Construction of the Voronoi Diagram for "One Million" Generators in Single-Precision Arithmetic

KOKICHI SUGIHARA, MEMBER, IEEE, AND MASAO IRI, FELLOW, IEEE

Invited Paper

The paper presents a numerically stable algorithm for constructing Voronoi diagrams in the plane. In this algorithm higher priority is placed on the topological structure than on numerical values, so that, however large the numerical errors, the algorithm will never come across topological inconsistency and thus can always complete its task. The behavior of the algorithm is shown with examples, including one for as many as 10^6 generators.

I. INTRODUCTION

The Voronoi diagram is one of the fundamental concepts in computational geometry with many applications [2], [5], [9], [11], and a number of "efficient" algorithms for its construction have been proposed [1], [6], [8], [14], [15], [18] (see also [4], [10], and [16]). However, those algorithms are usually designed on the (implicit) assumption that there is no numerical error in the course of computation. In real computation, on the other hand, numerical errors cannot be avoided completely if we use an ordinary floating-point arithmetic, so that it is difficult to always judge correctly, for example, whether a point is inside, outside, or exactly on a circle. Misjudgment on geometric relations often results in topological inconsistency and causes a "theoretically correct algorithm" to fail. Thus, there is an insurmountable gap between "theoretically correct" algorithms and "practically valid" computer programs. The same gap is pointed out for many problems in computational geometry [3], [7]. We sometimes hear it said that a simple divide-and-conquer algorithm can construct Voronoi diagrams with at best 1000 generators in single-precision arithmetic and those with 3000 generators in double-precision arithmetic. (We know an incremental algorithm [15] has constructed diagrams with as many as 30 000 generators in double-precision arithmetic.) The purpose of this paper is to present a new algorithm that is both theoretically correct and practically robust in a certain new sense and thus to fill the gap between theoretical algorithms and practical programming.

Among many algorithms for Voronoi diagram construction, a rather sophisticated implementation of the incremental-type algorithm [15] is most practical because, on the average, it runs in $O(n)$ time for n generators, while the other algorithms run in $O(n \log n)$ time both in the worst case and in the average [1], [6], [14], [18] (see also [15]). (Recently it was pointed out that a variant of the divide-and-conquer algorithm runs in $O(n)$ average time [13].) Here we redesign the incremental-type algorithm so as to make it work well in finite-precision environment.

Our basic idea [12], [19], [20] is the following. We assume that numerical errors may arise in the course of computation, so that no judgment based on numerical computation is absolutely reliable. On this assumption, it is obviously impossible to always construct the Voronoi diagram correctly. Hence, we change our goal; we try to construct a diagram which shares some topological properties with the true Voronoi diagram. For this purpose the basic structure of the algorithm is designed only in terms of combinatorial computation, and numerical results are used as lower-priority information for selecting a more probable structure of the Voronoi diagram from among all possible structures.

The algorithm thus designed does not come across any topological inconsistency; it always carries out its task and gives some output. Moreover, the output "converges" to the true Voronoi diagram as the precision in computation becomes higher (the meaning of "converge" will be discussed later).

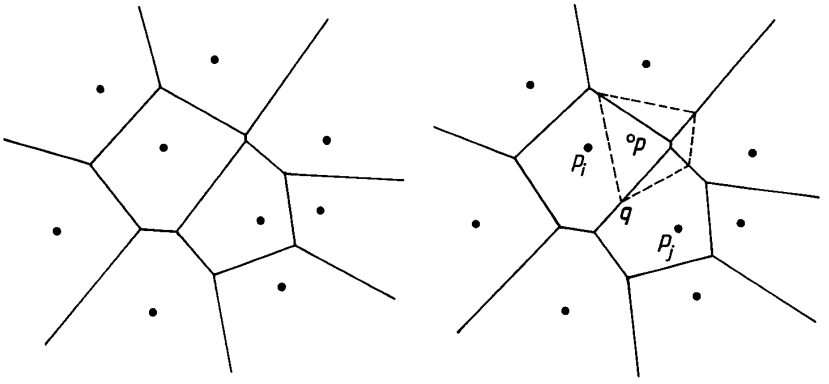
We consider topological consistency as being more fundamental than numerical results. However, this does not mean that numerical computation is less important. We tuned the way of computing numerical values, and as a re-

Manuscript received March 6, 1990; revised April 16, 1992. This work was supported by a Grant in Aid for Scientific Research (01550279) from the Ministry of Education, Science and Culture of Japan.

The authors are with the Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, 7–3–1 Hongo, Bunkyo-ku, Tokyo 113, Japan.

IEEE Log Number 9204255.

0018–9219/92$03.00 © 1992 IEEE



(a) (b)

Fig. 1. Voronoi diagram and the incremental-type method for constructing it: (a) example of a Voronoi diagram; (b) addition of a new generator.

sult our experimental computer program [21] can construct, for example, a Voronoi diagram for as many as 1 000 000 generators in single precision.

This paper is a preliminary report of our new topology oriented approach to the design of numerically robust geometric algorithms; a detailed discussion will be given in [22].

II. Voronoi Diagram and the Incremental Construction

For two points p and q, let $d(p, q)$ denote the Euclidean distance between p and q. For a finite set $P = \{p_1, \cdots, p_n\}$ of points in the plane, region $R(p_i)$ is defined by

$$R(p_i) = \{p \mid d(p, p_i) < d(p, p_j) \text{ for any } j(\neq i)\}$$

and is called the Voronoi region of p_i. The Voronoi regions $R(p_1), \cdots, R(p_n)$ make a partition of the plane, and this partition is called the Voronoi diagram for P. An element of P is called a generator of the Voronoi diagram. A common boundary of two Voronoi regions is called a Voronoi edge. A point at which boundaries of three or more Voronoi regions meet is called a Voronoi point.

An example of a Voronoi diagram is given in Fig. 1(a), where the generators are represented by the blobs.

The following properties are direct consequences of the definitions of the Voronoi diagram.

(P1) A Voronoi edge is part of the perpendicular bisector of the two generators associated with the right and the left Voronoi regions.

(P2) A Voronoi point is the circumcenter, i.e., the center of the circle circumscribing the triangle with the three generators (whose regions are incident to the point) as the vertices.

(P3) Voronoi region $R(p_i)$ is a convex polygon containing the corresponding generator p_i in its interior.

There are two types of Voronoi edges. One is a line segment connecting two Voronoi points, and the other is a half line emanating from a Voronoi point. We call the former an ordinary Voronoi edge and the latter an infinite Voronoi edge. (Strictly speaking, there is another type of Voronoi edges; when all the generators align on a line, the Voronoi edges are parallel lines extending infinitely in both directions. However, we need not consider such a special case because in our algorithm we augment the generator set by adding three additional generators that are not collinear; see subsection III-A).

A Voronoi region is bounded if the boundary of the region consists of ordinary Voronoi edges alone, while it is unbounded if its boundary contains an infinite Voronoi edge. The unbounded Voronoi region is characterized in the following manner.

(P4) Voronoi region $R(p_i)$ is unbounded if and only if p_i is on the boundary of the convex hull of P.

Properties (P1) through (P4) reflect both a topological and a metrical nature, while some of the properties can be stated in a purely combinatorial-topological (which we call topological hereafter) manner. For example, we have the following properties.

(P5) A Voronoi diagram partitions the plane into as many regions as the generators.

(P6) Two Voronoi regions do not share two or more edges as a common part of their boundaries.

Property (P6) comes from the convexity of a Voronoi region.

The incremental-type algorithm for constructing the Voronoi diagram starts with a simple Voronoi diagram for two or three generators, and modifies it step by step by adding new generators one by one.

One step of the algorithm is illustrated in Fig. 1. Let us assume that the Voronoi diagram shown in (a) has been obtained, and that point p, represented by a hollow circle in (b), is to be added as a new generator. Then, the Voronoi region of the new generator p is constructed in the following way. First, we find the generator, say p_i, whose region contains p and draw the perpendicular bisector of p and

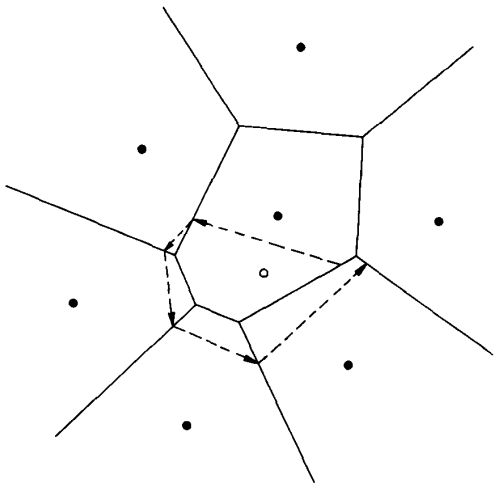


Fig. 2. Topological inconsistency arising from numerical errors in the incremental construction of the Voronoi diagram.

p_i; in (b) the bisector is represented by a broken line. The bisector crosses the boundary of $R(p_i)$ at two points. Let one of them be q. At q the bisector enters the neighboring region $R(p_j)$. Next, we draw the bisector of p and p_j to find the point of intersection (other than q) of the bisector with the boundary of $R(p_j)$. In this way, we construct a sequence of the bisectors between p and the neighboring generators until we return to the boundary of the starting region $R(p_i)$. Removing the points and edges enclosed by the closed sequence of part of the bisectors, we finally obtain the Voronoi region of the new generator p.

A sophisticated data structure with a quarternary tree and with buckets enables us to find the starting generator p_i in constant expected time, and to keep the average number of edges on the boundary of the new Voronoi region constant [15]. Hence, the incremental method carries out the addition of one generator in constant expected time, so that it constructs the Voronoi diagram for n generators in O(n) expected time. This expected time complexity is theoretically ensured for randomly distributed generators, and empirically shown for a wide class of distributions [15].

The incremental method is simple in principle, and it is usually said that this method is also robust against numerical errors; indeed, a computer program based on the incremental method has been used for a number of applications [8], [11], [15].

However, the incremental-type algorithm as well as other algorithms is unstable when degeneracy takes place. An example of a situation in which the conventional incremental-type algorithm fails is shown in Fig. 2, where the perpendicular bisectors between the new generator and the nearest old generator pass near a Voronoi point, and the boundary of the Voronoi region of the new generator does not form a closed cycle because of numerical error.

Hence, the avoidance of inconsistency arising from numerical errors is an important problem for practical implementation of the algorithm.

III. Design of a Robust Algorithm

A. Placing the Highest Priority on Topological Consistency

A Voronoi diagram can be regarded as a planar graph embedded in a plane. Let G_i be the embedded graph associated with the Voronoi diagram for i generators $p_1, p_2, \cdots, p_i$. From a topological point of view, the addition of a new generator p_l to the Voronoi diagram for $l-1$ generators $p_1, p_2, \cdots, p_{l-1}$ can be considered the task for changing G_{l-1} to G_l. This task is done by the next procedure.

Procedure A

A1. Select a subset, say T, of the vertex set of G_{l-1}.
A2. For every edge connecting a vertex in T with a vertex not in T, generate a new vertex on it and thus divide the edge into two edges.
A3. Generate new edges connecting the vertices generated in A2 in such a way that the new edges form a cycle that encloses the vertices in T and them only.
A4. Remove the vertices in T and the edges incident to them (and regard the interior of the cycle as the Voronoi region of p_l), and let the resulting embedded graph be G_l.

An example of the behavior of this procedure is illustrated in Fig. 3(a). Suppose that the solid lines represent a portion of the embedded graph G_{l-1} and that the four solid circles represent the vertices in T chosen in step A1. Then, the six vertices represented by hollow circles are generated in step A2, the cycle represented by the broken lines is generated in step A3, and the substructure enclosed by this cycle is removed in step A4.

Note that Procedure A is described in purely combinatorial terms, so that this procedure is not affected by numerical errors. However, there is an ambiguity in the choice of T in step A1. Next, we consider what conditions should be satisfied by T in order for Procedure A to be the correct procedure for constructing the Voronoi diagram.

Let us consider a triangle that is large enough to include all the generators and regard the three vertices of this triangle as the additional generators. We renumber the generators in such a way that p_1, p_2, and p_3 are the additional generators and $p_4, p_5, \cdots, p_n$ the original generators (now, n is the number of the original generators plus 3), and try to construct the Voronoi diagram for $P = \{p_1, p_2, \cdots, p_n\}$.

The Voronoi diagram for the three generators p_1, p_2, and p_3 consists of three infinite edges, as shown in Fig. 4(a). To represent the topological structure of this Voronoi diagram we consider the embedded graph G_3 shown in Fig. 4(b), where we introduce a sufficiently large closed curve and consider that the infinite Voronoi edges have their terminal points on this closed curve, as represented by the small solid triangles. With this convention, any Voronoi region is explicitly represented by a cycle of the embedded graph. We start with G_3, and add the other generators $p_4, p_5, \cdots, p_n$ one by one.

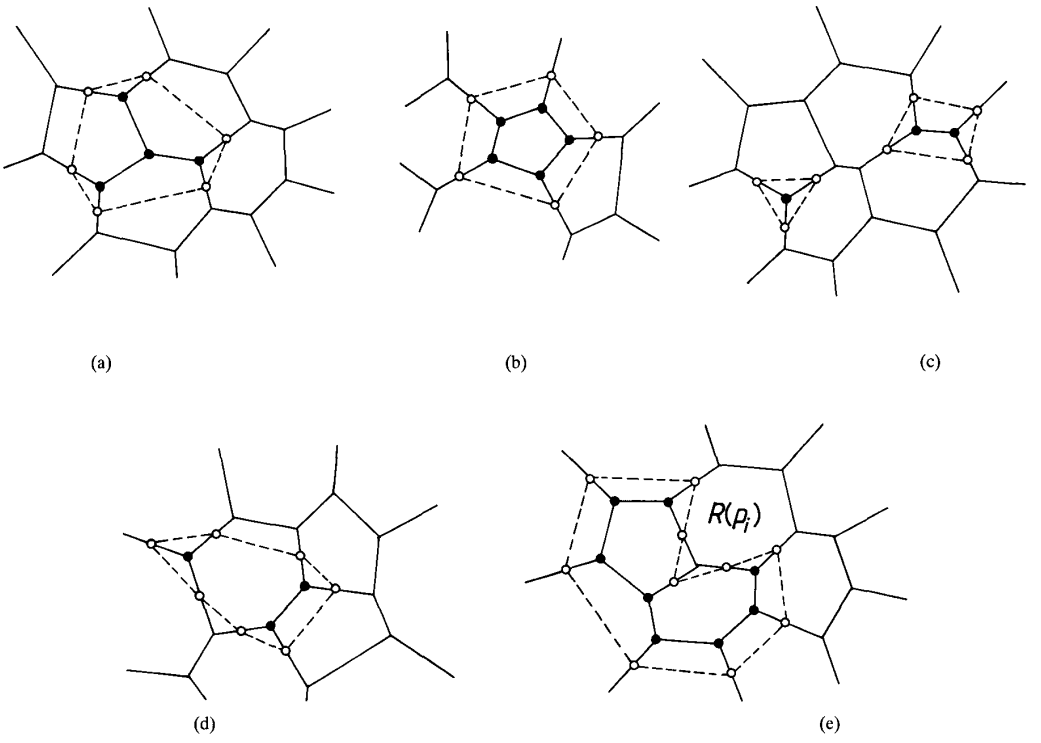


Fig. 3. Topological aspect of the addition of a new generator.

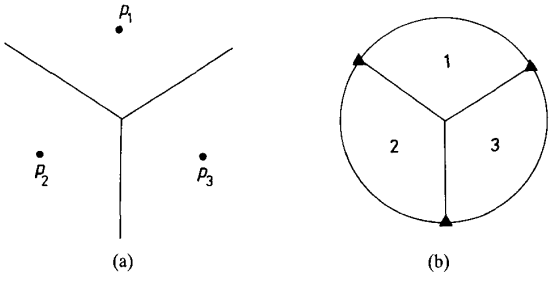


Fig. 4. Voronoi diagram for three generators and the corresponding embedded graph.

Note that p_1, p_2, and p_3 are the only generators that are on the boundary of the convex hull of P. Hence, it follows from property (P4) that in adding a new generator the Voronoi region of the new generator is always bounded; an infinite edge is never generated.

For any subset T of the vertex set of G_{l-1}, let $G_{l-1}(T)$ denote the subgraph of G_{l-1} that consists of the vertices in T and the edges connecting two vertices in T. The subgraph $G_{l-1}(T)$ is also regarded as an embedded graph. Let us call an elementary cycle C of G_{l-1} primary if C encloses no vertex or edge in its interior (a primary cycle corresponds to the boundary of a Voronoi region), and let $C \cap G_{l-1}(T)$ denote the subgraph of G_{l-1} consisting of vertices and edges belonging to both C and $G_{l-1}(T)$.

If Procedure A corresponds to the correct addition of the new generator p_l, the subset T should satisfy at least the following conditions.

(C1) T is nonempty.
(C2) $G_{l-1}(T)$ is a tree.
(C3) For any primary cycle C of G_{l-1}, $C \cap G_{l-1}(T)$ is connected.

(C1) is obvious because the new generator p_l should have its own nonempty Voronoi region. If $G_{l-1}(T)$ has a cycle, the Voronoi region of an old generator is removed completely, as shown in Fig. 3(b), which is a contradiction. If $G_{l-1}(T)$ is disconnected, then either the Voronoi region of p_l is disconnected, as shown in (c), or an old Voronoi region becomes disconnected, as shown in (d). Thus, $G_{l-1}(T)$ is a tree. If $C \cap G_{l-1}(T)$ is not connected as in (e) (where C is the cycle forming the boundary of the region $R(p_i)$), then the Voronoi regions $R(p_i)$ and $R(p_l)$ have two or more edges in common, which contradicts property (P6).

Now we have seen that in step A1 of Procedure A the subset T should be chosen in such a way that conditions (C1) through (C3) are satisfied. Note that (C1), (C2), and (C3) are all combinatorial conditions; hence we can check them without worrying about numerical errors.

B. Use of Numerical Values as Lower-Priority Information

In the previous subsection we constructed the basic part of the incremental algorithm only in terms of combinatorial tasks. However, there is an ambiguity in the choice of T. To clear up this ambiguity we employ numerical computation.

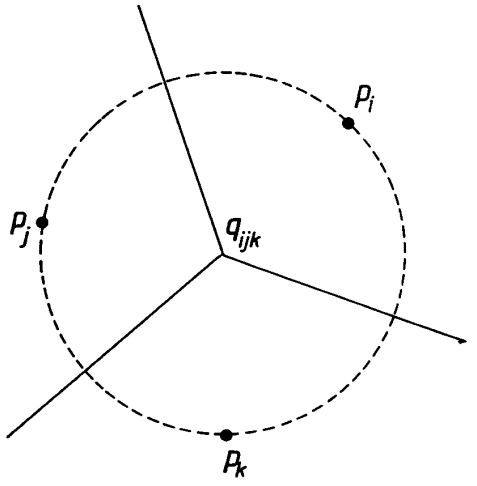


Fig. 5. Voronoi point and the surrounding generators.

In order to construct the correct Voronoi diagram, the subset T in Procedure A should consist of those Voronoi points of the Voronoi diagram for $p_1, p_2, \cdots, p_{l-1}$ which lie in the interior of the Voronoi region of p_l. If there is no numerical error, these Voronoi points are discerned in the following way.

Let (x_i, y_i) be the coordinates of generator p_i $(i = 1, 2, \cdots, n)$ and (x, y) be those of any point p with respect to a right-handed Cartesian coordinate system. As shown in Fig. 5, let q_{ijk} denote the Voronoi point incident to three Voronoi regions $R(p_i)$, $R(p_j)$, and $R(p_k)$ which enclose the Voronoi point counterclockwise in this order. Then, q_{ijk} is inside the Voronoi region of the new generator p_l if and only if p_l is inside the circle that passes through p_i, p_j, and p_k.

Let us define $H(p_i, p_j, p_k, p)$ by

$$H(p_i,p_j,p_k,p) = \begin{vmatrix} 1 & x_i & y_i & (x_i^2+y_i^2)/2 \\ 1 & x_j & y_j & (x_j^2+y_j^2)/2 \\ 1 & x_k & y_k & (x_k^2+y_k^2)/2 \\ 1 & x & y & (x^2+y^2)/2 \end{vmatrix}. \quad (1)$$

With p as a variable point, the circle passing through p_i, p_j, and p_k is represented by the equation $H(p_i, p_j, p_k, p) = 0$. Point p is inside this circle if $H(p_i, p_j, p_k, p) < 0$ and outside if $H(p_i, p_j, p_k, p) > 0$. Hence, in the error-free world, q_{ijk} belongs to T if $H(p_i, p_j, p_k, p_l) < 0$, q_{ijk} does not belong to T if $H(p_i, p_j, p_k, p_l) > 0$, and some exceptional treatment is required if $H(p_i, p_j, p_k, p_l) = 0$.

However, we assume that numerical errors may have taken place in the course of computation, so that the sign of $H(p_i, p_j, p_k, p_l)$ is not necessarily determined correctly. Therefore, in the choice of T we place the highest priority on the conditions (C1) through (C3); we use the numerical values of $H(p_i, p_j, p_k, p_l)$ only to select the most promising set as T provided that (C1), (C2), and (C3) are fulfilled. For this purpose we employ the next procedure to find the subset T for the addition of the new generator g_l.

Procedure B

B1. Find the generator, say p_i, that is closest to p_l, and, among the Voronoi points q_{ijk} on the boundary of the Voronoi region $R(p_i)$, find the one that gives the smallest value of $H(p_i, p_j, p_k, p_l)$. Let T be the singleton consisting of this Voronoi point.

B2. Repeat B2.1 until T cannot be augmented any more.

B2.1. For each Voronoi point q_{ijk} that is connected by a Voronoi edge to an element of T, add q_{ijk} to T if $H(p_i, p_j, p_k, p_l) < 0$ and if the resultant T satisfies conditions (C2) and (C3).

Note that, if there is no degeneracy and no numerical error, Procedure B constructs as T the set of Voronoi points that should be deleted in the addition of the new generator p_l. Even if numerical errors take place, the set T constructed by Procedure B satisfies conditions (C1) through (C3), because at least one element is chosen in step B1, and (C2) and (C3) are kept satisfied while T is augmented in step B2.

We use Procedure B as step A1 of Procedure A. That is, given the set $P = \{p_1, p_2, \cdots, p_n\}$ of generators, we start with the embedded graph G_3 in Fig. 4(b), and construct G_4, $G_5, \cdots$ by Procedure A, in which Procedure B is employed for step A1, until G_n is obtained. However large the numerical errors may be, our algorithm terminates with some embedded graph G_n, which is "topologically consistent" in the sense that G_n is a planar graph partitioning the plane into n regions and no two regions have two or more edges in common.

Conditions (C2) and (C3) can be checked efficiently in the following manner. We assign three labels, "in," "out," and "undecided," to the Voronoi points; "in" is assigned to the Voronoi points that are added to T, "out" to the Voronoi points that are judged not to be added to T, and "undecided" to the others. Also, we assign two labels, "incident" and "nonincident," to the Voronoi regions; "incident" to the Voronoi regions whose boundary has a vertex in T, and "nonincident" to the other Voronoi regions. In step B2.1, q_{ijk} is chosen among the Voronoi points that are labeled "undecided" and that are adjacent to a vertex in T. Then, $G_{l-1}(T \cup \{q_{ijk}\})$ is a tree if and only if

(C4) q_{ijk} is not adjacent to two or more "in" Voronoi points.

Also, for any primary cycle C of G_{l-1}, the subgraph $C \cap (G_{l-1}(T \cup \{q_{ijk}\})$ is connected if and only if the following condition is satisfied:

(C5) For any "incident" Voronoi region having q_{ijk} on its boundary, q_{ijk} is adjacent to an "in" Voronoi point on this boundary.

Hence, conditions (C2) and (C3) can be examined locally by checking the labels of the Voronoi regions incident to q_{ijk} and those of the Voronoi points adjacent to q_{ijk}.

For the initial embedded graph G_3, we assign "out" labels to the three vertices on the outmost closed curve (i.e., the

three vertices represented by solid triangles in Fig. 4(b)), and these labels are permanently fixed. This is because the Voronoi region of a new generator is always bounded, so that these three vertices should not belong to T. At each start of Procedure B, all the other Voronoi points are labeled "undecided," and all the Voronoi regions are labeled "nonincident." Each time a Voronoi point is added to T, its label is changed from "undecided" to "in" and the labels of Voronoi regions incident to the Voronoi point, if they are "nonincident," are changed to "incident." Similarly, each time a Voronoi point is judged not to be added to T, its label is changed from "undecided" to "out." During Procedure B the Voronoi points and the Voronoi regions whose labels are changed are listed, and at the end of Procedure B these labels are cleared (that is, "in" and "out" are changed to "undecided," and "incident" is changed to "nonincident"). In this way step B2 can be executed in time proportional to the size of the substructure that is deleted for the addition of the new generator.

C. Tuning the Way of Numerical Computation

We place higher priority on topological consistency than on numerical values, but this does not mean that numerical computation is less important. In our algorithm, the most important numerical computation is that of $H(p_i, p_j, p_k, p_l)$. Hence, we search for ways of computing this value with fewer numerical errors.

A typical numerical method for computing the determinant of a matrix is to employ Gaussian elimination. We basically follow this. However, we also consider the fact that the algorithm evaluates the determinants of many matrices which differ only partially from each other. That is, $H(p_i, p_j, p_k, p_l)$ is computed for fixed i, j, and k (forming a Voronoi point) and for many values of l. So it is also important to reuse the results of partial computations whenever possible. From these two points of view, we consider two methods for computing $H(p_i, p_j, p_k, p_l)$.

First, $H(p_i, p_j, p_k, p_l)$ can be rewritten in the following way:

$$\begin{aligned} H(p_i,p_j,p_k,p_l) &= \begin{vmatrix} 1 & x_i & y_i & (x_i^2+y_i^2)/2 \\ 1 & x_j & y_j & (x_j^2+y_j^2)/2 \\ 1 & x_k & y_k & (x_k^2+y_k^2)/2 \\ 0 & x_l-x_m & y_l-y_m & ((x_l+x_m)(x_l-x_m)+(y_l+y_m)(y_l-y_m))/2 \end{vmatrix} \\ &= H^2_{ijk}(x_l-x_m) - H^3_{ijk}(y_l-y_m) \\ &\qquad + \frac{1}{2}H^4_{ijk}((x_l+x_m)(x_l-x_m)+(y_l+y_m)(y_l-y_m)), \end{aligned} \tag{2}$$

where m is any one of i or j or k, and H^s_{ijk} $(s = 2, 3, 4)$ is the determinant of the 3×3 submatrix obtained from the upper 3×4 matrix in (2) by deleting the sth column:

$$\begin{aligned} H^2_{ijk} &= \begin{vmatrix} 1 & y_i & (x_i^2+y_i^2)/2 \\ 1 & y_j & (x_j^2+y_j^2)/2 \\ 1 & y_k & (x_k^2+y_k^2)/2 \end{vmatrix} \\ &= \begin{vmatrix} y_j-y_i & ((x_j+x_i)(x_j-x_i)+(y_j+y_i)(y_j-y_i))/2 \\ y_k-y_i & ((x_k+x_i)(x_k-x_i)+(y_k+y_i)(y_k-y_i))/2 \end{vmatrix}, \end{aligned} \tag{3}$$

$$\begin{aligned} H^3_{ijk} &= \begin{vmatrix} 1 & x_i & (x_i^2+y_i^2)/2 \\ 1 & x_j & (x_j^2+y_j^2)/2 \\ 1 & x_k & (x_k^2+y_k^2)/2 \end{vmatrix} \\ &= \begin{vmatrix} x_j-x_i & ((x_j+x_i)(x_j-x_i)+(y_j+y_i)(y_j-y_i))/2 \\ x_k-x_i & ((x_k+x_i)(x_k-x_i)+(y_k+y_i)(y_k-y_i))/2 \end{vmatrix}, \end{aligned} \tag{4}$$

$$\begin{aligned} H^4_{ijk} &= \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{vmatrix} \\ &= \begin{vmatrix} x_j-x_i & y_j-y_i \\ x_k-x_i & y_k-y_i \end{vmatrix}. \end{aligned} \tag{5}$$

From this expression, we get the following method for computing $H(p_i, p_j, p_k, p_l)$.

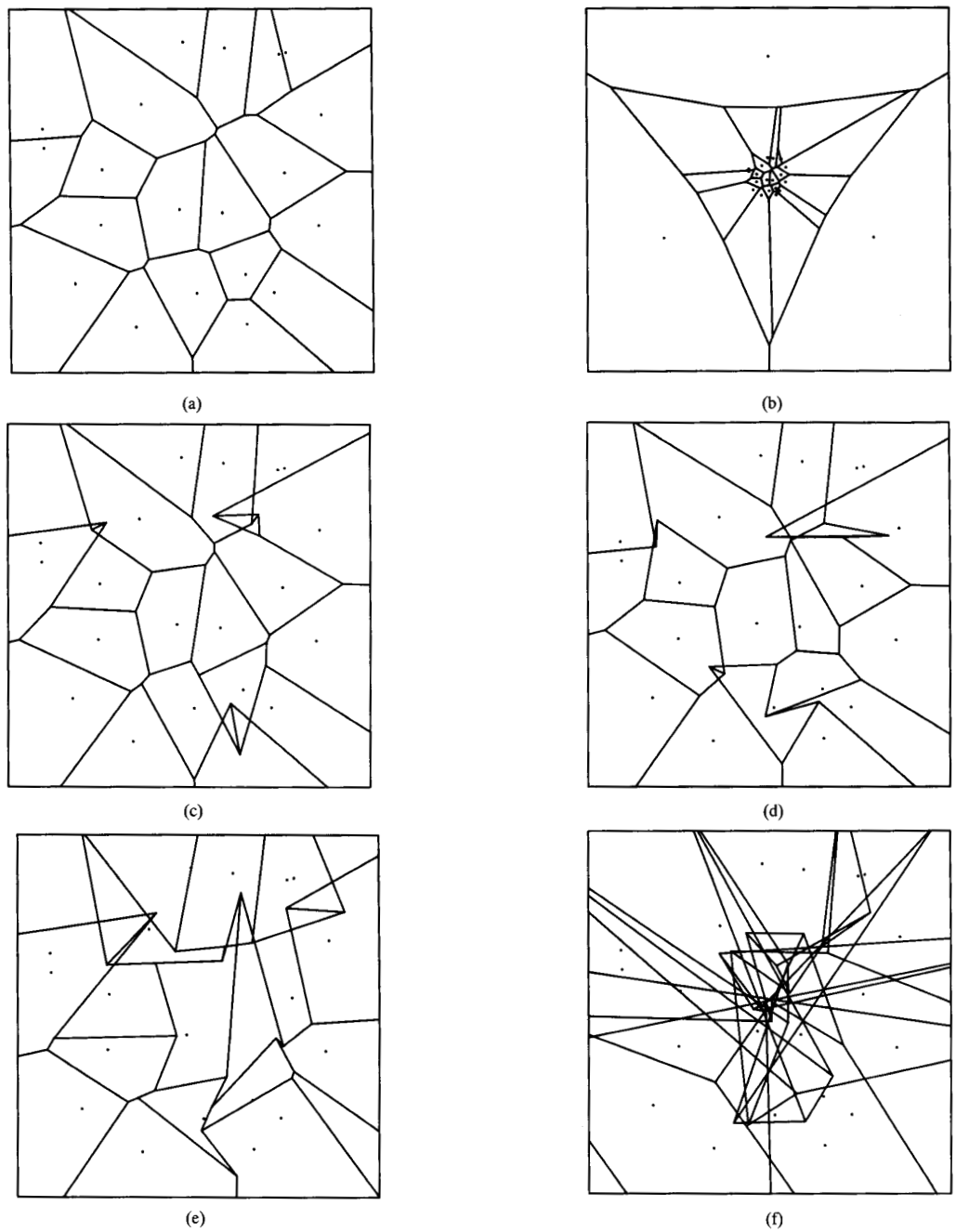


Fig. 6. Output of the program for 20 generators placed at random in the unit square: (a) output given in single-precision computation; (b) wider view of the same output; (c) output obtained when random numbers generated uniformly in $[-r, r)$ were added to the values of $H(p_i, p_j, p_k, p_l)$ for $r = 2^{-9}$; (d) $r = 2^{-8}$; (e) $r = 2^{-7}$; (f) output obtained when the values of $H(p_i, p_j, p_k, p_l)$ were replaced by random numbers.

Method 1: Each time we generate a new Voronoi point q_{ijk}, we compute H^2_{ijk}, H^3_{ijk}, and H^4_{ijk} using the last expressions in (3), (4), and (5) (or the ones in which the role of index i is replaced by j or k) and store them. $H(p_i, p_j, p_k, p_l)$ is computed by the right-hand-side expression in (2).

The coordinates of the Voronoi point q_{ijk} are represented by

$$(H^2_{ijk}/H^4_{ijk}, \quad -H^3_{ijk}/H^4_{ijk}). \tag{6}$$

Consequently, $(H^2_{ijk}, -H^3_{ijk}, H^4_{ijk})$ are the homogeneous coordinates of q_{ijk}. Hence, the partial results H^2_{ijk}, H^3_{ijk},

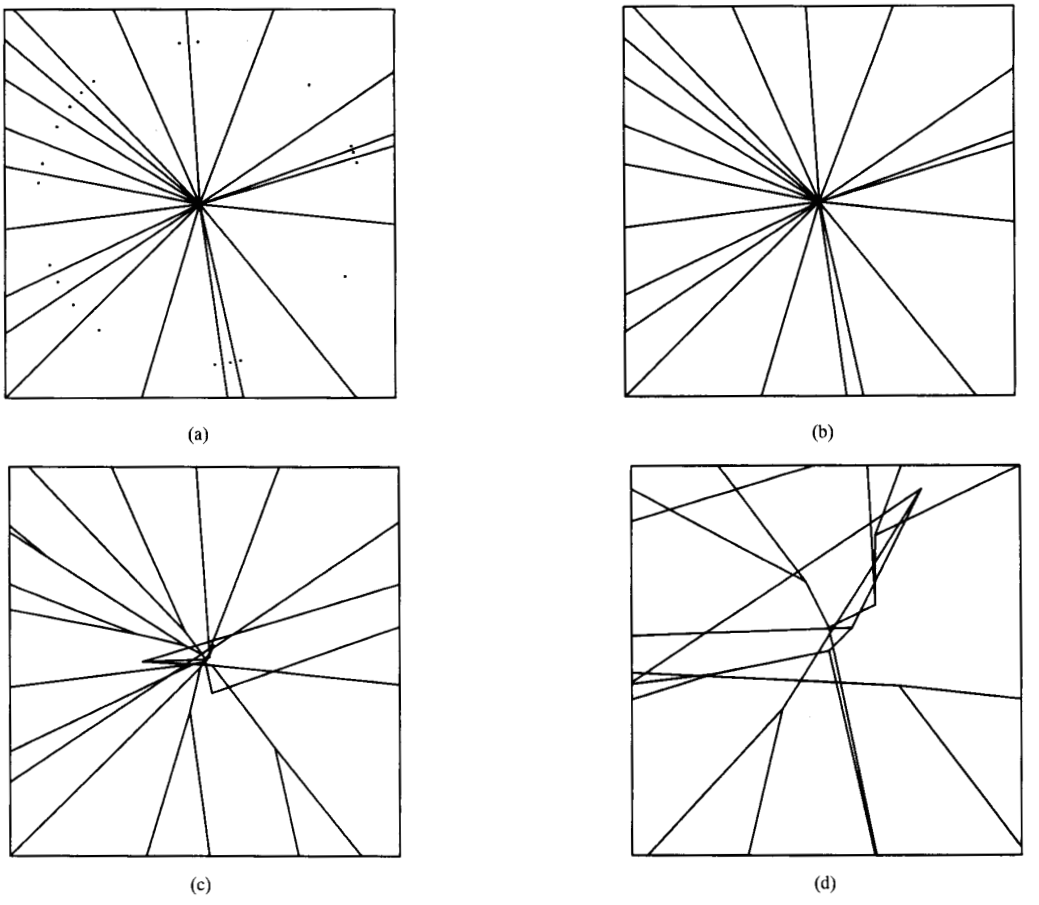


Fig. 7. Output of the program for 20 generators placed on a common circle: (a) output; (b) magnified by 10^3; (c) magnified by 10^5; (d) magnified by 10^6.

and H^4_{ijk} can also be reused for computing the coordinates of the Voronoi points.

Next, let us rewrite $H(p_i, p_j, p_k, p_l)$ in a different way:

$$H(p_i, p_j, p_k, p_l) = J^2_{ijk}(x_l - x_k) - J^3_{ijk}(y_l - y_k) + \frac{1}{2}J^4_{ijk}((x_l - x_k)^2 + (y_l - y_k)^2), \quad (7)$$

where

$$J^2_{ijk} = \begin{vmatrix} y_i - y_k & ((x_i - x_k)^2 + (y_i - y_k)^2)/2 \\ y_j - y_k & ((x_j - x_k)^2 + (y_j - y_k)^2)/2 \end{vmatrix} \quad (8)$$

$$J^3_{ijk} = \begin{vmatrix} x_i - x_k & ((x_i - x_k)^2 + (y_i - y_k)^2)/2 \\ x_j - x_k & ((x_j - x_k)^2 + (y_j - y_k)^2)/2 \end{vmatrix} \quad (9)$$

$$J^4_{ijk} = \begin{vmatrix} x_i - x_k & y_i - y_k \\ x_j - x_k & y_j - y_k \end{vmatrix}. \quad (10)$$

From this expression we get another method for computing $H(p_i, p_j, p_k, p_l)$ in the following way.

Method 2: Each time we generate a new Voronoi point q_{ijk}, we compute J^2_{ijk}, J^3_{ijk}, and J^4_{ijk} using (8), (9), and (10), and store them. $H(p_i, p_j, p_k, p_l)$ is computed using the expression in (7).

The coordinates of q_{ijk} are represented by

$$(-J^2_{ijk}/J^4_{ijk} + x_k, \quad J^3_{ijk}/J^4_{ijk} + y_k).$$

Hence $(-J^2_{ijk}, J^3_{ijk}, J^4_{ijk})$ can be considered the homogeneous coordinates of q_{ijk} with respect to the origin at (x_k, y_k).

In Method 2, the generator p_k has a special role, and there is ambiguity in choosing one of the three generators as p_k. We choose p_k in such a way that, among the three angles of the triangle $p_ip_jp_k$, the angle at p_k is closest to $\pi/2$, because $|J^4_{ijk}/2|$ equals the area of this triangle and the area can be computed most accurately for the above choice of p_k.

In the incremental-type algorithm, the structure of the Voronoi diagram is changed locally; hence the values of $H(p_i, p_j, p_k, p_l)$ are computed for Voronoi points that are close to the new generator p_l. Consequently, in general, the four generators p_i, p_j, p_k, and p_l are close to each other. Therefore, Method 2 can be expected to give smaller nu-

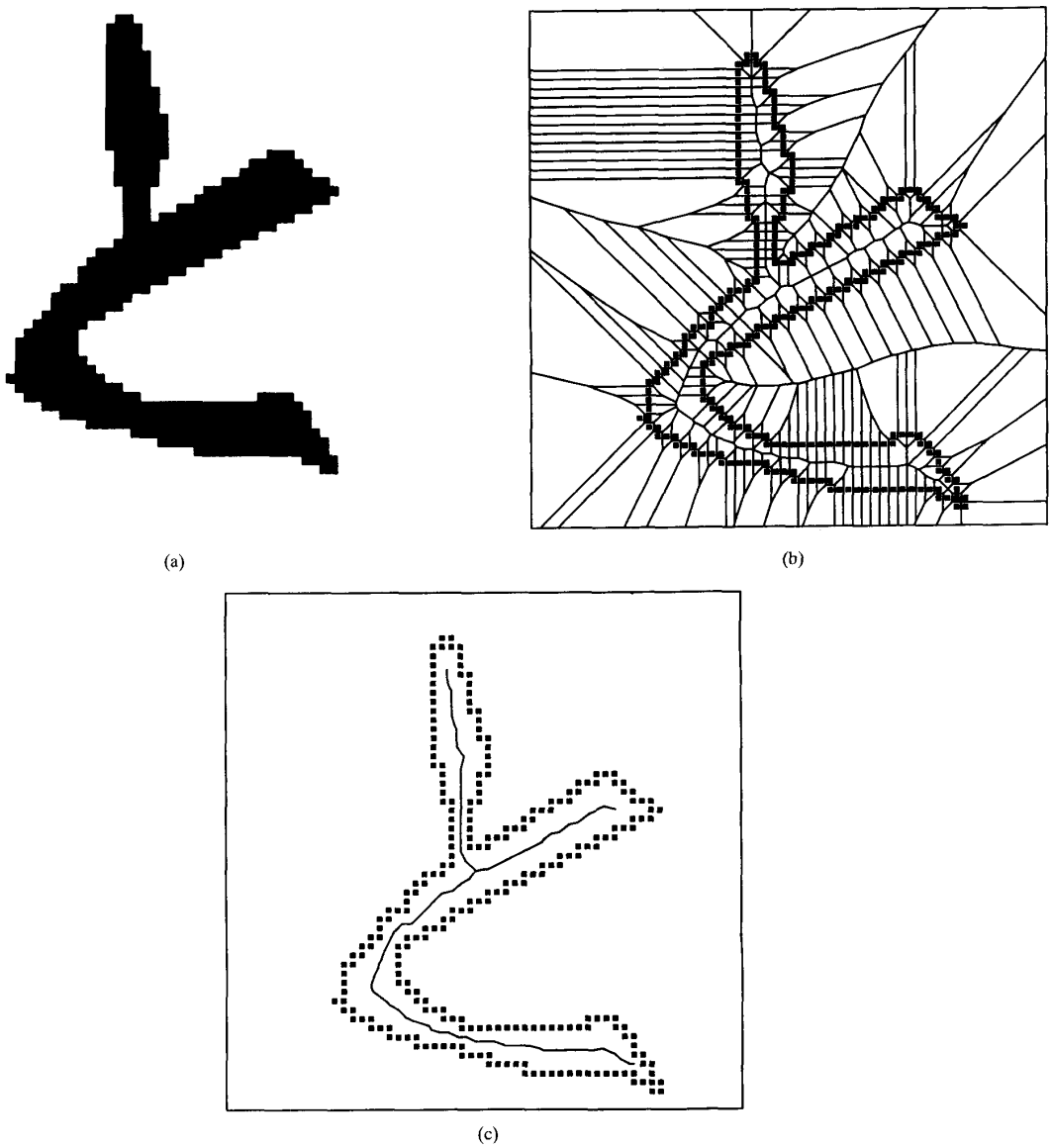


Fig. 8. Extraction of the strokes from a character image: (a) digital image of a Japanese character; (b) Voronoi diagram for the pixels on the boundary as generators; (c) extracted strokes.

merical errors than Method 1 because all the computations are done using p_k as the origin.

Methods 1 and 2 are both very similar to Gaussian elimination. Indeed most of the computations in those methods are the same as Gaussian elimination with respect to particular pivoting elements. Both our methods differ from Gaussian elimination only in the final step, where the partial results we want to reuse would be destroyed if we simply followed the Gaussian elimination procedure. The simple Gaussian elimination requires 13 multiplications, three divisions, and 18 additions, while Method 1 requires 15 multiplications and 24 additions, and Method 2 requires 15 multiplications and 14 additions. In general, numerical errors are larger in additions than in multiplications and divisions. From this point of view also, we can expect that the numerical errors will be smaller in Method 2 than in Method 1 and the simple Gaussian method. Method 1 requires more additions than the simple Gaussian elimination, but this is due to the use of expression $(a + b)(a - b)$ instead of $a^2 - b^2$; if we use the expression $a^2 - b^2$, Method 1 requires the same number of additions as the simple Gaussian method. Since $(a+b)(a-b)$ usually gives smaller numerical errors than $a^2 - b^2$, we can expect that numerical errors are smaller in Method 1 than in the simple Gaussian elimination. Moreover, the simple Gaussian elimination is disadvantageous in that we cannot reuse the results of partial computations. (A more detailed discussion of numerical errors will be given in [22].)

IV. Experiments in Single-Precision Arithmetic

A. Behavior for Small Sets of Generators

Our algorithm was implemented in a computer program written in FORTRAN [21]. The main part of the program consists of about 1200 lines, and the peripheral part including the routines for generating the generators and plotting the diagrams consists of about 1000 lines. All the floating-point computations were done in single precision. The computations were carried out on an NEC personal computer PC-9801VX with the MS-DOS operating system and an NEC FORTRAN77 compiler (Version 2.20).

Figure 6(a) shows the output of the program for 20 generators placed at random in the unit square $\{(x,y) \mid 0 \leq x,y \leq 1\}$; this output can be considered as a "correct" Voronoi diagram in the sense that we visually find no difference from the true Voronoi diagram. Part (b) shows the same output drawn in a wider range, so that we can see the three additional generators that form a triangle including all the other generators. Part (c) is the case where small random numbers were added to the computed values of $H(p_i,p_j,p_k,p_l)$; we added a random value generated uniformly in $[-r,r)$ for $r = 2^{-9}$. Parts (d) and (e) are the cases where random values were generated in greater ranges. Part (f) is the case where all values of $H(p_i,p_j,p_k,p_l)$ were replaced by random values. In all the cases, the program did not come across topological inconsistency, as was guaranteed theoretically; it carried out the task and gave the output. Moreover, the output is "topologically consistent" in the sense that it is a planar graph satisfying properties (P5) and (P6). If we see the outputs (f), (e), (d), (c), and (a) in this order, we can see that the output becomes nearer to the true Voronoi diagram as the precision in computation becomes higher. Indeed the output "converges" to the true Voronoi diagram in the sense that the output diagram becomes isomorphic to the true Voronoi diagram if we merge the two terminal vertices of each length-zero Voronoi edge into a degenerate Voronoi point.

Figure 7(a) shows the output for 20 generators placed at random on a common circle, and parts (b), (c), and (d) of the figure represent the same output magnified by 10^3, 10^5, and 10^6, respectively. Part (a) seems to be a correct Voronoi diagram, but as (c) and (d) show, the central portion of the diagram is far from the correct Voronoi diagram. This set of generators gives a highly degenerate case, and the crisscross structure of the central portion seems natural as the output obtained in single precision. It should be noted that even for such a degenerate set of generators the program carried out its task and gave the output.

The next example is an application of the Voronoi diagram to the extraction of strokes from a character image, where numerical robustness plays an important role. Figure 8(a) is a digital image of a Japanese phonetic character, (b) is the Voronoi diagram whose generators are the pixels on the boundary of the image, and (c) is the collection of those Voronoi edges that are inside the image and whose side regions have generators not close to each other on the closed sequence of pixels forming the boundary of the character. The structure of the strokes of this kind, which are also called medial axes or skeletons, is useful for character recognition [16]. The pixels of a digital image align on the vertical and horizontal lines, so that the set of generators chosen from pixels often give degeneracy. Indeed, there are many Voronoi points having four or more Voronoi edges in Fig. 8. As shown in this example, our program works well even if such degeneracy takes place.

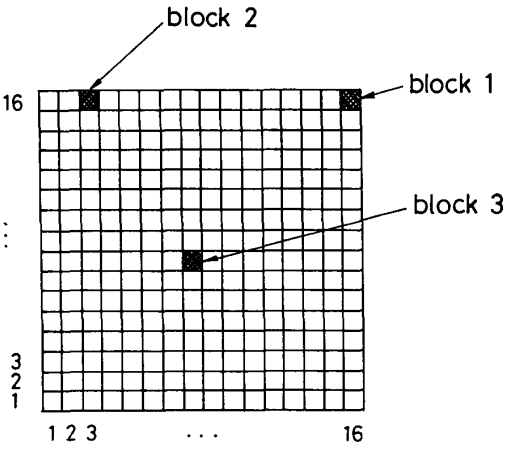


Fig. 9. 16 × 16 blocks covering the unit square.

B. Behavior for 1 000 000 Generators

Experiments with larger sets of generators were done on a Hitachi M-680H large computer with the VOS3 operating system at the Computer Centre of the University of Tokyo. The following are examples of the behavior of the program for 1 000 000 generators placed at random in the unit square $\{(x,y) \mid 0 \leq x,y \leq 1\}$. The x and y coordinates of the generators were given in single-precision floating-point representation using a maximum-length linearly recurring sequence (i.e., M-sequence) based on the characteristic polynomial $1 + x^{32} + x^{521}$. Among 1 000 000 generators thus given, no two occupied the same position. The size of the memory used by the program was about 1.8×10^8 bytes.

In order to show the output, let us consider the partition of the unit square into 16 × 16 (= 256) small square blocks of the same size, as shown in Fig. 9. Let (i,j) represent the position of the blocks located at the ith column from the left and at the jth row from the bottom. We choose three squares:

block 1 at $(16,16)$,
block 2 at $(3,16)$,
block 3 at $(8,8)$.

Parts (a), (b), and (c) of Fig. 10 show the output of the program in blocks 1, 2, and 3, respectively, where numerical computation was done by Method 1. Each part represents the diagram in the block as well as in its marginal region; the square that is a little smaller than the outermost square represents the boundary of the block. In block 3

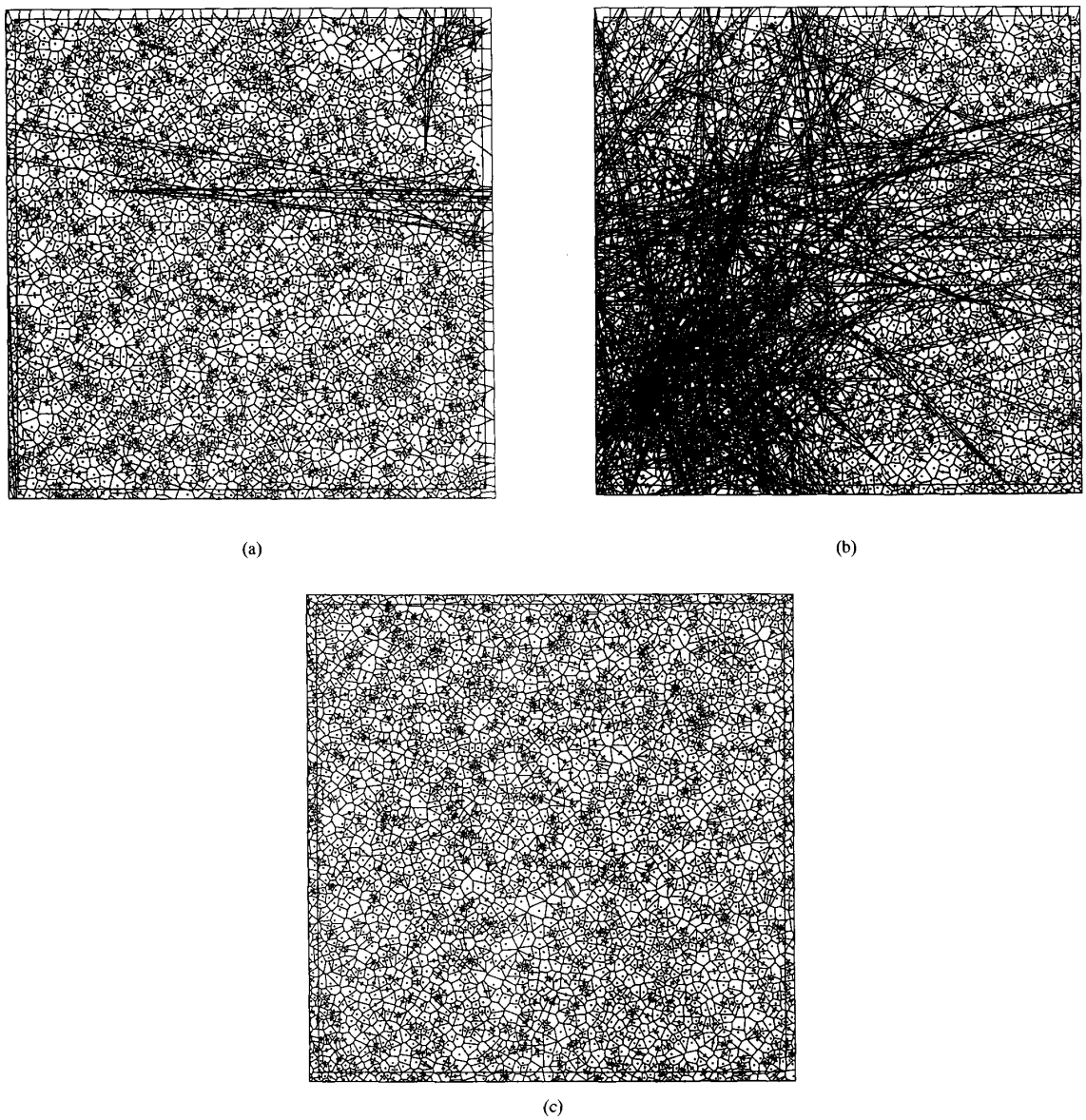
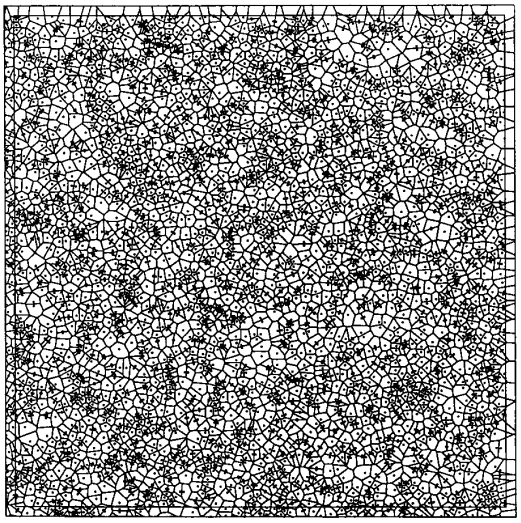


Fig. 10. Output given by Method 1 for 1 000 000 generators: (a) block 1; (b) block 2; (c) block 3.

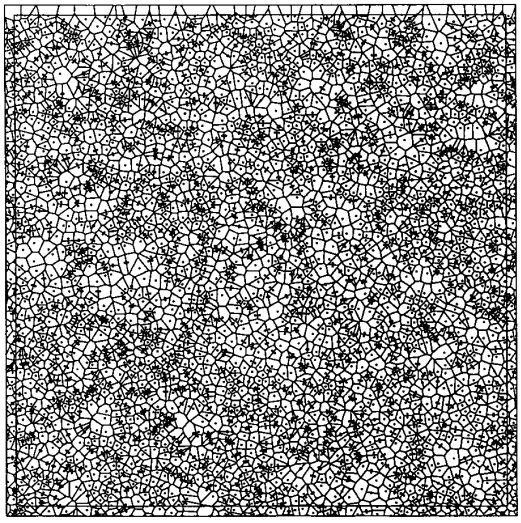
the "correct" Voronoi diagram was computed, which was the typical case for the block not close to the boundary of the unit square. In block 1 the diagram had small disturbance, which was the typical case for the block close to the boundary of the unit square. This observation seems reasonable because a generator around the boundary of the square region is apt to have one of the three additional generators as its neighbor. For this reason, the distances to the neighboring generators have a wide range, which might cause larger numerical errors in the computation of $H(p_i, p_j, p_k, p_l)$. In block 2 the diagram was far from the correct Voronoi diagram. Indeed, this block was the one that had the greatest disturbance for this particular set of generators.

When Method 2 was applied, on the other hand, such disturbance was not observed. Parts (a) and (b) of Fig. 11 show the output in blocks 1 and 2, respectively, of the program in which Method 2 was used. Indeed, in Method 2 the Voronoi diagram was constructed "correctly" in the whole unit square.

In the construction of the Voronoi diagram for the 1 000 000 generators, the values of $H(p_i, p_j, p_k, p_l)$ were computed for about 3×10^7 different combinations of i, j, k, and l. We counted the number of values of $H(p_i, p_j, p_k, p_l)$ whose sign was opposite to the result of computation in double precision. The number was 3442 in Method 1, while it was 171 in Method 2. Hence, the computation based on Method 2 is more accurate than that based on Method 1.



(a)



(b)

Fig. 11. Output given by Method 2: (a) block 1; (b) block 2.

Tables 1 and 2 represent certain other statistical data. Table 1 represents the distribution of the number of steps required in step B1 in Procedure B to reach the old generator that is closest to the new generator. In our program, the unit square was divided into 256×256 (= 65 536) buckets, and the starting generator was chosen in the same manner as in [15] and was replaced with the neighboring generator that was closer. Table 1 represents the distribution of the number of such replacements. The largest number of replacements was 9 in both Methods 1 and 2; we can see that in both cases step B1 was carried out in a constant average time.

Table 2 represents the distribution of the number of vertices of the new Voronoi region generated in the ad-

Table 1 Distribution of the Number of Steps Required to Reach the Nearest Generator

No. Steps	Method 1	Method 2
1	111413	111309
2	393354	393185
3	319254	319402
4	134452	134537
5	35009	35050
6	5832	5833
7	632	630
8	48	48
9	6	6

Table 2 Distribution of the Number of Vertices of the Voronoi Region Generated by the Addition of a New Generator

No. Vertices	Method 1	Method 2
3	12982	12896
4	104220	104081
5	234138	234122
6	264904	265018
7	192286	192436
8	106710	106846
9	49457	49522
10	20656	20664
11	8441	8425
12	3430	3432
13	1447	1412
14	611	591
15	301	292
16	178	163
17	69	57
18	37	29
19	17	8
20	13	4
21	11	0
22	7	2
23	5	0
24	11	0
25 or more	69	0

dition of a generator. Theoretically, the average number of such vertices is $6 + O(1/n)$ for uniformly and randomly distributed generators; in both methods the peak of the distribution was close to this theoretical average.

The two tables show that Procedure A was carried out in almost constant time. Indeed, the average lap time for adding every 10^4 generators was about 9 s in both Methods 1 and 2, and the lap time was almost constant in the whole processing of the 1 000 000 generators (because of the virtual memory managing system, there was perturbation in more than 10% of the CPU time, so that the lap time could be measured only roughly). Thus, the expected computational complexity of $O(n)$ was attained in this program.

V. Concluding Remarks

We have proposed a numerically stable algorithm for constructing the Voronoi diagram, in which higher priority is placed on the consistency of the topological structure than on numerical values.

The present program not only is robust against numerical errors; it also possesses other remarkable features. First, the average time complexity is linear in the number of generators; our modification does not destroy the expected time complexity attained by the incremental-type algorithm originally proposed in [15]. Second, the structure of the program is much simpler than the conventional one, because the program need not take care of special processing for degenerate cases. Recall that, in finite-precision arithmetic, we cannot discern whether degeneracy takes place or not; even if a numerical result suggests degeneracy, it means no more than that the situation is close to a degenerate case. Hence, in a finite-precision world, all cases are considered nondegenerate.

Moreover, we tuned up the way of computing numerical values, and thus succeeded in constructing the "correct" Voronoi diagram for as many as 1 000 000 generators in single precision. We would like to emphasize that the tuning was done efficiently because we had a numerically robust algorithm. Indeed, Method 1 could construct a Voronoi diagram for 2×10^5 generators "correctly", and failed in constructing a Voronoi diagram for 3×10^5 generators. An ordinary program, when it fails in processing, will go into an endless loop, will distroy the data, etc; it is almost impossible to trace the behavior of the program in order to find the reason of the failure for such extensive input data. Our program, on the other hand, always gives an output, and the output enables us to compare different ways of numerical computation easily.

In this paper we have shown that it is possible for us to design a geometric algorithm in finite-precision environment; we need not assume the error-free world to prove the validity of the algorithm—all that we can and must do is to slightly change the concept of the "validity." The basic principle of the present paper is to place the highest priority on topological consistency and thus to make an algorithm that is robust against numerical errors and whose output is at least "topologically consistent." Such an output may have numerical disturbances, but still can be used for many applications if application algorithms are also redesigned by placing the highest priority on topological consistency; further discussion will be given in [22].

The principle used in this paper seems useful in the design of a wide range of algorithms in computational geometry. The present study is only a first step of our long-range project to introduce a new concept of "an algorithm that works well in finite-precision arithmetic" and thus to establish a methodology for dealing with numerical problems within the framework of the theory of algorithms.

References

[1] F. Aurenhammer, "Power diagrams — Properties, algorithms and applications" *SIAM J. Comput.*, vol. 16, pp. 78–96, 1987.

[2] L. DeFloriani, "A pyramid data structure for triangle-based surface description," *Computer Graphics and Applications*, vol. 9, pp. 67–78, 1989.

[3] D. Dobkin and D. Silver, "Recipes for geometry and numerical analysis — Part I, An empirical study," in *Proc. 4th ACM Annual Symp. Computational Geometry* (Urbana-Champaign), June 1988, pp. 93–105.

[4] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*. Berlin: Springer-Verlag, 1987.

[5] J. Fairfield, "Segmenting dot patterns by Voronoi diagram concavity," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 5, pp. 104–110, 1983.

[6] S. Fortune, "A sweepline algorithm for Voronoi diagrams," *Algorithmica*, vol. 2, pp. 153–174, 1987.

[7] C. M. Hoffmann, "The problems of accuracy and robustness in geometric computation," *Computer*, vol. 22, no. 3, pp. 31–41, Mar. 1989.

[8] R. N. Horspool, "Constructing the Voronoi diagram in the plane," Tech. Rep. SOCS-79.12, School of Computer Science, McGill University, Montreal, Canada, 1979.

[9] M. Iri, "Practical computational methods in geometrical/geographical optimization problems," in *Methods of Operations Research*, 54 (Proceedings of the X. Symposium on Operations Research, München, 1985), M. J. Beckmann, K.-W. Gaede, K. Ritter, and H. Schneeweiss, Eds. Verlag Anton Hain, Part II, 1986, pp. 17–37.

[10] M. Iri and T. Koshizuka, Eds. *Computational Geometry and Geographic Information Processing* (in Japanese). Tokyo: Kyoritsu-Shuppan, 1986.

[11] M. Iri, K. Murota, and T. Ohya, "A fast Voronoi-diagram algorithm with applications to geographical optimization problems," *Lecture Notes in Control and Information Sciences*, 59 (System Modeling and Optimization — Proceedings of the 11th IFIP Conference, Copenhagen, 1983), Springer-Verlag, 1984, pp. 273–288.

[12] M. Iri and K. Sugihara, "Geometric algorithms robust against numerical errors," Note of the Special Interest Group on Algorithms of the Information Processing Society of Japan, SIGAL 1–1, May 1988.

[13] J. Katajainen and M. Koppinen, "Constructing Delaunay triangulations by merging buckets in quadtree order," *Fundamenta Informaticae*, vol. XI, pp. 275–288, 1988.

[14] D. T. Lee and B. J. Schachter, "Two algorithms for constructing a Delaunay triangulation," *Int. J. Computer and Information Sciences*, vol. 9, pp. 219–242, 1980.

[15] T. Ohya, M. Iri, and K. Murota, "Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms," *J. Oper. Res. Soc. Japan*, vol. 27, pp. 306–336, 1984.

[16] F. P. Preparata and M. I. Shamos, *Computational Geometry — An Introduction*. New York: Springer-Verlag, 1985.

[17] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, 2nd ed., vols. 1 and 2. Orlando, FL: Academic Press, 1982.

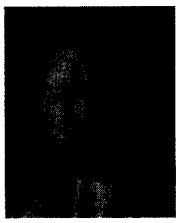
[18] M. I. Shamos and D. Hoey, "Closest-point problems," in *Proc. 16th Annual IEEE Symp. Foundation of Computer Science*, 1975, pp. 151–162.

[19] K. Sugihara and M. Iri, "Geometric algorithms in finite-precision arithmetic," in *Abstracts 13th Int. Symp. Mathematical Programming* (Tokyo), 1988, WE/3K2, p. 196. (The text presented was printed as Research Memorandum RMI 88–10, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, Tokyo, 1988.)
[20] K. Sugihara and M. Iri, "Two design principles of geometric algorithms in finite-precision arithmetic," *Appl. Math. Lett.*, vol. 2, pp. 203–206, 1989.
[21] K. Sugihara and M. Iri, "VORONOI2 reference manual," Research Memorandum RMI 89–04, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, Tokyo, 1989.
[22] K. Sugihara and M. Iri, "Topology-oriented incremental method for constructing Voronoi diagrams robust against numerical errors" Preprint, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, Tokyo, 1992.

Kokichi Sugihara (Member, IEEE) received the B.Eng., M.Eng., and D.Eng. degrees from the University of Tokyo in 1971, 1973, and 1980, respectively.

From 1973 to 1981 he was with the Electrotechnical Laboratory of the Ministry of International Trade and Industry of Japan. From 1981 to 1986, he was an Associate Professor of the Faculty of Engineering of Nagoya University. Since 1986 he has been with the Department of Mathematical Engineering and Information Physics of the Faculty of Engineering of the University of Tokyo, where he is now a professor. During the period 1990–1991 he was a visiting associate professor in the Department of Computer Science at Purdue University, West Lafayette, IN.

His research interests include computational geometry, computer vision, and computer graphics. He is the author of *Machine Interpretation of Line Drawings* (MIT Press, 1986) and coauthor of *Spatial Tessellations—Concepts and Applications of Voronoi Diagrams* (Wiley, 1992). He received best paper awards in 1979, 1987, and 1991 from the Information Processing Society of Japan. In 1992 and 1993 he will chair the Special Interest Group on Computer Vision of the Information Processing Society of Japan. He is an associate editor of *Pattern Recognition* (the journal of the Pattern Recognition Society). He is a member of the Information Processing Society of Japan, the Operations Research Society of Japan, the Japan Society for Industrial and Applied Mathematics, the Institute of Electronics, Communication and Information Engineers (of Japan), the Robotics Society of Japan, the Society of Instrument and Control Engineers (of Japan), and ACM.



Masao Iri (Fellow, IEEE) received the B.Eng., M.Eng., and D.Eng. degrees from the University of Tokyo in 1955, 1957, and 1960, respectively. He has been engaged in teaching and research on network theory, numerical methods, and other topics of applied mathematics at Kyushu University and the University of Tokyo. Since 1973, he has been with the Department of Mathematical Engineering and Information Physics of the Faculty of Engineering of the University of Tokyo as Professor. He was dean of the Faculty of Engineering of the University of Tokyo from April 1987 to March 1989 and vice-president of the University of Tokyo from April 1989 to March 1991.

Dr. Iri did pioneering work on the application of matroids to circuits and systems problems. He has authored 16 books and about 200 research papers and has edited the proceedings of two international conferences. He was awarded the Matsunaga Prize in 1965, paper prizes from IEICE (1969 and 1976) and the Information Processing Society of Japan (1981, 1988, and 1990), and an achievement prize from the IEICE in 1989. In January 1989 he was made a fellow of the IEEE for "contributions to the theory of networks and its applications to circuits and system design."

Dr. Iri is on the editorial boards of 15 international journals and is a member of more than ten domestic and international academic societies (among them the IEICE, the Information Processing Society of Japan, the Operations Research Society of Japan, the Mathematical Programming Society, and Sigma Xi). He was vice-president of IFORS (International Federation of Operational Research Societies) from 1983 to 1985 and is currently president of APORS (Asian Pacific Operational Research Societies within IFORS), president of the Operations Research Society of Japan, and vice-president of the Japan Society for Industrial and Applied Mathematics.