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

Leo J. Guibas and Jorge Stolfi
Xerox PARC and Stanford University

Abstract: We discuss the following problem: 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. We give two algorithms, one that constructs the Voronoi diagram in $O(n \lg n)$ time, and another that inserts a new site in $O(n)$ time. Both are based on the use of the Voronoi dual, the 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.

0. 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 is 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 $O(n \lg n)$ time and stored in $O(n)$ space; these bounds have been shown to be optimal in the worst case [Sh]. 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 $O(\lg n)$ time [Kil].

The work of Jorge 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 (CNPq).

Shamos [Sh] 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 $\Omega(n \lg n)$ time in the worst case, so these Voronoi-based algorithms are asymptotically optimal.

The complexity of the $O(n \lg n)$ Voronoi algorithms that can be found in the literature is a serious barrier to their widespread utilization. To the authors' knowledge, they so far have never been used in any of the significant practical applications of closest-point problems in statistics, operations research, geography, and other areas. In every case the authors of those programs chose to use asymptotically slower $O(n^2)$ algorithms, which were much simpler to code and almost certainly faster for the ranges of interest in n. Furthermore, the presentation of Voronoi algorithms in the literature has often been insufficiently precise. Authors typically confine themselves only to a very high-level description of their algorithms. As with many geometric problems, difficulties can arise in the implementation when degeneracies occur, as for example when three of the given points happen to be cocircular.

In this paper we present a novel way of looking at the standard Voronoi computation techniques, such as the divide and conquer [SH] and incremental [GS] methods, that results in Voronoi algorithms that are very concise and substantially easier to read, implement and verify. One of the main reasons for this simplicity is that we work with the duals of the Voronoi diagrams, which are known as *Delaunay triangulations*, rather than with the diagrams themselves. The other major reason is the clean separation that we are able to make between topological and geometrical aspects of the problem. In sections 6 through 10 we show that the hardest part of constructing a Voronoi diagram or Delaunay triangulation is the determination of its topological structure, that is, the incidence relations between vertices, edges, and faces. Once the topological properties of the diagram are known, the geometrical ones (coordinates, angles, lengths, etc.) can be computed in time linear in the size of the diagram.

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, that 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. 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 through 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 [Ba]. 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 both 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 presents the very important ideas of the dual of a subdivision, and the edge algebra associated with a subdivision. The edge algebra is a combinatorial structure on the edges of the subdivision that we claim captures all the topological information associated with the subdivision. Section 3 is more technical and may be omitted on a first reading. It formalizes and then proves the above claim about edge algebras by showing that isomorphism of edge algebras is equivalent to topological homeomorphism between the corresponding subdivisions. 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 Delaunay/Voronoi computation. Section 7 reviews some properties of the Voronoi/Delauanay subdivision and section 8 presents our main geometric primitive for their computation, the *InCircle* test and its properties. Section 9 presents in detail and proves correct a divide and conquer algorithm for Voronoi computations, and section 10 discusses incremental techniques.

1. Subdivisions

In this section we will 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 [Ba, MS]. 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, and so forth) 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 [IK]. 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 $B^1 = (0\ \ 1)$ of the real line. A *disk* of M is a subspace homeomorphic to the open circle of unit radius $B^2 = \{ x \in R^2 : |x| < 1 \}$. 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 the *faces* (denoted respectively by $\mathcal{V}S$, $\mathcal{E}S$, and $\mathcal{F}S$), 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 1.1 shows some examples of subdivisions.

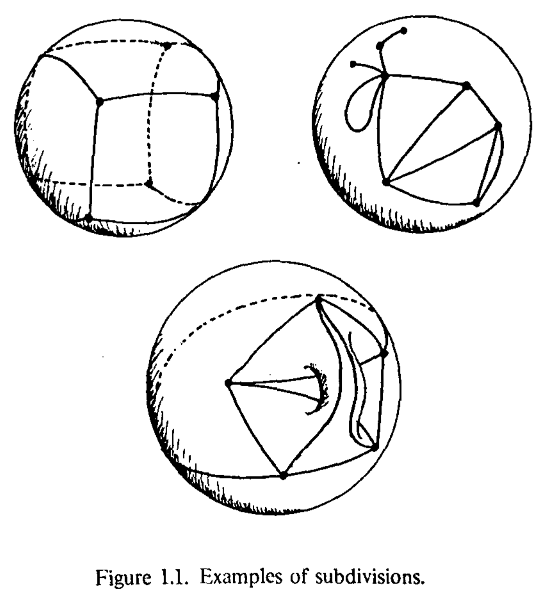


Figure 1.1. Examples of subdivisions.

Condition S4 needs some explanation. We will denote by $\overline{B}^2$ the *closed* circle of unit radius, and by S^1 its circumference. Let us define a *simple path* in S^1 as a partition of S^1 into a finite sequence of isolated points and open arcs. The precise meaning of S4 is then the following: for every face F there is a simple path π in S^1 and a continuous mapping ϕ_F from $\overline{B}^2$ onto the closure of F that (i) maps homeomorphically 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 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 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 1.1 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. 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 edge; if they are the same, then the edge is a *loop*, and its closure is homeomorphic to the circle S^1.

Since every element of S is in the closure of some face, and since the closed disk $\overline{B}^2$ 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; any manifold can be made compact by adding a dummy "point at infinity" that is by definition an accumulation point of all sequences with no other accumulation points. This operation transforms the Euclidean plane 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, edges into edges, and will preserve the incidence relationships among them. 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 doesn't follow that they are equivalent, or that M and M' are homeomorphic. Figure 1.2 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.

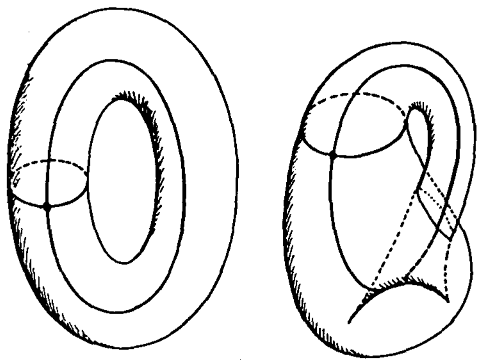


Figure 1.2. Two subdivisions with isomorphic graphs that are not equivalent.

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 [Har], 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 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 subsection 2.2). One important consequence of this restriction is stated below:

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 the 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. The edge algebra of a subdivision

In this section we will 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 will develop first the theory and representation for arbitrary compact manifolds, and then we will show that certain important simplifications can be made in the particular case when 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* of 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 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, 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 will denote by VS, ES and FS 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.

A sufficiently small disk containing the vertex $v = e\,Org$, it can be mapped homeomorphically onto the unit disk 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 ray of B^2. Traversing the boundary of D in the counterclockwise direction (as defined by the orientation of v) establishes a cyclical ordering of those edges. If each edge is oriented so as to agree with v, and directed away from D, we obtain what is called the *ring of edges out of* v. We can define the *next edge with same origin*, $e\,Onext$, as the one immediately following e (counterclockwise) in this ring (see figure 2.1). 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 occur only once in each circuit, and we will never encounter $e\,Flip$.

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.

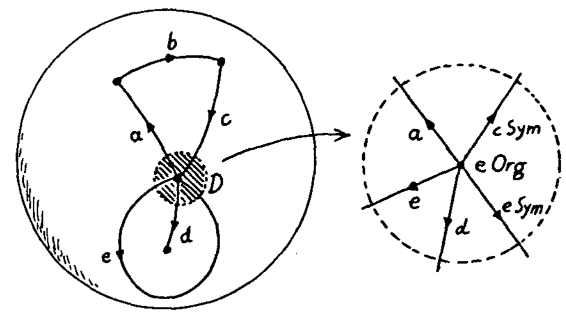


Figure 2.1. The ring of edges out of a vertex.

2.2. Duality

The dual of a planar graph G can be defined intuitively 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 ES and ES^*, between VS and FS^*, and between FS and VS^*, 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 2.2 shows a subdivision of the extended plane (solid lines) superimposed on its dual (dotted lines). Note that the two subdivisions of figure 2.2 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 $(e\,Dual)\,Org$.

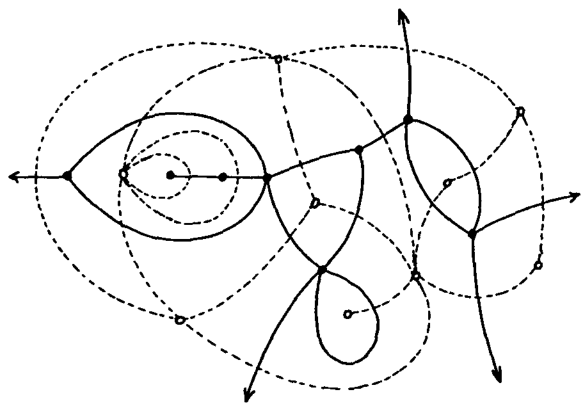


Figure 2.2. A subdivision of the extended plane (solid lines) and a strict dual (dashed lines).

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 ES to ES^* without this implicit "flipping". The edge $e\,Rot$ is called the *rotated* version of e; it is the undirected 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, like in figure 2.2, 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 fails 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. $e \in ES$ iff $e\,Rot \in ES^*$

E5. $e \in ES$ iff $e\,Onext \in ES$

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. $e \in ES$ iff $e\,Flip \in ES$

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

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

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:

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

The orientation and direction of the returned 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

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

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 2.3 illustrates these various functions.

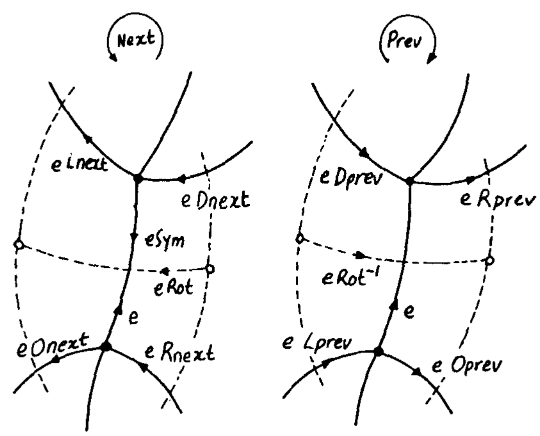


Figure 2.3. The edge functions.

2.4. Edge algebras

Definition 2.2. 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 F1–F5 and E1–E5.

The axioms 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. The orbits of *Onext* in E are in one-to-one correspondence with the vertices of S, and therefore also with the faces of S^*. These orbits are joined by *Flip* in pairs of opposite orientation. We can therefore define $e\,Org$ in an edge algebra as the orbit of e under *Onext*, and similarly $e\,Left = e\,Rot^{-1}\,Org$, $e\,Right = e\,Rot\,Org$, $e\,Dest = e\,Sym\,Org$. Notice that $e\,Flip\,Org$ is the same as $e\,Org\,Flip$ (the set obtained by *Flip*ping every element of $e\,Org$).

Edge algebras are a purely combinatorial abstraction of subdivisions, in the same way that undirected graphs are an abstraction of the subdivision graphs. In the next section we will defend the position that *all topological properties of a subdivision S are accurately captured by an edge algebra.* As a consequence this algebra, which is a finite object, can be used as a computer representation of S. 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.

3. From edge algebras back to subdivisions

In the present section we will show that the topological properties of an arbitrary subdivision S are accurately and unambiguously represented by its associated edge algebra. The concepts and theorems developed in this section are essential for showing the consistency and completeness of the data structure but are not used in the rest of the paper, so the reader whose interest is mostly practical can skip to section 4. The major idea will be to 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.

3.1. Completions

Definition 3.1. 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 c_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.

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 $\mathcal{V}\Sigma$, $\mathcal{C}\Sigma$, 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 primal one.* These links are denoted $\mathcal{L}\Sigma$, $\mathcal{L}^*\Sigma$, and $\mathcal{K}\Sigma$, in that order. Figure 3.1 shows a completion of a subdivision of the extended plane.

Definition 3.1 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 around π_F. To describe this process precisely, let ϕ_F be any continuous function from $\overline{\mathcal{B}}^2$ to the closure of F that establishes condition S4. Let $\pi = (u_1, \alpha_1, u_2, \alpha_2, \ldots, u_n, \alpha_n, u_{n+1} = u_1)$ be the path in the circle 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 $\phi_F(\alpha_i)$. Take $\phi_F((0,0))$ to be the dual vertex v_F; connect in $\overline{\mathcal{B}}^2$ 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 to 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.

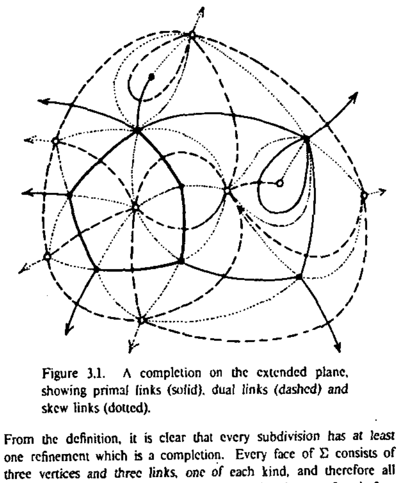


Figure 3.1. A completion on the extended plane, showing primal links (solid), dual links (dashed) and skew links (dotted).

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 $\overline{\mathcal{B}}^2$ bounded by two rays and an arc $u_i c_i$ or $c_i u_{i+1}$, 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 between 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 $\mathcal{C}\Sigma_1$, $\mathcal{L}\Sigma_1$, $\mathcal{L}^*\Sigma_1$, 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 $e^* = \ell_1 \cup \{c_e\} \cup \ell_2$, 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 of 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) $(\ell_1, t_1, \ell_2, t_2, \ldots, \ell_n, t_n, \ell_{n+1} = \ell_1)$ 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 esablishes 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 3.2. Let Σ be a completion. Let *Rot* be the function from $ES\Sigma \cup ES^*\Sigma$ into itself defined as follows: for every edge $e \in ES\Sigma$, 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 $e \in ES^*\Sigma$ 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 $A\Sigma = (ES\Sigma, ES^*\Sigma, Onext, Rot, Flip)$

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 $ES\Sigma$ (or $ES^*\Sigma$) 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 $ES\Sigma$ or $ES^*\Sigma$.

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's 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 $e\,Dual = e\,Flip\,Rot$ satisfied 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 $S^*\Sigma$).

3.2. 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 a 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 $A_i = (ES_i, ES_i^*, Onext_i, Rot_i, Flip_i)$, and let η be the homeomorphism between the manifolds of S_1 and S_2 that establishes their equivalence. An orientation or direction for an element of S_1 determines via η a unique orientation or direction for the corresponding element in S_2, and so η is also a one-to-one correspondence between ES_1 and ES_2. From the definition of *Onext* we can conclude that $\eta(e\,Onext_1) = \eta(e)\,Onext_2$ for all $e \in ES_1$; the same holds for *Lnext* and *Flip*.

Let us now define the function ξ from $ES_1 \cup ES_1^*$ to $ES_2 \cup ES_2^*$ as

$$\xi(e) = \begin{cases}\eta(e) & \text{if } e \in ES_1,\\ \eta(e\,Rot_1^{-1})\,Rot_2 & \text{if } e \in ES_1^*.\end{cases}$$

Clearly ξ is one-to-one, for Rot_i is one-to-one from ES_i to ES_i^*. Let us now show that $\xi(e\,Onext_1) = \xi(e)\,Onext_2$. If $e \in ES$ the proof is trivial. If $e \in ES_1^*$, then $e\,Onext_1 \in ES_1^*$, and

$$\begin{aligned}\xi(e\,Onext_1) &= \eta(e\,Onext_1\,Rot_1^{-1})\,Rot_2\\ &= \eta(e\,Rot_1^{-1}\,Lprev_1\,Rot_1\,Rot_1^{-1})\,Rot_2\\ &= \eta(e\,Rot_1^{-1}\,Lprev_1)\,Rot_2\\ &= \eta(e\,Rot_1^{-1})\,Lprev_2\,Rot_2 \quad (\text{since } e\,Rot_1^{-1} \in ES)\\ &= \eta(e\,Rot_1^{-1})\,Rot_2\,Rot_2^{-1}\,Lprev_2\,Rot_2\\ &= \xi(e)\,Onext_2\end{aligned}$$

The proof for $\xi(e\,Flip_1) = \xi(e)\,Flip_2$ is entirely similar, using $e\,Flip_1 = e\,Rot_1^{-1}\,Flip_1\,Rot_1$, and finally $\xi(e\,Rot_1) = \xi(e)\,Rot_2$ is trivial. □

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 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 $e\,Org \cup e\,Org\,Flip$ 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 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 of $\overline{T}$ onto $\overline{T'}$ that agrees with η on ∂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 3.5. *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 3.6. *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 lemma 3.4 Σ_1 and Σ_2 are equivalent; by lemma 3.5 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. Theorems 3.3 and 3.6 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.

3.3. Realizability of algebras

To conclude our thoretical 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 leaves axioms E1–E5 and F1–F5 invariant corresponds to a valid operation on manifolds.

Theorem 3.9. *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 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 corner 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 intersecting 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 disk-like 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

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

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

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 disk-like neighborhood.

We conclude that the triangles $\mathcal{T}$ 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 $\bar{e}\,Rot^r\,Flip^f$, where $r \in \{0,1,2,3\}$, $f \in \{0,1\}$, and $\bar{e}$ 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 $\bar{e}\,Rot^r$. See figure 4.1a. A generic edge $e = \bar{e}\,Rot^r\,Flip^f$ is represented by the triplet $(\mathtt{e}, r, f)$, 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" of 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 non-topological information about the edge $\bar{e}\,Rot^r$. This field neither affects nor is affected by the topological operations that we will describe, so its contents and format is entirely dependent on the application.

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

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

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 implies that

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

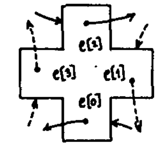
$$\begin{aligned} (\mathtt{e}, r, 1)\,Onext &= (\mathtt{e}[r+1].\mathtt{Next})\,Rot\,Flip \\ &= (\mathtt{e}, r, 0)\,Rot\,Onext\,Rot\,Flip \\ &= (\mathtt{e}, r, 0)\,Oprev\,Flip \end{aligned}$$

i.e., the 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

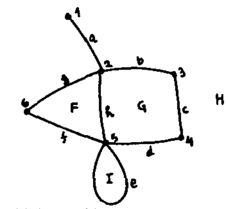
$$\begin{aligned} (\mathtt{e}, r, f)\,Sym &= (\mathtt{e}, r+2, f) \\ (\mathtt{e}, r, f)\,Rot^{-1} &= (\mathtt{e}, r+3+2f, f) \\ (\mathtt{e}, r, f)\,Oprev &= (\mathtt{e}[r+1-f].\mathtt{Next})\,Rot^{1-f}\,Flip^f, \end{aligned}$$

and so forth.

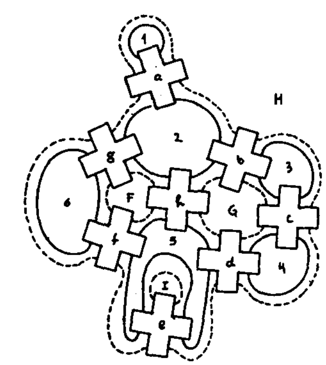
The record of an arbitrary edge belongs to four circular lists, corresponding to its two endpoints and its two faces. Figure 4.1 illustrates a portion of a subdivision and its quad edge data structure. We may think of each record as belonging to four circular lists, corresponding to the two vertices 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 4.2, where the canonical representative of edge a has orientation opposite to that of the others.



(a) Edge record showing `Next` links.



(b) A subdivision of the sphere.



(c) The data structure for the subdivision (b).

Figure 4.1. The quad-edge data structure.

The quad edge data structure contains no separate records for vertices 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 using the vertex 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 *eRot*.

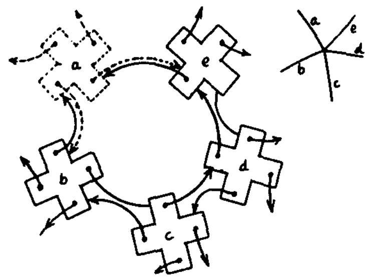


Figure 4.2. An *Onext* ring with canonical representatives on both sides of the manifold.

4.1. Simplifications for orientable manifolds

In many applications, including the Voronoi/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. Therefore 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

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

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 $|EP| \times$ (8 record pointers + 12 bits). The simplification for orientable manifolds reduces those 12 bits to 8. This compares favorably with the winged-edge representation [Ba] and with the Muller-Preparata variant [MP]. 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 an 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 $O(|EP|)$ and a boolean mark bit on each directed edge [Kn]. With a few more bits per edge, we can do away with the stack entirely [Ev]. 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 [Br, MS].

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 fig. 5.1).

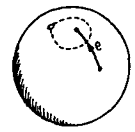


Figure 5.1. The result of `MakeEdge`.

Apart from orientation and direction, *e* will be the only edge of the subdivision, and will not be a loop; we have $eOrg \neq eDest$, $eLeft = eRight$, $eLnext = eRnext = eSym$, and $eOnext = eOprev = e$. To construct a loop, we may use $e \leftarrow$ `MakeEdge[].Rot`; then we will have $eOrg = eDest$, $eLeft \neq eRight$, $eLnext = eRnext = e$, and $eOnext = eOprev = eSym$.

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 *aOrg* and *bOrg*, and, independently, the two edge rings *aLeft* and *bLeft*. In each case,

- if the two rings are distinct, `Splice` will combine them into one;
- if the two are exactly the same ring, `Splice` will break it in two separate pieces;
- 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 *aOrg* and *bOrg*, the cuts will occur immediately *after* *a* and *b* (in counterclockwise order); for the rings *aLeft* and *bLeft*, the cut will occur immediately *before* *aRot* and *bRot*. Figure 5.2a 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.

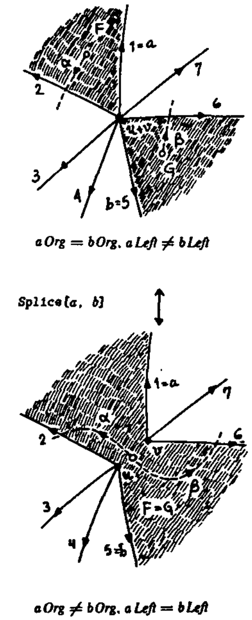


Figure 5.2a. The effect of `Splice`: trading a vertex for a face.

Figure 5.2b 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. The quad-edge data structure contains no mechanism to distinguish between these two cases, or 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. Figure 5.2b also illustrates the case when both left faces and origins are distinct.

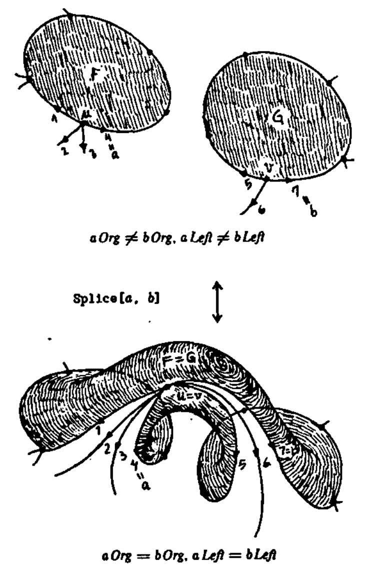


Figure 5.2b. The effect of `Splice`: Changing the connectivity of the manifold.

In the edge algebra, the *Org* and *Left* rings of an edge *e* are the orbits under *Onext* of *e* and *eOnextRot*, respectively. The effect of `Splice` can be described as the construction of a new edge algebra $A' = (E, E^*, Onext', Rot, Flip)$ 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 $\alpha = aOnextRot$ and $\beta = bOnextRot$, basically all we have to do is to interchange the values of *aOnext* with *bOnext* 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 [Kn].

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*:

$$
\begin{aligned}
a\,Onext' &= b\,Onext\\
b\,Onext' &= a\,Onext\\
\alpha\,Onext' &= \beta\,Onext\\
\beta\,Onext' &= \alpha\,Onext\\
(b\,Onext\,Flip)\,Onext' &= a\,Flip\\
(a\,Onext\,Flip)\,Onext' &= b\,Flip\\
(\beta\,Onext\,Flip)\,Onext' &= \alpha\,Flip\\
(\alpha\,Onext\,Flip)\,Onext' &= \beta\,Flip
\end{aligned}
\qquad (2)
$$

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 $b\,Onext \in E$, 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[†]. Another problematic situation is when $b = a\,Onext\,Flip$: according to equations (2) 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, i.e. $Onext' = Onext$. 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 `Splice[a, b]` will not destroy E5.

Now let's show E2 holds in A', i.e. $e\,Rot\,Onext'\,Rot\,Onext' = e$. Let X be the set of edges whose *Onext* has been changed, i.e.

$$X = \{a, b, \alpha, \beta, a\,Onext\,Flip, b\,Onext\,Flip, \alpha\,Onext\,Flip, \beta\,Onext\,Flip\}.$$

First, if $e\,Rot \not\in X$, then $e\,Rot\,Onext\,Rot \not\in X\,Onext\,Rot = X$, 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[α, β]`, 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, or $e\,Flip(Onext')^n \neq e$ 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 $a\,Org = (a_1\ a_2 \dots a_{m-1}\ a_m(= a))$ be the orbit of a under the original *Onext*. The orbit of $a\,Flip$ under *Onext* is then $a\,Flip\,Org = (a'_m\ a'_{m-1} \dots a'_2\ a'_1)$, where $a'_i = a_i\,Flip$ for all i. These two orbits are disjoint: they cannot contain any of the edges α, β, $\alpha\,Onext\,Flip$, or $\beta\,Onext\,Flip$, because they are 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 5.3):

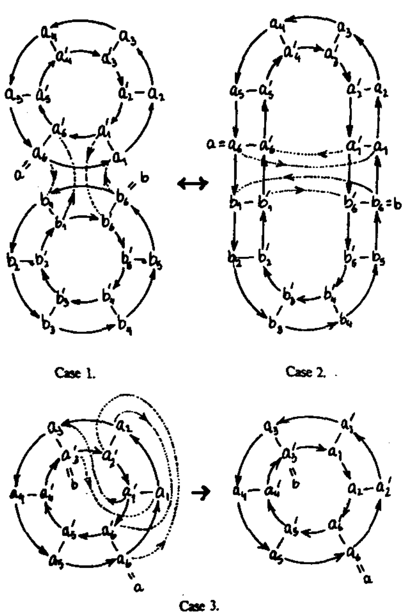


Figure 5.3. The effect of `Splice` on the *Onext* orbits.

Case 1: the edge b is neither in $a\,Org$ nor in $a\,Flip\,Org$. Then let $b\,Org = (b_1\ b_2 \dots b_{n-1}\ b_n(= b))$ and $b\,Flip\,Org = (b'_n\ b'_{n-1} \dots b'_2\ b'_1)$. According to (2), 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 $a\,Org' = (a_1\ a_2 \dots a_{m-1}\ a_m(= a)\ b_1\ b_2 \dots b_{n-1}\ b_n(= b))$ and $a\,Flip\,Org' = (b'_n\ b'_{n-1} \dots b'_2\ b'_1\ a'_m\ a'_{m-1} \dots a'_2\ a'_1)$.

Case 2: the edge b occurs in $a\,Org$. Then $b = a_i$ for some i, $1 \leq i \leq m$. 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 $b\,Org' = (a_1\ a_2 \dots a_i)$, $a\,Org' = (a_{i+1}\ a_{i+2} \dots a_m)$, $a\,Flip\,Org' = (a'_m\ a'_{m-1} \dots a'_{i+1})$, and $b\,Flip\,Org' = (a'_i\ a'_{i-1} \dots a_1)$.

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, $2 \leq i \leq m$. 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 $a\,Org' = (a'_{i-1}\ a'_{i-2} \dots a'_1\ a_i\ a_{i+1} \dots a_{m-1}\ a_m)$ and $a\,Flip\,Org' = (a'_m\ a'_{m-1} \dots a'_{i+1}\ a'_i\ a_1\ a_2 \dots a_{i-1}\ a_i)$.

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 dicussion for case 3 helps in the understanding of figure 5.2c. 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

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

and

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

allow us to rewrite (2) as

$$
\begin{aligned}
a\,Onext &\leftarrow b\,Onext & (\alpha\,Flip\,Rot)\,Onext &\leftarrow \beta\,Flip\\
b\,Onext &\leftarrow a\,Onext & (b\,Flip\,Rot)\,Onext &\leftarrow \alpha\,Flip\\
\alpha\,Onext &\leftarrow \beta\,Onext & (a\,Flip\,Rot)\,Onext &\leftarrow b\,Flip\\
\beta\,Onext &\leftarrow \alpha\,Onext & (\beta\,Flip\,Rot)\,Onext &\leftarrow a\,Flip
\end{aligned}
\qquad (3)
$$

Only one of the two assignments in each line of (3) 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 the equations (1). 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 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 (3), and the test for $b = a\,Onext\,Flip$ is meaningless.

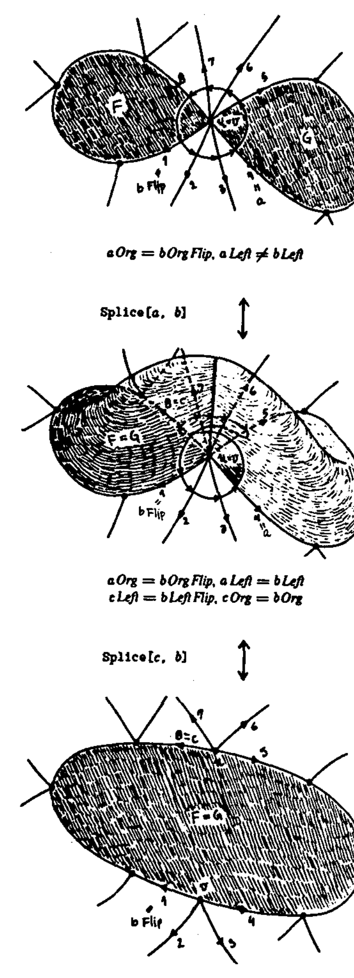


Figure 5.2c. The effect of `Splice`: Adding or removing a cross-cap.

† 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.

As an example of the use of `Splice`, consider the operation `SwapDiagonal[e]` that we will be using later on: 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 fig 5.4). This is equivalent to

```
a ← e.Oprev;
b ← e.Sym.Oprev;
Splice[e, a]; Splice[e.Sym, b];
Splice[e, a.Lnext]; Splice[e.Sym, b.Lnext]
```

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

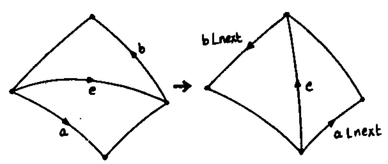


Figure 5.4. The effect of `SwapEdge[e]`.

Theorem 5.1. *An arbitrary subdivision P can be transformed into a collection of $|EP|$ isolated edges by the application of at most $2|EP|$ `Splice` operations.*

Proof: Let e be an arbitrary edge of P. Then the operations `Splice[`e, $eOprev$`]` and `Splice[`$eSym$, $eSymOprev$`]` will have the effect of removing e from P and placing 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 P can be constructed, in $O(|EP|)$ 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 `Splice`s), or initialize the `Data` links right from the beginning with the values they must have in the final structure.

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 $eOrg$, 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.

The topological manipulations performed by the algorithms on the Delaunay/Voronoi diagrams can be reduced to a pair of higher-level topological operators, defined here in terms of `Splice` and `MakeEdge`. The operation e ← `Connect[`a, b, *side*`]` will add a new edge e connecting the destination of a to the origin of b, across either a*Left* or a*Right* depending on the *side* flag. For added convenience, it will also set the `Org` and `Dest` fields of the new edge to `a.Dest` and `b.Org`, respectively.

```
e ← MakeEdge[];
e.Org ← a.Dest;
e.Dest ← b.Org;
Splice[e, IF Side=left THEN a.Lnext ELSE a.Sym];
Splice[e.Sym, IF Side=left THEN b ELSE b.Oprev]
```

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

```
Splice[e, e.Oprev];
Splice[e.Sym, e.Sym.Oprev]
```

7. Voronoi and Delaunay - some basic facts

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 [Sh], [Le], or [SH].

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 inter-site bisectors, and whose vertices are circumcenters of triangles defined by three of the sites. An example Voronoi diagram for a small collection of sites is shown in fig. 7.1.

As mentioned above, most of the time we will actually deal with a dual of the Voronoi subdivision, commonly called the Delaunay triangulation. As the name implies, this is a planar subdivision whose vertices are the n sites and whose edges are straight line segments that connect every pair of sites associated with regions that share a common edge. If no four of the sites happen to be cocircular, then this dual is in fact a triangulation and, in any case, additional edges can be introduced to make it one. Although our algorithms can handle the degeneracies implied by four cocircular points, in this section only we will assume that such degeneracies do not arise, so we can speak unambiguously of the Delaunay triangulation of the n sites. If S denotes our collection of sites, then $D(S)$ will denote the Delaunay triangulation of S.

We say that a circle is *point-free* if none of the given sites is contained in its interior. It is known that the circumcircle of each Delaunay triangle is point-free. Actually the converse is also true. If the circumcircle of a triangle defined by three of the sites is point-free, then that triangle is a Delaunay triangle. And if it is possible to find a point-free circle passing through two sites, then these sites will be connected by a Delaunay edge. For a discussion of these facts see [Le]. We will speak of these circles as *witnesses* to the "Delaunayhood" of the corresponding triangle or edge respectively. For completeness we also state here a couple of other obvious properties of the Delaunay triangulation which will be useful in the sequel.

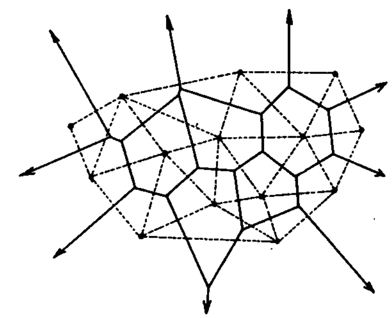


Figure 7.1. The Voronoi diagram (solid) and the Delaunay diagram (dashed).

Lemma 7.1. *Each triangulation of n points of which k lie on the convex hull has $2(n-1)-k$ triangles and $3(n-1)-k$ edges.*

Proof: Let t be the number of triangles and e the number of edges. By counting edges we have $3t+k=2e$, and from Euler's relation $n-e+(t+1)=2$. We can solve this system of two equations and obtain the lemma. □

Lemma 7.2. *Each convex hull edge is a Delaunay edge.*

Proof: The half-plane supporting the edge and not containing the convex hull is a degenerate circle witness to the Delaunayhood of the edge. □

Lemma 7.3. *If S and T are two point sets then each Delaunay edge $e \in D(S \bigcup T)$, both of whose endpoints are in S, is also a Delaunay edge in $D(S)$.*

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

Proof: There is a point-free circle through e in $S \bigcup T$, and therefore *a fortiori* in S. □

8. The InCircle test - implications for Delaunay

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 fig. 8.1.

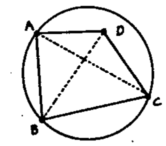


Figure 8.1. The *InCircle* test.

Definition 8.1. The predicate $InCircle(A, B, C, D)$ is defined to be true if and only if point D lies interior to the region of the plane bounded by the circle passing through A, B, and C and lying to the left of that circle when the latter is traversed in the direction from A to B to C.

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 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*

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

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 $z = x^2 + y^2$. See fig. 8.2 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.

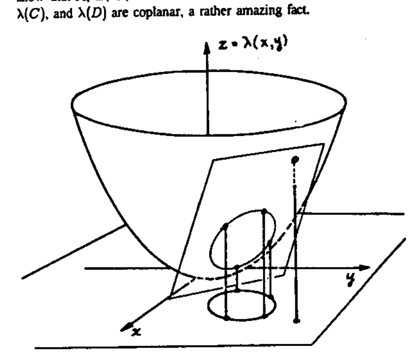


Figure 8.2. The quadratic map for computing *InCircle*.

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

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

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 center and r the radius of the circumcircle of triangle ABC. Then A, B, and C satisfy equation (1) 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 (1) as well, i.e., 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 upwards, 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 correspondance 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 *InCircle* test possesses several interesting properties:

Lemma 8.2. *If A, B, C, D are any four non-cocircular 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 $InCircle(A, B, C, D)$, $InCircle(B, C, D, A)$, $InCircle(C, D, A, B)$, $InCircle(D, A, B, C)$ is either T(rue), F(alse), T, F, or F, T, F, T.*

Proof: This is obvious from the properties of determinants. □

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. If A, B, C, D define a counterclockwise oriented quadrilateral and $InCircle(A, B, C, D)$ is true, then $InCircle(C, B, A, D)$ is false, so reversing the orientation flips the value of the predicate. Whenever we consider applying this test to a (rooted) quadrilateral in the sequel, we will take the quadrilateral to be counterclockwise oriented.

For a convex quadrilateral $ABCD$ there are two ways to triangulate it. We can either draw diagonal AC or diagonal BD. By lemma 7.2 the sides AB, BC, CD, and DA are always Delaunay edges. If $InCircle(A, B, C, D)$ is false, then circle ABC is point-free, so the diagonal AC completes the Delaunay triangulation for the four points A, B, C, D (else the diagonal BD does). More generally, for any collection of points, all the convex hull edges are Delaunay edges. For each non-hull Delaunay edge BD, if the two triangles ABD and BDC on its two sides form a convex quadrilateral, then $InCircle(A, B, C, D)$ is true. A converse of this also holds [Le]. This is commonly expressed in terms of the "swapping rule", an implementation of which was presented in section 5. This rule states that if there exists a convex quadrilateral $ABCD$ that has been triangulated with diagonal AC, but $InCircle(A, B, C, D)$ is true, then this diagonal should be erased and diagonal BD should be drawn. Success of the swapping rule on an edge (AC in the previous sentence) is a witness for non-Delaunayhood. The above converse then states that a triangulation stationary under the swapping rule is the Delaunay triangulation. In fact, the swapping rule provides a new necessary and sufficient condition for the Delaunayhood of an edge, namely that the rule should fail for any quadrilateral defined by the endpoints of the edge and any two other points, one on each side.

A useful intuition about the *InCircle* test comes from the following observation. Given two points in the plane X and Y, the set of circles passing through X and Y forms a one parameter family. For example, we can think of these circles as the family $\{C_t\}$, where t denotes the signed distance of the center of such a circle along the bisector of XY, measured from the midpoint of XY. Thus $C_{-\infty}$ denotes the circle corresponding to the half-plane to the left of the line XY, C_0 denoted the circle with diameter XY, and C_{∞} denotes the circle corresponding to right half-plane 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 fig. 8.3. The next lemma uses this intuition to derive another property of the Delaunay triangulation that will be useful to us in the next section.

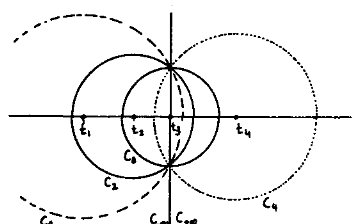


Figure 8.3. The set of circles passing through two given points.

Lemma 8.3. *Let A be a site and consider any line ℓ through A. For convenience of terminology, assume that ℓ is horizontal. Let $N_1, N_2, \ldots, N_k$ denote the Delaunay neighbors of A lying above ℓ, listed in counterclockwise order. If X is any point of ℓ to the right of A, let Γ_i denote the portion of the circumcircle of 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 $\{X_i\}$ moves monotonically to the left.

Consider, as in fig. 8.4, three consecutive Delaunay neighbors N_i, N_{i+1}, and N_{i+2} of A. The point N_{i+2} is outside 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 our assertion.

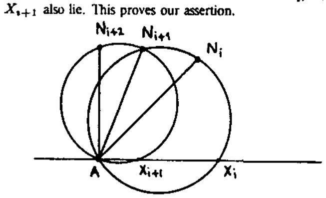


Figure 8.4. A property of the neighbors of A.

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. □

We will refer to lemma 8.3 as the *unimodality lemma*. Another useful property of the Delaunay triangulation that can be derived with the help of the *InCircle* test is:

Lemma 8.4. *Let A be a site and let X and Y denote any two distinct Delaunay neighbors of A. Then all other Delaunay neighbors of A that lie inside the convex angle $\angle XAY$ are inside the circumcircle of triangle AXY, and all neighbors lying outside the angle are outside that same circle.*

Proof: Without loss of generality let us assume that in the convex angle $\angle XAY$ the ray AY is counterclockwise of AX. Suppose the lemma is false inside the convex angle $\angle XAY$. Let N be the Delaunay neighbor of A outside the circle AXY which is the first to be encounterd as we sweep this angle in counterclockwise fashion. Let also M denote the predecessor of N, as in fig. 8.5. Note that N is also outside the circle AMY since X, M, and N are all on the same side of AY. Therefore Y is inside the circle AMN, contradicting Delaunayhood. A similar argument proves the lemma outside $\angle XAY$. □

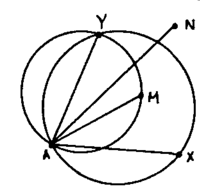


Figure 8.5. A property of the Delaunay diagram.

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, e.g., the coordinates of the corresponding Voronoi vertices, we are simultaneously computing the Voronoi diagram as well. Our algorithm is the dual of that proposed by Shamos and Hoey [SH] and, like theirs, runs in time $O(n \lg n)$ and uses linear storage. In [Le] D. T. Lee proposes a dual algorithm which is similar to but more complex than ours. Unlike both of these, our algorithm need not compute straight line intersections unless the coordinates of Voronoi vertices are needed. The only geometric primitives used by our algorithm 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. The latter is a standard tool in geometric algorithms, and is equivalent to reporting on which side of a line a point lies. It is equivalent to $InCircle(A, B, C, D)$, for D chosen as the barycenter of triangle ABC.

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 triangulation of each half. Finally we need to marry the two half triangulations into the Delaunay triangulation of the whole set.

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 *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 fig. 9.1.

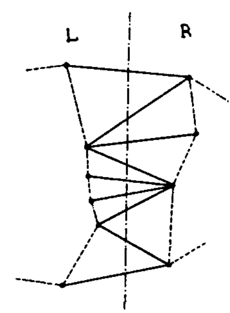


Figure 9.1. The structure of the L-R edges.

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 end-point of *basel* to one of the R-neighbors of the right end-point lying "above" *basel*, or symmetrically, it will be an edge from the right end-point of *basel* to one of the L-neighbors of the left end-point lying "above" *basel*. In the program below edges from the left end-point of *basel* to its candidate L-neighbors will be held in the variable *lcand*, and their symmetric counterparts in *rcand*. See fig. 9.2.

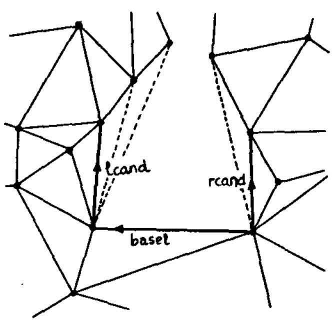


Figure 9.2. The variables lcand, rcand, and basel.

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 fig. 9.3. 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 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 edge *ldi* is the clockwise convex hull edge out of the rightmost L point, and *rdi* is the counterclockwise convex hull edge out of the leftmost R point. We assume these are returned by the recursive calls of the triangulation procedure to L and R. In the program below prose in {} is comments; certain portions of code that are just symmetric variants of others shown are elided and are indicated by " ".

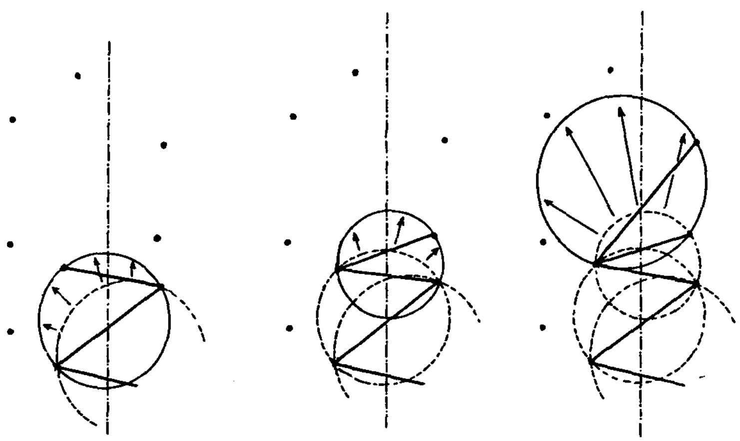


Figure 9.3. The rising bubble.

```
{ create a first cross edge (basel) by computing the lower common tangent of L and R }
DO WHILE CCW[ldi.Org, ldi.Dest, rdi.Org] DO ldi ← ldi.Lnext OD;
  IF CCW[rdi.Dest, rdi.Org, ldi.Org] THEN rdi ← rdi.Rprev ELSE EXIT;
  OD;

basel ← Connect[rdi.Sym, ldi, left];

{ initialize lcand to the counterclockwise first edge incident to basel.Dest and above basel }
{ symmetrically for rcand }
lcand ← basel.Sym.Onext; rcand ← basel.Oprev;

{ let ValidL and ValidR be procedures which test whether lcand and rcand are above basel }
ValidL[l] : CCW[basel.Org, basel.Dest, l.Dest];
ValidR[r] : CCW[basel.Org, basel.Dest, r.Dest];

{ this is the merge iteration }
DO
  { advance lcand counterclockwise, deleting the old lcand edge, until the InCircle test fails }
  IF ValidL[lcand] AND ValidL[lcand.Onext] THEN
  WHILE InCircle[lcand.Dest, lcand.Onext.Dest, lcand.Org, basel.Org] DO
     t ← lcand.Onext;
     DeleteEdge[lcand];
     lcand ← t;
     OD;

  { symmetrically, advance rcand clockwise }
  . . .

  { if both lcand and rcand are invalid then basel is the upper common tangent }
  IF NOT ValidL[lcand] AND NOT ValidR[rcand] THEN EXIT;

  { connect to either lcand or rcand }
  { if both are valid, then choose the appropriate one using the InCircle test }
  { if lcand is the winner then make the next cross edge close the triangle with lcand and basel. }
  { else make the next cross edge close the triangle with rcand and basel }
  { advance basel and reinitialize lcand or rcand }
  IF NOT ValidL[lcand] OR
     (ValidR[rcand] AND InCircle[lcand.Dest, lcand.Org, rcand.Org, rcand.Dest])
     THEN BEGIN { connect to the right }
     basel ← Connect[rcand, basel.Sym, left];
     rcand ← basel.Sym.Lnext;
     END
  ELSE . . .  { connect to the left }
  OD
```

We now elaborate on the above program. 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 $L-L$ 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 above algorithm.

Lemma 9.2. *When the lcand iteration stops, the circumcircle of the triangle defined by lcand and basel is free of other L points. Furthermore all deleted edges are not in the final Delaunay triangulation.*

Proof: Let A and X denote respectively the left and right endpoints of *basel*, and, as in lemma 8.3, let N_i, $1 \leq i \leq k$, denote the Delaunay neighbors of A in L that are above *basel*. We prove our lemma inductively on the sequence of cross edges discovered.

Assuming that the algorithm has worked correctly up to now, then the circumcircle of the triangle defined by *basel* and the previous cross edge is the circumcircle of a Delaunay triangle and therefore it is point-free. As in our earlier discussion, suppose that we "push" this circle upwards while always making it pass through A and X. At some point it will encounter a new L point N for the first time. This point will be above *basel*. We wish to show that N is the point that the *lcand* iteration will discover. To see this note first of all that N must be a Delaunay neighbor of A, since our circle is a Delaunayhood witness. The *lcand* iteration proceeds until the first time X falls outside the circle AN_iN_{i+1}, as shown in the code. By the unimodality lemma, this gives us the neighbor $N = N_i$ which determines the circle AXN_i with the smallest extent above *basel*. This circle now forms the basis for the next iteration through the main loop.

The inclusion of X in the circle AN_mN_{m+1} for $m = 1, 2, \ldots, m-1$ establishes that AN_m is not a Delaunay edge in the final triangulation.

since each circle with AN_m as chord contains either N_{m+1} or X in its interior. Thus the edge deletions performed are valid. It is easy to check that the cross edges drawn, together with the remaining edges of L and R define a triangulation of the n sites.

This is essentially the proof, modulo expending some care on certain boundary cases. We must first of all show that the *lcand* iteration, if started at all, never gets to neighbors on the bottom side of *basel*. For if the iteration was invoked, then there must be at least two neighbors above *basel*. We claim that the moment *lcand Onext* crosses to the bottom of *basel*, the `WHILE` *InCircle* test fails. See fig. 9.4.

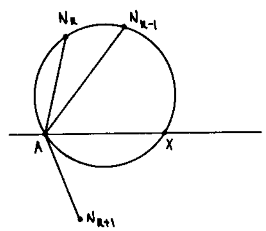


Figure 9.4. End of `lcand` loop.

In this case we had that X was inside the circle $AN_{k-1}N_k$. If the angle $\angle N_kAN_{k+1}$ is convex then X is obviously outside the circle AN_kN_{k+1}, so the *InCircle* test fails. If the angle $\angle N_kAN_{k+1}$ is concave, then two things happen. First of all the circle AN_kN_{k+1} reverses orientation, and secondly, since N_{k+1} is outside the circle $AN_{k-1}N_k$ but on the same side of AN_k as N_{k-1} and X, X now falls inside the circle AN_kN_{k+1}. So in this case the *InCircle* test fails also.

Finally note that the main loop starts with our ascending circle being the halfplane below the lower outer common tangent of L and R, and ends with that circle being the halfplane above the upper such tangent. □

A symmetric statement holds for the *rcand* iteration. Furthermore, the final choice between *lcand* and *rcand* is again done by the *InCircle* test, namely *rcand* will be chosen if `InCircle[lcand.Dest, lcand.Org, rcand.Org, rcand.Dest]` is true. It now follows that:

Lemma 9.3. *The triangle defined by basel and the next cross edge as computed by the above algorithm has a circumcircle free of both L and R points, and thus is a Delaunay triangle. Furthermore, after the completion of the cross edge iteration, all remaining L or R triangles are also Delaunay triangles for the full configuration.*

Proof: The first statement is obvious from the above discussion. The second statement is most easily proven by showing that each remaining $L - -L$ or $R - -R$ edge is stationary under the swapping rule. The only possible suspect edges are those bordering triangles containg cross edges. It follows that these edges were *lcand* or *rcand* at some point in the algorithm and were not deleted. So the *InCircle* test of the inner iterations failed there, and this establishes that these edges are safe under the swapping rule. □

These lemmas complete the proof that the above 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 "downwards" looking faces of this polyhedron are in a one-to-one correspondance with the Delaunay faces of the sites. The upwards facing 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 [Sh] 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 [PH] algorithm for computing the convex hull of n points in three dimensions. If the *InCircle* test is replaced by a "negative volume" test, as obtained by substituting in $D(A, B, C, D)$ 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 [PH].

10. Incremental techniques

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. The machinery we have developed so far yields also simple and elegant insertion algorithms. Deletions present problems.

It will simplify our discussion to assume that our points are always within some large convex polygon (say a triangle) whose vertices are considered to be among the given points. Thus the entire interior of this polygon will have been triangulated, and when a new point is given, we can be sure that it will fall within one of the existing triangular faces. The algorithm for updating the Delaunay triangulation starts out by locating the new point in one of the triangular faces of the subdivision. This can be done by Kirkpatrick's method in time $O(\lg n)$ [Kil]. Our concern is with how the triangulation is to be updated.

Lemma 10.1. *The edges connecting the new point to vertices of its enclosing triangle are Delaunay edges.*

Proof: From fig. 10.1 it is clear the we can always find a circle contained inside the circumcircle of a triangle ABC, and passing through one of A, B, or C, and a specified interior point X of that circle. Such a circle is point-free. □

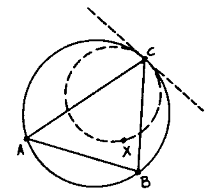


Figure 10.1. A circle proving the "Delaunayhood" of XC.

However, the edges of the enclosing triangle are themselves suspect. We can remove suspect edges by the swapping rule discussed in section 8. This gives rise to a very simple algorithm, given in high-level form below.

```
place the three edges of the enclosing triangle
  on a stack in ccw order;

WHILE the stack is nonempty DO
  BEGIN
    { each edge on the stack forms a triangle with the new point }
   remove an edge from the stack;
   consider the quadrilateral whose diagonal
     is this edge;
   if this edge passes the swapping rule test,
     then forget about it,
     else apply the swapping rule and stack,
       in ccw order, the two edges not
       incident to the new point;
  OD;
```

At any one time the set of triangles with the new point X as a vertex form a star-shaped polygon around X. This is because the polygon can only get larger with each application of the swap rule on a bounding edge, and if it does grow, then the two new stacked sides are in the angular sector with vertex X and delimited by the deleted edge. Furthermore, because we stack edges in a consistent order, the stacked edges always form a contiguous section of the perimeter of that polygon, with the rest of the perimeter consisiting of the forgotten edges.

Lemma 10.2. *All edges incident to the new point introduced during applications of the swap rule in the above algorithm are Delaunay edges. So is each forgotten edge.*

Proof: Let X be the new point and suppose that the swap rule was just applied to quadrilateral $XLMN$ where it replaced diagonal LN with diagonal XM, as in fig. 10.2. Then X must be interior to the circle LMN and by the same argument as in the proof of lemma 10.1 it follows that XM is a Delaunay edge. If the swap rule failed on $XLMN$, then X is outside the circle LMN, and so edge LN is Delaunay and can be forgotten. □

Lemma 10.3. *No edge is ever stacked twice.*

Proof: One of two things can happen at each application of the swapping rule. If the swap succeeds, then the polygon grows past the old bounding edge, so that edge can never be on its boundary again. If it fails, then that bounding edge is frozen and the algorithm will never again touch any edge in the angular sector with vertex X and delimited by that edge. □

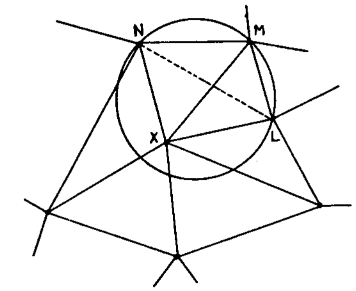


Figure 10.2. Proving correctness of the incremental method.

Lemma 10.4. *When the above loop terminates the swap test fails everywhere in the triangulation.*

Proof: From lemma 10.2 we know that all edges leading to the new point X and introduced during the above algorithm are Delaunay edges. The only other edges bounding new triangles are the edges of the star-shaped polygon discussed in the proof of lemma 10.3. At the termination of the loop the stack is empty and so all the bounding edges have been forgotten at some point. Thus there are no suspect edges left. □

The above lemmas imply that this loop correctly computes the final Delaunay triangulation, and that it does so in at most $O(n)$ time. More precisely, if k edges need to be added or deleted to update the Delaunay structure, then our algorithm works in time $O(k)$, since each edge added by the algorithm must be added, and each edge that is deleted must be deleted.

We saw that the forgotten edges will always define a path around the new point, which will eventually close. If e is an edge on the path, then the next edge is *e Lnext Oprev*. This observation gives us a way to implement the above algorithm without any auxiliary storage, such as the stack, by a method that is similar to that used in the previous section. Here *basel* starts out (arbitrarily) as one of the edges connecting the new point to one of the vertices of its enclosing triangle. By convention we think of the new point as being the left end point of *basel*. Then *lcand* will always be invalid (non-existent, in fact) and we will consider for *rcand* the various edges out of the other endpoint of *basel*. The successively found *rcands*, 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 [Ki2] 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 $O(k \lg k)$, 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.

11. Conclusions.

In this paper we have presented a new data structure for planar subdivisions which 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.

12. References.

[Ba] Baumgart, B. G. *A Polyhedron Representation for Computer Vision.* AFIPS Conference Proceedings, Vol. 44, 1975 National Computer Conference, pp. 589-596..

[Br] Braid, I. C., Hillyard, R. C., and Stroud, I. A. *Stepwise construction of Polyhedra in Geometric Modeling*; in Brodlie, K. W. (ed.), *Mathematical Methods in Computer Graphics and Design.* Academic Press, London, 1980, pp. 123–141..

[Ev] Even, S. *Algorithmic Combinatorics.* Macmillan, N. Y., 1973..

[GS] Green, P. J., and Sibson, R. *Computing Dirichlet Tesselation in the Plane.* Computer Journal, Vol. 21, no. 2, 1977, pp. 168–173..

[Har] Harary, F. *Graph Theory.* Addison-Wesley, 1972, p. 105..

[IK] Iyanaga, S., and Kawada, Y. *Encyclopedic Dictionary of Mathematics* (2nd. ed.). MIT Press, Cambridge, Mass., 1968..

[Ki1] Kirkpatrick, D. *Optimal Search in Planar Subdivisions.* Technical Report 81-13, Department of Computer Science, University of British Columbia, 1981..

[Ki2] Kirkpatrick, D. *Efficient Computaion of Continuous Skeletons.* Proceedings of the 20th Annual Symposium on Foundations of Computer Science, 1979, pp. 18–27..

[Kn] Knuth, D. E. *The Art of Computer Programming*, Vol. I: Fundamental Algorithms (2nd. ed.). Addison-Wesley, 1975..

[Le] Lee, D. T. *Proximity and Reachability in the Plane.* Report R-831, Coordinated Science Laboratory, University of Illinois, Urbana, 1978..

[MS] Mantyla, M. J., and Sulonen, R. *GWB: a Solid Modeler with Euler Operators.* IEEE Computer Graphics and Applications, Vol. 2 N. 5 (September 1982) pp. 17–31..

[MP] Muller, D. E., and Preparata, F. P. *Finding the Intersection of Two Convex Polyhedra.* Theoretical Computer Science Vol. 7, 1978, pp. 217–236..

[PH] Preparata, F. P., and Hong, S. J. *Convex Hulls of Finite Sets of Points in Two and Three Dimensions.* CACM, Vol. 20, 1977, pp. 87–93.

[Sh] Shamos, M. I. *Computational Geometry.* Ph. D. dissertation, Yale University, 1977..

[SH] Shamos, M. I., and and Hoey, D. *Closest-Point Problems.* 16-th Annual Symposium on Foundations of Computer Science, 1975, pp. 151–162..