Voronoi diagram for multiply-connected polygonal domains I: Algorithm

by Vijay Srinivasan
Lee R. Nackman

Voronoi diagrams of multiply-connected polygonal domains (polygons with holes) can be of use in computer-aided design. We describe a simple algorithm that computes such Voronoi diagrams in $O(N(\log_2 N + H))$ time, where N is the number of edges and H is the number of holes.

1. Introduction

Voronoi diagrams have been an active research topic in computational geometry for the past decade. Much of the earlier work, and some of the current work, concerns the Voronoi diagrams of a set of discrete points. This domain has been extended by Kirkpatrick [1], Lee and Drysdale [2], Lee [3], Yap [4], and others to cover a collection of two-dimensional objects such as line segments, circular arcs, and polygons.

Several applications based on two-dimensional geometric modeling require the computation of Voronoi diagrams of boundaries of multiply-connected domains. Geometrical modelers usually provide a boundary description of two-dimensional, multiply-connected polygonal domains ("polygons with holes") as a tree of loops, in which each loop is a list of edges. If N is the total number of edges on the boundary of a multiply-connected polygonal domain, then algorithms of $O(N \log_2 N)$ time complexity are known [1, 2, 4] to compute the Voronoi diagram, but they are too complex for practical implementation. In this paper, we propose a simpler algorithm to compute the Voronoi diagram of a multiply-connected polygonal domain. (Here and henceforth, we shall drop the use of the term "boundary of" and refer to the Voronoi diagram of the boundary of a domain as the Voronoi diagram of the domain itself.) If H is the number of holes, the algorithm requires $O(N(\log_2 N + H))$ time. An implementation and application of the algorithm to a VLSI problem is presented in a companion paper [5].

The algorithm described in this paper computes the Voronoi diagram of multiply-connected polygonal domains. Informally, a multiply-connected polygonal domain is a planar domain with holes in which all boundaries are polygons (see **Figure 1**). A formal definition follows.

Definition 1

A *closed line segment* $[a, b]$ is the union of two *endpoints* a and b and the *open line segment* (a, b). Points or open line segments are called *elements.* □

Definition 2

A *multiply-connected polygonal domain* Ω is the closure of a nonempty, bounded, connected, open (in the relative

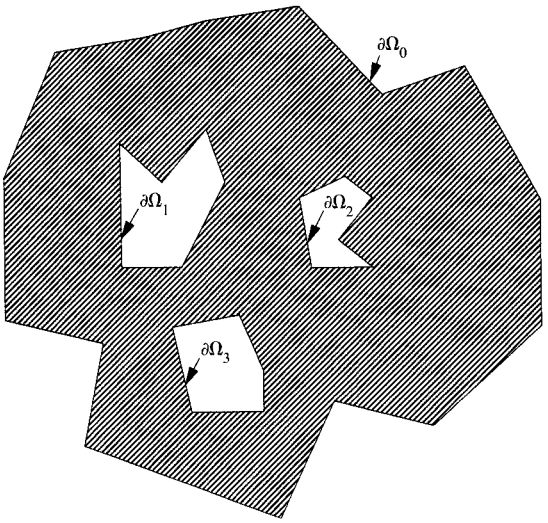

Figure 1

Boundary notation for a polygonal domain. Shading indicates presence of material.

topology) subset of $\mathbb{R}^2$ whose boundary is the union of a finite number of closed line segments. □

The boundary of Ω, which is denoted by $\partial\Omega$, consists of one or more disjoint subsets. Each of these subsets partitions $\mathbb{R}^2$ into two regions, one bounded (the interior region) and the other unbounded (the exterior region). The interior region of one, and only one, of these subsets contains Ω, and is called the *outer boundary* and is denoted by $\partial\Omega_0$. The remaining subsets are called the *inner boundaries* of Ω and are denoted by $\partial\Omega_i$, $1 \le i \le H$, where H denotes the number of inner boundaries (holes). Our notation is illustrated in Figure 1.

Definition 3

The *vertices* of $\partial\Omega$ are the points of intersection of the closed line segments which constitute $\partial\Omega$. The *edges* of $\partial\Omega$ are the open line segments obtained by deleting the endpoints of the closed line segments which constitute $\partial\Omega$. □

We denote the number of elements in $\partial\Omega_i$ by n_i. The jth element of $\partial\Omega_i$ is denoted by e_i^j, $1 \le j \le n_i$, where the elements are numbered so that while traversing the boundary in increasing order, the domain lies to the left of the boundary.

2. Voronoi diagram definition

The Voronoi diagram of a set of elements is defined in terms of various distances and half-planes. These are the subject of this section. (Definitions 4 through 8 are similar, but not identical, to definitions in [3].)

Definition 4

The *distance* between a point p and a point q is denoted $d(p, q)$. The *distance* between a point p and a nonempty set X, denoted $d(p, X)$, is $d(p, X) = \text{glb}\{d(p, q): q \in X\}$. □

Definition 5

The *projection* $p(q, [a, b])$ of a point q onto a closed segment $[a, b]$ is the intersection of the line through a and b and the line perpendicular to $[a, b]$ and passing through q. A similar definition holds for $p(q, (a, b))$. □

The distance $d(q, [a, b])$ between a point q and a closed segment $[a, b]$ is the distance between the point q and its projection onto $[a, b]$ if $p(q, [a, b])$ belongs to $[a, b]$, and is $\min(d(q, a), d(q, b))$ otherwise. Similarly, the distance $d(q, (a, b))$ between a point q and an open line segment (a, b) is the distance between the point q and its projection onto (a, b) if $p(q, (a, b))$ belongs to (a, b), and is $\min(d(q, a), d(q, b))$ otherwise.

Definition 6

The *image* $I(q, [a, b])$ of a point q on a closed segment $[a, b]$ is the projection $p(q, [a, b])$ if the projection is in $[a, b]$; otherwise it is the endpoint of $[a, b]$ closest to the projection. An analogous definition holds for the image on an open line segment. □

Definition 7

The *bisector* $B(e_i, e_j)$ of two elements e_i and e_j is the locus of points equidistant from e_i and e_j. The bisector $B(X, Y)$ of two sets of elements X and Y is the locus of points equidistant from X and Y. The *oriented* bisector $\bar{B}(e_i, e_j)$ is the bisector $B(e_i, e_j)$ with a direction imposed upon it so that elements e_i and e_j lie respectively to the *left* and to the *right* of it. An oriented bisector $\bar{B}(X, Y)$ is defined similarly. □

Definition 8

The *half-plane* $h(e_i, e_j)$ is the set of points closer to element e_i than to element e_j. Its complement, $\bar{h}(e_i, e_j)$, is the set of points not closer to element e_i than to element e_j. □

Note that the boundary of a half-plane $h(e_i, e_j)$ is the bisector $B(e_i, e_j)$, and $\bar{h}(e_i, e_j)$ includes this bisector. **Figure 2** illustrates all possible half-planes and associated bisectors relevant to edges and vertices of polygons.

Note that when both e_i and e_j are points [Figure 2(a)], the bisector is a straight line. When one of the elements is a point and the other is an open line segment [Figure 2(b)], the bisector consists of three connected pieces: one parabolic arc and two half-lines. For the special case where the point is one of the endpoints of the open line segment [Figure 2(c)], the bisector is a straight line passing through the point and perpendicular to the open line segment. When both elements

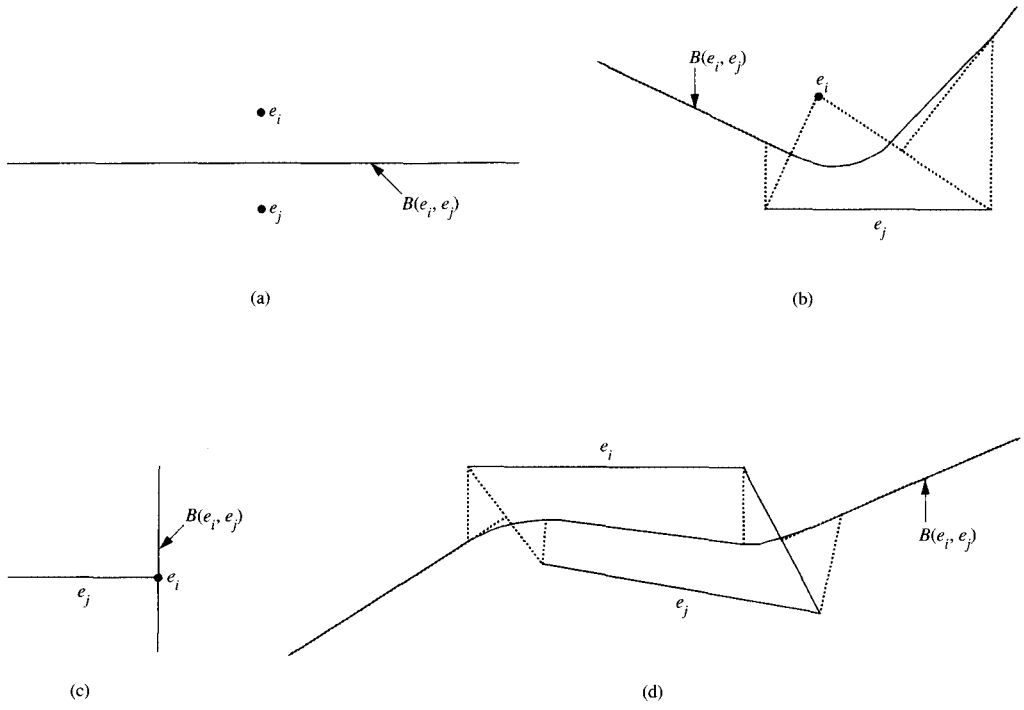

Figure 2

Half-planes and bisectors. The construction lines are shown as dotted lines, and they are not parts of the bisectors. (a) Both elements are points; the bisector is a line. (b) One element is a point and the other is an open line segment; the bisector consists of a parabolic arc and two half-lines. (c) As in (b), but the point is an endpoint of the line segment; the bisector is a line perpendicular to the line segment. (d) Both elements are open line segments; the bisector consists of two parabolic arcs, a line segment, and two half-lines.

are open line segments [Figure 2(d)], the bisector consists of five connected pieces: two parabolic arcs, a line segment, and two half-lines.

Definition 9

Let S_1 and S_2 be two disjoint sets of elements. The *Voronoi region* $V(S_1, S_2)$ of S_1 with respect to S_2 is the set of all points closer to S_1 than to S_2. □

Lemma 1

$$V(S_1, S_2) = \bigcup_{e_i \in S_1} \bigcap_{e_j \in S_2} h(e_i, e_j).$$

Proof Let p be in $V(S_1, S_2)$. By definition, p is closer to some element e_i in S_1 than to any element in S_2. Therefore p is in $h(e_i, e_j)$, for all e_j in S_2. Hence, p is in $\cup_{e_i \in S_1} \cap_{e_j \in S_2} h(e_i, e_j)$. By a similar argument, it is easy to show the converse. □

Two corollaries follow immediately.

Corollary 1

$$V(e_i, S) = \bigcap_{e_j \in S} h(e_i, e_j). \square$$

Corollary 2

$$V(S_1, S_2) = \bigcup_{e_i \in S_1} V(e_i, S_2). \square$$

Since the boundary of the union (intersection) of a finite number of sets is a subset of the union of their boundaries, the boundary of a Voronoi region consists of pieces of straight lines and parabolas (as illustrated in Figure 2). Each such piece is called a *Voronoi edge* of the Voronoi region; the endpoints of a Voronoi edge are called *Voronoi vertices.*

Definition 10

The *Voronoi diagram, VOD*(S), of a set of elements $S = \{e_i\}$ is $\cup_{e_i \in S} V(e_i, S - e_i)$. □

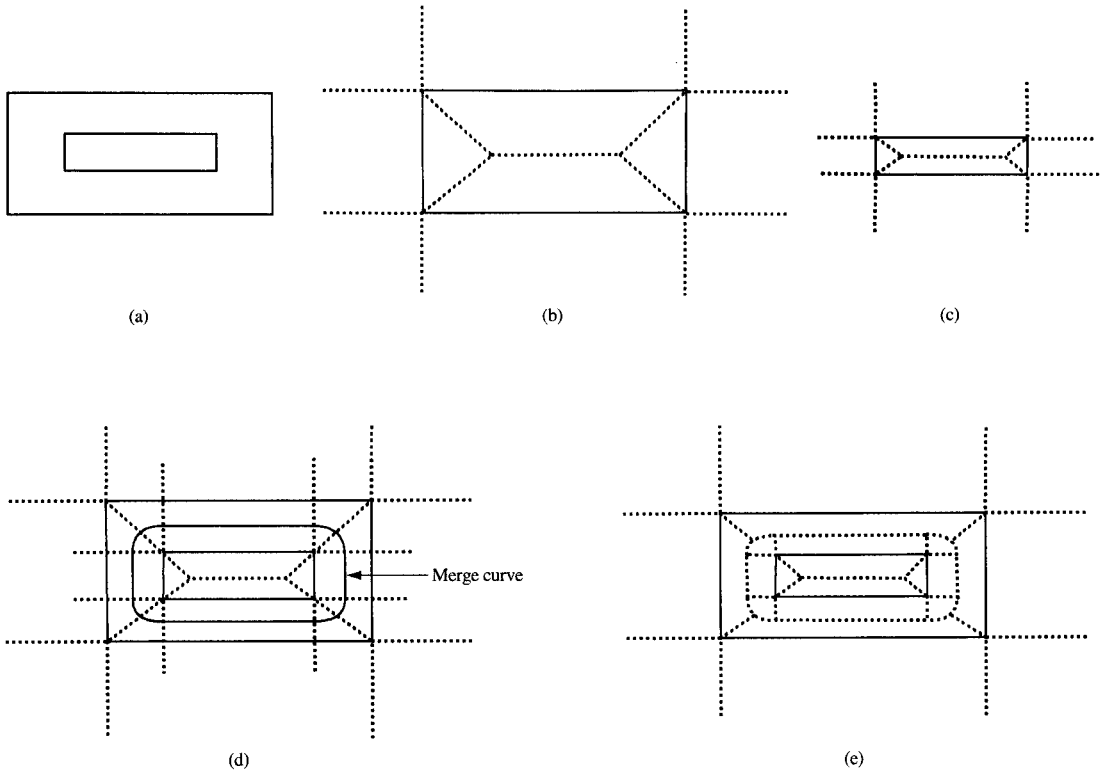

Figure 3

Simple example illustrating the algorithm: (a) Multiply-connected polygonal domain. (b) and (c) Individual Voronoi diagrams. (d) Superposition of individual Voronoi diagrams and the merge curve. (e) Voronoi diagram of the multiply-connected polygonal domain.

Often $\cup_{e_i \in S} \partial V(e_i, S - e_i)$ is referred to as the *Voronoi diagram* of S. These two definitions are complementary.

3. Algorithm overview

Our objective is to compute the Voronoi diagram of a multiply-connected polygonal domain, $VOD(\cup_{j=0}^{H} \partial\Omega_j)$. The algorithm we describe does this by computing the individual Voronoi diagrams $VOD(\partial\Omega_j)$, $j = 0, \cdots, H$, and then merging them.

For example, consider the multiply-connected polygonal domain shown in **Figure 3(a).** The individual Voronoi diagrams of the outer and inner boundaries are shown in **Figures 3(b)** and **3(c).** These can be computed using a simple extension of the algorithm described in [3] for computing the Voronoi diagram of a simply-connected polygonal domain. In **Figure 3(d),** the two individual Voronoi diagrams are shown superposed together with their *merge curve.* The merge curve is the bisector between the boundaries whose Voronoi diagrams are being merged. In this example, the merge curve is the locus of points equidistant between the inner and outer boundaries. The merged Voronoi diagram, shown in **Figure 3(e),** consists of that portion of the Voronoi diagram of the outer boundary that lies outside the merge curve together with that portion of the Voronoi diagram of the inner boundary that lies inside the merge curve.

More generally, the algorithm consists of the following steps:

1. Compute the Voronoi diagram of the outer boundary.
2. Sort the inner boundaries.
3. For each of the inner boundaries, do the following:
   a. Compute the Voronoi diagram of the inner boundary.
   b. Compute the merge curve between the Voronoi diagram of the inner boundary and the merged Voronoi diagram computed thus far.

c. Discard the extraneous portions of the original Voronoi diagrams, thus obtaining the new merged Voronoi diagram.

The heart of the algorithm, and the fundamental difference between this algorithm and those described in [2, 3], is the way in which the merge curve is computed. All of these algorithms compute the merge curve by first finding a starting point on the curve and then traversing the merge curve starting from that point. We propose a simple method for finding a starting point by exploiting properties of Voronoi diagrams of multiply-connected polygonal domains (as distinguished from arbitrary sets of points and open line segments). The following section describes properties of Voronoi diagrams that we shall need to develop the algorithm.

4. Properties of Voronoi diagrams

For our purposes, the most important property of Voronoi diagrams of multiply-connected polygonal domains is that, under certain conditions, merge curves are simple, closed curves. This result, which is stated below in Theorem 1, is proved by showing that certain Voronoi regions are simply-connected, path-connected, and bounded.

Definition 11
A planar region R is *generalized-star-shaped* with *nucleus* N, $N \subseteq R$, if for any point $r \in R$ there exists a point $n \in N$ such that the closed line segment $[r, n]$ lies completely in R. □

Lemma 2
The Voronoi region $V(e_i, S)$ is generalized-star-shaped, with nucleus e_i.

Proof Lemma 1 of [2]. □

Lemma 3
$V(S_1, S_2)$ is generalized-star-shaped, with nucleus S_1.

Proof Let p be a point in $V(S_1, S_2)$. By Corollary 2, p is in $V(e_i, S_2)$ for some e_i in S_1. Therefore, by Lemma 2, there is a closed segment between p and e_i which is contained entirely in $V(e_i, S_2)$ and hence in $V(S_1, S_2)$. □

Lemma 4
$V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is bounded.

Proof Let us denote the kth element of the boundary $\partial\Omega_{i+1}$ by e_{i+1}^k. We first prove that $V(e_{i+1}^k, \cup_{j=0}^{i} \partial\Omega_j)$ is bounded. We prove this by contradiction. Assume that the region is unbounded. Since the region enclosed by the outer boundary $\partial\Omega_0$ is bounded, there must exist a point p that lies outside $\partial\Omega_0$ but in the Voronoi region $V(e_{i+1}^k, \cup_{j=0}^{i} \partial\Omega_j)$. See **Figure 4.** Furthermore, e_{i+1}^k is an element of $\partial\Omega_{i+1}$, and therefore lies

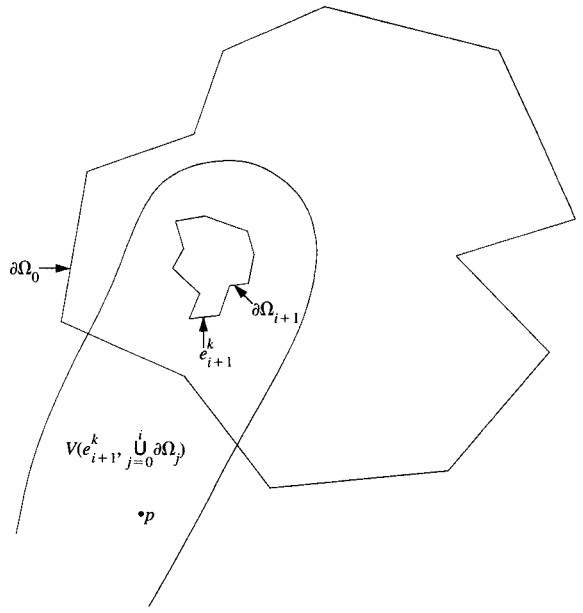

Figure 4
A point p outside $\partial\Omega_0$ and inside $V(e_{i+1}^k, \bigcup_{j=0}^{i} \partial\Omega_j)$.

inside $\partial\Omega_0$. But, since p is outside $\partial\Omega_0$, it must be closer to a point on $\partial\Omega_0$ than to any point contained in the region enclosed by $\partial\Omega_0$. Therefore, p is closer to $\partial\Omega_0$ than to e_{i+1}^k. This contradicts the assumption that p is in $V(e_{i+1}^k, \cup_{j=0}^{i} \partial\Omega_j)$, proving that $V(e_{i+1}^k, \cup_{j=0}^{i} \partial\Omega_j)$ is bounded. From Corollary 2, we know that $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j) = \cup_{k=1}^{n_{i+1}} V(e_{i+1}^k, \cup_{j=0}^{i} \partial\Omega_j)$. Hence, $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is bounded. □

Lemma 5
$V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is path-connected.

Proof A path can be constructed between any two points p and q in $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$, as illustrated in **Figure 5.** The path begins at p, goes to its image on $\partial\Omega_{i+1}$, then to the image of q on $\partial\Omega_{i+1}$, ending at q. The portions of the path between p and $\partial\Omega_{i+1}$ and between q and $\partial\Omega_{i+1}$ lie entirely within $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ by Lemma 3. The portion of the path between the images of p and q on $\partial\Omega_{i+1}$ is path-connected because, by definition, $\partial\Omega_{i+1}$ is path-connected. □

Let $CH(\partial\Omega_i)$ denote the interior of the convex hull of $\partial\Omega_i$.

Lemma 6
Let $\partial\Omega_i$ and $\partial\Omega_j$ be two polygonal hole boundaries. If $CH(\partial\Omega_j)$ does not completely contain $\partial\Omega_i$, then there exist a vertex $v \in \partial\Omega_i$ and a half-line l_v starting at v, such that l_v is completely contained in $V(\partial\Omega_i, \partial\Omega_j)$.

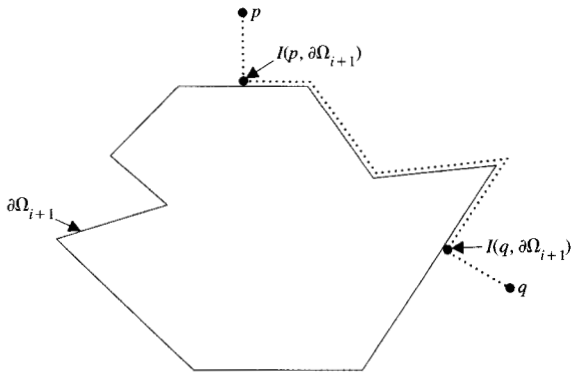

Figure 5

Connected path between points p and q.

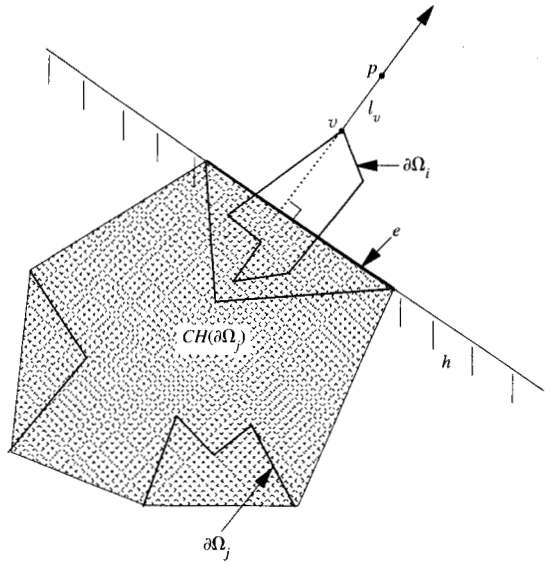

Figure 6

$\partial\Omega_i$ and $CH(\partial\Omega_j)$.

Proof Since $CH(\partial\Omega_j)$ does not completely contain $\partial\Omega_i$, there must exist a vertex $v \in \partial\Omega_i$ that lies outside of $CH(\partial\Omega_j)$. Also, there must exist an edge e on the boundary of $CH(\partial\Omega_j)$ such that a half-plane h constructed on e completely contains $CH(\partial\Omega_j)$ but not v. See **Figure 6.** Construct a half-line l_v in $\bar{h}$ such that l_v starts at v and is perpendicular to e.

Let p be any point in l_v. Since l_v is perpendicular to the boundary of h, p is closer to v than to any element in $\partial\Omega_j$. Hence $p \in V(\partial\Omega_i, \partial\Omega_j)$. Since the choice of p on l_v is arbitrary, l_v is completely contained in $V(\partial\Omega_i, \partial\Omega_j)$. □

Lemma 7

If $CH(\partial\Omega_{i+1})$ does not completely contain any $\partial\Omega_j$, $j = 1, \cdots, i$, then $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is simply connected.

Proof We must show that any simple, closed curve in $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ can be shrunk to a point without leaving $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$. Let C be such a curve and let R_C denote the region enclosed by C. To show that C can be shrunk to a point without leaving $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$, we must show that all points in R_C are also in $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$. We show this by contradiction. Let p be any point in R_C. Assume that p is not in $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$. Then $p \in \text{cl}\, V(\cup_{j=0}^{i} \partial\Omega_j, \partial\Omega_{i+1})$, and, by the argument used in the proof of Lemma 3, there must exist a $\partial\Omega_k$, $0 \le k \le i$, such that the line segment from p to its image on $\partial\Omega_k$ is completely contained in $\text{cl}\, V(\cup_{j=0}^{i} \partial\Omega_j, \partial\Omega_{i+1})$. Also, since $CH(\partial\Omega_{i+1})$ does not completely contain $\partial\Omega_k$, by Lemma 6 there must exist a vertex $v \in \partial\Omega_k$ and a half-line l_v such that l_v is completely contained in $V(\partial\Omega_k, \partial\Omega_{i+1})$. We can now construct a continuous path C_p from p to its image $I(p, \partial\Omega_k)$, to vertex $v \in \partial\Omega_k$, and to infinity along l_v. See **Figure 7.** Note that C_p is completely contained in $\text{cl}\, V(\cup_{j=0}^{i} \partial\Omega_j, \partial\Omega_{i+1})$, and since $p \in R_C$ and C_p is unbounded, C_p must intersect C at, say, q. Since $q \in C_p$, q must be contained in $\text{cl}\, V(\cup_{j=0}^{i} \partial\Omega_j, \partial\Omega_{i+1})$. This contradicts the assumption that C is in $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$. □

Note that the condition stated in Lemma 7 is a sufficient but not a necessary condition for the Voronoi region to be simply-connected. We now show that given H polygonal holes, we can always sort the holes so that the condition stated in Lemma 7 is satisfied.

Lemma 8

Let Y_j be the y-coordinate of the topmost vertex on $\partial\Omega_j$. If the hole boundaries have been sorted such that $Y_{j+1} \le Y_j$, then $CH(\partial\Omega_{i+1})$ does not completely contain any $\partial\Omega_j$, $j = 1, \cdots, i$.

Proof We prove this by contradiction. Assume that $CH(\partial\Omega_{i+1})$ completely contains a $\partial\Omega_j$, $1 \le j \le i$. Then $Y_{i+1} > Y_j$. This contradicts the sort criterion that $Y_{j+1} \le Y_j$. □

Henceforth, we assume that the holes have been sorted so that the condition stated in Lemma 7 is satisfied.

Corollary 3

$V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is bounded, path-connected, and simply-connected.

Proof Follows immediately from Lemmas 4, 5, and 7. □

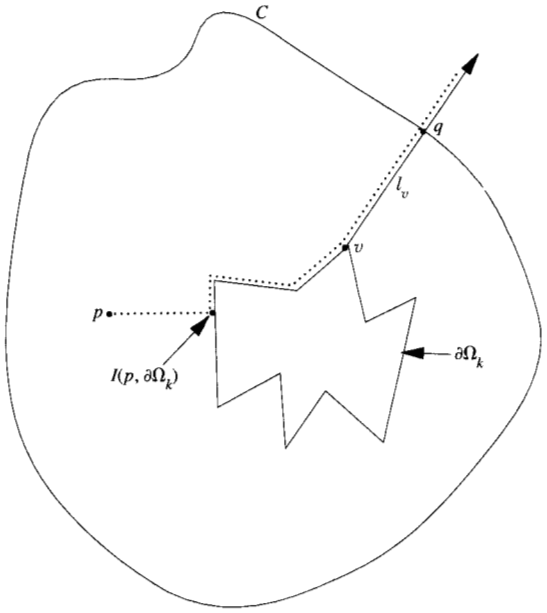

Figure 7

Intersection of C and C_p. Dotted curve is C_p.

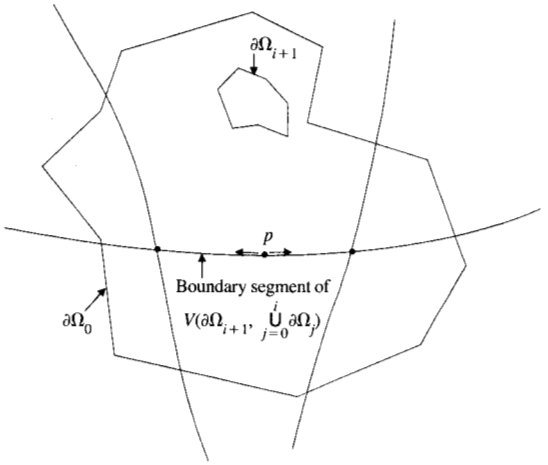

Figure 8

Intersection of half-plane boundaries.

Theorem 1
The boundary of $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is a simple, closed curve.

Proof As noted in the discussion following Corollary 2, the boundary of $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ consists of the union of connected portions of half-plane boundaries, each of which we call a half-plane boundary segment. We now argue that the boundary consists of one or more cycles of such half-plane boundary segments. Pick a point on the boundary and traverse the boundary in some direction. Since $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is bounded (Lemma 4) and no half-plane boundary closes on itself, the traversal can neither go to infinity nor close on itself in the same half-plane boundary segment. Hence, it must reach an endpoint of the half-plane boundary segment. But, as illustrated in **Figure 8**, an endpoint can only be created by the intersection with another half-plane boundary. At such an intersection, the traversal continues on a segment of the second half-plane boundary. Since, by reversing the direction of traversal, the same argument applies to the other endpoint of the half-plane boundary segment, we conclude that each half-plane boundary segment is connected to at least one other half-plane boundary segment at each endpoint. Thus, each half-plane boundary segment is contained in a boundary cycle. Since, by Corollary 3, $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is both path-connected and simply-connected, there can only be one such boundary cycle. □

Definition 12
The bisector $B(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ is called a *merge curve*. □

Note that the merge curve is also the boundary of $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$.

Lemma 9
$VOD(\cup_{j=0}^{i+1} \partial\Omega_j)$ consists of the union of that portion of $VOD(\cup_{j=0}^{i} \partial\Omega_j)$ lying outside the merge curve $B(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$ and that portion of $VOD(\partial\Omega_{i+1})$ lying inside the merge curve.

Proof Let us first establish some simple notations. Let $\partial_{\text{old}} = \cup_{j=0}^{i} \partial\Omega_j$, $\partial_{\text{hole}} = \partial\Omega_{i+1}$, and $\partial_{\text{new}} = \cup_{j=0}^{i+1} \partial\Omega_j$. By Definition 10,

$$VOD(\partial_{\text{new}}) = \bigcup_{e_k \in \partial_{\text{new}}} V(e_k, \partial_{\text{new}} - e_k)$$

$$= \left[\bigcup_{e_k \in \partial_{\text{old}}} V(e_k, \partial_{\text{new}} - e_k) \right]$$

$$\cup \left[\bigcup_{e_k \in \partial_{\text{hole}}} V(e_k, \partial_{\text{new}} - e_k) \right].$$

We can expand the first subexpression as

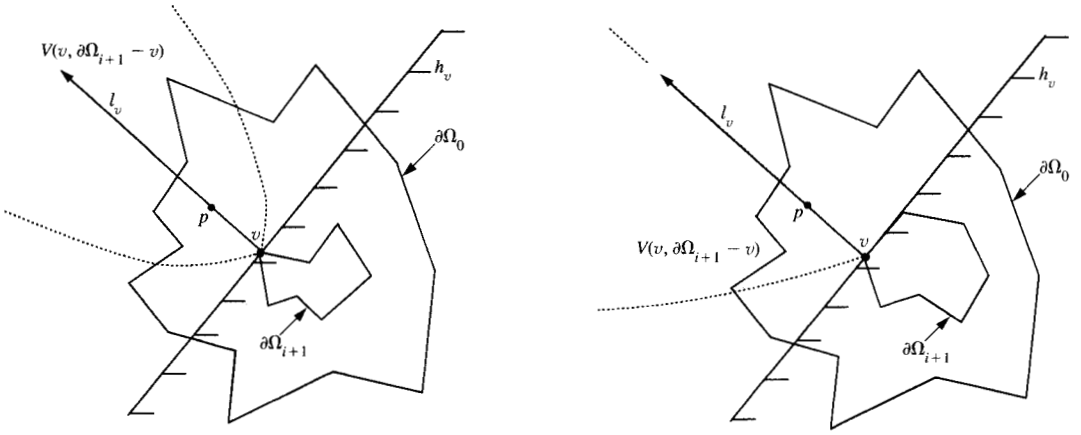

Figure 9

Two possible cases for the inclusion of l_v: (a) l_v is completely contained in $V(v, \partial\Omega_{i+1} - v)$. (b) l_v is on the boundary of $V(v, \partial\Omega_{i+1} - v)$.

$$\bigcup_{e_k \in \partial_{\text{old}}} V(e_k, \partial_{\text{new}} - e_k) = \bigcup_{e_k \in \partial_{\text{old}}} \bigcap_{e_j \in \partial_{\text{new}} - e_k} h(e_k, e_j)$$

$$= \bigcup_{e_k \in \partial_{\text{old}}} \left[\left\{ \bigcap_{e_j \in \partial_{\text{old}} - e_k} h(e_k, e_j) \right\} \cap \left\{ \bigcap_{e_j \in \partial_{\text{hole}}} h(e_k, e_j) \right\} \right]$$

$$= \left[\bigcup_{e_k \in \partial_{\text{old}}} V(e_k, \partial_{\text{old}} - e_k) \right] \cap [V(\partial_{\text{old}}, \partial_{\text{hole}})]$$

$$= VOD(\partial_{\text{old}}) \cap V(\partial_{\text{old}}, \partial_{\text{hole}}).$$

Similarly, we can expand the second subexpression as

$$\bigcup_{e_k \in \partial_{\text{hole}}} V(e_k, \partial_{\text{new}} - e_k) = \bigcup_{e_k \in \partial_{\text{hole}}} \bigcap_{e_j \in \partial_{\text{new}} - e_k} h(e_k, e_j)$$

$$= \bigcup_{e_k \in \partial_{\text{hole}}} \left[\left\{ \bigcap_{e_j \in \partial_{\text{old}}} h(e_k, e_j) \right\} \cap \left\{ \bigcap_{e_j \in \partial_{\text{hole}} - e_k} h(e_k, e_j) \right\} \right]$$

$$= [V(\partial_{\text{hole}}, \partial_{\text{old}})] \cap \left[\bigcup_{e_k \in \partial_{\text{hole}}} V(e_k, \partial_{\text{hole}} - e_k) \right]$$

$$= V(\partial_{\text{hole}}, \partial_{\text{old}}) \cap VOD(\partial_{\text{hole}}).$$

The result follows immediately. □

Our goal is to obtain $VOD(\bigcup_{j=0}^{i+1} \partial\Omega_j)$ from $VOD(\bigcup_{j=0}^{i} \partial\Omega_j)$ and $VOD(\partial\Omega_{i+1})$. We do this by constructing the merge curve and using Lemma 9 to discard unwanted portions of $VOD(\bigcup_{j=0}^{i} \partial\Omega_j)$ and $VOD(\partial\Omega_{i+1})$. The first step is to obtain a starting point, which is any point on the merge curve. Once a starting point is found, the entire merge curve is constructed by using a merge algorithm. We shall need the following lemmas.

Lemma 10

Let v be a vertex on the convex hull of an inner boundary $\partial\Omega_{i+1}$, and h_v be a half-plane such that h_v completely contains the convex hull and the boundary of h_v contains the vertex v. Also, let l_v be a half-line in the complement of h_v such that l_v starts at v and is perpendicular to the boundary of h_v (see **Figure 9**). Then l_v is completely contained in the closure of the Voronoi region $V(v, \partial\Omega_{i+1} - v)$ of this vertex.

Proof Since h_v completely contains the convex hull, it also contains $\partial\Omega_{i+1}$. Let p be a point on l_v. Since l_v is perpendicular to the boundary of h_v and h_v completely contains $\partial\Omega_{i+1}$, p is no farther from v than from any other element of $\partial\Omega_{i+1}$. Hence, p is contained completely inside or on the boundary of $V(v, \partial\Omega_{i+1} - v)$. Since the choice of p in l_v is arbitrary, it follows that l_v is completely contained in the closure of $V(v, \partial\Omega_{i+1} - v)$. □

Lemma 11

l_v of Lemma 10 intersects the merge curve $B(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$.

Proof Since $v \in \partial\Omega_{i+1}$, v is contained inside the merge curve. l_v is a half-line that starts at v, and, by Theorem 1, the merge curve is a simple closed curve. Therefore, l_v must intersect the merge curve. □

Any point on the intersection between l_v and the merge curve can be a starting point. In an implementation, we can pick v to be the topmost vertex (i.e., having the largest y-coordinate), or one of the topmost vertices, of $\partial\Omega_{i+1}$. This will ensure that v is on the convex hull of $\partial\Omega_{i+1}$.

5. Algorithm

The algorithm for computing the Voronoi diagram of a multiply-connected polygonal domain is shown in **Figure 10.** The algorithm first computes the Voronoi diagram of the outer boundary (step 1), sorts the hole boundaries (step 2), and then computes and merges in the Voronoi diagrams of the inner boundaries (step 3). The Voronoi diagrams of the individual boundaries can be computed using an extension of Lee's algorithm [3]. [Lee's algorithm computes the inner Voronoi diagram of a simply-connected polygon (i.e., a polygon without holes). In other words, his algorithm computes the portion of the Voronoi diagram that is inside the polygon. An extended version of his algorithm also computes the outer Voronoi diagram of a simply-connected polygon.] An algorithm for finding a starting point on the merge curve (step 3b) is described in the following and is shown in Figure 11. An algorithm for computing the merge curve and merging in the Voronoi diagram of an inner boundary (step 3c) is then described and is shown in Figure 12.

The worst-case execution time of this algorithm is easy to determine. Lee has shown [3] that $VOD(\partial\Omega_j)$ can be computed in $O(n_j \log_2 n_j)$ time. Thus, the worst-case execution time for step 1 is $O(n_0 \log_2 n_0)$. Sorting of the hole boundaries in step 2 simply reorders the sequence in which the holes are inserted. The sorting can be done by first computing the largest y-coordinate among the vertices of each hole boundary, which can totally take $O(\sum_{i=1}^{H} n_i)$ time, and then sorting these y-coordinates in descending order, which can take $O(H \log_2 H)$ time. Therefore, the total time taken for the execution of step 2 is $O(H \log_2 H + \sum_{i=1}^{H} n_i)$. The worst-case execution time for step 3a is $O(n_{i+1} \log_2 n_{i+1})$. We show below that a starting point can be found (step 3b) in $O(\sum_{j=0}^{i+1} n_j)$ time and that the merge (step 3c) can also be done in $O(\sum_{j=0}^{i+1} n_j)$ time. Therefore, an iteration of step 3 takes at most $O(n_{i+1} \log_2 n_{i+1} + \sum_{j=0}^{i+1} n_j)$ time.

The total worst-case time is thus $O(n_0 \log_2 n_0) + O(H \log_2 H + \sum_{i=1}^{H} n_i) + \sum_{i=1}^{H} O(n_i \log_2 n_i + \sum_{j=0}^{i} n_j)$. This can be written as $\sum_{i=0}^{H} O(n_i \log_2 n_i) + \sum_{i=1}^{H} O(\sum_{j=0}^{i} n_j)$. Let $N = \sum_{j=0}^{H} n_i$, that is, the total number of elements to be processed. The total time can then be written as $O(N \log_2 N) + \sum_{i=1}^{H} O(\sum_{j=0}^{i} n_j)$, which simplifies to $O(N(\log_2 N + H))$.

Input: $\bigcup_{j=0}^{H} \partial\Omega_j$.

Output: $VOD(\bigcup_{j=0}^{H} \partial\Omega_j)$.

1. Compute $VOD(\partial\Omega_0)$.
2. Sort the hole boundaries.
3. **For** $i = 0$ **to** $H - 1$ **do Begin**
   a. Compute $VOD(\partial\Omega_{i+1})$.
   b. Find a starting point $s_0 \in B(\partial\Omega_{i+1}, \bigcup_{j=0}^{i} \partial\Omega_j)$, along with vertex $v \in \partial\Omega_{i+1}$ and element $e_k \in \bigcup_{j=0}^{i} \partial\Omega_j$ that define s_0.
   c. Merge $VOD(\bigcup_{j=0}^{i} \partial\Omega_j)$ and $VOD(\partial\Omega_{i+1})$ to obtain $VOD(\bigcup_{j=0}^{i+1} \partial\Omega_j)$.

   End

Figure 10

Algorithm for computing $VOD(\bigcup_{j=0}^{H} \partial\Omega_j)$.

• *Finding a starting point*

Recall that we wish to obtain a starting point on the merge curve and then to construct the merge curve through that point. Any point on the merge curve will suffice. By Lemma 11 we know that if we pick a vertex v, which is the topmost vertex or one of the topmost vertices on the inner boundary, the half-line l_v, which starts at v and is directed vertically upwards, intersects the merge curve. That point of intersection will be our starting point. However, since the merge curve is not yet known, we need some other test that will determine where l_v intersects the merge curve.

The merge curve is the boundary of the Voronoi region $V(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$, and therefore consists of the union of portions of bisectors. The starting point must then be the intersection of l_v with some (yet to be determined) bisector. Observe that the starting point must be contained in $\mathrm{cl}\, V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$, for some $e_k \in \cup_{j=0}^{i} \partial\Omega_j$. We claim that $B(v, e_k)$, the bisector between v and e_k, is the desired bisector. This is proved in the following lemma.

Lemma 12

Let s_0 be a point. Then, $s_0 \in B(v, e_k) \cap l_v \cap \mathrm{cl}\, V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$ if and only if $s_0 \in B(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j) \cap l_v \cap \mathrm{cl}\, V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$.

Proof Assume $s_0 \in B(v, e_k) \cap l_v \cap \mathrm{cl}\, V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$. We must show that $s_0 \in B(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$, that is, that s_0 is equidistant from $\partial\Omega_{i+1}$ and $\cup_{j=0}^{i} \partial\Omega_j$. Since $s_0 \in B(v, e_k)$, it is equidistant from v and e_k. Moreover, since $s_0 \in l_v$, there is

Input: $VOD(\bigcup_{j=0}^{i} \partial\Omega_j)$, $VOD(\partial\Omega_{i+1})$, $\partial\Omega_{i+1}$.

Output: s_0, v, e_k.

1. Find the topmost vertex (or one of the topmost vertices) v on $\partial\Omega_{i+1}$.
2. Find $e_k \in \bigcup_{j=0}^{i} \partial\Omega_j$ such that v is contained in the closure of $V(e_k, \bigcup_{j=0}^{i} \partial\Omega_j - e_k)$.
3. **While true do Begin**
   a. $l_s \leftarrow l_v \cap clV(e_k, \bigcup_{j=0}^{i} \partial\Omega_j - e_k)$.
   b. $s_0 \leftarrow l_s \cap B(v, e_k)$.
   c. **If** $s_0 \neq \phi$ **then return** s_0, v, e_k
      else Begin
      1) Find $e_\lambda \in \bigcup_{j=0}^{i} \partial\Omega_j$ such that l_v enters into $V(e_\lambda, \bigcup_{j=0}^{i} \partial\Omega_j - e_\lambda)$ from $V(e_k, \bigcup_{j=0}^{i} \partial\Omega_j - e_k)$.
      2) $e_k \leftarrow e_\lambda$.
      End
   End

Figure 11

Algorithm for finding a starting point.

no element of $\partial\Omega_{i+1}$ that is closer to s_0 than v is. Similarly, since $s_0 \in$ cl $V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$, there is no element of $\cup_{j=0}^{i} \partial\Omega_j$ that is closer to s_0 than e_k is. Hence, s_0 is equidistant from $\partial\Omega_{i+1}$ and $\cup_{j=0}^{i} \partial\Omega_j$. Thus, $s_0 \in B(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j)$.

Now assume $s_0 \in B(\partial\Omega_{i+1}, \cup_{j=0}^{i} \partial\Omega_j) \cap l_v \cap$ cl $V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$. We must show that $s_0 \in B(v, e_k)$, that is, that s_0 is equidistant from v and e_k. Since s_0 is in cl $V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$, there is no element in $\cup_{j=0}^{i} \partial\Omega_j$ which is closer to s_0 than to e_k. Also, since s_0 is in l_v, there is no element of $\partial\Omega_{i+1}$ that is closer to s_0 than v. Therefore, since s_0 is equidistant from $\partial\Omega_{i+1}$ and $\cup_{j=0}^{i} \partial\Omega_j$, it must be equidistant between v and e_k. Thus, $s_0 \in B(v, e_k)$. □

Since the desired bisector depends on the region $V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$ in which the starting point lies, the algorithm scans along l_v starting from v and determines for each Voronoi region encountered whether the starting point lies on l_v in that region. By Lemma 12, this is equivalent to determining whether or not the bisector $B(v, e_k)$ intersects l_v in the region $V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$.

A detailed algorithm for finding a starting point is shown in **Figure 11.** The algorithm begins (step 1) by finding the topmost vertex (or one of the topmost vertices) on $\partial\Omega_{i+1}$. This can be done in $O(n_{i+1})$ time by finding the vertex v of $\partial\Omega_{i+1}$ with the largest y-coordinate.

Step 2 finds the element e_k in $\cup_{j=0}^{i} \partial\Omega_j$ whose Voronoi region $V(e_k, \cup_{j=0}^{i} \partial\Omega_j - e_k)$ contains the vertex v. (When v lies on the common boundary of two or more Voronoi regions, the algorithm can choose any element such that l_v enters that element's Voronoi region.) This can be done in $O(\sum_{j=0}^{i} n_j)$ time by scanning all the elements in $\cup_{j=0}^{i} \partial\Omega_j$ and finding that particular element e_k which is the closest to v.

Step 3 implements the scanning process to determine the starting point. Step 3a computes the intersection of the half-line l_v and the Voronoi region of e_k. This involves finding the intersection of l_v with the edges of the Voronoi region of e_k, and in the worst case, we may have to do this for all the elements in $\cup_{j=0}^{i} \partial\Omega_j$. Since there are $O(N)$ Voronoi edges in the Voronoi regions of N elements [2], the total time taken to execute step 3a (i.e., over all iterations) is $O(\sum_{j=0}^{i} n_j)$.

Step 3b can be executed in constant time. Since this step can potentially be executed $O(\sum_{j=0}^{i} n_j)$ times, the total time taken to execute this step is $O(\sum_{j=0}^{i} n_j)$.

Execution of step 3c1 on subsequent loop iterations usually takes constant time because the termination point (i.e., the last point in the direction of l_v) on l_s lies on a Voronoi edge. As a result, the region into which l_v enters must be the other Voronoi region that shares this edge. However, in the worst case, all of the termination points could fall on Voronoi vertices. For each such occurrence, all of the Voronoi edges incident upon the Voronoi vertex must be examined to determine e_k. Since there are a total of $O(\sum_{j=0}^{i} n_j)$ edges in $VOD(\cup_{j=0}^{i} \partial\Omega_j)$ and step 3c2 can be done in constant time, the total time taken to execute step 3c (i.e., over all iterations) is $O(\sum_{j=0}^{i} n_j)$.

Hence, the total time taken to execute step 3 is $O(\sum_{j=0}^{i} n_j)$. The algorithm terminates because, by Lemma 11, there must be a starting point on l_v. Since step 3 scans along l_v, it must eventually find the starting point and terminate. Thus the time complexity of the algorithm to find the starting point is $O(\sum_{j=0}^{i} n_j)$.

- *Merging Voronoi diagrams*

Once a starting point on the merge curve is found, the next step is to traverse the entire merge curve. Recall that the merge curve consists of pieces of bisectors between pairs of elements. The strategy we adopt is to identify element pairs (one element belonging to $\partial\Omega_{i+1}$, the other to $\cup_{j=0}^{i} \partial\Omega_j$), whose bisectors contribute to the merge curve. Efficiency requires that the merge algorithm exploit the continuity of the merge curve to avoid exhaustive searching. The same approach to merging two Voronoi diagrams has been used by Shamos and Hoey [6], Lee and Drysdale [2], and Lee [3]. The algorithm for computing the merge curve is shown in **Figure 12.** The algorithm given here has a simple termination condition based on the fact that the merge curve is a simple closed curve.

After the initializations in step 1, step 2 computes the merge curve. In step 2a1, $\bar{b}$ inherits the direction from the oriented bisector $\bar{B}(l, r)$. Since $\bar{B}(l, r)$ has a direction, we can identify the "first" continuous piece, $\bar{b}_1$, of the indicated intersection. (Since the bisectors can involve parabolas, and the Voronoi regions are not always convex, the indicated intersections can be a set of discontinuous pieces, hence the need to identify the first continuous piece.) We can also easily determine the termination point p (i.e., the last point on $\bar{b}_1$) along the direction of $\bar{b}_1$. When step 2a is executed for the first time, it may take, in the worst case, $O(\sum_{j=0}^{i+1} n_j)$ time. But in subsequent iterations of step 2a, we can use the tactics of Lee and Drysdale [2], where the boundary of $V(l, \partial\Omega_{i+1} - l)$ is traversed in the counterclockwise direction from the last examined Voronoi edge, and the boundary of $V(r, \cup_{j=0}^{i} \partial\Omega_j - r)$ is traversed in the clockwise direction from the last examined edge, to obtain their intersection with $\bar{B}(l, r)$. This avoids backtracking, and Lee and Drysdale have shown that two Voronoi diagrams, one of m elements, the other of n elements, can be merged in $O(m + n)$ time in this manner. Their analysis holds here as well. Therefore, the total time taken to execute step 2a (i.e., over all iterations) is $O(\sum_{j=0}^{i+1} n_j)$.

Execution of steps 2b and 2c on subsequent loop iterations usually takes constant time because the termination point p [i.e., the last point on $\bar{b}_1$ in the direction of $\bar{B}(l, r)$] on $\bar{b}_1$ usually lies on a Voronoi edge. As a result, the region which $\bar{b}_1$ tries to enter must be the other Voronoi region that shares this edge. However, in the worst case, all of the termination points could fall on Voronoi vertices. For each such occurrence, all of the Voronoi edges incident upon the Voronoi vertex must be examined to determine λ or ρ. Since there are a total of $O(\sum_{j=0}^{i+1} n_j)$ edges in $VOD(\cup_{j=0}^{i} \partial\Omega_j)$ and $VOD(\partial\Omega_{i+1})$, the total time taken to execute steps 2b and 2c (i.e., over all iterations) is $O(\sum_{j=0}^{i+1} n_j)$.

Hence, the total time taken to execute step 2 is $O(\sum_{j=0}^{i+1} n_j)$. Step 2 terminates because, by Theorem 1, the merge curve must close on itself.

Once the merge curve itself is constructed, the new Voronoi diagram is obtained (step 3) by discarding portions of the original Voronoi diagrams as described in Lemma 9. In an implementation, this step can be carried out simultaneously with step 2 by proper updating of the Voronoi regions of various elements involved. Therefore, the merge algorithm (Figure 12) can be executed in $O(\sum_{j=0}^{i+1} n_j)$ time.

Input: $VOD(\bigcup_{j=0}^{i} \partial\Omega_j)$, $VOD(\partial\Omega_{i+1})$, s_0, v, e_k.

Output: $VOD(\bigcup_{j=0}^{i+1} \partial\Omega_j)$.

1. $l \leftarrow v$, $r \leftarrow e_k$, and initialize the merge curve.
2. **Repeat Begin**
   a.
      1) $\bar{b} \leftarrow \{V(l, \partial\Omega_{i+1} - l) \cap \bar{B}(l, r)\} \cap \{V(r, \bigcup_{j=0}^{i} \partial\Omega_j - r) \cap \bar{B}(l, r)\}$.
      2) $\bar{b}_1 \leftarrow$ first continuous piece of $\bar{b}$, and add $\bar{b}_1$ to the merge curve.
      $p \leftarrow$ termination point of $\bar{b}_1$.
   b. **If** p is on a Voronoi edge or on a Voronoi vertex of $V(l, \partial\Omega_{i+1} - l)$
      then Begin
      1) Find $\lambda \in \partial\Omega_{i+1}$ such that $\bar{b}_1$ tries to enter into $V(\lambda, \partial\Omega_{i+1} - \lambda)$ from $V(l, \partial\Omega_{i+1} - l)$.
      2) $l \leftarrow \lambda$.
      End
   c. **If** p is on a Voronoi edge or on a Voronoi vertex of $V(r, \bigcup_{j=0}^{i} \partial\Omega_j - r)$
      then Begin
      1) Find $\rho \in \bigcup_{j=0}^{i} \partial\Omega_j$ such that $\bar{b}_1$ tries to enter into $V(\rho, \bigcup_{j=0}^{i} \partial\Omega_j - \rho)$ from $V(r, \bigcup_{j=0}^{i} \partial\Omega_j - r)$.
      2) $r \leftarrow \rho$.
      End
   End Until $l = v$ and $r = e_k$.
3. Discard all portions of $VOD(\bigcup_{j=0}^{i} \partial\Omega_j)$ that lie within $B(\partial\Omega_{i+1}, \bigcup_{j=0}^{i} \partial\Omega_j)$ and all portions of $VOD(\partial\Omega_{i+1})$ that lie outside of $B(\partial\Omega_{i+1}, \bigcup_{j=0}^{i} \partial\Omega_j)$.

Figure 12

Algorithm for merging $VOD(\bigcup_{j=0}^{i} \partial\Omega_j)$ and $VOD(\partial\Omega_{i+1})$.

6. Concluding remarks

We have presented an $O(N(\log_2 N + H))$ algorithm to compute the Voronoi diagram of a multiply-connected polygonal domain. It is not an optimal algorithm, but it is simple and implementable. Moreover, in most practical applications the number of holes is far less than the number of edges in the input domain, which brings the worst-case efficiency of our algorithm closer to the optimum. This is an example of an algorithm in which we have traded worst-case complexity for simplicity.

The algorithm has been implemented and applied by Meshkat and Sakkas [5] to solve an important problem in VLSI design. After we completed the theoretical work on the algorithm [7] and the implementation was well under way, we came across a sweepline algorithm by Fortune [8] which computes the Voronoi diagram of points and line segments

in $O(N \log_2 N)$ time. If Fortune's algorithm can be implemented to include both points and line segments and if the sweepline technique is implemented to handle degeneracies (i.e., many vertices lying on the same sweepline, as is the case in VLSI applications), it should also be applicable to such VLSI problems.

The paper by Meshkat and Sakkas [5] reports some experimental data on the run-time efficiency of our algorithm. A theoretical average-case analysis of the algorithm would be very useful to compare with such experimental data. This would entail defining some practically useful notion of what "average" means in, for example, VLSI applications. Another useful exploration would be to find some way of quantifying the above-mentioned trade-off between the complexity of an algorithm and the simplicity of its implementation.

It is reasonable to expect that with modest additional effort one can compute Voronoi diagrams of multiply-connected domains whose boundary consists of curvilinear segments as well. A more challenging task is to find algorithms that handle higher-dimensional spaces, i.e., polyhedra, where important applications can be found.

7. Acknowledgments

We are indebted to two of our colleagues who helped us with the algorithm reported in this paper. Michael A. O'Connor suggested Lemma 10 and its use in finding a starting point; this simplified considerably an earlier starting-point algorithm. V. Thomas Rajan carefully read a draft of this paper and pointed out a major flaw in an earlier version of Lemma 7. He also suggested a proof for the current version of the same lemma, and the sorting of the hole boundaries.

References

1. D. G. Kirkpatrick, "Efficient Computation of Continuous Skeletons," *IEEE 20th Annual Symposium on Foundations of Computer Science*, 1979, pp. 18–27.
2. D. T. Lee and R. L. Drysdale, "Generalization of Voronoi Diagrams in the Plane," *SIAM J. Computing* **10,** No. 1, 73–87 (February 1981).
3. D. T. Lee, "Medial Axis Transformation of a Planar Shape," *IEEE Trans. Pattern Anal. & Machine Intell.* **PAMI-4,** No. 4, 363–369 (July 1982).
4. C. K. Yap, "An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments," preliminary version of a report, Courant Institute of Mathematical Sciences, New York University, New York, October 1984.
5. Siavash N. Meshkat and Constantine M. Sakkas, "Voronoi Diagram for Multiply-Connected Polygonal Domains II: Implementation and Application," *IBM J. Res. Develop.* **31,** No. 3, 373–381 (May 1987, this issue).
6. M. I. Shamos and D. Hoey, "Closest-Point Problems," *IEEE 16th Annual Symposium on Foundations of Computer Science*, 1975, pp. 151–162.
7. V. Srinivasan and L. R. Nackman, "An Algorithm to Compute the Voronoi Diagram of a Multiply Connected Polygonal Domain," *Research Report RC-11605*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, December 1985.
8. S. Fortune, "A Sweepline Algorithm for Voronoi Diagrams," *Proceedings of the Second Annual Symposium on Computational Geometry*, Yorktown Heights, NY, June 1986, pp. 313–322.

Received October 1, 1986; accepted for publication January 6, 1987

Vijay Srinivasan *IBM Thomas J. Watson Research Center, P. O. Box 218, Yorktown Heights, New York 10598.* Dr. Srinivasan joined IBM in 1983 as a Research Staff Member and currently manages the Design Automation Science Project at the IBM Thomas J. Watson Research Center. He received his B. Tech. degree in 1976 and his Ph.D. in 1980, both in mechanical engineering, from the Indian Institute of Technology, Madras, India. His research interests include finite-element modeling, dynamics of flexible systems, geometric modeling, theory of tolerances, and mechanical design theory. Dr. Srinivasan is a member of the Design Automation Committee of the American Society of Mechanical Engineers, and is an adjunct faculty member of the Department of Mechanical Engineering at Columbia University, New York.

Lee R. Nackman *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Nackman has been a Research Staff Member at the IBM Thomas J. Watson Research Center since 1982 and is now manager of the Design Automation Systems project. He received an Sc.B. degree in computer science from Brown University, Providence, Rhode Island, in 1976 and a Ph.D. degree in computer science from the University of North Carolina at Chapel Hill in 1982. In 1983, he was also an adjunct assistant professor of computer science at the Manhattanville campus of New York University. His current research is in geometric algorithms and software system structures for computer-aided design systems, especially solid modeling systems.