Voronoi diagram for multiply-connected polygonal domains II: Implementation and application

by Siavash N. Meshkat
Constantine M. Sakkas

Voronoi diagrams have many novel applications in computer-aided design. In this paper, an implementation of a Voronoi diagram algorithm described in a companion paper by Srinivasan and Nackman is presented. This Voronoi diagram is then used for an application in which equivalent resistance networks are derived from a boundary representation of a two-dimensional VLSI geometry.

1. Introduction

Voronoi diagrams have interesting geometrical properties that make them attractive for many computer-aided design applications. In particular, an important problem in the electrical analysis of VLSI designs is the extraction of an equivalent resistance network from a polygonal representation of its geometry. This problem can be solved efficiently by generating the skeleton of the polygon and converting the skeleton directly into an electrical network graph with the appropriate branch resistances. Interestingly, the skeleton (also called the symmetric axis) of a given domain is a subset of the corresponding Voronoi diagram [1]; hence the network can be extracted by first obtaining the Voronoi diagram.

There are theoretical Voronoi diagram algorithms reported in the literature that handle general cases of discrete points and line segments in a plane [2–4], and subsume the case of multiply-connected polygons that are of interest to us. However, these algorithms are too complex for implementation, and to our knowledge no details of their implementation exist. Only one, due to Lee [1], shows an implementation of a Voronoi diagram algorithm for simply-connected polygonal domains. The algorithm due to Srinivasan and Nackman [5], described in a companion paper, is applicable to our multiply-connected polygonal domains, efficient in terms of its time complexity, and relatively simple to implement.

In Section 2, the data structures and basic geometrical computations involved in our implementation of that algorithm are discussed. Section 3 gives important details of our implementation and compares the theoretical upper bound for the time complexity with experimental data derived from our program execution. We describe in Section

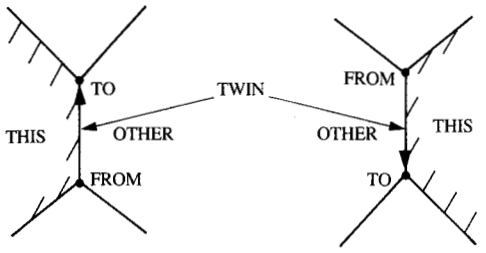

Figure 1

Edge data structure: Each directed edge is a twin of the other. FROM and TO are the endpoints of the directed edge. THIS and OTHER are the elements of the original polygonal boundary whose bisector contains the edge.

4 how an equivalent resistance network can be derived using the Voronoi diagram of a two-dimensional VLSI geometric specification. In Section 5, an approach for obtaining the branch resistances of the resistance network is presented.

2. Data structures and geometrical computations

The choice of data structures is one of the most critical decisions in designing a geometrical algorithm. To make this choice, one must first derive data abstractions for the objects to be modeled and their relationships. There are three basic objects that can represent a Voronoi diagram—the same objects that describe a polygon: regions, edges, and vertices.

Each region is bounded by a chain of edges, and each edge is shared by exactly two regions. Each edge is bounded by two vertices (one of them could be at infinity), and each vertex can be shared by many edges. It is convenient to represent an edge by two directed edges that oppose each other, and to associate each directed edge with a region that lies to the left of it. These two directed edges are twins. Note that each Voronoi region is associated with a vertex or an edge of the original polygon, and it is convenient to refer to these regions in terms of the corresponding vertex or edge. This model is essentially equivalent to the winged-edge data structure [6] or the doubly-chained edge list (DCEL) [7]. **Figure 1** summarizes the edge data structure that we have adopted.

In addition to the incidence and connectivity relations described above, some geometrical data should be stored. The boundary edges of our Voronoi regions are line segments, parabolic arcs, or half-lines. In processing intermediate data, there is also a need to represent entire lines. One field called STYLE in the data structure for an edge can take on a value STRAIGHT or PARABOLA, and another field TYPE can take on a value OPEN, OPENTO, OPENFROM, or CLOSED to represent the geometric form of an edge.

The endpoints of the edges are the vertices that can be stored in the fields FROM and TO, one or both of which can be at infinity. A line segment is completely determined by its endpoints. A half-line is determined by one of its endpoints and by a direction which can be inferred from the THIS and the OTHER fields, which give the elements whose directed bisector contains the half-line. A line is also similarly determined by the THIS and the OTHER fields. In the case of a parabolic arc, FROM and TO fields give the endpoints, and THIS and OTHER fields give the directrix and the focus of the directed parabola that contains the arc. Optional geometric information can also be stored to speed up the processing. For example, coefficients of the line equation associated with lines, half-lines, and line segments can be stored with a straight edge. Three optional parameters stored with a parabolic edge can speed up certain linear transformations needed in geometric calculations.

The most frequently encountered geometric calculation in the Voronoi algorithm is the intersection of bisectors. Intersection of lines can be computed simply by solving two linear equations. Intersection computation of a line and a parabola involves solving a quadratic equation, which is easily available in closed form. Intersection of two arbitrary parabolas involves the solution of a fourth-degree polynomial equation. However, the parabolas which are required to intersect in the Voronoi algorithm have the interesting property that they share either the directrix or the focus. This reduces the problem to one of computing the intersection of one of the parabolas and a line (which is the bisector between the unshared directrices or foci), which can be done easily in closed form.

Another important geometric computation is to determine whether a bisector enters a Voronoi region, or, equivalently, whether a bisector enters the positive side of an edge. If the bisector and the edge are linear, this computation is trivial. If the bisector or the edge or both are parabolic, we use the tangent(s) at the point of intersection to answer this geometric question.

All of our geometrical computations were carried out using floating-point numbers, and we compared two floating-point numbers to within a small value ε. The choice of ε can be made relative to the coordinate ranges in the problem space. In our implementation, ε was chosen to be $10^{-4}u$, where u is the smallest unit of coordinates. In most cases $u = 1$.

3. Algorithm implementation and performance

The Srinivasan-Nackman algorithm [5] for multiply-connected polygonal domains uses the internal and external

Voronoi diagrams of simply-connected polygonal domains, and then merges them in a specific sequence. We implemented the $O(n \log_2 n)$ algorithm due to Lee [1], which uses a divide and conquer strategy, to compute these internal Voronoi diagrams of simply-connected domains. We also extended Lee's algorithm to compute the external Voronoi diagrams of simply-connected domains. A major part of the implementation of Lee's algorithm was the merge procedure, which merges the Voronoi diagrams of two chains of edges. In fact, much of this code was also used in computing the merge curve in the Srinivasan-Nackman algorithm.

Srinivasan and Nackman have outlined a procedure to compute the starting point on their merge curve (see [5] for details). In a part of this procedure they obtain a segment of a half-line l_v that lies in the closure of a Voronoi region, and then intersect this segment with a bisector. In our implementation, we first computed the intersection between the half-line l_v and the bisector, and then checked whether this intersection is contained in the closure of the Voronoi region. This change was made in order to economize on new code, and it does not change the worst-case complexity of the algorithm.

The procedure to obtain the merge curve outlined by Srinivasan and Nackman was implemented using essentially the same code that was used in the merge step of Lee's algorithm, with some additional code for the termination condition and postprocessing. The quantity of code added for the termination condition is relatively small. A postprocessing step was needed for the following reason. In Lee's algorithm the starting point on the merge curve is also a Voronoi vertex, and it lies on the boundary of two of the unmerged Voronoi regions. Starting at this point, we computed which of the two Voronoi regions was exited first by the bisector between the elements that own the Voronoi regions. The Voronoi region that we exited first was updated by including new edges and vertices and discarding old edges and vertices. In the Srinivasan-Nackman algorithm, the starting point usually lies in the interior of two Voronoi regions, and upon exiting any of these Voronoi regions we could not update the Voronoi region that we exited. We kept the information regarding the first two Voronoi regions and updated them in the postprocessing step.

Storage management was quite critical in our implementation since we allocate and free a very large number of objects in our VLSI applications. On the basis of theoretical upper limits, storage pools were obtained for objects such as vertices and edges. Then each area was managed by a storage management subsystem. This subsystem provided services such as allocating and freeing each object as well as keeping track of the usage of these objects. In the worst case, our scheme might use slightly more storage than allocation and deallocation for each object instance, but it improves performance and prevents internal storage fragmentation.

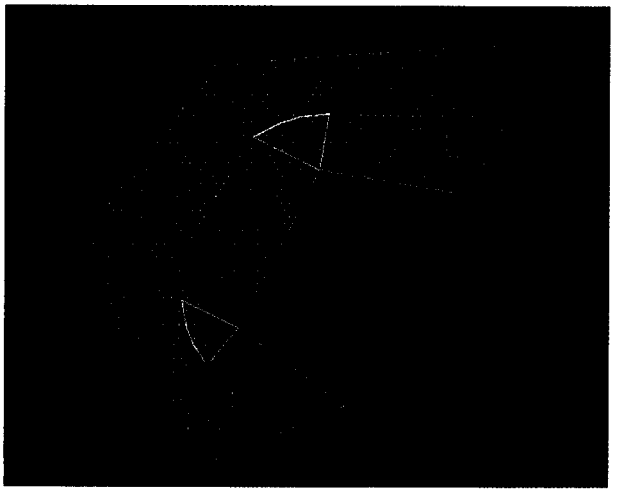

Figure 2

Voronoi diagram of a simply-connected nonorthogonal object.

We implemented the algorithm in PL/I in about 4000 lines of code organized in 73 procedures. The procedures were grouped into algorithmic, geometric, arithmetic, storage management, and input/output subsystems. The algorithmic subsystem is a collection of modules that implement the Lee and Srinivasan-Nackman algorithms. It governs the flow of control and the termination conditions. The geometric subsystem performs all of the manipulations on vertices, edges, and regions. Its tasks include the following: the computation of bisectors, distance between edges and vertices, and point of intersection between straight and parabolic bisectors; and the linear transformation of coordinate systems. The arithmetic subsystem performs tests for approximate equality of floating-point values. The storage management subsystem is a specialized set of routines to allocate, replace, free, and monitor the usage of points and edges. The input/output subsystem reads and sorts the vertices of the input polygons and creates printable and plot-ready output. It also contains optional plotting of intermediate results for debugging and step-by-step demonstration. The work from initial understanding of the algorithm to producing a running code was done in four person-months.

Examples of output from our implementation are presented in several figures. **Figure 2** shows the Voronoi diagram of a simply-connected nonorthogonal shape. The boundary edges of the Voronoi diagram are color-coded. The edges of the input polygon are shown in blue, and the interior is lightly shaded in blue. Each bisector of adjacent vertices is green, while bisectors of nonadjacent edges are red. The bisectors of a vertex and its adjacent edges are depicted in turquoise. The pink bisectors lie between a vertex and its nonadjacent edges.

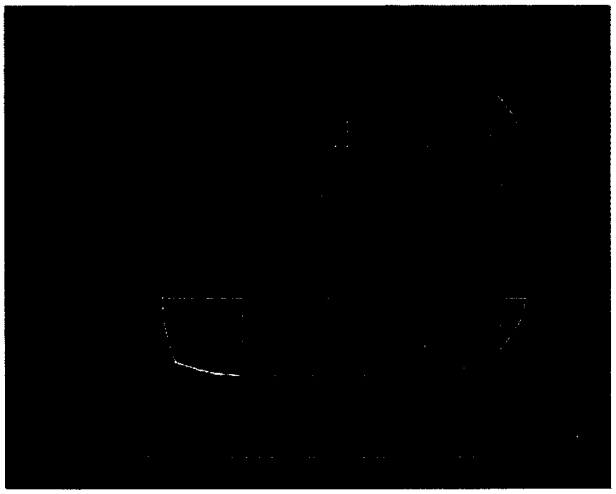

Figure 3

Voronoi diagram of a simply-connected orthogonal object.

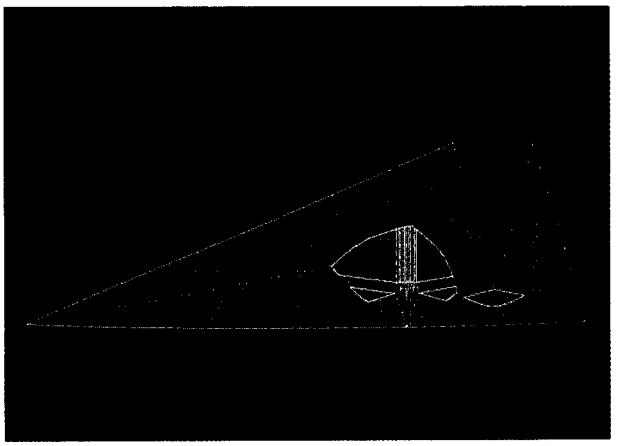

Figure 5

Voronoi diagram of a multiply-connected nonorthogonal object.

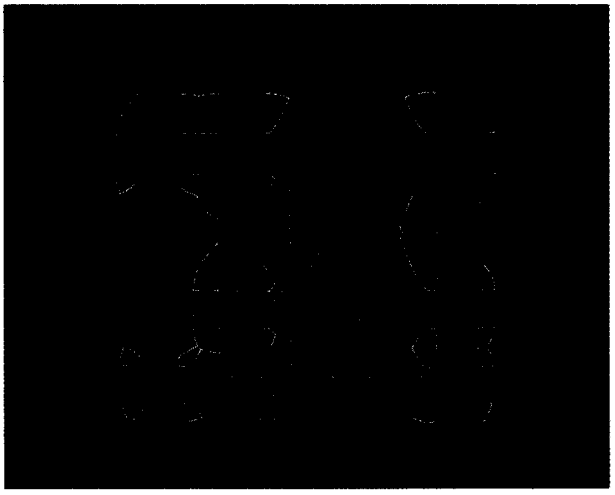

Figure 4

Voronoi diagram of a multiply-connected orthogonal object.

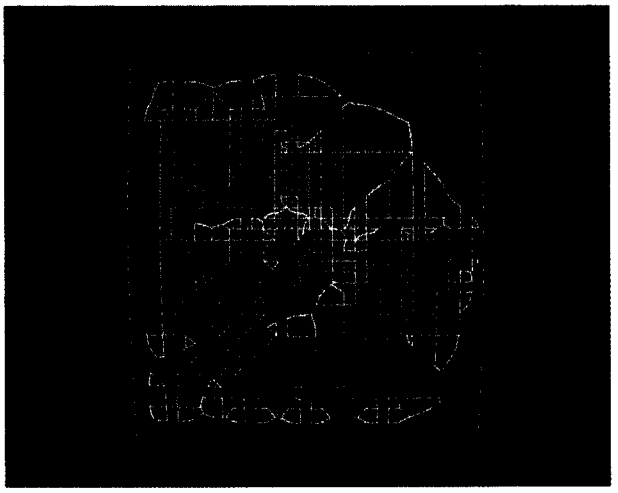

Figure 6

Voronoi diagram of a multiply-connected mixed-mode object.

Table 1 Run-time in IBM 3081 CPU seconds: N is the number of vertices (or edges) and H is the number of holes in the input polygonal domain.

N	H								
	0	*10*	*20*	*30*	*40*	*100*	*200*	*300*	*400*
114	0.37	0.43	0.49	—	—	—	—	—	—
164	0.43	0.58	0.65	0.72	0.75	—	—	—	—
424	1.11	1.16	1.43	1.58	1.72	2.34	—	—	—
1644	3.49	4.20	4.99	5.50	6.10	9.88	15.09	18.66	20.00

Figure 3 displays the Voronoi diagram of a simply-connected orthogonal polygon. In this figure parabolic (pink) Voronoi edges exist at concave vertices of the original polygon. We can associate two "widths," one in each of the two orthogonal directions, with each concave vertex. If λ is the ratio of the smaller width to the larger width at each concave vertex, the concave vertices in this figure have λ values of 1.0, 0.5, 0.33, and 0.25. For $0.5 < \lambda \leq 1.0$, there are two parabolic arcs associated with the corresponding concave vertex. For $\lambda \leq 0.5$, there is only one parabolic arc associated with the corresponding concave vertex. This property can easily be proven theoretically.

Figure 4 shows a multiply-connected polygonal domain with one hole. The hole boundary is blue, and the interior of the hole is heavily shaded in blue. In this figure we have an additional class of bisectors, namely the bisector between two vertices, which are shown in yellow. **Figure 5** shows a nonorthogonal case with a hole placed very close to one of the exterior edges. The closed merge curve is very evident in this figure as the closed set of red and pink edges surrounding the hole. Inside the closed merge curve only the external Voronoi diagram of the hole is present. Outside the merge curve, only the internal Voronoi diagram of the outer boundary is present.

Figure 6 shows a multiple hole mix of orthogonal and nonorthogonal geometries which occur in real cases. Each hole has a closed merge curve associated with it, but some of the edges of the merge curve may be shared with other holes. **Figure 7** is the complete Voronoi diagram of the outline of a human figure.

Experimental timing data support the worst-case analysis [5] while pointing out their conservative nature. There are two main variables that control the run-time: total number of vertices (N) and number of holes (H). The theoretical time complexity of the algorithm is $O(N(\log_2 N + H))$. It is always true that $H < N/3$ and for orthogonal geometries $H < N/4$. The upper bound $O(NH)$ seems to be too conservative in practice, due to the fact that holes are processed in a sorted order as demanded in [5], which often has the beneficial side effect of reducing redundant or (eventually) useless computation.

Table 1 presents some collected data. In some cases $H = N/3$, which was close to the worst case. We did not attempt to find the worst possible hole-insertion sequence. Analysis of these and other collected data shows the following:

1. For given H and values of N between 4 and 2000, the run-time variation is practically linear, with no trace of $\log_2 N$ factor (which should contribute at least one order of decimal magnitude in that range).
2. For a given N, the run-time variation is sublinear with respect to H. The reason (or speculation) for this was given earlier.

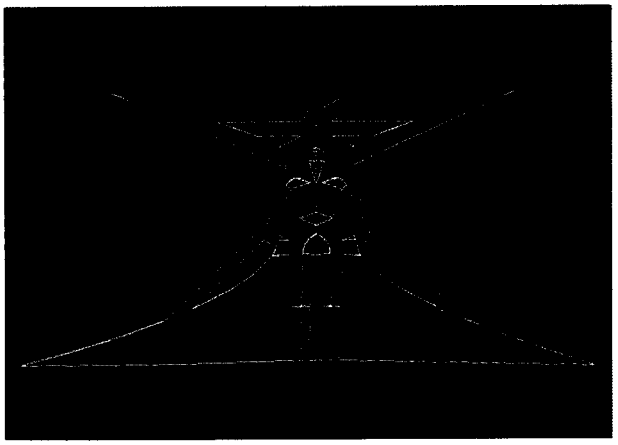

Figure 7

Internal and external Voronoi diagrams of a human figure.

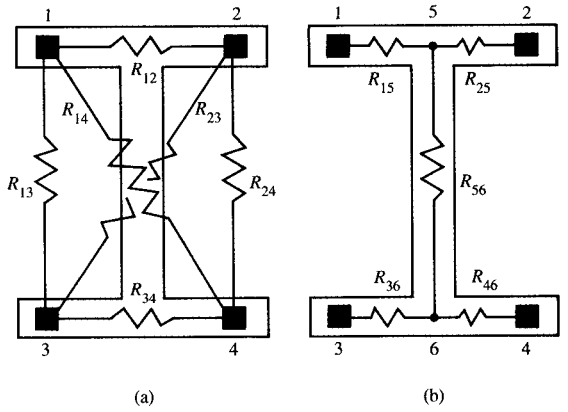

Figure 8

Two resistance networks: (a) Equivalent resistance network for a simple VLSI cell; (b) alternate equivalent resistance network for the same cell.

4. Equivalent resistance network

An important part of a dc electrical analysis of a VLSI design is to compute its equivalent resistance network from the physical description of the design. This network, representing the design, consists of ports (contacts or terminals) as the nodes and resistors as the edges of the network. Given a set $\{p_i\}$ of P ports, the complete resistance matrix will consist of elements R_{ij}, where $1 \leq i < j \leq P$. **Figure 8(a)** shows the complete network computed in this manner for a simple design. The most common approach to

Input: Voronoi diagram of multiply-connected polygonal domain.
Output: Electrical network graph.

1. Create pseudoports at junctions.
   a. Identify Voronoi edges ("branches") that are bisectors of nonadjacent edges of the input polygons.
   b. Collect groups of ordered Voronoi vertices, such that each group defines a pseudoport and the ordered vertices within the group define a polygon for the pseudoport.
2. Locate real ports within Voronoi regions. For each real port,
   a. If it coincides with a pseudoport, flag the pseudoport as a real port.
   b. Else move the real port to the nearest branch, and split the branch into two.
3. Convert the branches and the ports (pseudo and real) into edges and vertices of an electrical network graph.

Figure 9

Algorithm to extract electrical network from Voronoi diagram.

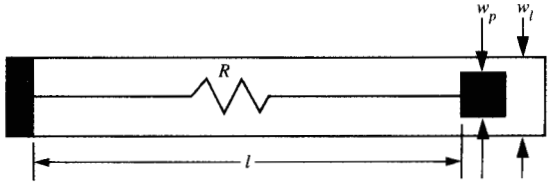

Figure 10

Branch resistance between ports.

deriving such complete resistance networks is the use of the finite-element method [8]. In this approach the domain is divided into many finite elements, and the global electrical conductivity matrix for the entire domain is obtained by assembling the elemental conductivity matrices. If M is the number of nodes in the finite-element discretization, then by proper node numbering one can obtain the global conductivity matrix having M rows and $\sqrt{M}$ bandwidth. This banded matrix can be triangulated in $O(M^2)$ time. The triangulated matrix also has a bandwidth of $\sqrt{M}$ by use of which one solution vector can be obtained in $O(M^{3/2})$ time. For P ports, $P-1$ independent solutions must be obtained in order to compute the total resistance matrix, and this takes $O(M^{3/2}P)$ time. Hence the total resistance matrix can be obtained in $O(M^2 + M^{3/2}P)$ time. This resistance matrix is then used in further electrical simulation calculations.

The above-mentioned approach is very accurate and can handle any kind of geometric design. However, the computational time can be quite large, especially when complex geometries and many ports are involved. Moreover, the complete resistance matrix is not necessary for most applications. In **Figure 8(b)** we propose an equivalent resistance network for the same design. This network is produced using the idea of Voronoi diagrams. Additional ports, called pseudoports (5 and 6), are placed in every location of geometric irregularity (corners, T-junctions, etc.) in the design. Then all real ports and pseudoports that lie along the same conducting path are connected to produce the resistance network. The introduction of pseudoports considerably simplifies the resistance network and permits the calculation of the resistive elements by the use of a simple formulation described in the next section. In addition, the solution of this network provides more information about the design, since every resistance element is related to a particular path of the design. Current densities of the conducting paths, voltage drops, and so on can easily be obtained and interpreted.

There have been attempts in the past [9] to develop algorithms that produce an equivalent resistance network given a design consisting of long rectangular shapes. They employ a pattern recognition approach with a time complexity $O(N^2)$, where N is the number of vertices. A major disadvantage of this approach is its inability to deal with multiply-connected geometries (shapes with holes) or even geometries with general irregularities such as nonorthogonal corners. Our approach exploits the power of the Voronoi diagram and solves the problem of arbitrary geometric complexity of the physical design. The time needed to obtain the solution is wholly based on the Voronoi algorithm and is independent of the geometric complexity of the design.

An algorithm for computing an electrical network graph from the Voronoi diagram is shown in **Figure 9**. Only the Voronoi edges that are bisectors of nonadjacent edges in the input polygons are considered as branches in the electrical network. These Voronoi edges are line-segments which are connected by parabolic arcs (bisectors between edges and vertices) and line-segments (bisectors between vertices). The branches are identified (step 1a in Figure 9) by examining the Voronoi edges of each of the Voronoi regions; their connectivity is also determined (step 1b) in this process. Since there are $O(N)$ Voronoi edges in a Voronoi diagram and each Voronoi edge is examined only twice in this

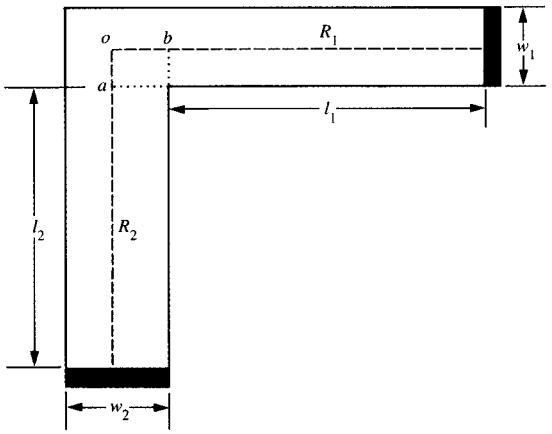

Figure 11

Branch resistances in an L-shaped design.

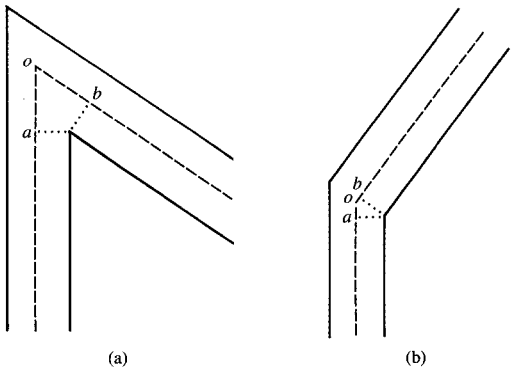

Figure 12

Corner conditions with (a) acute and (b) obtuse angles.

process, the entire step 1 takes $O(N)$ time. At the end of step 1, we have the branches and the pseudoports in our network.

Step 2 locates the real port within this network. Each real port is considered to be a point, and this point can be located within a Voronoi region in $O(\log_2 N)$ time. Since there are P ports, all of them can be located in $O(P \log_2 N)$ time. If a real port coincides with a pseudoport (within a small distance metric) the pseudoport is flagged as a real port. Otherwise, the real port is moved to the nearest point on a branch associated with the Voronoi region, and the branch is split into two branches meeting at a real port.

The final step, 3, converts the branch and port information into a network graph having edges and vertices. With proper data structuring, we can achieve this trivially at the completion of steps 1 and 2; hence the total time taken is $O(N + P \log_2 N)$ for the algorithm shown in Figure 9. Usually $P \ll N$, and the Voronoi algorithm takes $O(N(\log_2 N + H))$ time. The total time taken for the entire process is usually $O(N(\log_2 N + H))$.

In addition to the network graph information, we also need the branch resistance values for the electrical network simulation. We address this in the next section.

5. Branch resistances

The resistance R of a long rectangular conductor of length l and width w is simply represented by $R = (l/w)R_{\square}$, where $R_{\square}$ is the square resistance. This relation assumes that the two ends of the conductor are completely covered by existing ports. As the width of a port decreases in relation to the width of the conductor, its resistance increases. **Table 2,** which was obtained through numerous runs of a finite-element program, gives a correction factor C as a function of the ratio of the port width w_p to the line width w_l. Thus, the resistance of the line shown in **Figure 10** would be $R = \{(l/w_l) + C\}R_{\square}$, where C is the correction factor for w_p/w_l.

Table 2 Correction factor due to ports as a function of width ratio.

w_p/w_l	C
1.0	0.0000
0.9	0.0028
0.8	0.0083
0.7	0.0170
0.6	0.0294
0.5	0.0455
0.4	0.0667
0.3	0.0932
0.2	0.1272
0.1	0.1737

The effect of a corner on two branch resistances is considered next. It was observed through numerous runs of a finite-element program that the resistances of two branches shown in **Figure 11** were affected only by the ratio of the lengths l_{oa} and l_{ob} to the sum of the widths. Here a and b are the feet of the perpendiculars from the concave corner to the axes of the two branches, l_{oa} is the distance between o and a, and l_{ob} is the distance between o and b. Note that a and b are also Voronoi vertices. l_{oa} and l_{ob} are not always half of the widths of the branches because they depend on the angle between the branches. For example, in **Figure 12(a)** the two branches meet at an acute angle, and l_{oa} and l_{ob} are larger than the semi-widths. Similarly, in **Figure 12(b)** the two

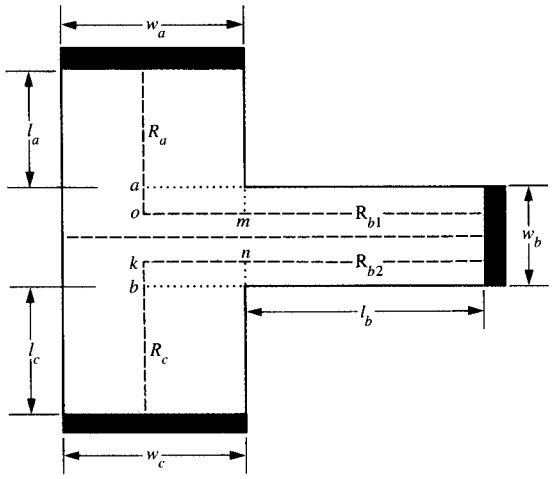

Figure 13

Branch resistances in a T-shaped design.

Table 3 Correction factor due to corner as a function of width ratio.

w_1/w_2	C
1.0	0.568
0.9	0.570
0.8	0.576
0.7	0.589
0.6	0.612
0.5	0.652
0.4	0.713
0.3	0.823
0.2	0.996
0.1	1.362

branches meet at an obtuse angle, resulting in smaller values for l_{oa} and l_{ob} than the semi-widths. **Table 3** gives the correction factor C for the branch resistances as a function of the ratio of the branch widths. The two resistances are calculated as

$$R_1 = \{(l_1/w_1) + 2Cl_{ob}/(w_1 + w_2)\}R_\square,$$

$$R_2 = \{(l_2/w_2) + 2Cl_{oa}/(w_1 + w_2)\}R_\square,$$

where C is the correction factor for w_1/w_2 from Table 3. These equations are also applicable for branches that meet at angles other than 90°.

Using Tables 2 and 3, we can derive all branch resistances after the introduction of pseudoports. As an example, consider a T-shaped junction, as shown in **Figure 13.** We can calculate four resistances as

$$R_a = \{(l_a/w_a) + 2Cl_{oa}/(w_a + 0.5w_b)\}R_\square,$$

$$R_c = \{(l_c/w_c) + 2Cl_{kb}/(w_c + 0.5w_b)\}R_\square,$$

$$R_{b1} = \{(2l_b/w_b) + 2Cl_{om}/(w_a + 0.5w_b)\}R_\square,$$

$$R_{b2} = \{(2l_b/w_b) + 2Cl_{kn}/(w_c + 0.5w_b)\}R_\square.$$

The final equation for the resistance of the horizontal branch is derived by adding R_{b1} and R_{b2} in parallel, giving

$$R_b = R_{b1}R_{b2}/(R_{b1} + R_{b2}).$$

6. Conclusions

Voronoi diagrams are practical and valuable tools for performing a variety of geometric tasks. The algorithms proposed by Lee [1] and by Srinivasan and Nackman [5] have proven to have industrial strength and practicality. The run-times of these algorithms are quite satisfactory.

A Voronoi diagram of VLSI geometry can be used to obtain extremely accurate information about the nature of the layout and the terrain changes: e.g., to reduce a typical VLSI cell layout to its equivalent resistance network. The accuracy of this approach for designs with many thin rectangular segments compares well with that of finite-element analysis, while the computational time and the main memory requirements are reduced considerably.

Our experience with Voronoi diagrams indicates that they should be integrated into the data model of any CAD/CAM database. This will require development and implementation of Voronoi diagram algorithms for more general objects. Once the Voronoi diagrams of all the objects are computed, it should be possible to answer just about any geometric query in linear time with respect to the number of Voronoi edges.

7. Acknowledgments

We are grateful to Vijay Srinivasan and Lee R. Nackman for many helpful technical suggestions in implementing the Voronoi diagram algorithm. During the development of the program, many useful technical comments were made by our colleagues William C. Bakker and L. William Dewey III. We also acknowledge Steven E. Washburn and John A. Brunner for their help in testing the modules and integrating them into an engineering design system.

References

1. D. T. Lee, "Medial Axis Transformation of a Planar Shape," *IEEE Trans. Pattern Anal. & Machine Intell.* **PAMI-4,** No. 4, 363–369 (July 1982).
2. D. G. Kirkpatrick, "Efficient Computation of Continuous Skeletons," *IEEE 20th Annual Symposium on Foundations of Computer Science,* 1979, pp. 18–27.
3. D. T. Lee and R. L. Drysdale, "Generalization of Voronoi Diagrams in the Plane," *SIAM J. Computing* **10,** No. 1, 73–87 (February 1981).
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. V. Srinivasan and L. R. Nackman, "Voronoi Diagram for Multiply-Connected Polygonal Domains I: Algorithm," *IBM J. Res. Develop.* **31,** No. 3, 361–372 (May 1987, this issue).
6. B. G. Baumgart, "A Polyhedral Representation for Computer Vision," *Proc. 1975 Nat. Comput. Sci. Conf.* (AFIPS Press, Arlington, VA) **44,** 589–596 (1975).
7. F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
8. C. M. Sakkas, "Potential Distribution and Multi-Terminal DC Resistance Computations for LSI Technology," *IBM J. Res. Develop.* **23,** No. 6, 640–651 (November 1979).
9. P. J. Russel, "A DC Resistance Calculator," *Technical Report TR.12.121*, IBM United Kingdom Laboratory, Hursley, England, April 1974.

Received October 15, 1986; accepted for publication January 9, 1987

Siavash N. Meshkat *IBM Research Division, 650 Harry Road, San Jose, California 95120.* Mr. Meshkat is currently a staff programmer, in the area of mechanical design automation. From June 1986 to April 1987, he was an IMB assignee at the Rolm Corporation, in Santa Clara, California. Previously, he was a staff programmer in the Engineering Design System organization of the IBM General Technology Division, East Fishkill, New York. He joined the General Technology Division in East Fishkill in 1981, after receiving his M.S. in computer science from the University of Missouri, Columbia. He received his B.S. in mathematics (1981) and B.S. in computer science (1980) from Missouri Western State College, St. Joseph, Missouri. Mr. Meshkat is a member of the Association for Computing Machinery and the Computer Society of the Institute of Electrical and Electronics Engineers.

Constantine M. Sakkas *IBM General Technology Division, East Fishkill facility, Route 52, Hopewell Junction, New York 12533.* Mr. Sakkas received a B.S. in 1963 and an M.S. in 1965, both in mechanical engineering, from the University of Missouri, Columbia, where he also served as a teaching assistant instructor in applied mathematics from 1963 to 1965. He joined IBM in 1965 in a scientific computation department in Poughkeepsie, New York, where he worked on the development of software tools in the area of thermal, stress, and electrical analysis. In 1970, he joined an engineering design system area where as a member of a four-man team of an advanced design techniques department he participated in the development of the FET automatic placement program. In 1982 he received an IBM Outstanding Innovation Award for the formulation and development of an electrical analysis and checking software tool for VLSI technologies. In 1986 he was promoted to senior engineer. He is currently working on the development of packaging analysis tools as a member of a custom layout analysis department. Mr. Sakkas is a member of Pi Tau Sigma and Tau Beta Pi.