Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams

LEONIDAS GUIBAS and JORGE STOLFI
Xerox Palo Alto Research Center and Stanford University

The following problem is discussed: Given n points in the plane (the sites) and an arbitrary query point q, find the site that is closest to q. This problem can be solved by constructing the Voronoi diagram of the given sites and then locating the query point in one of its regions. Two algorithms are given, one that constructs the Voronoi diagram in $O(n \log n)$ time, and another that inserts a new site in $O(n)$ time. Both are based on the use of the Voronoi dual, or Delaunay triangulation, and are simple enough to be of practical value. The simplicity of both algorithms can be attributed to the separation of the geometrical and topological aspects of the problem and to the use of two simple but powerful primitives, a geometric predicate and an operator for manipulating the topology of the diagram. The topology is represented by a new data structure for generalized diagrams, that is, embeddings of graphs in two-dimensional manifolds. This structure represents simultaneously an embedding, its dual, and its mirror image. Furthermore, just two operators are sufficient for building and modifying arbitrary diagrams.

Categories and Subject Descriptors: E.1 [**Data**]: Data Structures—*graphs*; E.2 [**Data**]: Data Storage Representations—*linked representations*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*curve, surface, solid, and object representations; geometric algorithms, languages, and systems*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Voronoi and Delaunay diagrams, closest point, nearest neighbors, point location, triangulations, representation of polyhedra, planar graphs, convex hull, geometric primitives, computational topology, Euler operators

INTRODUCTION

One of the fundamental data structures of computational geometry is the Voronoi diagram. This diagram arises from consideration of the following natural problem. Let n points in the plane be given, called *sites*. We wish to preprocess them into a data structure, so that given a new query point q, we can efficiently locate the nearest neighbor of q among the sites. The n sites in fact partition the plane into a collection of n regions, each associated with one of the sites. If region P is associated with site p, then P is the locus of all points in the plane closer to p than to any of the other $n-1$ sites. This partition is known as the *Voronoi* diagram (or the *Dirichlet*, or *Thiessen*, tesselation) determined by the given sites.

The closest site problem can therefore be solved by constructing the Voronoi diagram and then locating the query point in it. Using the currently best available algorithms, the Voronoi diagram of n points can be computed in $O(n \log n)$ time and stored in $O(n)$ space; these bounds have been shown to be optimal in the worst case [18]. Once we have the Voronoi diagram, we can construct in linear further time a structure with which we can do point location in a planar subdivision in $O(\log n)$ time [11].

Shamos [18] first pointed out that the Voronoi diagram can be used as a powerful tool to give efficient algorithms for a wide variety of other geometric problems. Given the Voronoi, we can compute in linear time the closest pair of sites, or the closest neighbor of each site, or the Euclidean minimum spanning tree of the n sites, or the largest point-free circle with center inside their convex hull, etc. Several of these problems are known to require $\Omega(n \log n)$ time in the worst case, so these Voronoi-based algorithms are asymptotically optimal.

Few of the previously published $O(n \log n)$ Voronoi algorithms [19] have been amenable to a practical implementation. The reasons have been varied, ranging from the complexity of the algorithms, to their insufficiently precise specification, to their improper handling of degenerate cases. For example, many of those algorithms may fail if the input includes four or more cocircular sites.

It turns out that the hardest part of constructing a Voronoi diagram is the determination of its topological structure, that is, the incidence relation between vertices, edges, and faces. Once the topological properties of the diagram are known, its geometrical properties (coordinates, lengths, angles, etc.) can be computed in time linear in the number of sites. Boots [2, 20] was apparently the first to observe that the computation of a Voronoi diagram can be greatly simplified by working with its dual, which is known as the *Delaunay diagram* of the given sites. This allows a cleaner separation between the topological and geometrical aspects of the problem. In this paper we push further in this direction, aiming for conciseness and completeness at the same time. The result is a one-page description of an $O(n \log n)$ algorithm that can be translated almost mechanically into any typical high-level language and correctly handles degenerate cases. For completeness, we apply the same methodology to a simpler (but asymptotically slower) incremental algorithm due to Green and Sibson [7].

Our algorithms are built using essentially two primitives: a geometric predicate and a topological operator for manipulating the structure of the diagrams. The geometrical primitive, which we call the InCircle test, encapsulates the essential geometric information that determines the topological structure of the Voronoi diagram and is a powerful tool not only in the coding of the algorithms but also in proving their correctness. As evidence for its importance, we show that it possesses many interesting properties and can be defined in a number of equivalent ways.

The topological structure of a Voronoi or Delaunay diagram is equivalent to that of a particular embedding of some undirected graph in the Euclidean plane.

One of the authors of this paper is an Associate Editor of Transactions on Graphics; he was not involved in the editorial decision process that resulted in acceptance of this paper for publication.

The work of J. Stolfi, who is on leave from the University of São Paulo, São Paulo, Brazil, was partially supported by a grant from Conselho Nacional de Desenvolvimento Científico e Tecnológico.

Authors' present address: Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301.

ACM Transactions on Graphics, Vol. 4, No. 2, April 1985, Pages 74–123.

We have found it convenient to consider such diagrams as being drawn on the sphere rather than on the plane; topologically that is equivalent to augmenting the Euclidean plane by a dummy *point at infinity*. This allows us to represent such things as infinite edges and faces in the same way as their finite counterparts. In Sections 1–5 we will establish the mathematical properties of such embeddings, define a notation for talking about them, and describe a data structure for their representation.

It turns out that the data structure we propose is general enough to allow the representation of undirected graphs embedded in arbitrary two-dimensional manifolds. In fact, it may be seen as a variant of the "winged edge" representation for polyhedral surfaces [1]. We show that a single topological operator, which we call *Splice*, together with a single primitive for the creation of isolated edges, is sufficient for the construction and modification of arbitrary diagrams. Our data structure has the ability to represent simultaneously and uniformly the primal, the dual, and the mirror-image diagrams, and to switch arbitrarily from one of these domains to another, in constant time. Finally, the design of the data structure enables us to manipulate its geometrical and topological parameters independently of each other. As it will become clear in the sequel, these properties have the effect of producing programs that are at once simple, elegant, efficient from a practical point of view, and asymptotically optimal in time and space.

Since this paper is quite long, some guidance to the forthcoming sections may be advisable. Section 1 introduces the concept of a simple subdivision of a manifold and discusses some of the conventions we adopt as compared to the extant literature. Section 2 develops a notation for expressing relationships between elements of a subdivision and explores its properties. Section 3 defines the important concept of an edge algebra, a combinatorial structure on the edges of the subdivision that we claim captures all topological properties of the latter. We spend most of Section 3 proving this claim, by showing that isomorphism of edge algebras is equivalent to topological homeomorphism between the corresponding subdivisions. The proof is somewhat technical and may be omitted on a first reading. In Section 4 we present a computer representation for an edge algebra, which is our *quad-edge* data structure. Section 5 introduces the topological primitives that we use to manipulate this structure and discusses their properties and implementation. Section 6 tailors these primitives to the application on hand, namely, the construction of Delaunay/Voronoi diagrams. Section 7 reviews some properties of such diagrams, and Section 8 presents our main geometric for their computation, the InCircle test. Section 9 describes a divide-and-conquer algorithm for Voronoi computations, and Section 10 presents an incremental version that is slower but simpler.

1. SUBDIVISIONS

In this section we give a precise definition for the informal concept of an embedding of an undirected graph on a surface. Special instances of this concept are sometimes referred to as a subdivision of the plane, a generalized polyhedron, a two-dimensional diagram, or by other similar names. They have been extensively discussed in the solid modeling literature of computer graphics [1, 15]. We want a definition that accurately reflects the topological properties one would

intuitively expect of such embeddings (for instance, that every edge is on the boundary of two faces, every face is bounded by a closed chain of edges and vertices, every vertex is surrounded by a cyclical sequence of faces and edges, etc.) and at the same time is as general as possible and leads to a clean theory and data structure.

We assume the reader is familiar with a few basic concepts of point-set topology, such as topological space, continuity, and homeomorphism [9]. Two subsets A and B of a topological space M are said to be *separable* if some neighborhood of A is disjoint from some neighborhood of B; otherwise, they are said to be *incident* on each other. A *line* of M is a subspace of M homeomorphic to the open interval $\mathbf{B}_1 = (0\ 1)$ of the real line. A *disk* of M is a subspace homeomorphic to the open circle of unit radius $\mathbf{B}_2 = \{x \in \mathbf{R}^2\colon |x| < 1\}$. Recall that a *two-dimensional manifold* is a topological space with the property that every point has an open neighborhood which is a disk (all manifolds in this paper will be two dimensional).

Definition 1.1. A *subdivision* of a manifold M is a partition S of M into three finite collections of disjoint parts, the *vertices*, the *edges*, and *faces* (denoted, respectively, by $\mathcal{V}\mathcal{S}$, $\mathcal{E}\mathcal{S}$, and $\mathcal{F}\mathcal{S}$), with the following properties:

S1. Every vertex is a point of M.
S2. Every edge is a line of M.
S3. Every face is a disk of M.
S4. The boundary of every face is a closed path of edges and vertices.

The vertices, edges, and faces of a subdivision are called its *elements*. Figure 1 shows some examples of subdivisions.

Condition S4 needs some explanation. We denote by $\mathbf{B}_2^*$ the *closed* circle of unit radius, and by $\mathbf{S}_1$ its circumference. Let us define a *simple path* in $\mathbf{S}_1$ as a partition of $\mathbf{S}_1$ into a finite sequence of isolated points and open arcs. The precise meaning of S4 is then the following: Every face F there is a simple path π in $\mathbf{S}_1$ and a continuous mapping ϕ_F from $\mathbf{B}_2^*$ onto the closure of F that (i) maps homeomorphically $\mathbf{B}_2$ onto F, (ii) maps homeomorphically each arc of π into an edge of S, and (iii) maps each isolated point of π to a vertex of S.

Condition S4 has a number of important implications. Clearly the points and arcs of π must alternate as we go around $\mathbf{S}_1$; if α is the arc between two consecutive points a and b of π, then its image $\phi_F(\alpha)$ is an edge incident to the points $\phi_F(a)$ and $\phi_F(b)$. Therefore, the images of the elements of π, taken in the order in which they occur around $\mathbf{S}_1$, constitute a closed, connected path π_F of edges and vertices of S, whose union is the boundary of F. Notice that the bounding path π_F need not be simple, since ϕ_F may take two or more distinct arcs or points of π to the same element of S. Therefore the closure of a face may not be homeomorphic to a disk, as Figure 1 shows.

Since it is impossible to cover a disk with only a finite number of edges and vertices, every edge and every vertex in a subdivision of a manifold must be incident to some face. Using condition S4 we conclude that every edge is entirely contained in the boundary of some face, and that it is incident to two (not necessarily distinct) vertices of S. These vertices are called the *endpoints* of the

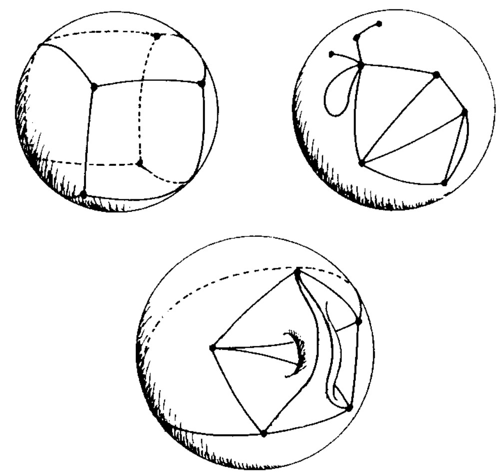

Fig. 1. Examples of subdivisions.

edge; if they are the same, then the edge is a *loop*, and its closure is homeomorphic to the circle $\mathbf{S}_1$.

Since every element of S is in the closure of some face, and since the closed disk $\mathbf{B}_2^\circ$ is compact, the manifold M is the union of a finite number of compact sets and therefore is itself compact. In fact, condition S4 can be replaced by the requirement that M be compact, that the edges be pairwise separable, and that every vertex is incident to some edge. Furthermore, every compact manifold has a subdivision. We will not attempt to prove these statements, since they are too technical for the scope of this paper.

Informally speaking, a compact two-dimensional manifold is a surface that closes upon itself, has no boundary, and in which every infinite sequence has an accumulation point. The sphere, the torus, and the projective plane are such manifolds; the disk, the line segment, the whole plane, and the Möbius strip are not. The compactness condition is not as restrictive as it may seem; most surfaces of practical interest can be transformed into a compact manifold by the addition of a finite number of dummy faces, edges, and vertices. In particular, the addition of a single "point at infinity," which by definition is an accumulation point of all sequences with no other accumulation points, transforms the Euclidean plane $\mathbf{R}^2$ into the *extended plane*, which is homeomorphic to the sphere.

1.1 Equivalence and Connectivity

Definition 1.2. Let S and S' be two subdivisions of the manifolds M and M'. An *isomorphism* from S to S' is a homeomorphism of M onto M' that maps each element of S onto an element of S'. When such a mapping exists, we say that S and S' are *equivalent*, and we write $S \sim S'$.

Such an isomorphism will perforce map vertices into vertices, faces into faces, and edges into edges, and will preserve the incidence relationships among them.

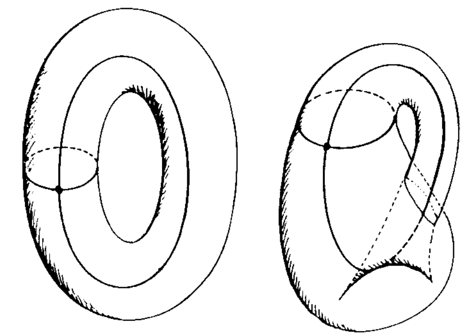

Fig. 2. A pair of noneqivalent subdivisions that have isomorphic graphs.

A *topological property* of subdivisions is a property that is invariant under equivalence. Our goal will be to develop a computer representation that fully captures all topological properties of subdivisions.

The collection of all edges and vertices of a subdivision S constitutes an undirected graph, the *graph of* S. The graphs of two equivalent subdivisions S and S' are obviously isomorphic. The converse is not always true: if S and S' have isomorphic graphs, it does not follow that they are equivalent, or that M and M' are homeomorphic. Figure 2 shows an example. Note that the subdivisions are not equivalent even though there also is a one-to-one correspondence between the faces of S and S' with the property that corresponding faces are incident to corresponding edges and vertices. This example shows that the *set* of edges and vertices on the boundary of a face is not enough information to characterize its relationship to the rest of the manifold.

This fact is the main source of complexity in the theoretical treatment of subdivisions, notably in the proof that our data structure is a consistent representation of a general subdivision. It is possible to define subdivisions in such a way that their topological structure is completely determined by that of their graphs. For example, if the manifold is restricted to be a sphere and the graph is triply connected [8], then the subdivision is determined up to equivalence. However, any set of conditions strong enough to achieve this goal would probably outlaw "degeneracies" such as loops, multiple edges with the same endpoints, faces with nonsimple boundaries, and so forth. Subdivisions with such degeneracies are much more common than it may seem: they inevitably arise as intermediate objects in the transformation of a "well-behaved" subdivision into another. An even stronger reason for adopting our liberal Definition 1.1 is that it leads to more flexible data structures and simpler atomic operations with weaker preconditions.

On the other hand, we depart from the common solid modeling tradition by insisting that every face be a simple disk, without "handles" or "holes," even though the whole manifold is allowed to have arbitrary connectivity. The main reason for this requirement is to enable a clean and unambiguous definition of the dual subdivision (see Section 2.2). One important consequence of this restriction is the following.

THEOREM 1.1. *The graph of a simple subdivision is connected iff the manifold is connected.*

PROOF. Since every face is incident to some edge, if the graph is connected, then the whole manifold is too. Now assume that the graph is not connected, but the manifold is. Since the faces are pairwise separable and their addition to the graph makes it connected, some face is incident to two distinct components of the graph. By condition S4 the boundary of that face is connected, a contradiction. □

Therefore, the connected components of the manifold are in one-to-one correspondence with the connected components of the underlying graph.

2. EDGE FUNCTIONS AND THEIR PROPERTIES

In this section we develop a convenient notation for describing relationships among edges of a subdivision and a mathematical framework that will justify the choice of our data structure. We first develop the theory and representation for arbitrary compact manifolds, and then we show that certain important simplifications can be made in the particular case in which the manifold is orientable. For many applications, including the computation of Voronoi diagrams, the only relevant manifold will be the extended plane.

2.1 Basic Edge Functions

On any disk D of a manifold there are exactly two ways of defining a local "clockwise" sense of rotation; these are called the two possible *orientations on* D. An *oriented element* of a subdivision P is an element x of P together with an orientation of a disk containing x. There are also exactly two consistent ways of defining a linear order among the points of a line l; each of these orderings is called a *direction along* l. A *directed edge* of a subdivision P is an edge of P together with a direction along it. Since directions and orientations can be chosen independently, for every edge of a subdivision there are four directed, oriented edges. Observe that this is true even if the edge is a loop or is incident twice to the same face of P.

For any oriented and directed edge e we can define unambiguously its vertex of *origin*, *e Org*, its *destination*, *e Dest*, its *left face*, *e Left*, and its *right face*, *e Right*. We define also the *flipped* version *e Flip* of an edge e as being the same unoriented edge taken with *opposite orientation* and same direction, as well as the *symmetric* of e, *e Sym*, as being the same undirected edge with the *opposite direction* but the same orientation as e. We can picture the orientation and direction of an edge e as a small bug sitting on the surface over the midpoint of the edge and facing along it. Then the operation *e Sym* corresponds to the bug making a half turn on the same spot, and *e Flip* corresponds to the bug hanging upside down from the other side of the surface, but still at the same point of the edge and facing the same way.

The elements *e Org*, *e Left*, *e Right*, and *e Dest* are taken by definition with the orientation that agrees locally with that of e. More precisely, the orientation of *e Org* agrees with that of some initial segment of e, and that of *e Dest* agrees with some final segment of e. Note that for some loops *e Org* and *e Dest* may have

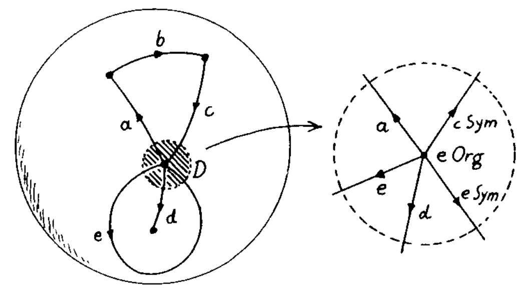

Fig. 3. The ring of edges out of a vertex.

opposite orientations, in spite of being the same (unoriented) vertex. Similarly, the orientation of *e Left* agrees with e along the "left margin" of e, and that of *e Right* agrees along its "right margin." If e is a bridge in the graph of P, it may be the case that *e Left* and *e Right* have different orientations, in spite of being the same (unoriented) face. By extending our previous notation, we denote by $\mathcal{V}S$, $\mathcal{E}S$, and $\mathcal{F}S$ the sets of directed and oriented elements of a subdivision S. In the rest of this section, unless otherwise specified, all subdivision elements are assumed to be oriented, and directed if edges.

Consider an edge e incident to an oriented vertex v. If the edge is not a loop, then there is a natural way to extend the orientation of v into an orientation of e. This may not be possible if e is a loop and the manifold is nonorientable. However, given a sufficiently small disk D containing v, we can always extend the orientation of v to each portion of e inside that disk. If small enough, D can be mapped homeomorphically onto the unit disk $\mathbf{B}_2$ in such a way that v is mapped to the origin, and the intersection of D with every edge incident to v is a line of M that is mapped to a radius of $\mathbf{B}_2$. These edge fragments can be oriented consistently with v and directed away from v. Traversing the boundary of D in the counterclockwise direction (as defined by the orientation of v) establishes a cyclical ordering of the fragments. If for each fragment we take the corresponding edge, with orientation and direction as specified by the fragment, we obtain what is called the *ring of edges out of* v. Note that if e is a loop, it will occur twice in the ring of edges out of v. To be precise, both e and an oppositely directed version of it (either *e Sym* or *e Sym Flip*) will occur once each: since the manifold around v is like a disk, e will appear only once in each circuit, and we will never encounter *e Flip*.

We can define the *next edge with same origin*, *e Onext*, as the one immediately following e (counterclockwise) in this ring (see Figure 3). Similarly, given an edge e we define the *next counterclockwise edge with same left face*, denoted by *e Lnext*, as being the first edge we encounter after e when moving along the boundary of the face $F = e\ Left$ in the counterclockwise sense as determined by the orientation of F. The edge *e Lnext* is oriented and directed so that *e Lnext Left* $= F$ (including orientation). The successive images of e under *Lnext* give precisely the edges of the bounding path π_F of condition S4 (in one of the two possible orders). As in the case of *Onext*, the edge e appears exactly once in this list, and either *e Sym* or *e Flip* (but not *e Sym Flip*) may appear once.

2.2 Duality

The dual of a planar graph G can be informally defined as a graph G^* obtained from G by interchanging vertices and faces while preserving the incidence relationships. The definition below extends this intuitive concept to arbitrary subdivisions.

Definition 2.1. Two subdivisions S and S^* are said to be *dual* of each other if for every directed and oriented edge e of either subdivision there is another edge e *Dual* of the other such that

D1. $(e\ Dual)\ Dual = e$.
D2. $(e\ Sym)\ Dual = (e\ Dual)\ Sym$.
D3. $(e\ Flip)\ Dual = (e\ Dual)\ Flip\ Sym$.
D4. $(e\ Lnext)\ Dual = (e\ Dual)\ Onext^{-1}$.

Equation D4 states that moving counterclockwise around the left face of e in one subdivision is the same as moving clockwise around the origin of (e *Dual*) in the other subdivision. To see why, note that the edges on the boundary of the face $F = e\ Left$, in counterclockwise order, are

$$\langle e\ Lnext,\ e\ Lnext^2, \ldots, e\ Lnext^m = e \rangle$$

for some $m \geq 1$. This path maps through *Dual* to the sequence

$$\langle (e\ Dual)\ Onext^{-1},\ (e\ Dual)\ Onext^{-2}, \ldots, (e\ Dual)\ Onext^{-m} = e\ Dual \rangle$$

of all edges coming out of the vertex $v = (e\ Dual)\ Org$ of S^*, in clockwise order around v.

We can therefore extend *Dual* to vertices and faces of the two subdivisions by defining $(e\ Left)\ Dual = (e\ Dual)\ Org$ and $(e\ Org)\ Dual = (e\ Dual)\ Left$. Equations D2 and D3 imply that any two edges that differ only in orientation and direction will be mapped to two versions of the same undirected edge. Combining this with the preceding argument we conclude that *Dual* establishes a correspondence between $\mathcal{E}\mathcal{S}$ and $\mathcal{E}\mathcal{S}^*$, between $\mathcal{V}\mathcal{S}$ and $\mathcal{F}\mathcal{S}^*$, and between $\mathcal{F}\mathcal{S}$ and $\mathcal{V}\mathcal{S}^*$, such that incident elements of S correspond to incident elements of S^*, and vice versa. It follows that two vertices of one subdivision are connected by an edge whenever (and as many times as) the corresponding faces of the other are incident to a common edge. So, in the particular case when S and S^* are subdivisions of the sphere, the graphs of S and S^* are duals of each other in the sense of graph theory.

Figure 4 shows a subdivision of the extended plane (solid lines) superimposed on its dual (dotted lines). Note that the two subdivisions of Figure 4 have the property that each undirected edge of one meets (and crosses) only the corresponding dual edge of the other, and that each vertex of one is in the corresponding dual face of the other. When this happens, we say that S and S^* are *strict duals* of each other. In that case, the dual of an oriented and directed edge e is the edge of the dual subdivision that crosses e from left to right, but taken with orientation *opposite* to that of e. That is, the dual subdivision should be looked from the other side of the manifold, or the manifold should be turned inside out. This reflects the correspondence between counterclockwise traversal of $e\ Left$ to clockwise traversal of $(e\ Dual)\ Org$.

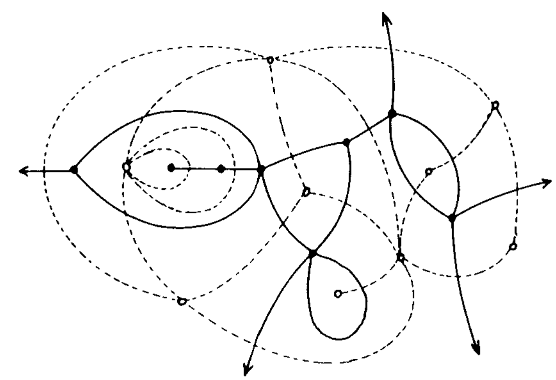

Fig. 4. A subdivision of the extended plane (solid lines) and a strict dual (dashed lines).

This implicit "flipping" of the manifold is unavoidable if S and S^* are superimposed as strict duals and we insist that *Dual* be its own inverse. It has the serious drawback of making the calculus of the edge functions much less intuitive. It is therefore preferable to relate the two dual subdivisions by means of the function

$$e\ Rot = e\ Flip\ Dual = e\ Dual\ Flip\ Sym,$$

which maps $\mathcal{E}\mathcal{S}$ to $\mathcal{E}\mathcal{S}^*$ without this implicit "flipping." The edge $e\ Rot$ is called the *rotated* version of e; it is the dual of e, directed from $e\ Right$ to $e\ Left$ and oriented so that moving counterclockwise around the right face of e corresponds to moving counterclockwise around the origin of $e\ Rot$. If the two subdivisions are superimposed as strict duals, as in Figure 4, then we may say that $e\ Rot$ is e "rotated 90° counterclockwise" around the crossing point. In fact, the only reason for not defining duality in terms of *Rot* (rather than *Dual*) is that it falls short of being its own inverse: $(e\ Rot)\ Rot$ gives $e\ Sym$ instead of e.

2.3 Properties of Edge Functions

The functions *Flip*, *Rot*, and *Onext* satisfy the following properties:

E1. $e\ Rot^4 = e$.
E2. $e\ Rot\ Onext\ Rot\ Onext = e$.
E3. $e\ Rot^2 \neq e$.
E4. $e \in \mathcal{E}S$ iff $e\ Rot \in \mathcal{E}S^*$.
E5. $e \in \mathcal{E}S$ iff $e\ Onext \in \mathcal{E}S$.
F1. $e\ Flip^2 = e$.
F2. $e\ Flip\ Onext\ Flip\ Onext = e$.
F3. $e\ Flip\ Onext^n \neq e$ for any n.
F4. $e\ Flip\ Rot\ Flip\ Rot = e$.
F5. $e \in \mathcal{E}S$ iff $e\ Flip \in \mathcal{E}S$.

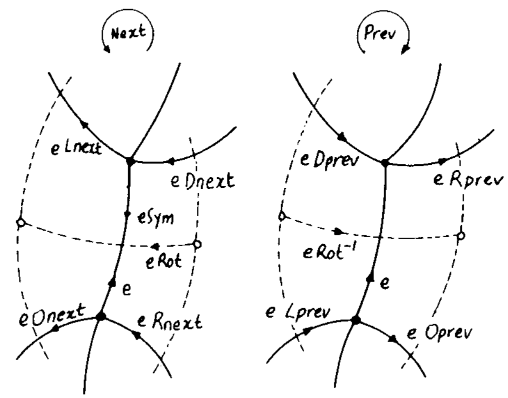

Fig. 5. The edge functions.

A number of useful properties can be deduced from these, as for example

$$\begin{aligned} e\ Flip^{-1} &= e\ Flip, \\ e\ Sym &= e\ Rot^2, \\ e\ Rot^{-1} &= e\ Rot^3 = e\ Flip\ Rot\ Flip, \\ e\ Dual &= e\ Flip\ Rot, \\ e\ Onext^{-1} &= e\ Rot\ Onext\ Rot = e\ Flip\ Onext\ Flip, \end{aligned} \tag{1}$$

and so forth. For added convenience in talking about subdivisions, we introduce some derived functions. By analogy with *e Lnext* and *e Onext*, for a given *e* we define the *next edge with same right face*, *e Rnext*, and *with same destination*, *e Dnext*, as the first edges that we encounter when moving counterclockwise from *e* around *e Right* and *e Dest*, respectively. These functions satisfy also the following equations:

$$\begin{aligned} e\ Lnext &= e\ Rot^{-1}\ Onext\ Rot, \\ e\ Rnext &= e\ Rot\ Onext\ Rot^{-1}, \\ e\ Dnext &= e\ Sym\ Onext\ Sym. \end{aligned} \tag{2}$$

The orientation and direction of these edges is defined so that *e Lnext Left* = *e Left*, *e Rnext Right* = *e Right*, and *e Dnext Dest* = *e Dest*. Note that *e Rnext Dest* = *e Org*, rather than vice versa. By moving *clockwise* around a fixed endpoint or face, we get the inverse functions, defined by

$$\begin{aligned} e\ Oprev &= e\ Onext^{-1} = e\ Rot\ Onext\ Rot, \\ e\ Lprev &= e\ Lnext^{-1} = e\ Onext\ Sym, \\ e\ Rprev &= e\ Rnext^{-1} = e\ Sym\ Onext, \\ e\ Dprev &= e\ Dnext^{-1} = e\ Rot^{-1}\ Onext\ Rot^{-1}. \end{aligned} \tag{3}$$

It is important to notice that every function defined so far (except *Flip*) can be expressed as the composition of a constant number of *Rot* and *Onext* operations, independently of the size or complexity of the subdivision. Figure 5 illustrates these various functions.

3. EDGE ALGEBRAS

In this section we develop the notion of an *edge algebra*, a finite combinatorial object that we prove accurately captures all the topological properties of a subdivision. Edge algebras will be the basis of our data structure for representing subdivisions.

Definition 3.1. An *edge algebra* is an abstract algebra (E, E^*, *Onext*, *Rot*, *Flip*) where E and E^* are arbitrary finite sets and *Onext*, *Rot*, and *Flip* are functions on E and E^* satisfying properties E1–E5 and F1–F5.

An edge algebra represents simultaneously a pair of dual subdivisions; as we remarked before, this allows us to express all our edge functions in terms of only three basic primitives, *Flip*, *Rot*, and *Onext*. Other advantages of this primal/dual representation will be encountered later on, and we will see that they are obtained at a negligible cost in storage and time.

Axioms E1–F5 imply that *Rot* is a bijection from E to E^* and from E^* to E. Also, *Flip* and *Onext* each define permutations acting on E and E^* separately. We define *e Org* in an edge algebra as the orbit of *e* under *Onext*, that is, the cyclic sequence of edges

$$\langle \ldots,\ e,\ e\,Onext,\ e\,Onext^2,\ \ldots,\ e\,Onext^{-1},\ e,\ \ldots \rangle.$$

Note that *e Flip Org* is the sequence obtained by *Flip*ping each element of *e Org* and listing them in reverse order, that is,

$$\begin{aligned} e\,Flip\,Org &= \langle \ldots,\ e\,Flip,\ e\,Flip\,Onext,\ e\,Flip\,Onext^2,\ \ldots, \\ &\qquad e\,Flip\,Onext^{-1},\ e\ \ldots \rangle \\ &= \langle \ldots,\ e\,Flip,\ e\,Onext^{-1}\,Flip,\ e\,Onext^{-2}\,Flip,\ \ldots, \\ &\qquad e\,Onext\,Flip,\ e\,Flip,\ \ldots \rangle. \end{aligned}$$

Similarly, we define *e Left* = *e Rot*$^{-1}$*Org*, *e Right* = *e Rot Org*, and *e Dest* = *e Sym Org*. We also take eqs. (2) as the definition of the functions *Lnext*, *Rnext*, and *Dnext* for arbitrary edge algebras. From the axioms it follows that *Onext* and these derived functions have inverses, which we denote by *Lprev*, *Rprev*, *Dprev*, and *Oprev* and which can be shown to satisfy eqs. (3).

3.1 Completions

We now proceed to show that the topology of a subdivision is completely determined by its edge algebra, and vice versa. To prove this thesis, we will show that a general subdivision S can be fully characterized by the graph of a standard refinement of S, which in turn is closely related to the edge algebra of S. The concepts and theorems developed in the rest of Section 3 are essential for showing the consistency and completeness of our proposed data structure but are not used in the rest of the paper. The reader whose interest is mostly practical may skip to Section 4.

Definition 3.2 Let S and Σ be subdivisions of a manifold M. We say that Σ is a *completion* of S if it is a refinement of S obtained by adding one vertex v_e on each edge e and one vertex v_F in each face F and then connecting v_F by new edges to every vertex (old or new) on the boundary of F.

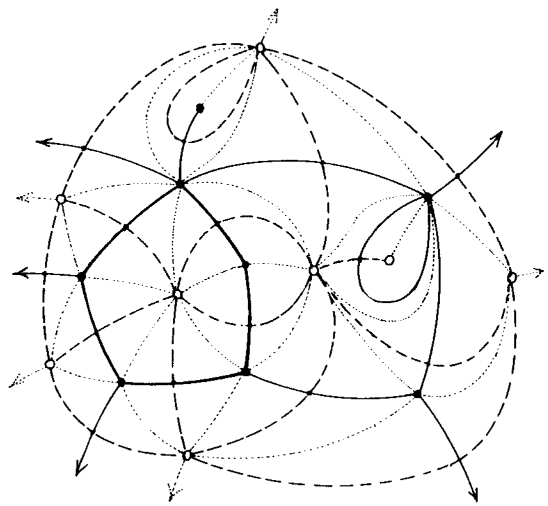

Fig. 6. A completion of the extended plane showing primal links (solid), dual links (dashed), and skew links (dotted).

The vertices of Σ are called *primal*, *crossing*, or *dual*, depending on whether they lie on vertices, edges, or faces of S; they are denoted by $\mathcal{V}\Sigma$, $\mathcal{E}\Sigma$, and $\mathcal{V}^*\Sigma$, respectively. Every edge of S is split by its crossing vertex in two *primal links* of Σ; the new edges added in each face are called *dual links* if they connect a dual vertex to a crossing point and *skew links* if they connect a dual vertex to a primal one. These links are denoted $\mathcal{S}\Sigma$, $\mathcal{S}^*\Sigma$, and $\mathcal{K}\Sigma$, in that order. Figure 6 shows a completion of a subdivision of the extended plane.

Definition 3.2 must be understood appropriately in the case of a face F whose bounding path π_F is not simple. If π_F passes k times through a vertex or crossing point p, then p is to be connected to v_F by exactly k new links, and their order around v_F should be the same as the order of the crossings on π_F. To describe this process precisely, let ϕ_F be any continuous function from $\mathbf{B}_2^*$ to the closure of F that establishes condition S4. Let $\pi = \langle u_1, \alpha_2, u_2, \alpha_2, \ldots, u_n, \alpha_n, u_{n+1} = u_1 \rangle$ be the path in the circle $\mathbf{S}_1$ that is mapped to π_F by φ_F; in each arc α_i there is a point c_i that is mapped to the crossing vertex of the edge $\varphi_F(\alpha_i)$. Take $\varphi_F((0, 0))$ to be the dual vertex v_F; connect in $\mathbf{B}_2^*$ the origin $(0, 0)$ to each u_i and to each c_i by a straight line segment, and let the images of these segments under φ_F be respectively the dual and skew links for the face F. Note that the restriction of faces to simple disks is essential for a simple and unambiguous definition of the completion.

From the definition, it is clear that every subdivision has at least one refinement which is a completion. Every face of Σ consists of three vertices and three links, one of each kind, and therefore all distinct. An important consequence is that the closure of each face is homeomorphic to (not just the continuous image of) the sector of $\mathbf{B}_2^*$ bounded by two rays and an arc u_ic_i or c_iu_{i+1}, which in turn is homeomorphic to a disk. In fact, the closure of a face of Σ is homeomorphic to any planar triangle, with each corner mapping to a vertex and each side to a link.

For this reason, we will refer to the faces of Σ as *triangles* and denote them by $\mathcal{T}\Sigma$.

It is also apparent from the definition that every edge of Σ has two distinct endpoints and is incident to exactly two triangles (which may or may not lie in the same face of S). A completion may have more than one link connecting any given pair of vertices, but it has no loops. Every crossing vertex c_e is incident to exactly four links, two primal and two dual, and to four distinct triangles. The vertex c_e and these eight elements constitute a disk of M that contains the edge e. It can be seen also that, given a primal link l and a dual link l^* that are incident to the same (crossing) vertex, there is exactly one triangle that is incident to both l and l^*.

We consider the distinction among primal, dual, and crossing vertices to be an integral part of the description of Σ, so S is uniquely determined by it. We call S the *primal subdivision* of Σ, denoted by $S\Sigma$. In the same spirit, we say that two completions Σ_1 and Σ_2 are equivalent only if there is a homeomorphism that maps each element of Σ_1 to an element of Σ_2, takes $\mathcal{V}\Sigma_1$ to $\mathcal{V}\Sigma_2$, and takes $\mathcal{V}^*\Sigma_1$ to $\mathcal{V}^*\Sigma_2$. Such an homeomorphism will clearly take $\mathcal{E}\Sigma_1$, $\mathcal{S}\Sigma_1$, $\mathcal{S}^*\Sigma_1$, and $\mathcal{K}\Sigma_1$ to the corresponding components of Σ_2.

3.2 Existence of Duals and Algebras

As it was defined, the edge algebra of a subdivision S seems to depend not only on S itself, but also on the choice of a dual subdivision S^*, and of the function *Dual*(or *Rot*) that connects the two. The first part of our theoretical justification is the proof that such S^* and *Dual* always exist and that the edge functions of S and S^* satisfy axioms E1–E5 and F1–F5.

Let Σ be a completion on a manifold M. For every crossing c_e of Σ, define the *dual* of the (unoriented and undirected) edge e of $S\Sigma$ as the set $e^* = l_1 \cup \{c_e\} \cup l_2$, where l_1, l_2 are the two dual links incident to c_e. Denote by $\mathcal{E}^*\Sigma$ the set of all such objects. Define the *dual* F_v^* of a primal vertex v as the union of $\{v\}$ and all elements of Σ incident to v. Let $\mathcal{F}^*\Sigma$ be the set of all those objects.

LEMMA 3.1. *The triplet* $S^*\Sigma = (\mathcal{V}^*\Sigma, \mathcal{E}^*\Sigma, \mathcal{F}^*\Sigma)$ *is a subdivision of* M.

PROOF. Besides v itself, the dual F_v^* of a vertex v contains only triangles, primal links, and skew links incident to v. Each link of F_v^* is incident to exactly two distinct triangles of F_v^*, and conversely each triangle is incident to two distinct links of F_v^*, one primal and one skew. Therefore, these links and triangles can be arranged in one or more sequences (without repetitions) $\langle l_1, t_1, l_2, t_2, \ldots, l_n, t_n, l_{n+1} = l_1 \rangle$, where the t_i are triangles, the l_i are alternately primal and skew links, and each t_i is incident to l_i and to l_{i+1}. Each such sequence plus v is a disk containing v; since M is a manifold, there can be only one such disk.

We conclude that F_v^* is a disk of M. Furthermore, it is clear that we can construct a continuous function ϕ from the closed ball onto the closure F_v^* that establishes condition S4. Since a triangle or primal link cannot be incident to two distinct primal vertices, the elements of $\mathcal{F}^*\Sigma$ are pairwise disjoint. Clearly the elements of $\mathcal{E}^*\Sigma$ are lines of M that are pairwise disjoint and also disjoint from the members of $\mathcal{F}^*\Sigma$ and $\mathcal{V}^*\Sigma$. Therefore, $S^*\Sigma$ is a subdivision of M. □

Definition 3.3. Let Σ be a completion. Let *Rot* be the function from $\mathscr{E}S\Sigma \cup \mathscr{E}S^*\Sigma$ into itself defined as follows. For every edge $e \in \mathscr{E}S\Sigma$, let $e\,Rot$ be the dual edge e^* of $S^*\Sigma$, directed so as to cross e from right to left and oriented so as to agree with the orientation of e at the crossing point. Similarly, for each element $e \in \mathscr{E}S^*\Sigma$ let $e\,Rot$ be the edge of $S\Sigma$ of which e is the dual, directed and oriented according to the same rules with respect to e. The *standard edge algebra of* Σ is by definition $A\Sigma = (\mathscr{E}S\Sigma, \mathscr{E}S^*\Sigma, Onext, Rot, Flip)$.

THEOREM 3.2. *The standard edge algebra $A\Sigma$ of any completion Σ satisfies axioms E1–E5 and F1–F5, and $S^*\Sigma$ is a (strict) dual of $S\Sigma$.*

PROOF. Each oriented and directed edge e of $\mathscr{E}S\Sigma$ (or $\mathscr{E}S^*\Sigma$) can be represented unambiguously by a pair of links (e_o, e_r), where e_o is the origin half of e, and e_r is the dual (or primal) link of Σ that is incident to the crossing vertex of e and lies to its right. Conversely, any pair (x, y) of adjacent links (one primal and one dual) corresponds to a unique edge of $\mathscr{E}S\Sigma$ or $\mathscr{E}S^*\Sigma$.

For any link pair (x, y) of this kind there is a unique triangle T of Σ incident to x and y, and a unique triangle T' sharing a skew link with T. Let us call the *opposite* of the pair (x, y) the link pair (r, s) such that r and s are on the boundary of T' and are of the same sort (primal/dual) as x and y, respectively. Let x' denote the link of the same sort (primal/dual) as x and incident to the same crossing.

According to this notation, we have $(a, b)\,Flip = (a, b')$, $(a, b)\,Rot = (b, a')$, and $(a, b)\,Onext = (x, y)$, where (x, y) is opposite to (a, b'). Now it is easy to check that the algebra $A\Sigma$ satisfies E1–E5 and F1–F5. For example, let (x, y) be the opposite of (b, a); then (a, b) is the opposite of (y, x), and we have

$$\begin{aligned}(a, b)\,Rot\,Onext\,Rot\,Onext &= (b, a')\,Onext\,Rot\,Onext\\ &= (x, y)\,Rot\,Onext\\ &= (y, x')\,Onext\\ &= (a, b),\end{aligned}$$

and so forth. The function $e\,Dual = e\,Flip\,Rot$ satisfies D1–D4, since these conditions can be proved from E1–E5 and F1–F5. We conclude that $S\Sigma$ and $S^*\Sigma$ are (strict) duals of each other. □

For any subdivision S there is a completion Σ such that $S = S\Sigma$, and therefore a dual $S^*\Sigma$ and a valid edge algebra $A\Sigma$ that describes S (and $S^*\Sigma$).

3.3 Equivalence and Isomorphism

The second part of our argument shows that the edge algebra of a subdivision is determined up to isomorphism, and conversely the subdivision of an edge algebra is unique up to equivalence.

THEOREM 3.3. *Let $A_i\,(i = 1, 2)$ be an edge algebra for a pair of dual subdivisions S_i and S_i^*. If S_1 is equivalent to S_2, then A_1 and A_2 are isomorphic algebras.*

PROOF. Let $A_i = (\mathscr{E}S_i, \mathscr{E}S_i^*, Onext_i, Rot_i, Flip_i)$, and let h be the homeomorphism between the manifolds of S_1 and S_2 that establishes $S_1 \sim S_2$. An orientation or direction for an element of S_1 determines via h a unique orientation or direction for the corresponding element in S_2 and therefore defines also a one-to-one correspondence η between $\mathscr{E}S_1$ and $\mathscr{E}S_2$. From the definition of *Onext* we can conclude that $\eta(e\,Onext_1) = \eta(e)\,Onext_2$ for all $e \in \mathscr{E}S_1$; the same holds for *Sym* and *Flip*.

Let us now define the function ξ from $\mathscr{E}S_1 \cup \mathscr{E}S_1^*$ to $\mathscr{E}S_2 \cup \mathscr{E}S_2^*$ as

$$\xi(e) = \begin{cases}\eta(e) & \text{if } e \in \mathscr{E}S_1,\\ \eta(e\,Rot_1^{-1})\,Rot_2 & \text{if } e \in \mathscr{E}S_1^*.\end{cases}$$

Clearly ξ is one-to-one, for Rot_i is one-to-one from $\mathscr{E}S_i$ to $\mathscr{E}S_i^*$.

We prove that $\xi(e\,Rot_1) = \xi(e)\,Rot_2$ as follows. If $e \in \mathscr{E}S_1$, we have $e\,Rot_1 \in \mathscr{E}S_1^*$, so

$$\xi(e\,Rot_1) = \eta(e\,Rot_1\,Rot_1^{-1})\,Rot_2 = \eta(e)\,Rot_2 = \xi(e)\,Rot_2.$$

If $e \in \mathscr{E}S_1^*$, then $e\,Rot_1 \in \mathscr{E}S_1$, and so

$$\begin{aligned}\xi(e\,Rot_1) &= \eta(e\,Rot_1) = \eta(e\,Rot_1^{-1}\,Sym_1)\\ &= \eta(e\,Rot_1^{-1})\,Sym_2 = \eta(e\,Rot_1^{-1})\,Rot_2\,Rot_2\\ &= \xi(e)\,Rot_2.\end{aligned}$$

Let us now show that $\xi(e\,Onext_1) = \xi(e)\,Onext_2$. If $e \in \mathscr{E}\mathscr{S}$, the proof is trivial. If $e \in \mathscr{E}S_1^*$, then $e\,Onext_1 \in \mathscr{E}S_1^*$, and

$$\begin{aligned}\xi(e\,Onext_1) &= \eta(e\,Onext_1\,Rot_1^{-1})\,Rot_2\\ &= \eta(e\,Rot_1^{-1}\,Onext_1^{-1}\,Rot_1^{-1}\,Rot_1^{-1})\,Rot_2\\ &= \eta(e\,Rot_1^{-1}\,Onext_1^{-1}\,Sym_1)\,Rot_2\\ &= \eta(e\,Rot_1^{-1})\,Onext_2^{-1}\,Sym_2\,Rot_2 \qquad (\text{since } e\,Rot_1^{-1} \in \mathscr{E}S)\\ &= \eta(e\,Rot_1^{-1})\,Rot_2\,Onext_2\\ &= \xi(e)\,Onext_2.\end{aligned}$$

The proof for $\xi(e\,Flip_1) = \xi(e)\,Flip_2$ is entirely similar, using $e\,Flip_1 = e\,Rot_1^{-1}\,Flip_1\,Rot_1$. □

We say that two completions are *similar* if there is an isomorphism of the graph of Σ_1 to that of Σ_2 that takes primal vertices to primal vertices and dual vertices to dual vertices.

LEMMA 3.4. *Let Σ_1 and Σ_2 be two completions. If their edge algebras $A\Sigma_1$ and $A\Sigma_2$ are isomorphic algebras, then Σ_1 and Σ_2 are similar.*

PROOF. For any completion Σ, we establish one-to-one mappings between certain subsets of oriented and directed edges of the algebra $A\Sigma$ and the primal links, dual links, and vertices of Σ in the following way. To each primal (or dual) link l of Σ there corresponds a unique pair of primal (or dual) elements of $A\Sigma$ of the form $\{e, e\,Flip\}$; these elements are the directed and oriented edges of $S\Sigma$ (or $S^*\Sigma$) of which l is the "origin" half. To each primal vertex of Σ there corresponds an orbit of $A\Sigma$ under *Onext* and *Flip* (i.e., a set of the form $e\,Org \cup e\,Flip\,Org$ for some edge e); similarly, to each crossing of Σ, there corresponds an orbit of $A\Sigma$ under *Rot* and *Flip*. These mappings are one-to-one, and a primal or dual link of Σ is incident to a vertex if and only if the corresponding orbits in $A\Sigma$ intersect.

We also associate each skew link of Σ to a set of the form

$$\{e, e\,Flip, e\,Rot^{-1}, e\,Rot^{-1}\,Flip, f, f\,Flip, f\,Rot, f\,Rot\,Flip\},$$

where $f = e\,Onext$, in the following way. There are exactly two triangles of Σ incident to s, each incident also to a primal and to a dual link. We take s' to be the union of the four subsets of $A\Sigma$ that correspond to those four links. It is easy to check that these subsets have the form above, and that s is incident to a primal or dual vertex of Σ if and only if an element of s' intersects the orbit corresponding to that vertex. Conversely, every set of the form above determines a unique skew link by this rule.

The isomorphism between $A\Sigma_1$ and $A\Sigma_2$ maps those representative subsets of $A\Sigma_1$ to subsets of $A\Sigma_2$ having the same form, and therefore it establishes a one-to-one correspondence ξ between the primal (or dual) links and vertices of Σ_1 and those of Σ_2. Since intersecting subsets are mapped to intersecting subsets, ξ preserves incidence. We conclude that Σ_1 and Σ_2 are similar. □

LEMMA 3.5. *If two completions Σ_1 and Σ_2 are similar, then they are equivalent.*

PROOF. Let ξ be the isomorphism between the graphs of Σ_1 and Σ_2 that establishes their similarity. We will construct from it an homeomorphism η between the manifolds of the two completions that establishes their equivalence. First, we define η on the vertices of Σ_1 as being the same as ξ. For every link r of Σ_1 with endpoints u and v, we can always find an homeomorphism η_r from the closure of r to that of $\xi(r)$ that takes u to $\xi(u)$ and v to $\xi(v)$; we define $\eta(p) = \eta_r(p)$ for all points p of r. Clearly, η is an homeomorphism of the graph of Σ_1 onto that of Σ_2.

Since any pair of adjacent links of which one is primal and the other dual determines a unique triangle, the similarity of the two completions gives also a one-to-one correspondence between their triangles that preserves incidence. For each pair of corresponding triangles T and T' there is a homeomorphism η_T from the closure of T onto the closure of T' that agrees with η on the boundary of T; this follows readily from the fact that both closures are homeomorphic to closed disks. So η and all η_T constitute a finite collection of continuous maps of closed subsets of M into M', with the property that any two of them agree in the intersection of their domains. Their union η^* is therefore a continuous map from M into M'. Clearly, η^* is one-to-one and onto, so it is an homeomorphism. By construction, it maps elements of Σ_1 to elements of Σ_2. □

LEMMA 3.6. *If two completions Σ_1 and Σ_2 are equivalent, then so are $S\Sigma_1$ and $S\Sigma_2$.*

PROOF. Each face of $S\Sigma_i$ is the union of a dual vertex and all elements of Σ_i that are incident to it. Each edge of $S\Sigma_i$ is the union of a crossing and all (two) primal links of Σ_i incident to it. The homeomorphism η that establishes the equivalence of the two completions preserves incidence and the primal/dual character of links and vertices, so it maps elements of $S\Sigma_1$ *to* $S\Sigma_2$, establishing their equivalence. □

THEOREM 3.7. *Let A_1 and A_2 be edge algebras for two subdivisions S_1 and S_2. If A_1 and A_2 are isomorphic, then S_1 and S_2 are equivalent.*

PROOF. Let Σ_1 and Σ_2 be any two completions of S_1 and S_2. By Theorem 3.3 we have $A_1 \sim A\Sigma_1$ and $A_2 \sim A\Sigma_2$, and therefore $A\Sigma_1 \sim A\Sigma_2$. Then by Lemmas 3.4 and 3.5, the subdivisions Σ_1 and Σ_2 are equivalent; by Lemma 3.6 the same is true of S_1 and S_2. □

Therefore, the topological structure of a subdivision is completely and uniquely characterized by its edge algebra. An analogous theorem seems to have been discovered independently by Damphousse [5]. Theorems 3.3 and 3.7 also imply that all completions of a subdivision are equivalent and that two subdivisions are equivalent if and only if their duals are equivalent. Therefore, the dual of a simple subdivision is unique up to equivalence.

3.4 Realizability of Algebras

To conclude our theoretical justification, we will show that every edge algebra corresponds to a subdivision of some manifold. This fact is of great practical importance, for it guarantees that any modification to the data structure that preserves axioms E1–E5 and F1–F5 corresponds to a valid operation on manifolds.

THEOREM 3.8. *Every edge algebra can be realized by some subdivision.*

PROOF. Let $A = (E, E^*, Flip, Rot, Onext)$ be an edge algebra. We will prove this by constructing a completion Σ such that $A\Sigma$ is isomorphic to A. The manifold of Σ is constructed by taking a collection of disjoint closed triangles (that will become the triangles of a completion) and "pasting" their edges together as specified by A.

Let then U be the set of all *unoriented edges* of A, that is, the set of all unordered pairs $\{e, e\,Flip\}$, where $e \in E$. Similarly, let U^* denote the unoriented edges of E^*. We define a *corner* of the algebra as being a pair of unoriented edges of the form $\{\{e, e\,Flip\}, \{e\,Rot, e\,Rot\,Flip\}\}$, where e is an edge. Notice that there are $|E|$ distinct corners in the algebra and that every unoriented edge belongs to exactly two corners. Let $\mathcal{T}$ be a collection of $|E|$ disjoint closed triangles on the plane, each triangle T_r associated to a unique and distinct corner r of the algebra. Label the three vertices of each triangle with the symbols V, E, F.

For each unoriented edge $u \in U$, take the two corners r and s to which u belongs, and identify homeomorphically the VE sides of the two triangles T_r and T_s (matching V with V and E with E). That common side minus its two endpoints is the *primal link* corresponding to u. In the same manner, for every $u^* \in U^*$ take the two corners r and s containing u^* and identify the FE sides of T_r and T_s; the common side will become the *dual link* corresponding to u^*.

Finally, for every corner

$$r = \{\{e, e\,Flip\}, \{e\,Rot, e\,Rot\,Flip\}\},$$

there is exactly one *opposite corner*,

$$s = \{\{f, f\,Flip\}, \{f\,Rot, f\,Rot\,Flip\}\},$$

such that $f = e\,Rot\,Onext$ and $e = f\,Rot\,Onext$. Identify the VF sides of T_r and T_s. Call the seam segment a *vertex–face link*.

Clearly any point interior to a triangle has a neighborhood homeomorphic to a disk. Every side of every triangle is joined with exactly one side of a distinct triangle, so a point on a link also has a disklike neighborhood. Now consider a vertex v of some triangle and all other points that have been identified with it; they have all the same label by construction. An E type vertex v belongs to exactly four triangles, corresponding to the corners

$$\{\{e\,Rot^k,\ e\,Rot^k\,Flip\},\ \{e\,Rot^k\,Rot,\ e\,Rot^k\,Rot\,Flip\}\}$$

for $0 \le k < 4$ and some edge e. Each triangle is pasted to the next one by a primal or dual link incident at v, so as to form a quadrilateral with center v. A V- or F-type vertex v is common to $2n$ triangles (for some $n \ge 1$) corresponding to the corners

$$\{\{e_k,\ e_k\,Flip\},\ \{e_k\,Rot,\ e_k\,Rot\,Flip\}\}$$

and

$$\{\{e_k\,Flip,\ e_k\},\ \{e_k\,Flip\,Rot,\ e_k\,Flip\,Rot\,Flip\}\},$$

where $e_k = e\,Onext^k$ for some edge e and $0 \le k < n$. These triangles are pasted alternately by vertex–face links and primal or dual links, so as to form a $2n$-sided polygon around v. In all cases, the vertex v has a disklike neighborhood.

We conclude that the triangles $\mathscr{T}$ pasted as above constitute a manifold. The links, the triangle interiors, and the identified vertices obviously define a completion Σ of this manifold, and $A\Sigma$ is isomorphic to A. □

4. THE QUAD-EDGE DATA STRUCTURE

We represent a subdivision S (and simultaneously a dual subdivision S^*) by means of the *quad-edge data structure*, which is a natural computer implementation of the corresponding edge algebra. The edges of the algebra can be partitioned in groups of eight: each group consists of the four oriented and directed versions of an undirected edge of S plus the four versions of its dual edge. The group containing a particular edge e is therefore the orbit of e under the subalgebra generated by *Rot* and *Flip*. To build the data structure, we select arbitrarily a *canonical representative* in each group. Then any edge e can be written as $e_0\,Rot^r\,Flip^f$, where $r \in \{0, 1, 2, 3\}$, $f \in \{0, 1\}$, and e_0 is the canonical representative of the group to which e belongs.

The group of edges containing e is represented in the data structure by one *edge record* e, divided into four parts e[0] through e[3]. Part e[r] corresponds to the edge $e_0\,Rot^r$. See Figure 7a. A generic edge $e = e_0\,Rot^r\,Flip^f$ is represented by the triplet (e r, f), called an *edge reference*. We may think of this triplet as a pointer to the "quarter-record" e[r] plus a bit f that tells whether we should look at it from "above" or from "below."

Each part e[r] of an edge record contains two fields, `Data` and `Next`. The `Data` field is used to hold geometrical and other nontopological information about the edge $e_0\,Rot^r$. This field neither affects nor is affected by the topological

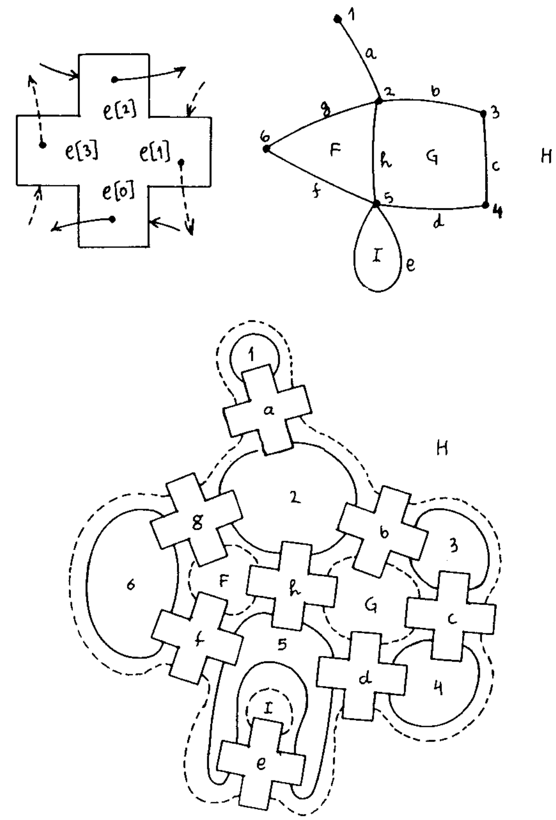

Fig. 7. (a) Edge record showing `Next` links. (b) A subdivision of the sphere. (c) The data structure for the subdivision (b).

operations that we will describe, so its contents and format are entirely dependent on the application.

The `Next` field of e[r] contains a reference to the edge $e_0\,Rot^r\,Onext$. Given an arbitrary edge reference (e, r, f), the three basic edge functions *Rot*, *Flip*, and *Onext* are given by the formulas

$$\begin{aligned}(\mathrm{e}, r, f)\,Rot &= (\mathrm{e}, r + 1 + 2f, f),\\ (\mathrm{e}, r, f)\,Flip &= (\mathrm{e}, r, f + 1),\\ (\mathrm{e}, r, f)\,Onext &= (\mathrm{e}[r + f].\mathtt{Next})\ Rot^f Flip^f,\end{aligned} \tag{4}$$

where the r and f components are computed modulo 4 and modulo 2, respectively. In the first expression above, note that $r + 1 + 2f$ is congruent modulo 4 to $r + 1$ if $f = 0$, and $r - 1$ if $f = 1$; this corresponds to saying that rotating e 90° counterclockwise, as seen from one side of the manifold, is the same as rotating it 90° clockwise as seen from the other side. Similarly, the third expression

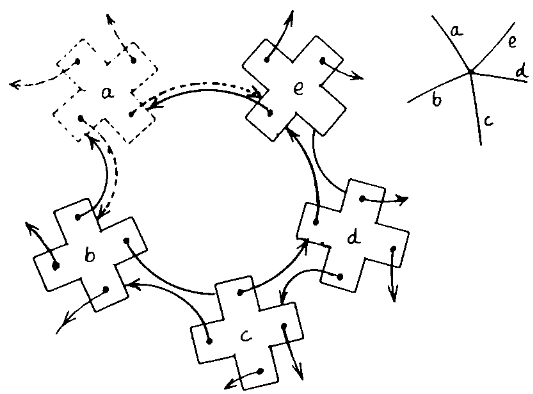

Fig. 8. An *Onext* ring with canonical representatives on both sides of the manifold.

implies that

$$(\mathrm{e}, r, 0)\,Onext = \mathtt{e}[r + f].\mathtt{Next}$$

and

$$\begin{aligned}(\mathrm{e}, r, 1)\,Onext &= (\mathtt{e}[r + 1].\mathtt{Next})\,Rot\,Flip\\ &= (\mathrm{e}, r, 0)\,Rot\,Onext\,Rot\,Flip\\ &= (\mathrm{e}, r, 0)\,Onext^{-1}\,Flip,\end{aligned}$$

that is, moving counterclockwise around a vertex is the same as moving clockwise on the other side of the manifold. From these formulas it follows also that

$$\begin{aligned}(\mathrm{e}, r, f)\,Sym &= (\mathrm{e}, r + 2, f),\\ (\mathrm{e}, r, f)\,Rot^{-1} &= (\mathrm{e}, r + 3 + 2f, f),\\ (\mathrm{e}, r, f)\,Oprev &= (\mathtt{e}[r + 1 - f].\mathtt{Next})\,Rot^{1-f}\,Flip^{f},\end{aligned}$$

and so forth.

Figure 7 illustrates a portion of a subdivision and its quad-edge data structure. We may think of each record as belonging to four circular lists, corresponding to the two vertices and two faces incident to the edge. Note however that to traverse those lists we have to use the *Onext* function, not just the `Next` pointers. Consider for example the situation depicted in Figure 8, where the canonical representative of edge *a* has orientation opposite to that of the others.

The quad-edge data structure contains no separate records for vertice or faces; a vertex is implicitly defined as a ring of edges, and the standard way to refer to it is to specify one of its outgoing edges. This has the added advantage of specifying a reference point on its edge ring, which is frequently necessary when the vertex is used as a parameter to topological operations. Similarly, the standard way of referring to a connected component of the edge structure is by giving one of its directed edges. In this way, we are also specifying one of the two dual subdivisions and a "starting place" and "starting direction" on it. Therefore a subdivision referred to by the edge *e* can be "instantaneously" transformed into its dual by taking *e Rot*.

4.1 Simplifications for Orientable Manifolds

In many applications, including the Voronoi and Delaunay algorithms that we are going to discuss, all manifolds to be handled are orientable. This means we can assign a specific orientation to each edge, vertex, and face of the subdivision so that any two incident elements have compatible orientations. This happens if and only if the elements of the edge algebra can be partitioned in two sets, each closed under *Rot* and *Onext*, and each the image of the other under *Flip*. Then we don't need the *f* bit in edge references, and the formulas simplify to

$$\begin{aligned}(\mathrm{e}, r)\,Rot &= (\mathrm{e}, r + 1),\\ (\mathrm{e}, r)\,Onext &= \mathtt{e}[r].\mathtt{Next},\\ (\mathrm{e}, r)\,Sym &= (\mathrm{e}, r + 2),\\ (\mathrm{e}, r)\,Rot^{-1} &= (\mathrm{e}, r + 3),\\ (\mathrm{e}, r)\,Oprev &= (\mathtt{e}[r + 1].\mathtt{Next})\,Rot,\end{aligned}$$

and so forth.

We can represent a simple subdivision (without its dual) by a "simple edge algebra" that has only *Onext* and *Sym* as the primitive operators. Then we can get *Dnext*, *Lprev*, and *Rprev* in constant time, but not their inverses. However, this may be adequate for some applications. We save two pointers (and perhaps two data fields) in each edge record. Note that this optimization cannot be used with *Flip*.

4.2 Additional Comments on the Data Structure

The storage space required by the quad-edge data structure, including the `Data` fields, is $|\mathscr{E}S| \times$ (8 record pointers + 12 bits). The simplification for orientable manifolds reduces those 12 bits to 8. This compares favorably with the winged-edge representation [1] and with the Muller–Preparata variant [16]. Indeed, all three representations use essentially the same pointers: each edge is connected to the four "immediately adjacent" ones (*Onext*, *Oprev*, *Dnext*, *Dprev*), and the four `Data` fields of our structure may be seen as corresponding to the vertex and face links of theirs.

Compared with the two versions mentioned above, the quad-edge data structure has the advantage of allowing uniform access to the dual and mirror-image subdivisions. As we shall see, this capability allows us to cut in half the number of primitive and derived operations, since these usually come in pairs whose members are "dual" of each other. As an illustration of the flexibility of the quad-edge structure, consider the problem of constructing a diagram which is a cube joined to an octahedron: we can construct two cubes (calling twice the same procedure) and join one to the dual of the other.

The systematic enumeration of all edges in a (connected) subdivision is a straightforward programming exercise, given an auxiliary stack of size $O(|\mathscr{E}S|)$ and a Boolean mark bit on each directed edge [12]. With a few more bits per edge, we can do away with the stack entirely [6]. A slight modification of those algorithms can be used to enumerate the vertices of the subdivision, in the sense of visiting exactly one edge out of every vertex. If we take the dual subdivision, we get an enumeration of the faces. In all cases the running time is linear in the number of edges. Recall also that from Euler's relation it follows that the number of vertices, edges, and faces of a subdivision are linearly related.

5. BASIC TOPOLOGICAL OPERATORS

Perhaps the main advantage of the quad-edge data structure is that the construction and modification of arbitrary diagrams can be effected by as few as two basic topological operators, in contrast to the half-dozen or more required by the previous versions [3, 15].

The first operator is denoted by $e \leftarrow$ `MakeEdge[ ]`. It takes no parameters, and returns an edge e of a newly created data structure representing a subdivision of the sphere (see Figure 9). Apart from orientation and direction, e will be the only edge of the subdivision and will not be a loop; we have $e\,Org \neq e\,Dest$, $e\,Left = e\,Right$, $e\,Lnext = e\,Rnext = e\,Sym$, and $e\,Onext = e\,Oprev = e$. To construct a loop, we may use $e \leftarrow$ `MakeEdge[ ].Rot`; then we will have $e\,Org = e\,Dest$, $e\,Left \neq e\,Right$, $e\,Lnext = e\,Rnext = e$, and $e\,Onext = e\,Oprev = e\,Sym$.

The second operator is denoted by `Splice`$[a, b]$ and takes as parameters two edges a and b, returning no value. This operation affects the two edge rings $a\,Org$ and $b\,Org$ and, independently, the two edge rings $a\,Left$ and $b\,Left$. In each case,

(a) if the two rings are distinct, `Splice` will combine them into one;
(b) if the two are exactly the same ring, `Splice` will break it in two separate pieces;
(c) if the two are the same ring taken with opposite orientations, `Splice` will *Flip* (and reverse the order) of a segment of that ring.

The parameters a and b determine the place where the edge rings will be cut and joined. For the rings $a\,Org$ and $b\,Org$, the cuts will occur immediately *after* a and b (in counterclockwise order); for the rings $a\,Left$ and $b\,Left$, the cut will occur immediately *before* $a\,Rot$ and $b\,Rot$. Figure 10 illustrates this process for one of the simplest cases, when a and b have the same origin and distinct left faces. In this case `Splice`$[a, b]$ splits the common origin of a and b in two separate vertices and joins their left faces. If the origins are distinct and the left faces are the same, the effect will be precisely the opposite: the vertices are joined and the left faces are split. Indeed, `Splice` is its own inverse: if we perform `Splice`$[a, b]$ twice in a row we will get back the same subdivision.

Figure 11 illustrates the effect of `Splice`$[a, b]$ in the case where a and b have distinct left faces and distinct origins. In this case, `Splice` will either join two components in a single one or add an extra "handle" to the manifold, depending on whether a and b are in the same component or not. Figure 11 also illustrates the case when both left faces and origins are distinct.

In the edge algebra, the *Org* and *Left* rings of an edge e are the orbits under *Onext* of e and $e\,Onext\,Rot$, respectively. The effect of `Splice` can be described as the construction of a new edge algebra $A' = (E, E^*, Rot, Flip)$ from an existing algebra $A = (E, E^*, Onext, Rot, Flip)$, where $Onext'$ is obtained from $Onext$ by redefining some of its values. The modifications needed to obtain the effect described above are actually quite simple. If we let $\alpha = a\,Onext\,Rot$ and $\beta = b\,Onext\,Rot$, basically all we have to do is to interchange the values of $a\,Onext$

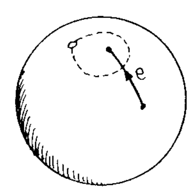

Fig. 9. The result of `MakeEdge`.

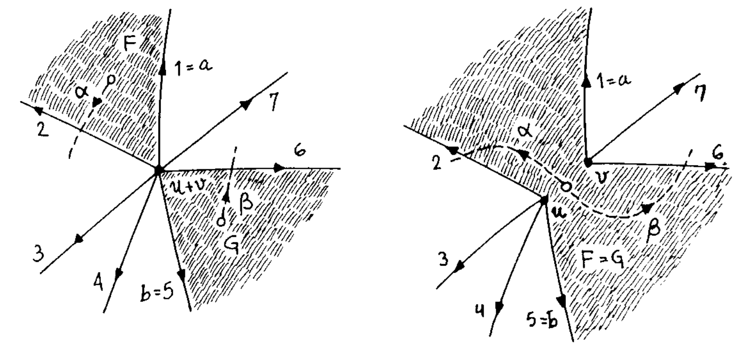

Fig. 10. The effect of `Splice`: Trading a vertex for a face. (a) $a\,Org = b\,Org$, $a\,Left \neq b\,Left$. (b) $a\,Org \neq b\,Org$, $a\,Left = b\,Left$.

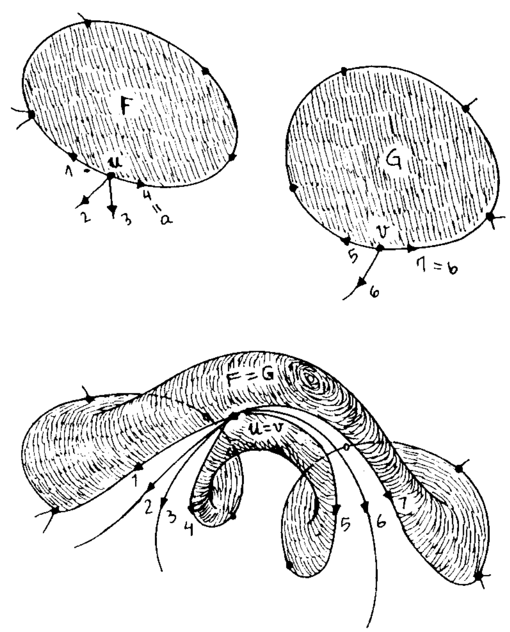

Fig. 11. The effect of `Splice`: Changing the connectivity of the manifold. (a) $a\,Org \neq b\,Org$, $a\,Left \neq b\,Left$. (b) $a\,Org = b\,Org$, $a\,Left = b\,Left$.

with $b\,Onext$ and $\alpha\,Onext$ with $\beta\,Onext$. The apparently complex behavior of `Splice` now can be recognized as the familiar effect of interchanging the next links of two circular list nodes [12].

As one may well expect, to preserve the validity of the axioms F1–F5 and E1–E5 we may have to make some additional changes to the *Onext* function. For example, whenever we redefine $e\,Onext'$ to be some edge f, we must also redefine $e\,Flip\,(Onext')^{-1}$ to be $f\,Flip$, or, equivalently, $f\,Flip\,Onext'$ to be $e\,Flip$. So, `Splice`$[a, b]$ must perform at least the following changes in the function *Onext*:

$$\begin{aligned}
a\,Onext' &= b\,Onext,\\
b\,Onext' &= a\,Onext;\\
\alpha\,Onext' &= \beta\,Onext,\\
\beta\,Onext' &= \alpha\,Onext;\\
(b\,Onext\,Flip)\,Onext' &= a\,Flip,\\
(a\,Onext\,Flip)\,Onext' &= b\,Flip;\\
(\beta\,Onext\,Flip)\,Onext' &= \alpha\,Flip,\\
(\alpha\,Onext\,Flip)\,Onext' &= \beta\,Flip.
\end{aligned}\tag{5}$$

Note that these equations reduce to $Onext' = Onext$ if $b = a$. Since $a\,Onext' = b\,Onext$, to satisfy axiom E5 we must have $a \in E$ iff $b\,Onset \in E$, which is equivalent to $a \in E$ iff $b \in E$. We will take this as a precondition for the validity of `Splice`$[a, b]$: the effect of this operation is not defined if a is a primal edge and b is dual, or vice-versa.[1] Another problematic situation is when $b = a\,Onext\,Flip$: according to eqs. (5) we would have $a\,Onext' = a\,Onext\,Flip\,Onext = a\,Flip$, which contradicts F3. In this particular case, it is more convenient to define the effect of `Splice`$[a, b]$ as being null, that is, $Onext' = Onext$. It turns out that with only these two exceptions, the equations above always define a valid edge algebra.

THEOREM 5.1. *If A is an edge algebra, a and b are both primal or both dual, and $b \neq a\,Onext\,Flip$, then the algebra A' obtained by performing the operation* `Splice`$[a, b]$ *on A is also an edge algebra.*

PROOF. Since `Splice` does not affect *Flip* and *Rot*, all axioms except F2, F3, E2, and E5 are automatically satisfied by A'. Since a and b are both primal or both dual, the same is true of α and β, $a\,Onext\,Flip$ and $b\,Onext\,Flip$, and $\alpha\,Onext\,Flip$ and $\beta\,Onext\,Flip$. Thus the assignments corresponding to the operation `Splice`$[a, b]$ will not destroy E5.

Now let us show that E4 holds in A', that is, $e\,Rot\,Onext'\,Rot\,Onext' = e$. Let X be the set of edges whose *Onext* has been changed, that is,

$$X = \left\{\begin{matrix} a, & b, \\ \alpha, & \beta, \\ a\,Onext\,Flip, & b\,Onext\,Flip, \\ \alpha\,Onext\,Flip, & \beta\,Onext\,Flip \end{matrix}\right\}.$$

[1] Note that if a and b lie in distinct subalgebras A_a and A_b of A, then the union of A_a and the dual of A_b is also a valid edge algebra. So, in practice we can always perform `Splice`$[a, b]$ when a and b lie in disjoint data structures.

First, if $e\,Rot \notin X$, then $e\,Rot\,Onex\,Rot \notin X\,Onext\,Rot = X$, and so

$$\begin{aligned}
e\,Rot\,Onext'\,Rot\,Onext' &= e\,Rot\,Onext\,Rot\,Onext'\\
&= (e\,Rot\,Onext\,Rot)\,Onext\\
&= e.
\end{aligned}$$

Now assume $e\,Rot \in X$. Notice that `Splice`$[a, b]$ does exactly the same thing as `Splice`$[b, a]$, `Splice`$[\alpha, \beta]$, and `Splice`$[a\,Onext\,Flip, b\,Onext\,Flip]$, so without loss of generality we can assume $e\,Rot = a$. Then

$$\begin{aligned}
e\,Rot\,Onext'\,Rot\,Onext' &= a\,Onext'\,Rot\,Onext' = b\,Onext\,Rot\,Onext'\\
&= \beta\,Onext' = \alpha\,Onext\\
&= a\,Onext\,Rot\,Onext = e\,Rot\,Onext\,Rot\,Onext\\
&= e.
\end{aligned}$$

In a similar way we can prove F2. To conclude, let us prove F3: $e\,Flip\,(Onext')^n \neq e$ for all n. In other words, we have to show that *Flip* always takes an $Onext'$ orbit to a different $Onext'$ orbit. It suffices to show this for the orbits of elements of X; in fact the symmetry of `Splice` implies it is sufficient to show this for the orbit of a.

Let $a\,Org = \langle a_1 a_2 \cdots a_{m-1} a_m (= a)\rangle$ be the orbit of a under the original *Onext*. The orbit of $a\,Flip$ under *Onext* is then $a\,Flip\,Org = \langle a'_m a'_{m-1} \cdots a'_2 a'_1\rangle$, where $a'_i = a_i\,Flip$ for all i. These two orbits are disjoint; and cannot contain any of the edges α, β, $\alpha\,Onext\,Flip$, or $\beta\,Onext\,Flip$, which lie in the dual subdivision. Furthermore, one contains b if and only if the other contains $b\,Flip$. There are then only three cases to consider (see Figure 12):

Case 1. The edge b is neither in $a\,Org$ nor in $a\,Flip\,Org$. Then let $b\,Org = \langle b_1 b_2 \cdots b_{n-1} b_n (= b)\rangle$ and $b\,Flip\,Org = \langle b'_n b'_{n-1} \cdots b'_2 b'_1\rangle$. According to eqs. (5), we will have $a_m\,Onext' = b_1$, $b_m\,Onext' = a_1$, $a'_1\,Onext' = b'_m$, $b'_1\,Onext' = a'_m$. Therefore, the orbits of a and $a\,Flip$ under $Onext'$ will be

$$\begin{aligned}
a\,Org' &= \langle a_1 a_2 \cdots a_{m-1} a_m (= a) b_1 b_2 \cdots b_{n-1} b_n (= b)\rangle,\\
a\,Flip\,Org' &= \langle b'_n b'_{n-1} \cdots b'_2 b'_1 a'_m a'_{m-1} \cdots a'_2 a'_1\rangle.
\end{aligned}$$

Case 2. The edge b occurs in $a\,Org$. Then $b = a_i$ for some i, $1 \le i \le m$. After `Splice` is executed we will have $a_m\,Onext' = a_{i+1}$, $a_i\,Onext' = a_1$, $a'_1\,Onext' = a'_i$, and $a'_{i+1}\,Onext' = a'_m$. If $i = m$ (i.e., if $a = b$), then $Onext' = Onext$ and we are done. If $i \neq m$, then under $Onext'$ the elements of $a\,Org$ and $a\,Flip\,Org$ will be split in the four orbits,

$$\begin{aligned}
b\,Org' &= \langle a_1 a_2 \cdots a_i\rangle, & a\,Org' &= \langle a_{i+1} a_{i+2} \cdots a_m\rangle,\\
a\,Flip\,Org' &= \langle a'_m a'_{m-1} \cdots a'_{i+1}\rangle, & b\,Flip\,Org' &= \langle a'_i a'_{i-1} \cdots a_1\rangle.
\end{aligned}$$

Case 3. The edge b occurs in $a\,Flip\,Org$. Since $b \neq a\,Onext\,Flip = a'_1$, we have $b = a'_i$ for some i, $2 \le i \le m$. After `Splice` is executed we will have $a_m\,Onext' = a'_{i-1}$, $a'_i\,Onext' = a_1$, $a'_1\,Onext' = a_i$, and $a_{i-1}\,Onext' = a'_m$. Then the orbits of $Onext'$ containing those elements will be

$$\begin{aligned}
a\,Org' &= \langle a'_{i-1} a'_{i-2} \cdots a'_1 a_i a_{i+1} \cdots a_{m-1} a_m\rangle,\\
a\,Flip\,Org' &= \langle a'_m a'_{m-1} \cdots a'_{i+1} a'_i a_1 a_2 \cdots a_{i-1} a_i\rangle.
\end{aligned}$$

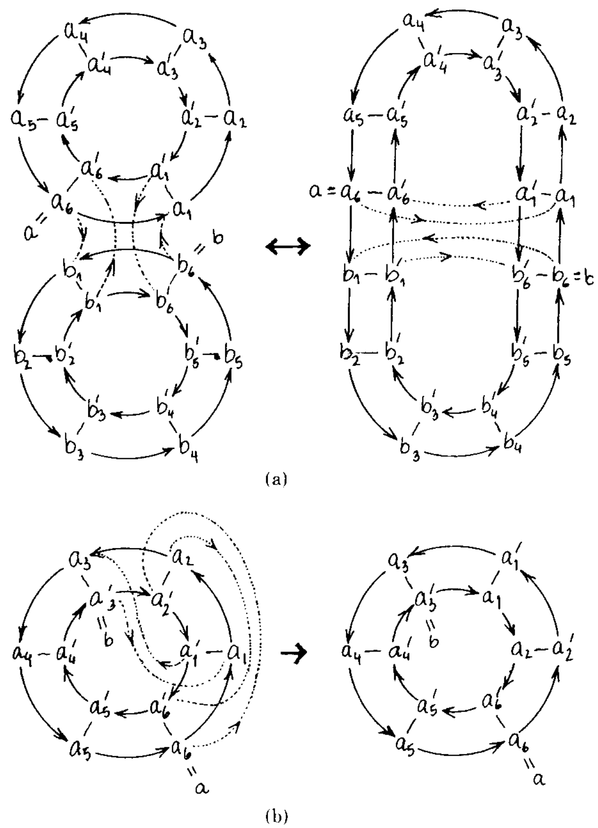

Fig. 12. The effect of `Splice` on the *Onext* orbits. (a) Case 1. (b) Case 2. (c) Case 3.

In all three cases, the orbits of e and $e\,Flip$ under $Onext'$ will be disjoint for all edges e. □

The proof of Theorem 5.1 gives a precise description of the effect of `Splice` on the edge rings. In particular, the discussion for case 3 helps in the understanding of Figure 13. In that case the effect of `Splice` is to add or remove a "cross cap" to the manifold.

In terms of the data structure, the `Splice` operation is even simpler. The identities

$$a\,Onext\,Flip = a\,Onext\,Rot\,Flip\,Rot = a\,Flip\,Rot$$

and

$$\alpha\,Onext\,Flip = a\,Onext\,Rot\,Onext\,Flip = a\,Flip\,Rot$$

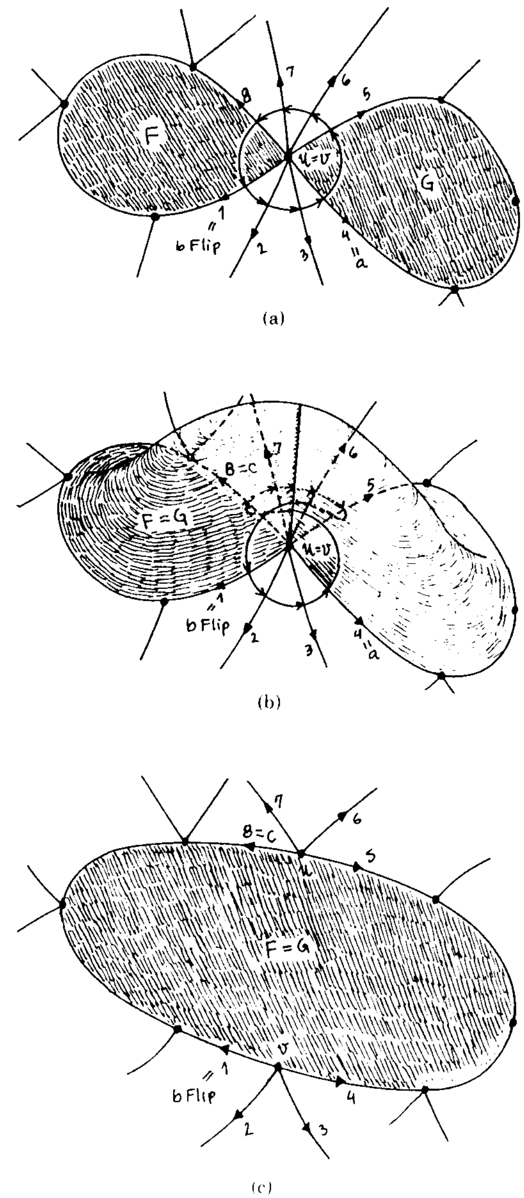

Fig. 13. The effect of `Splice`: Adding or removing a cross-cap. (a) $a\,Org = b\,Flip\,Org$, $a\,Left = b\,Left$. (b) $a\,Org = b\,Flip\,Org$, $a\,Left = b\,Left$; $c\,Left = b\,Flip\,Left$, $c\,Org = b\,Org$. (c) $c\,Left = b\,Flip\,Left$, $c\,Org \neq b\,Org$.

allow us to rewrite (5) as

$$\begin{array}{ll} a\,Onext \leftarrow b\,Onext; & (a\,Flip\,Rot)\,Onext \leftarrow \beta\,Flip; \\ b\,Onext \leftarrow a\,Onext; & (b\,Flip\,Rot)\,Onext \leftarrow \alpha\,Flip; \\ \alpha\,Onext \leftarrow \beta\,Onext; & (\alpha\,Flip\,Rot)\,Onext \leftarrow b\,Flip; \\ \beta\,Onext \leftarrow \alpha\,Onext; & (\beta\,Flip\,Rot)\,Onext \leftarrow a\,Flip. \end{array} \tag{6}$$

Only one of the two assignments in each line of (6) is meaningful. The reason is that only one of the receiving *Onext* fields actually exists in the structure; the value of the other is determined implicitly from existing links by (4). If the f bit of a is 0, then $a\,Onext$ exists, and `Splice` writes $b\,Onext$ into it. Otherwise $a\,Flip\,Rot$ has $f = 0$, and we can assign $b\,Flip\,Rot\,Onext$ to $a\,Flip\,Rot\,Onext$. The same applies to b, α, and β. Note that these assignments are simultaneous, that is, all right-hand sides are computed before any value is assigned to the left-hand sides. In addition, these assignments should be preceded by a test of whether $b = a\,Onext\,Flip$, in which case they should not be executed at all. Note however that there is no need to check for $a = b$.

Further reductions in the code of `Splice` occur in the case of orientable manifolds, when we can use the simplified data structure without *Flip* and the f bits. In that case, the meaningful assignments are precisely those in the left column of (6), and the test for $b = a\,Onext\,Flip$ is meaningless.

THEOREM 5.2 *An arbitrary subdivision S can be transformed into a collection of $|\mathscr{E}\mathscr{S}|$ isolated edges by the application of at most $2|\mathscr{E}S|$* `Splice` *operations.*

PROOF. Let e be an arbitrary edge of S. The operations

`Splice`[e, $e\,Oprev$]; `Splice`[$e\,Sym$, $e\,Sym\,Oprev$]

will remove e from S and place it as an isolated edge on a separate manifold homeomorphic to the sphere. By repeating this for every edge the theorem follows. □

From this theorem and from the fact that `Splice` is its own inverse, we can conclude that any simple subdivision S can be constructed, in $O(|\mathscr{E}S|)$ time and space by using only the `Splice` and `MakeEdge` operations.

The `Data` links are not affected by (and do not affect) the `MakeEdge` and `Splice` operations; if used at all, they can be set and updated at any time after the edge is created by plain assignment statements. Since they carry no topological information, there is no need to forbid or restrict assignments to them. Usually each application imposes geometrical or other constraints on the `Data` fields that may be affected by changes in the topology. Some care is required when enforcing those constraints; for example, the operation of joining two vertices may change the geometrical parameters of a large number of edges and faces, and updating all the corresponding `Data` fields every time may be too expensive. However, even in such applications it is frequently the case that we can defer those updates until they are really needed (so that their cost can be amortized over a large number of `Splices`) or initialize the `Data` links right from the beginning with the values they must have in the final structure.

Like its predecessors, the quad-edge data structure contains no mechanism to keep track automatically of the components and connectivity of the manifold. There seems to be no general way of doing this at a bounded cost per operation; on the other hand, in many applications this problem is trivial or straightforward, so it is best to solve this problem independently for each case.

6. TOPOLOGICAL OPERATORS FOR DELAUNAY DIAGRAMS

In the Voronoi/Delaunay algorithms described further on, all edge variables refer to edges of the Delaunay diagram. The `Data` field for a Delaunay edge e points to a record containing the coordinates of its origin $e\,Org$, which is one of the sites; accordingly, we will use e.`Org` as a synonym of e.`Data` in those algorithms. For convenience, we will also use e.`Dest` instead of e.`Sym.Org`. We emphasize again that these `Dest` and `Org` fields carry no topological meaning and are not updated by the `Splice` operation per se. The endpoints of the dual edges (Voronoi vertices) are neither computed nor used by the algorithms; if desired, they can be easily added to the structure, either during its construction or after it. The fields e.`Rot.Data` and e.`Rot`$^{-1}$.`Data` are not used.

Most topological manipulations performed by our algorithms on the Delaunay/Voronoi diagrams can be reduced to three higher-level topological operators, defined here in terms of `Splice` and `MakeEdge`. The operation $e \leftarrow$ `Connect`[a, b] will add a new edge e connecting the destination of a to the origin of b, in such a way that $a\,Left = e\,Left = b\,Left$ after the connection is complete. For added convenience it will also set the `Org` and `Dest` fields of the new edge to a.`Dest` and b.`Org`, respectively.

```
PROCEDURE Connect[a, b, side] RETURNS [e]
  e ← MakeEdge[ ];
  e.Org ← a.Dest;
  e.Dest ← b.Org;
  Splice[e, a.Lnext];
  Splice[e.Sym, b]
END Connect.
```

The operation `DeleteEdge`[e] will disconnect the edge e from the rest of the structure (this may cause the rest of the structure to fall apart in two separate components). In a sense, `DeleteEdge` is the inverse of `Connect`. It is equivalent to

```
PROCEDURE DeleteEdge[e]:
  Splice[e, e.Oprev];
  Splice[e.Sym, e.Sym.Oprev]
END DeleteEdge.
```

The operation `Swap`[e] below is used in the incremental algorithm described in Section 10. Given an edge e whose left and right faces are triangles, the problem is to delete e and connect the other two vertices of the quadrilateral thus formed (see Figure 14).

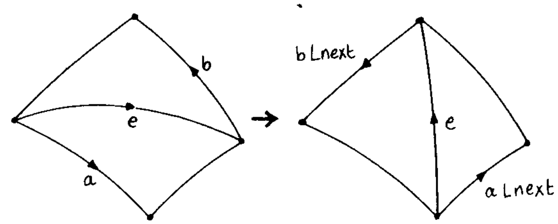

Fig. 14. The effect of Swap [e].

```
PROCEDURE Swap[e]:
  a ← e.Oprev;
  b ← e.Sym.Oprev;
  Splice[e, a]; Splice[e.Sym, b];
  Splice[e, a.Lnext]; Splice[e.Sym, b.Lnext];
  e.Org ← a.Dest; e.Dest ← b.Dest
END Swap.
```

The first pair of `Splices` disconnects e from the edge structure, and leaves it as the single edge of a separate spherical component. The last two `Splices` connect e again at the required position.

7. VORONOI AND DELAUNAY DIAGRAMS

In this section we recapitulate some of the most important properties of Voronoi diagrams and their duals, with an emphasis on the results we will need later on. For a fuller treatment of these topics the reader should consult refs. [13], [18], or [19].

If we are given only two sites, then the associated Voronoi regions are simply the two (open) half-planes delimited by the bisector of the two sites. More generally, when n sites are given, the region associated with a particular site p will be the intersection of all half-planes containing p and delimited by the bisectors between p and the other sites. It follows that the Voronoi regions are (possibly unbounded) convex polygons whose edges are portions of intersite bisectors and whose vertices (except the point at infinity) are circumcenters of triangles defined by three of the sites. An example Voronoi diagram for a small collection of sites is shown in Figure 15.

As mentioned in Section 1, most of the time we will be dealing with a dual of the Voronoi subdivision, commonly called the Delaunay diagram. This is a planar subdivision whose vertices are the given sites and whose edges are straight-line segments that connect every pair of sites having Voronoi regions sharing a common edge. It can be shown that Delaunay such edges do not cross each other.

We say that a circle is *point-free* if none of the given sites is contained in its interior. It follows readily from the definitions that two sites are connected by an edge in the Delaunay diagram if and only if there is a point-free circle passing through them and through no other site. In particular, every convex hull edge is in the Delaunay diagram. It can be shown also that three or more sites are the vertices of an interior face of the Delaunay diagram if and only if there is a point-free circle passing through them and through no other site. For a discussion of

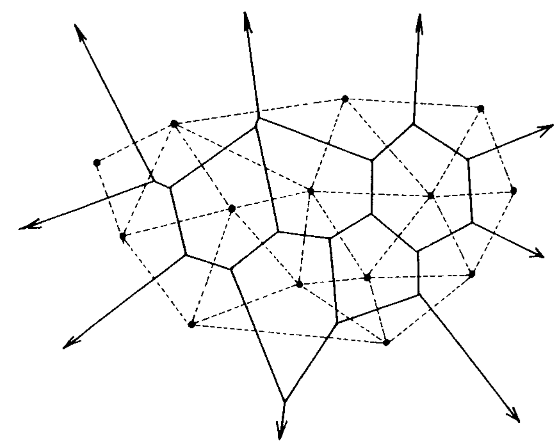

Fig. 15. The Voronoi diagram (solid) and the Delaunay diagram (dashed).

these facts see Lee's thesis [13]. The following obvious lemma will be important in the sequel.

LEMMA 7.1. *Let L and R be two sets of points. Any edge of the Delaunay diagram of $L \cup R$ whose endpoints are both in L is in the Delaunay diagram of L.*

In other words, the addition of new points does not introduce new edges between the old points.

7.1 Delaunay Triangulations

A *triangulation* of $n \geq 2$ sites is a straight-line subdivision of the extended plane whose vertices are the given sites and whose faces are all triangular except for one, which is the complement of the convex hull of the sites. It is easily shown that any triangulation of n sites, of which k lie on the convex hull, has $2(n - 1) - k$ triangles and $3(n - 1) - k$ edges.

If no four of the sites happen to be cocircular, then their Delaunay diagram is a triangulation; in any case, it can be made into one by introducing zero or more additional edges. The subdivisions obtained in this way are called *Delaunay triangulations* of the given sites. They are characterized by either of the following properties.

LEMMA 7.2. *A triangulation of $n \geq 2$ sites is Delaunay if and only if every edge has a point-free circle passing through its endpoints.*

LEMMA 7.3. *A triangulation of $n \geq 2$ sites is Delaunay if and only if the circumcircle of every interior face (triangle) is point-free.*

We will say that an edge or triangle is *Delaunay* when there is a point-free circle passing through its vertices. We speak of that circle as being *witness to the*

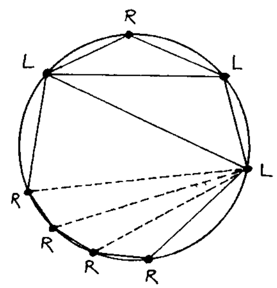

Fig. 16. Triangulating a face of the Delaunay.

Delaunayhood of the edge or triangle. Note that the circle may pass through other sites as well, so a Delaunay edge or triangle is not necessarily an element of the Delaunay diagram. In the few places where this distinction is relevant, we will refer to the edges and faces of the latter as being *strictly Delaunay*.

Lemma 7.1 can be extended to Delaunay triangulations, provided their non-uniqueness is taken into account:

LEMMA 7.4 *Let T_L and T_R be Delaunay triangulations with vertex sets L and R. Then we can always construct a Delaunay triangulation T for the set $L \cup R$ such that every edge of T that is not in T_L or in T_R has one endpoint in L and one in R.*

PROOF. This assertion holds for the edges of the Delaunay diagram D of $L \cup R$. We have only to show that we can triangulate every face of D without violating the above assertion, that is, by using only old edges from T_L and T_R, or new L–R edges.

Consider any face F of D with four or more vertices, and its circumcircle C. Note that any edge connecting two L (respectively, R) vertices that are adjacent along C is in fact an edge of the L (respectively, R) diagram and therefore in T_L (respectively, T_R). So all boundary edges of F are appropriate for our triangulation T. To complete the triangulation, we now add in all diagonals of F that are in T_L and finally connect each R vertex to the previous L vertex counterclockwise along C by an L–R edge. See Figure 16. □

8. THE INCIRCLE TEST

We now proceed to define the main geometric primitive we will use for Delaunay computations. This test is applied to four distinct points in the plane A, B, C, and D. See Figure 17.

Definition 8.1. The predicate InCircle(A, B, C, D) is defined to be true if and only if point D is interior to the region of the plane that is bounded by the oriented circle ABC and lies to the left of it.

In particular this implies that D should be *inside* the circle ABC if the points A, B, and C define a counterclockwise oriented triangle, and *outside* if they define a clockwise oriented one. (In case A, B, and C are collinear we interpret the line

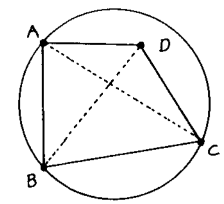

Fig. 17. The InCircle test.

as a circle by adding a point at infinity.) If A, B, C, and D are cocircular, then our predicate returns false. Notice that the test is equivalent to asking whether $\angle ABC + \angle CDA > \angle BCD + \angle DAB$. Another equivalent form of it is given below, based on the coordinates of the points.

LEMMA 8.1. *The test Incircle(A, B, C, D) is equivalent to*

$$\mathcal{D}(A, B, C, D) = \begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_D & y_D & x_D^2 + y_D^2 & 1 \end{vmatrix} > 0.$$

PROOF. We consider the following mapping from points in the plane to points in space:

$$\lambda : (x, y) \mapsto (x, y, x^2 + y^2),$$

which lifts each point on the x, y-plane onto the paraboloid of revolution $x = x^2 + y^2$. See Figure 18 for an illustration. We first show that A, B, C, and D are cocircular if and only if $\lambda(A)$, $\lambda(B)$, $\lambda(C)$, and $\lambda(D)$ are coplanar, a rather amazing fact.

Suppose first that A, B, C, and D are cocircular. If we have the degenerate case where they are collinear, then $\mathcal{D}(A, B, C, D)$ is zero, as we can see by expanding it by the third column. But $\mathcal{D}(A, B, C, D)$ is also the (signed) volume of the tetrahedron defined by $\lambda(A)$, $\lambda(B)$, $\lambda(C)$, and $\lambda(D)$. Since the volume is zero, the points must be coplanar. Otherwise let (p, q) denote the center and r the radius of the circle passing through the points A, B, C, D. We must have

$$(x_A - p)^2 + (y_A - q)^2 = r^2,$$

or equivalently,

$$-2p \cdot x_A - 2q \cdot y_A + 1 \cdot (x_A^2 + y_A^2) + (p^2 + q^2 - r^2) \cdot 1 = 0. \qquad (7)$$

This relation also holds for points B, C, and D, and therefore we have a linear dependence among the columns of the determinant $\mathcal{D}(A, B, C, D)$, which implies that its value is zero. So again we can conclude that $\lambda(A)$, $\lambda(B)$, $\lambda(C)$, and $\lambda(D)$ are coplanar.

Now conversely, suppose that $\lambda(A)$, $\lambda(B)$, $\lambda(C)$, and $\lambda(D)$ are coplanar. If all of A, B, C, and D are collinear, then we are done. So suppose, without loss of generality, that A, B, and C are not collinear. As above, let (p, q) denote the

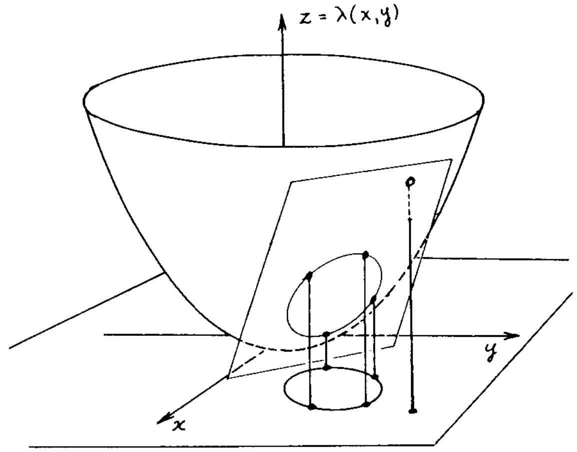

Fig. 18. The quadratic map for computing InCircle.

center and r the radius of the circumcircle of triangle ABC. Then A, B, and C satisfy eq. (7) above. Since A, B, and C are *not* collinear, the corresponding three rows of $\mathcal{D}(A, B, C, D)$ are linearly independent. But all four rows are linearly dependent, since the determinant is zero. So the last row can be expressed as a linear combination of the first three, and therefore point D satisfies (7) as well, that is, it is on the circle ABC.

The above result shows that planar sections of the paraboloid of revolution $z = x^2 + y^2$ project onto circles in the x, y-plane. The paraboloid is a surface that is convex upward, and therefore, in a section of it with a plane, the part below the plane projects to the interior of the corresponding circle in the x, y-plane, and the part above the plane to the exterior. From this and the standard right-handed orientation convention for the sign of volumes, the lemma follows. Notice that this establishes an interesting correspondence between circular queries in the plane and half-space queries in 3-space. □

As a side note we remark that Ptolemy's theorem in Euclidean plane geometry does not lead to a useful implementation of the InCircle test, as we always have

$$AB \times CD + BC \times AD \geq BD \times AC,$$

with equality only when the four points are cocircular. In fact, the quantity one obtains by rendering $AB \times CD + BC \times AD - BD \times AC$ radical-free is essentially the square of the determinant $\mathcal{D}(A, B, C, D)$ above.

The following property of the InCircle test is an obvious consequence of Lemma 8.1.

LEMMA 8.2. *If A, B, C, D are any four noncocircular points in the plane, then transposing any adjacent pair in the predicate InCircle(A, B, C, D) will change the value of the predicate from true to false or vice versa. In particular, the Boolean sequence*

$$InCircle(A, B, C, D), \quad InCircle(B, C, D, A), \quad InCircle(C, D, A, B), \quad InCircle(D, A, B, C)$$

is either T(rue), F(alse), T, F, or F, T, F, T.

In particular, if InCircle(A, B, C, D) is true, then InCircle(C, B, A, D) is false, so reversing the orientation flips the value of the predicate. Note however that InCircle is always false if the four points are cocircular, irrespective of their order. The last two lemmas show that in InCircle(A, B, C, D) all four points play a symmetric role, even though from the definition D seems to be special.

What use is the InCircle test in the construction of Delaunay diagrams? Consider for example the case of four sites that are the vertices of a convex quadrilateral $ABCD$. The sides AB, BC, CD, and DA are on the convex hull and therefore must be included. To complete the triangulation, we must add either diagonal AC or diagonal BD. We can decide between the two by evaluating InCircle(A, B, C, D). If it is false, then the circle ABC is point-free, and AC is Delaunay. Conversely, if InCircle(A, B, C, D) is true, then AC is not Delaunay. However, by Lemma 8.2, InCircle(B, C, D, A) must in that case be true. Thus the circle BCD is point-free, and BD is Delaunay.

This rule can be extended to more than four points, thanks to the following observation. Given two points X and Y on the plane, the set of circles passing through X and Y form a one-parameter family $\{C_t\}$, where the parameter t may be taken as the position of the center along the bisector of XY, measured from the midpoint of XY. Thus $C_{-\infty}$ denotes the half-plane to the left of the line XY, C_0 denotes the circle with diameter XY, and C_∞ denotes the half-plane to the right of XY. Note that the portion of these circles to the left of XY strictly decreases (by proper inclusion) as t increases, while the portion to the right of XY strictly increases. See Figure 19.

Now let X, Y be any pair of sites. The edge XY will be Delaunay if and only if there is a point-free circle passing through both sites. But this is possible if and only if every circle AXY with site A on the left side of the line XY corresponds to a value of t less than or equal to that of any circle YXB with B on the right side. This observation proves the following result:

LEMMA 8.3. *An edge XY is Delaunay if and only if InCircle(A, X, Y, B) is false for every pair of sites A and B, respectively, to the left and to the right of the line XY.*

In fact, to check whether a triangulation is Delaunay it is sufficient to consider just one pair of sites per edge, as shown below.

Definition 8.2. Let T be an arbitrary triangulation of the given sites, and XY be one of its edges. We say that XY *passes the circle test* if it is the boundary between two counterclockwise triangles AXY and YXB of T, and InCircle(A, X, Y, B) is false.

The counterclockwise-oriented (but not necessarily convex) polygon $AXBY$ is called the *edge quadrilateral* of XY. An edge that passes the circle test is not

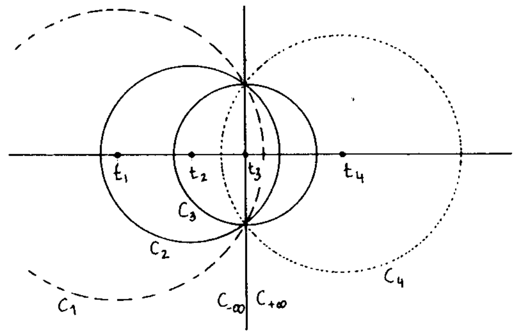

Fig. 19. The circles passing through two given points.

necessarily Delaunay, since the test considers just one pair of sites A, B. However, Lee [13] has proved the following result.

LEMMA 8.4. *A triangulation T is Delaunay if and only if all its edges pass the circle test.*

PROOF. If an edge XY fails the circle test, the two other vertices A and B of its edge quadrilateral establish its non-Delaunayhood, by Lemma 8.3. Conversely, if T is not Delaunay, there must be some edge XY of T and some pair of sites A and B, respectively, to the left and to the right of XY, for which InCircle(A, X, Y, B) is true. Among all such quadruplets X, Y, A, and B, choose one for which the sum of the angles $\angle YAX$ and $\angle XBY$ is maximum. It is easy to see that no other vertex or edge of T can enter the triangles AXY or YXB. Therefore, these triangles are the two faces of T incident to XY, and XY fails the circle test. □

9. THE DIVIDE-AND-CONQUER ALGORITHM

In this section we use the tools we have developed so far to describe, analyze, and prove correct a divide-and-conquer algorithm for computing the Delaunay triangulation of n points in the plane. Topologically the quad-edge data structure gives us the dual for free, so by associating some relevant geometric information with our face nodes, for example, the coordinates of the corresponding Voronoi vertices, we are simultaneously computing the Voronoi diagram as well. An advantage of working with the dual of the Voronoi diagram is that we need not compute straight-line intersections unless the coordinates of Voronoi vertices are needed. Our algorithm follows closely the one proposed by Lee and Schachter [14] and is the dual of that described by Shamos and Hoey [19]. Like theirs, it runs in time $O(n \log n)$ and uses linear storage. The reasons for including it here are twofold. First of all we wanted to illustrate the use of the quad-edge data structure on a concrete and important application. Secondly, our presentation is significantly more complete in both the details of the algorithm—which can be subtle—and its proof.

As one might expect, in the divide-and-conquer-algorithm we start by partitioning our points into two halves, the left half (L) and the right half (R), which are separated in the x-coordinate. We next recursively compute the Delaunay

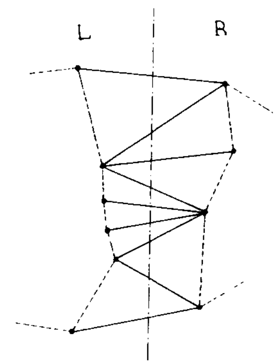

Fig. 20. The structure of the L–R edges.

triangulation of each half. Finally, we need to marry the two half triangulations into the Delaunay triangulation of the whole set. This recursive decomposition cannot be used if the number of sites is less than four, since in that case one or both of L and R would end up with a single site (recall that any subdivision, and hence any Delaunay diagram, must have at least one edge). Therefore, the two- and three-site cases must be handled separately.

We now elaborate on this brief description in stages. First of all it is advantageous to start out by sorting our points in increasing x-coordinate. When there are ties we resolve them by sorting in increasing y-coordinate and throwing away duplicate points. This makes all future splittings constant time operations. After splitting in the middle and recursively triangulating L and R, we must consider the merge step. Note that this may involve deleting some L–L or R–R edges and will certainly require adding some L–R (or so called *cross*) edges. By Lemma 7.3, however, no new L–L or R–R edges will be added.

What is the structure of the cross edges? All these edges must cross a line parallel to the y-axis and placed at the splitting x value. This establishes a linear ordering of the cross edges, so we can talk about successive cross edges, the bottom-most cross edge, etc. The algorithm we are about to present will produce the cross edges incrementally, in ascending y-order. See Figure 20.

LEMMA 9.1. *Any two cross edges adjacent in the y-ordering share a common vertex. The third side of the triangle they define is either an L–L or an R–R edge.*

PROOF. Any two consecutive intersections of a triangulation with a straight line must belong to the same triangular face. Therefore the two cross edges in question have one endpoint in common, and the third side of the triangle is fully to one side or the other of the vertical divider. □

Lemma 9.1 has the following important consequence. Let us call the current cross edge the base and write its directed variant going from right to left as `basel`. The successor to `basel` will either be an edge going from the left endpoint of `basel` to one of the R-neighbors of the right endpoint lying above `basel`, or, symmetrically, it will be an edge from the right endpoint of `basel` to one of the L-neighbors of the left endpoint lying above `basel`. In the program below edges from the left endpoint of `basel` to its candidate L-neighbors will be

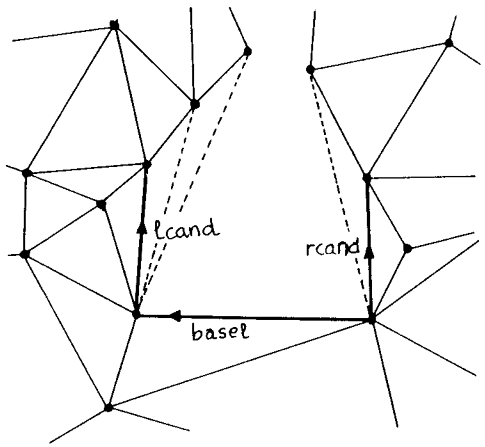

Fig. 21. The variables `lcand`, `rcand`, and `basel`.

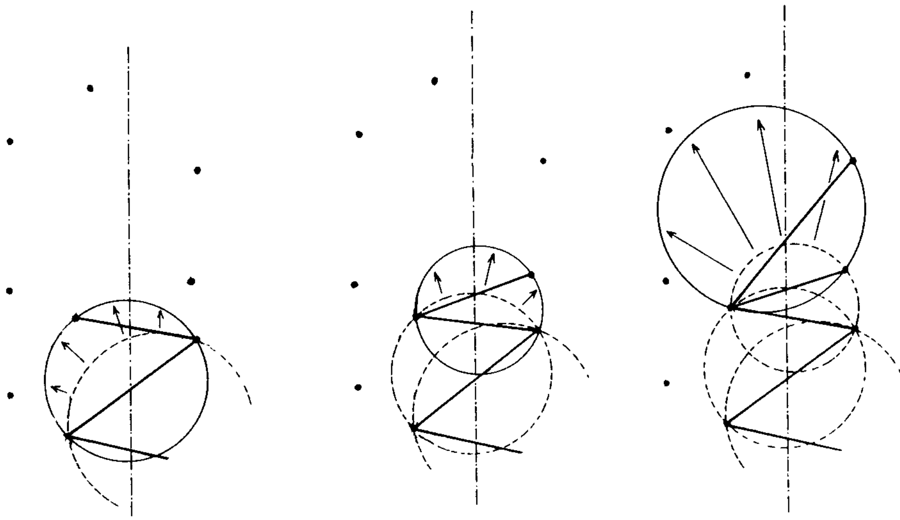

Fig. 22. The rising bubble.

held in the variable `lcand`, and their symmetric counterparts in `rcand` (see Figure 21).

We can intuitively view what happens by imagining that a circular bubble weaves its way in the space between L and R and in so doing gives us the cross edges. Inductively we have a point-free circle circumscribing the triangle defined by `basel` and the previous cross edge. Consider continuously transforming this circle into other circles having `basel` as a chord but lying further into the half-plane above `basel`. As we remarked, there is only a single degree of freedom, as the center of the circle is constrained to lie on the bisector of `basel` (see Figure 22). Our circles will be point free for a while, but unless `basel` is the upper

common tangent of L and R, at some point the circumference of our transforming circle will encounter a new point, belonging either to L or R, giving rise to a new triangle with a point-free circumcircle. The new L–R edge of this triangle is the next cross edge determined by the body of the main loop below.

In more detail, edge `lcand` is computed so as to have as destination the first L point to be encountered in this process, and `rcand` the first R point. A final test chooses the point among these two that would be encountered first. We start the cross edge iteration by computing the lower common tangent of L and R, which defines the first cross edge.

The divide-and-conquer algorithm is coded in Figure 9.5. Prose within {} is comments. The program computes the Delaunay triangulation of a point set S and returns two edges, `le` and `re`, which are the counterclockwise convex hull edge out of the leftmost vertex and the clockwise convex hull edge out of the rightmost vertex, respectively.

The only geometric primitives we use are the InCircle test and the predicate CCW(A, B, C), which is true if the points A, B, and C form a counterclockwise-oriented triangle.[2] The CCW test is frequently used to test whether a point X lies to the right or to the left of the line of a given edge e. These tests are conveniently expressed by the procedures

```
PROCEDURE RightOf[X, e]:
  RETURN CCW[X, e.Dest, e.Org]
END RightOf.

PROCEDURE LeftOf[X, e]:
  RETURN CCW[X, e.Org, e.Dest]
END LeftOf.
```

The procedure `Valid[e]` tests whether the edge e is above `basel`:

```
Valid[e] ≡ RightOf[e.Dest, basel]
         ≡ CCW[e.Dest, basel.Dest, basel.Org].
```

We now elaborate on the program in Figure 23. Recall first that the number of vertices, edges, and faces in a triangulation are all linearly related. Also, the lower common tangent computation takes linear time as either `ldi` or `rdi` advances at each step. What about the cost of the `lcand` computation? We can account for this loop by charging every iteration to the edge being deleted. Similarly, iterations of the `rcand` loop can be charged to deleted edges. The rest of the body of the main loop requires constant time and may be charged to the

[2] The predicate CCW(A, B, C) can be implemented as the test

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix} > 0$$

and tells us on which side of the line AB the point C lies. It is equivalent to InCircle(A, B, C, D), for D chosen as the barycenter of triangle ABC.

```
PROCEDURE Delaunay [S] RETURNS [le, re]:
  IF |S| = 2 THEN
    {Let s1, s2 be the two sites, in sorted order. Create an edge a from s1 to s2:}
    a ← MakeEdge[ ]; a.Org ← s1; a.Dest ← s2; RETURN [a, a.Sym]
  ELSIF |S| = 3 THEN
    {Let s1, s2, s3 be the three sites, in sorted order.}
    {Create edges a connecting s1 to s2 and b connecting s2 to s3:}
    a ← MakeEdge[ ]; b ← MakeEdge; Splice[a.Sym, b];
    a.Org ← s1; a.Dest ← b.Org ← s2; b.Dest ← s3;
    {Now close the triangle:}
    IF CCW[s1, s2, s3] THEN c ← Connect[b, a]; RETURN [a, b.Sym]
    ELSIF CCW[s1, s3, s2] THEN c ← Connect [b, a]; RETURN [c.Sym, c]
    ELSE {The three points are collinear:} RETURN [a, b.Sym] FI
  ELSE {|S| ≥ 4. Let L and R be the left and right halves of S.}
    [ldo, ldi] ← Delaunay[L]; [rdi, rdo] ← Delaunay[R];
    {Compute the lower common tangent of L and R:}
    DO
      IF LeftOf[rdi.Org, ldi] THEN ldi ← ldi.Lnext
      ELSIF RightOf[ldi.Org, rdi] THEN rdi ← rdi.Rprev
      ELSE EXIT FI
    OD;
    {Create a first cross edge basel from rdi.Org to ldi.Org:}
    basel ← Connect[rdi.Sym, ldi];
    IF ldi.Org = ldo.Org THEN ldo ← basel.Sym FI;
    IF rdi.Org = rdo.Org THEN rdo ← basel FI;
    DO {This is the merge loop.}
      {Locate the first L point (lcand.Dest) to be encountered by the rising bubble,}
      {and delete L edges out of basel.Dest that fail the circle test.}
      lcand ← basel.Sym.Onext;
      IF Valid[lcand] THEN
        WHILE InCircle
          [basel.Dest, basel.Org, lcand.Dest, lcand.Onext.Dest]
          DO t ← lcand.Onext; DeleteEdge[lcand]; lcand ← t OD
      FI;
      {Symmetrically, locate the first R point to be hit, and delete R edges:}
      rcand ← basel.Oprev;
      IF Valid[rcand] THEN
        WHILE InCircle
          [basel.Dest, basel.Org, rcand.Dest, rcand.Oprev.Dest]
          DO t ← rcand.Oprev; DeleteEdge[rcand]; rcand ← t OD
      FI;
      {If both lcand and rcand are invalid, then basel is the upper common tangent:}
      IF NOT Valid[lcand] AND NOT Valid[rcand] THEN EXIT FI;
      {The next cross edge is to be connected to either lcand.Dest or rcand.Dest.}
      {If both are valid, then choose the appropriate one using the InCircle test:}
      IF NOT Valid[lcand] OR
        (Valid[rcand] AND
          InCircle[lcand.Dest, lcand.Org, rcand.Org, rcand.Dest])
        THEN {Add cross edge basel from rcand.Dest to basel.Dest:}
          basel ← Connect[rcand, basel.Sym]
        ELSE {Add cross edge basel from basel.Org to lcand.Dest:}
          basel ← Connect[basel.Sym, lcand.Sym]
      FI
    OD;
    RETURN [ldo, rdo]
  FI
END Delaunay.
```

Fig. 23. The divide-and-conquer algorithm.

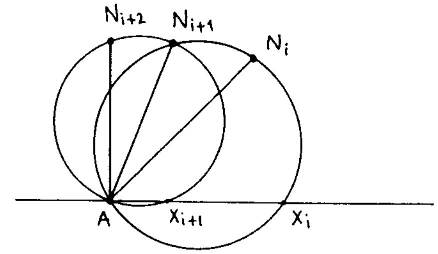

Fig. 24. A property of the neighbors of A.

L–L or R–R edge closing the next triangle. This shows that the overall cost of the merge pass is linear in the size of L and R.

We now formally state the lemmas that prove the correctness of the algorithm.

LEMMA 9.2. *Let $\mathcal{U}$ be any collection of sites, and consider a particular site A and a line l passing through A. For convenience of terminology, assume that l is horizontal. Let $N_1, N_2, \ldots, N_k$ ($k \geq 1$) be some Delaunay neighbors of A in $\mathcal{U}$, occurring in counterclockwise order from l and lying above l. If X is any point of l to the right of A, let Γ_i denote the portion of the circumcircle (disk) AXN_i that is above l. Then the sequence $\{\Gamma_i\}$ is unimodal, in the sense that there is some j, $1 \leq j \leq k$, such that for $1 \leq i < j$ we have $\Gamma_i \supseteq \Gamma_{i+1}$, while for $j \leq i < k$ we have $\Gamma_i \subseteq \Gamma_{i+1}$.*

PROOF. We first show that if X_i denotes the rightmost intersection of the circumcircle of triangle AN_iN_{i+1}, $i = 1, 2, \ldots, k-1$, with l, then the sequence of points $X_1, X_2, \ldots, X_{k-1}$ moves monotonically to the left.

Consider, as in Figure 24, three consecutive Delaunay neighbors N_i, N_{i+1}, and N_{i+2} of A. The point N_{i+2} is not inside the circle AN_iN_{i+1}, and points N_i and N_{i+2} are on opposite sides of AN_{i+1}. So to get to the circle $AN_{i+1}N_{i+2}$ from AN_iN_{i+1} while always passing through A and N_{i+1}, we must expand on the side of N_{i+2}, and therefore we must contract on the side of N_i, where X_i and X_{i+1} also lie. This proves that the X_i move toward A.

To prove the lemma now, note that as long as the X_i are to the right of X, then X is inside the circumcircle of AN_iN_{i+1}, or equivalently, N_{i+1} is inside the circumcircle of AN_iX. After the X_i move to the left of X, then N_{i+1} is outside the circumcircle of AN_iX. Thus the Γ_i behave as stated. □

LEMMA 9.3. *Assume that* `basel`*, as computed by the above algorithm, is a Delaunay edge for $L \cup R$, and that the* `lcand` *iteration stops with a valid edge. Then the circumcircle of the triangle defined by* `lcand` *and* `basel` *is free of other L points.*

PROOF. Recall that the edge `lcand` is valid if and only if its destination N lies above `basel`. Let X and Y be the origin and destination of `basel`. Consider the point-free circle C that established the Delaunayhood of `basel`. As in our earlier discussion, suppose that we "push" this circle upward while constraining it to pass through X and Y, until it encounters a new point $M \in L$ for the first time (if it encounters two or more such points at the same time, we let M be the one for which the angle XYM is smallest). The edge YM is then an L Delaunay edge, and in fact a Delaunay edge of L together with any R sites encountered so far. This implies that YN has not been deleted previously by the algorithm.

We wish to show that M is the same as N, the destination of `lcand` when the iteration stops. If there is only one Delaunay neighbor of Y above `basel`, then the iteration will stop right away. To see this, consider N', defined as the destination of the edge $(YN)Onext$. The site N' is below `basel` and outside C, and therefore outside the circle YXN, so InCircle(Y, X, N, N') fails.

If there are several Delaunay neighbors of Y above `basel`, then the `lcand` iteration proceeds until the first time that InCircle(Y, X, N, N') becomes false. So at this point we know that all current Delaunay neighbors of Y in L that are above `basel` and counterclockwise before N lie outside the circle YXN. The same is true of the neighbor N'. By the unimodality lemma, Lemma 9.2, this will also hold for any subsequent neighbors after N'. Therefore the circle YXN has the smallest extent above `basel` among all circles passing through Y, X, and some L Delaunay neighbor of Y above `basel`. This proves that N is the same as M. □

LEMMA 9.4. *Assume that* `basel` *is a Delaunay edge of $L \cup R$. The edges deleted during the* `lcand` *iteration are not Delaunay edges of $L \cup R$.*

PROOF. Let X and Y be the origin and destination of `basel`. When the `lcand` loop starts, we know that `lcand` (1) is above the line of `basel`, and (2) is also the first edge out of Y after `basel.Sym`. Therefore, the next edge `lcand.Onext` out of Y is either to the left of `lcand` or below the line XY (or both). As in the previous lemma, let N and N' respectively denote the destinations of `lcand` and `lcand.Onext`. If the text InCircle(Y, X, N, N')) succeeds, then N' is strictly inside the circle YXN. Now suppose N' were below `basel`; by Lemma 8.3 the pair N, N' would be a witness to the non-Delaunayhood of `basel`, contradicting the assumption. Therefore, N' too is above `basel`, and to the left of `lcand`. The sites N' and X lie on the opposite sides of `lcand`, and by Lemma 8.3 they establish the non-Delaunayhood of `lcand`. After `lcand` is deleted, it is set to the next edge YN', and properties (1) and (2) are restored; by induction, the argument applies to each iteration. A symmetric argument works for the edges deleted in the `rcand` loop. □

LEMMA 9.5. *After the merge pass is complete, the subdivision is a Delaunay triangulation for $L \cup R$.*

PROOF. First, let us show that the subdivision will be a triangulation of $L \cup R$. The cross edge `basel` starts with the lower common tangent of L and R, and at each major iteration it will intersect the separating line at a higher point. The loop will end with `basel` being the upper common tangent of L and R. The edges on the outer parts of the convex hulls of L and R will never be deleted and, together with the first and last cross edges, they will constitute the convex hull of $L \cup R$. The interior faces of the subdivision will be either original faces (triangles) of the Delaunay triangulations of L and R or will be new triangles delimited by two new L–R edges and an old L–L or R–R edge.

Now we have to show that the triangulation is Delaunay. Edges that are on the convex hull or are incident to two old triangles all pass the circle test. This includes the first cross edge `basel`. At each major iteration, we determine `lcand` such that the circumcircle of the triangle determined by `basel` and `lcand` is free from L points, and similarly for `rcand` (Lemma 9.3). The final choice between `lcand` and `rcand` ensures that the corresponding circle is free of both L and R points. This circle is a witness to the Delaunayhood of the triangle determined by `basel`, the winner, and the new cross edge. We conclude that all edges pass the circle test; by Lemma 8.4, the triangulation is Delaunay. □

These lemmas complete the proof that the algorithm correctly computes the Delaunay triangulation. The algorithm will work with cocircular sites and other degenerate cases. When both `lcand` and `rcand` are equally good, it arbitrarily favors `rcand`. In practice, floating-point errors in the computation of the InCircle test usually interfere with any effort to handle degenerate cases in a consistent way.

It is worthwhile to comment on a way to view this algorithm as operating in three-dimensional space, on the lifted images of our sites under the map λ discussed in the proof of Lemma 8.1. These lifted images are points on a convex surface, and therefore they define a convex polyhedron, corresponding to their convex hull. The discussion in the proof of Lemma 8.1 has also established that the "downward" looking faces of this polyhedron are in a one-to-one correspondence with the Delaunay faces of the sites. The upward looking faces of the polyhedron correspond to triangles of our collection of sites whose circumcircles enclose all the sites. These are of course the faces of the dual of the *furthest point* Voronoi diagram [18] for our collection of sites. Note that when our cross edge iteration ends, the ascending circle is the half-plane above the upper common tangent of L and R. The half-plane below this tangent is a circle containing all the sites. If we now let this circle contract until it hits the first site while always having as chord the last cross edge, we will have produced the next cross edge of the dual of the furthest point Voronoi. In fact, if our cross edge iteration is left to continue, but by using the furthest point versions of L and R and reversing the sense of the InCircle test, it will compute all the cross edges of this dual and will cycle back to the lower common tangent of L and R.

From the above discussion it is apparent that our divide-and-conquer algorithm is computing the convex hull of the lifted images of the sites. It is in fact exactly the Preparata–Hong [17] algorithm for computing the convex hull of n points in three dimensions. If the InCircle test is replaced by a "positive volume" test, as obtained by substituting in $\mathscr{D}(A, B, C, D)$ the third column by the z coordinates of the points, then the code given above implements the Preparata–Hong algorithm! The reader may have fun verifying that our expanding circles passing through a chord become rotating supporting planes around the lifted image of the chord. Thus we are computing the "sleeve" discussed in [17]. Brown [4] has observed that a similar correspondence can be obtained by lifting the sites stereographically onto a sphere.

10. AN INCREMENTAL ALGORITHM

The algorithm of the previous section assumes that all points are known at the beginning of time. For many applications we are interested in dynamic algorithms, which allow us to update our diagrams as new points are added or deleted. A simple algorithm of this sort was proposed by Green and Sibson [7]. We now proceed to describe this algorithm in detail, with the purpose of further illustrating the power of our tools.

10.1 Overview of the Algorithm

It will simplify our discussion to assume that our points are known to be strictly inside some large convex polygon (say a triangle) whose vertices are considered to be among the given sites. The Delaunay diagram of the sites inserted so far is thus a triangulation of the interior of this bounding polygon. When a new site X is given, our first step is to locate the interior triangle of the subdivision containing X, and add three new edges connecting its vertices to X. The new site X may happen to fall on some existing edge e; in that case, we delete e, and connect X to the four vertices of the quadrangular "hole" thus created. If the new site coincides with a previous one, we ignore it. See Figure 25.

Below we prove that the three or four new edges now incident to X are guaranteed to be Delaunay edges. However, some of the old edges may now be incorrect and might have to be replaced. In general, we will be in a situation where our new point X is surrounded by a collection of triangles defining a star-shaped polygon around X. The spokes connecting X to this polygon will be known to be Delaunay edges. The edges on the boundary, however, will be of two kinds. Some will be confirmed Delaunay edges, while others will be marked as "suspect." A suspect edge is one which is not known to pass the circle test defined in Section 8.

The incremental algorithm proceeds by choosing a suspect edge and applying the circle test to it. We prove below that if the edge passes the test, it is guaranteed to be a Delaunay edge and need not be considered further. If it fails, however, it is *swapped*, that is, replaced by the other diagonal of its quadrilateral. In that case the new diagonal can be shown to be Delaunay while the two sides opposite X become suspect, thus reestablishing the initial situation (see Figure 26). The algorithm terminates when no suspect edges remain.

In order to prove the correctness of the algorithm described above we need a few lemmas.

LEMMA 10.1. *The edges initially made incident to X are Delaunay.*

PROOF. Consider the circumcircle C of a Delaunay triangle that contains the new site (or is adjacent to the edge containing it). For each vertex Y of that triangle, consider the circle C' that passes through X and is tangent to C and Y. That circle is point-free, and it establishes the "Delaunayhood" of the edge XY. □

LEMMA 10.2. *Any edge made incident to X by swapping is Delaunay. So is any suspect edge that passes the circle text.*

PROOF. Suppose that the edge LN with quadrilateral $XLMN$ was swapped with the opposite diagonal XM, as in Figure 26. Then X must be the only site interior to the circle $C = LMN$. As in Lemma 10.1, we can find a circle contained in C and passing through X and M; this circle is point-free and proves XM is a

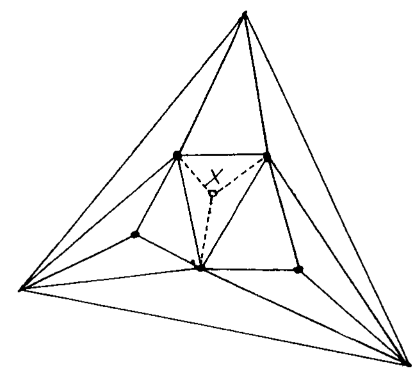

Fig. 25. Preliminary insertion of a new site.

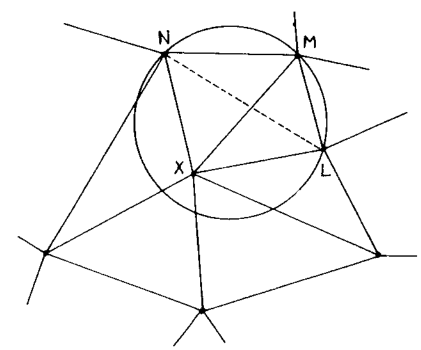

Fig. 26. Swapping a suspect edge.

Delaunay edge. If the edge LN passed the circle test, then X is outside C; C is still point-free and proves LN to be a Delaunay edge. □

LEMMA 10.3. *No edge is tested more than once (for each new site).*

PROOF. An edge e becomes suspect if and only if there is exactly one edge e' between it and the point X, and e' is swapped. Since the edges introduced by the swapping rule are always incident to X, this situation can occur at most once for each site insertion. □

In fact, the same argument shows that if a suspect edge e passes the circle test, the algorithm will never consider again any edge in the angular sector with vertex X and spanned by e.

LEMMA 10.4. *Once all suspect edges have been checked, all edges in the triangulation pass the circle test.*

PROOF. From Lemma 10.2 we know that all new edges incident to X are Delaunay edges and therefore they pass the circle test. Any old edge either became suspect and passed the test at some later time or never became suspect and was known to pass the test just before X was inserted. In either case, its quadrilateral did not change (otherwise it would have become suspect) since that time, and therefore it still satisfies the edge test. □

The above lemmas imply that the incremental algorithm terminates and correctly computes the updated Delaunay triangulation.

10.2 Coding the Algorithm

If at each iteration the next edge to be tested is chosen in a consistent way, the suspect edges will always form a continuous chain on the perimeter of the star-shaped polygon surrounding X. Therefore, if we know the first edge in that chain, we can get all the others by following the pointers of the quad-edge data structure. The following code implements this idea in more detail. We assume the procedure `Locate` returns an edge e of the current Delaunay diagram such that the given point X is either on e or strictly inside the left face of e.

```
PROCEDURE InsertSite[X]:
  e ← Locate[X];
  IF X = e.Org OR X = e.Dest THEN {Ignore it:} RETURN
  ELSIF X is on e THEN t ← e.Oprev; DeleteEdge[e]; e ← t FI;
  {Connect X to vertices around it.}
  base ← MakeEdge[ ];
  first ← e.Org; base.Org ← first; base.Dest ← X;
  Splice[base, e];
  REPEAT
    base ← Connect[e, base.Sym]; e ← base.Oprev
  UNTIL e.Dest = first;
  e ← base.Oprev;
  {The suspect edges (from top to bottom) are e(.Onext.Lprev)^k for k = 0, 1, ....}
  {The bottom edge has .Org = first.}
  DO
    t ← e.Oprev;
    IF RightOf[t.Dest, e] AND InCircle[e.Org, t.Dest, e.Dest, X]
    THEN Swap[e]; e ← t
    ELSIF e.Org = first THEN {No more suspect edges.} RETURN
    ELSE {Pop a suspect edge:} e ← e.Onext.Lprev FI
  OD
END InsertSite.
```

The main loop of this algorithm is in some aspects similar to the merge step of the one given in Section 9. Consider the set L as being reduced to the single point X, and R as including all the previous sites. The "cross edges" then are the "spokes," the new edges incident to X; instead of being linearly ordered along the separating line, they are cyclically ordered around X. The edge `e` here plays a role similar to that of `rcand`, and `lcand` is always invalid (nonexistent, in fact). The equivalent of `basel` is the last spoke added, that is, `e.Onext`. The successively found `rcand`s, as we proceed counterclockwise around the new point, will correspond to the forgotten edges of the previous algorithm. Note, however, that the incremental algorithm "connects ahead" after each deletion, while the algorithm of the previous section would connect all cross edges in strict counterclockwise order.

Something quite general is happening here. We have a method with which, given any two Delaunay triangulations L and R (not necessarily linearly separated) and a cross edge between them, we are able to find the next cross edge (if one exists) on a specified side of the original one. Thus we have another way to look at Kirkpatrick's [10] linear time merge of two arbitrary Voronoi subdivisions.

The above arguments show that it is possible to insert a new site into the Delaunay structure in total time $O(k)$, if k updates need to be made. Unfortunately we know of no $O(k)$ algorithm for handling the deletion of a site that leaves an untriangulated face of k sides. Our best algorithm has asymptotic complexity $O(k \log k)$, which in the worst case $k = \Theta(n)$ is as bad as rebuilding the subdivision from scratch. We do not know of a linear algorithm even if we assume that the deletion of the site leaves a convex face. We regard the handling of deletions as the major open problem in this area.

10.3 Locating a Point in the Delaunay

A suitable algorithm to use for the `Locate` procedure is the "walking" method described by Green and Sibson [7]. The idea is to start at some arbitrary place on the subdivision and then move one edge at a time in the general direction of the point X. More precisely, we have

```
PROCEDURE Locate[X] RETURNS [e]:
  e ← some edge;
  DO
    IF X = e.Org OR X = e.Dest THEN RETURN e
    ELSIF RightOf[X, e] THEN e ← e.Sym
    ELSIF NOT RightOf[X, e.Onext] THEN e ← e.Onext
    ELSIF NOT RightOf[X, e.Dprev] THEN e ← e.Dprev
    ELSE RETURN e FI
  OD
END Locate.
```

10.4 Analysis

The `Locate` procedure given above terminates in $O(n)$ time for a triangulation with n vertices. From this and Lemma 10.4 we derive an $O(n)$ worst-case bound for the cost of the insertion of the nth site. Point location methods that are asymptotically faster (in the worst case) have been described in the literature [11], but they would not improve the worst-case cost of site insertion and are probably too complex to be of practical use here. Moreover, those methods generally assume that subdivision is fixed, so the $O(n)$ cost of building the associated data structures can be spread out over many queries.

A more careful analysis shows that except for point location, the algorithm only does the work that needs to be done: it deletes only edges that have to be deleted and inserts only the edges that have to be inserted. If the updated Delaunay has k edges incident to the new site X, then the running time (exclusive of point location) will be $\Theta(k)$. The algorithm is therefore asymptotically optimal for the Delaunay update problem.

It is possible to select n sites in such a way that the incremental algorithm does $\Theta(k)$ work to insert the kth site in the diagram of the preceding $k - 1$ ones, for all k. The total time for the insertion of all n sites is therefore $\Theta(n^2)$ in the worst case. In spite of this, the simplicity of the incremental algorithm may make it competitive with the divide-and-conquer method even in the cases where the sites are all given in advance, and in most practical situations its performance can be quite acceptable. In particular, if the sites are independently sampled from a reasonably uniform probability distribution, the expected time for each insertion is small and roughly independent of n. The running time is then dominated by point location. For the simple walking algorithm, the expected time in this case will be roughly $O(n^{1/2})$ per site, or $O(n^{3/2})$ in total. Furthermore, in many practical applications successive sites tend to be close to each other; therefore, if the `Locate` procedure uses as its starting point the edge returned in the previous call, each insertion may take roughly constant time, on the average. In such cases the incremental method may beat the divide-and-conquer one even for thousands of sites [7, 14].

11. CONCLUSIONS

In this paper we have presented a new data structure for planar subdivisions that simultaneously represents the subdivision, its dual, and its mirror image. Our quad-edge structure is both general (it works for subdivisions on any two-dimensional manifold) and space efficient. We have shown that two topological operations, both simple to implement, suffice to build and dismantle any such structure.

We have also shown how by using the quad-edge structure and the InCircle primitive, we can get compact and efficient Voronoi/Delaunay algorithms. The InCircle test is shown to be of value both for implementing and reasoning about such algorithms. The code for these algorithms is sufficiently simple that we have practically given all of it in this paper.

REFERENCES

1. BAUMGART, B. G. A polyhedron representation for computer vision. In *1975 National Computer Conference*. AFIPS Conference Proceedings, vol. 44. AFIPS Press, Arlington, Va., 1975, pp. 589–596.
2. BOOTS, B. N. Delaunay triangles: An alternative approach to point pattern analysis. In *Proc. Assoc. Am. Geogr. 6* (1974), 26–29 (as cited by [20]).
3. BRAID, I. C., HILLYARD, R. C., AND STROUD, I. A. Stepwise construction of polyhedra in geometric modeling. In *Mathematical Methods in Computer Graphics and Design*, K. W. Brodlie, Ed. Academic Press, London, 1980, pp. 123–141.
4. BROWN, K. Q. Voronoi diagrams from convex hulls. *Inf. Proc. Lett. 9*, 5 (1979), 223–228.
5. DAMPHOUSSE, P. Cartographie topologique—La classification des cartes cellulaires. Unpublished Rep., Univ. of Tours, France.
6. EVEN, S. *Algorithmic Combinatorics*. Macmillan, N.Y., 1973.
7. GREEN, P. J., AND SIBSON, R. Computing Dirichlet tesselation in the plane. *Comput. J. 21*, 2 (1977), 168–173.
8. HARARY, F. *Graph Theory*. Addision-Wesley, Reading, Mass., 1972, p. 105.
9. IYANAGA, S., AND KAWADA, Y. *Encyclopedic Dictionary of Mathematics*, 2nd. ed. MIT Press, Cambridge, Mass., 1968.
10. KIRKPATRICK, D. Efficient computation of continuous skeletons. In *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science* (San Juan, Puerto Rico, Oct. 1979), IEEE, New York, pp. 18–27.
11. KIRKPATRICK, D. Optimal search in planar subdivisions. Tech. Rep. 81-13, Dep. of Computer Science, Univ. of British Columbia, 1981.

12. KNUTH, D. E. *The Art of Computer Programming, vol. I: Fundamental Algorithms*, 2nd. ed. Addision-Wesley, Reading, Mass., 1975.
13. LEE, D. T. Proximity and reachability in the plane. Tech. Rep. R-831, Coordinated Science Lab., Univ. of Illinois, Urbana, Ill., 1978.
14. LEE, D. T., AND SCHACHTER, B. J. Two algorithms for constructing the Delaunay triangulation. *Int. J. Comput. Inf. Sci. 9*, 3 (1980), 219–242.
15. MANTYLA, M. J., AND SULONEN, R. GWB: A solid modeler with Euler operators. *IEEE Comput. Graphics Appl. 2*, 5 (Sept. 1982), 17–31.
16. MULLER, D. E., AND PREPARATA, F. P. Finding the intersection of two convex polyhedra. *Theor. Comput. Sci. 7* (1978), 217–236.
17. PREPARATA, F. P., AND HONG, S. J. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM 20* (1977), 87–93.
18. SHAMOS, M. I. Computational geometry. Ph.D. Dissertation, Yale University, New Haven, Conn., 1977.
19. SHAMOS, M. I., AND HOEY, D. Closest-point problems. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science* (Berkeley, Calif., Oct. 1975), IEEE, New York, pp. 151–162.
20. WATSON, D. F. Computing the n-dimensional Delaunay tesselation with application to Voronoi polytopes. *Comput J. 24*, 2 (1981), 167–172.

Received March 1983; accepted March 1984