1991

A Robust Intersection Algorithm Based on Delaunay Triangulation

Kokichi Sugihara

Report Number:
91-011

A ROBUST INTERSECTION ALGORITHM BASED ON DELAUNAY TRIANGULATION

Kokichi Sugihara

CSD-TR-91-011
February 1991

A Robust Intersection Algorithm Based on Delaunay Triangulation

Kokichi Sugihara

Department of Computer Science, Purdue University
West Lafayette, Indiana 47907, U.S.A.

Department of Mathematical Engineering and Information Physics
University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113, Japan

Abstract

The paper presents a new robust method for finding points of intersection of line segments in the plane. In this method the subdivision of the plane based on the Delaunay triangulation plays the main role. First, the Delaunay triangulation spanning the end points of line segments is constructed. Next, for line segments that are not realized by Delaunay edges, midpoints are inserted recursively until the descendants of the line segments become realized by Delaunay edges or the areas containing points of intersection are sufficiently localized. The method is robust in the sense that in any imprecise arithmetic it gives a topologically consistent output, and is stable in the sense that it does not miss intersection that can be easily detected by naive pairwise check with the precision at hand.

1. Introduction

Rapid development in computational geometry has produced a huge amount of algorithms for solving a variety of geometric problems. However, most of them are designed in the assumption that numerical computation can be done precisely. In actual computation, on the other hand, it is usually difficult to avoid numerical errors, and hence straightforward implementations of these algorithms do not necessarily give practically valid computer programs; numerical errors often produce inconsistency, making algorithms to fail.

Traditionally the problems caused by numerical errors were left to implementors, but it has gradually been recognized that in order to solve the numerical error problems we need to consider numerical errors from the beginning of designing algorithms [Dobkin and Silver, 1988] [Hoffmann, 1989], and several such attempts are proposed. They include use of integer grids [Greene and Yao, 1986], use of higher-precision arithmetic [Ottmann, Thiemt and Ullrich, 1987] [Sugihara and Iri, 1989a] [Karasick, Lieber and Nackman, 1989], use of geometric reasoning [Hoffmann, Hopcroft and Karasick, 1988] [Karasick, 1989], use of enriched worlds [Milenkovic, 1988, 1989], use of three-value logic [Guibas, Salesin and Stolfi, 1989], use of approximate geometric predicates [Fortune, 1989] and use of combinatorial abstraction [Sugihara and Iri, 1989b].

In this paper we propose a new robust method for finding points of intersection of line segments in a two-dimensional plane. In this method the subdivision of the plane by the Delaunay triangulation is employed as a key tool for localizing the areas where points of intersection exist and for guaranteeing the topological consistency of the resultant arrangement. The method has several good properties. First it is robust in the sense that it always gives a topologically consistent output. Secondly it is stable in the sense that it does not miss points of intersection that are easily detectable in a given precision. Thirdly, it is adaptive in the sense that most of computational cost is paid to the areas where finding points of intersection is difficult.

The Delaunay triangulation is a kind of the space subdivision. The space subdivision is one of fundamental techniques in designing geometric algorithms. Typical examples are a slab method [Preparata and Shamos, 1985], a space pointer method [Ohsawa, 1986, 1990], convex subdivision [Vaněček, 1989] and quaternary or octant subdivision [Samet, 1990]. The present method is different from the slab method and the space pointer method in the sense that we divide the space according to intrinsic relations among objects whereas the other methods depend on the coordinate system. The present method is different from the convex subdivision because we divide the space according to the distance among objects. The present method has similarity with the quaternary/octant subdivision in the sense that finer subdivision is applied to the areas where the configurations are more complicated. However there is a great difference because the quaternary/octant subdivision uses a fixed grid structure whereas we divide the space using given objects as boundaries.

After reviewing numerical error problems in Section 2 and making some preparation in Section 3, we present the basic algorithm and its behavior in Sections 4 and 5. In Section 6 the algorithm is revised into a robust one, and computational experiments are shown in Section 7.

2. Numerical Problems in Finding Intersections

Let p_i be a point in the plane $\Re^2$ with coordinates (x_i, y_i) for $i = 1, 2, \ldots$ We denote by $l(p_i, p_j)$ the open line segment connecting p_i and p_j, and by $\bar{l}(p_i, p_j)$ the closed line segment connecting p_i and p_j. We denote the length of $\bar{l}(p_i, p_j)$ by $|\bar{l}(p_i, p_j)|$.

Two closed line segments $\bar{l}(p_1, p_2)$ and $\bar{l}(p_3, p_4)$ have a point of intersection (as shown in Fig. 2.1), if and only if there exist $0 \le \alpha, \beta \le 1$ satisfying

$$\alpha \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + (1-\alpha) \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \beta \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + (1-\beta) \begin{pmatrix} x_4 \\ y_4 \end{pmatrix}. \tag{2.1}$$

From this equation we get

$$\alpha = ((x_2 - x_4)(y_3 - y_4) - (y_2 - y_4)(x_3 - x_4))/\gamma, \tag{2.2}$$

$$\beta = (-(x_1 - x_2)(y_2 - y_4) + (y_1 - y_2)(x_2 - x_4))/\gamma, \tag{2.3}$$

where

$$\gamma = -(x_1 - x_2)(y_3 - y_4) + (y_1 - y_2)(x_3 - x_4). \tag{2.4}$$

If $\gamma \ne 0$, we can judge whether the two line segments have a point of intersection by computing α and β and by checking the condition $0 \le \alpha, \beta \le 1$. If $\gamma = 0$ (which happens when the two line segments are collinear), the two line segments have a common point of intersection if and only if

$$\min\{\xi, \eta\} \le 1 \quad \text{and} \quad \max\{\xi, \eta\} \ge 0, \tag{2.5}$$

where ξ and η are two values satisfying

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} = \xi \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + (1-\xi) \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}, \tag{2.6}$$

$$\begin{pmatrix} x_4 \\ y_4 \end{pmatrix} = \eta \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + (1-\eta) \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}. \tag{2.7}$$

Hence if n line segments are given, we can find all points of intersection in $O(n^2)$ time by computing γ and either α and β or ξ and η for all pairs of line segments.

However, the exhaustive pairwise check is expensive particularly when the number of points of intersection is not large. A typical technique for avoiding unnecessary pairwise check is a plane sweep method [Bentley and Ottmann, 1979]. In this method the plane is swept by a vertical line, called a sweepline, from left to right. The algorithm maintains a list, say L, of all the line segments intersecting with the sweepline in the increasing order with respect to the y coordinates of the points of intersection with the sweepline, and the pairwise check is done only for those pairs that are ever adjacent in the list L.

An example is shown in Fig. 2.2(a), where there are four line segments a, b, c, d. When the sweepline hits the left endpoint of line d (see the sweepline at S_1), d is inserted in L, resulting in $L = (a, b, c, d)$. When the sweepline hits the point of intersection of a and b (at S_2), they are interchanged in the list, resulting in $L = (b, a, c, d)$. When the sweepline pass through the right endpoint of d (at S_3), d is deleted from L and we get $L = (b, a, c)$. In this example, d is always above c whereas a and b are always below c; d never comes adjacent to a or b. Consequently the pairwise check is not done between a and d or between b and d. In this way we can avoid unnecessary pairwise check.

However, the plane sweep method is not stable, because in imprecise arithmetic it has chances to miss points of intersection that are easily detectable by the pairwise check [Kanazawa, 1990]. An example of such a situation is shown in Fig. 2.2(b). Since three line segments a, b, c have their left endpoints very close to each other, numerical errors may give misjudgment on the order of these line segments. Suppose that because of numerical errors the algorithm judged that $L = (a, c, b)$ after the sweepline hits all the left endpoints of these three lines and points of intersection if any (at S_4). When the sweepline hits the left endpoint of d (at S_5), d is judged above a and below c, and hence d is inserted between a and c, resulting in $L = (a, d, c, b)$. Thus, b has no chance to be adjacent to d in L, and hence the point of intersection between b and d cannot be detected.

Note that the endpoints of a, b, c come close to each other and hence the points of intersection among them may be missed. However, this is allowable because the configuration is so delicate and it is difficult to find all the point of intersection in imprecise arithmetic. On the other hand, b and d intersect "obviously"; to miss it is a serious problem.

Ottmann, Thiemt and Ullrich (1987) proposed a method for employing high precision arithmetic in order to avoid the unstableness. However, their method can be used only when higher precision computation is available. It is also important and useful to construct a stable algorithm that works even if high precision arithmetic is not available.

It is interesting to note that if we naively make the pairwise check between all pairs of line segments, we do not fail in detecting obvious intersections. It seems that the unstableness of the plane sweep method is the cost we should pay for obtaining efficiency in processing time.

The exhaustive pairwise check is also unstable for certain kinds of intersection problems. One of such examples is the intersection between polygonal lines. As shown in Fig. 2.3(a), suppose that we are given two polygonal lines, one consisting of two consecutive line segments $\bar{l}(p_1, p_2)$ and $\bar{l}(p_2, p_3)$ and the other consisting of one line segment $\bar{l}(p_4, p_5)$. Since the point p_2 is close to $\bar{l}(p_4, p_5)$, if we make the pairwise check between $\bar{l}(p_1, p_2)$ and $\bar{l}(p_4, p_5)$ and between $\bar{l}(p_2, p_3)$ and $\bar{l}(p_4, p_5)$ independently, it may happen that none of the two pairs has point of intersection, failing in finding the obvious point of intersection. In order to find the point of intersection stably, we need to use the information that p_2 is the common end point of $\bar{l}(p_1, p_2)$ and $\bar{l}(p_2, p_3)$.

Another example is shown in Fig. 2.3(b), where we want to find points of intersection between the boundaries of two polygons. Since the boundaries are closed, their points of intersection appear in pairs; if a traveler moving along one of the boundaries crosses the other and enters into its interior, he must cross it again to go out. However, if the pairwise check is done between line segments independently, one of the points may be missed. In that case the number of points of intersection becomes odd, though it should be even. This kind of topological inconsistency easily gives a fatal damage to many geometric operations, such as

Boolean operations between polygons.

Thus, what we want is an algorithm that does not miss obvious points of intersection and that gives topologically consistent output, even if the precision in computation is poor.

Milenkovic (1988) proposed an algorithm with these properties. He enriched the object world from line segments to pseudo line segments, and constructed a method to find the arrangement of given line segments always consistently. However, his method still requires a certain precision in computation.

In this paper, we further loosen the object world, and construct a simple algorithm which works in any poor arithmetic; no matter how poor the precision may be, the algorithm behaves reasonably in a given precision and always gives a consistent output.

3. Intersection Problem and the Delaunay Triangulation

Let $G = (V, E)$ be a finite graph having vertex set V and edge set E. We assume that G has no selfloop and no parallel edge, and hence V is a finite set and E is a collection of two-element subsets of V. If $e = \{v_i, v_j\} \in E$, we say that v_i and v_j are the end vertices of e. Let μ be a mapping from V to $\Re^2$. For $v_i \in V$, we write $\mu(v_i) = (x_i, y_i)$ and consider them as the two-dimensional coordinates of the vertex v_i. Placing the vertices in the plane by μ and drawing straight line segments between all pairs of the end vertices of the edges, we get a diagram composed of line segments. We call this diagram the *diagram* generated by G and μ, or *diagram* (G, μ) in short.

For edge $e \in E$, we denote by $\mu(e)$ the closed line segment connecting $\mu(v_i)$ and $\mu(v_j)$:

$$\mu(e) = \bar{l}(\mu(v_i), \mu(v_j)). \tag{3.1}$$

We denote by $L(G, \mu)$ the collection of all the closed line segments constituting the diagram (G, μ):

$$L(G, \mu) = \{\mu(e) \mid e \in E\}. \tag{3.2}$$

We denote by $D(G, \mu)$ the set of all points on at least one line segment in $L(G, \mu)$:

$$D(G, \mu) = \{p \mid p \in \mu(e) \text{ for some } e \in E\}. \tag{3.3}$$

Given the diagram (G, μ), we want to find all the points of intersection between two line segments in $L(G, \mu)$. Some of the points of intersection are given explicitly in the graph structure, that is, if E contains $\{v_i, v_j\}$ and $\{v_i, v_k\}$, $\bar{l}(v_i, v_j)$ and $\bar{l}(v_i, v_k)$ have v_i as their common point of intersection. So our problem is to find the points of intersection that are not represented explicitly in the graph structure.

An example of diagram (G, μ) is shown in Fig. 3.1(a), where G has seven vertices $v_1, v_2, \ldots, v_7$ and four edges $\{v_1, v_2\}$, $\{v_2, v_3\}$, $\{v_4, v_5\}$, $\{v_6, v_7\}$, and the images of the vertices by the mapping μ are as shown there. The diagram has points of intersection that are not represented explicitly by the graph G. Inserting new vertices at the points of intersection and dividing the edges at these points, we get another diagram (G', μ') shown in (b). The graph G' has nine vertices $v_1, v_2, \ldots, v_9$ and nine edges $\{v_1, v_8\}$, $\{v_8, v_2\}$, $\{v_2, v_9\}$, $\{v_9, v_3\}$, $\{v_4, v_8\}$, $\{v_8, v_9\}$, $\{v_9, v_5\}$, $\{v_6, v_9\}$, $\{v_9, v_7\}$. In this paper, by the problem of finding points of intersection in diagram (G, μ) we mean the problem of modifying (G, μ) to (G', μ') such that G' represents all the points of intersection explicitly.

More formally we define as follows. Diagram (G', μ') is called a *refinement* of diagram (G, μ) if the graph G' is obtained from G by inserting new vertices on edges of G and by dividing the edges into pieces at the new vertices and $D(G, \mu) = D(G', \mu')$. Diagram (G, μ) is said to be *coarser* than diagram (G', μ') if (G', μ') is a refinement of (G, μ). Diagram

(G,μ) is said to be *planar* if line segments in $L(G,\mu)$ do not intersect except at their end points. The diagram in Fig. 3.1(b) is the coarsest planar refinement of the diagram in (a). Thus, given a diagram (G,μ), our problem is to find the coarsest planar refinement of (G,μ),

If every vertex is the end point of exactly one edge, $L(G,\mu)$ is a collection of line segments, and hence our problem is to find all points of intersection between line segments. If G consists of exactly one path, the diagram (G,μ) is a polygonal line, and hence our problem is to find all points of self-intersection of the polygonal line. If G is disconnected and every connected component is a path, the diagram (G,μ) is a collection of polygonal lines and our problem is to find all points of self-intersection within each polygonal line and all point of intersection between polygonal lines. Thus, our general form of the problem includes many different types of intersection problems.

To solve this problem we employ the concept of the Delaunay triangulation. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane such that all the points are not on a common line. For subset P' of P, if P' contains at least three points and there is a circle passing through all the points in P' and contains no points in $P - P'$ on it or in its interior, the convex polygon defined by the vertex set P' is called a *Delaunay polygon*. The collection of all Delaunay polygons gives a partition of the convex hull of P. If all the Delaunay polygons are triangles, this partition is called the *Delaunay triangulation* spanning P. If there are nontriangular Delaunay polygons (this situation is called *degeneracy*), we arbitrarily triangulate them by adding diagonal edges; the resultant partition of the convex hull of P is also called the Delaunay triangulation spanning P. Let $\mathcal{T}(P)$ denote the Delaunay triangulation spanning P. If there is no degeneracy, $\mathcal{T}(P)$ is unique. Otherwise, there is freedom in the choice of diagonal edges for nontriangular Delaunay polygons. Throughout this paper, let us assume that we have a certain algorithm for constructing the Delaunay triangulation, and that $\mathcal{T}(P)$ represents the Delaunay triangulation obtained by that algorithm. Hence, even if degeneracy takes place, we consider that $\mathcal{T}(P)$ represents a uniquely fixed Delaunay triangulation.

A triangle appearing in $\mathcal{T}(P)$ is called a *Delaunay triangle*, and an edge of a Delaunay triangle is called a *Delaunay edge*. Let n_c represent the number of vertices of the convex hull of P. Then, the number of edges and that of triangles in $\mathcal{T}(P)$ are $3(n-1) - n_c$ and $2(n-1) - n_c$, respectively. Hence, both the number of edges and that of vertices are of $O(n)$.

There are many efficient methods for constructing the Delaunay triangulation [Shamos and Hoey, 1975] [Lee and Schachter, 1980] [Ohya, Iri and Murota, 1984] [Guibas and Stolfi, 1985] [Fortune, 1987] [Katajainen and Koppinen, 1988] [Sugihara and Iri, 1989b] [Sugihara, Ooishi and Imai, 1990], among which a robust version of an incremental method [Sugihara and Iri, 1989b] is useful for the present purpose because we want to modify the Delaunay triangulation by adding new points step by step.

4. Basic Idea.

In this section, we present the basic idea of our method. In order to simplify the argument, we assume for a while that there is no numerical error in computation and that no degeneracy takes place, that is, every line segment $\mu(e)$ has nonzero length and each point of intersection lies on exactly two line segments. In Section 6 we will remove all of these assumptions and design a robust version of the algorithm.

In order to apply the Delaunay triangulation, let us start with some properties which relate the Delaunay triangulation and the points of intersection. The next property is a direct consequence of the definition.

Property 1. Let p and q be two points in P. If there is a circle passing through p and q and containing no other points in P on it or in its interior, $\bar{l}(p,q)$ is a Delaunay edge of $\mathcal{T}(P)$.

Property 1 intuitively implies that if two points are sufficiently close to each other, they admit a Delaunay edge. Hence, we can expect the following. Let p and q be two points in P. They are not necessarily connected by a Delaunay edge in $\mathcal{T}(P)$. However, if we add sufficiently many points on the line segment $l(p,q)$ and construct the Delaunay triangulation for the augmented set of points, p and q will be connected by a sequence of Delaunay edges. This expectation is one of the basis of our method.

Our aim is to find points of intersections between line segments. For this purpose the next two properties, which are the contrapositions of each other, are useful; we omit the proofs because they are obvious.

Property 2. If $\bar{l}(p_1,p_2)$ and $\bar{l}(q_1,q_2)$ are both Delaunay edges in $\mathcal{T}(P)$, the open line segments $l(p_1,p_2)$ and $l(q_1,q_2)$ do not intersect.

Property 3. If $l(p_1,p_2)$ and $l(q_1,q_2)$ intersect, at least one of $\bar{l}(p_1,p_2)$ and $\bar{l}(q_1,q_2)$ is not a Delaunay edge in $\mathcal{T}(P)$.

On the basis of these properties we can consider the next method for finding points of intersection of the diagram (G,μ). We define P as the set of the images of the vertices by the mapping μ, i.e., $P = \{\mu(v) \mid v \in V\}$, and construct the Delaunay triangulation $\mathcal{T}(P)$. We say that edge e is *realized* in $\mathcal{T}(P)$ if $\mu(e)$ is a Delaunay edge in $\mathcal{T}(P)$, and *unrealized* otherwise. If all the edges in E are realized in $\mathcal{T}(P)$, we can conclude from Property 2 that there is no implicit point of intersection in (G,μ). If there are unrealized edges, on the other hand, we can say nothing about these edges; they may intersect with other edges or they may not. However, we do not want to make the pairwise check against all other edges. So, we insert new vertices at the midpoints of these edges, and construct the Delaunay triangulation again for the augmented set of vertices.

Recall that the Delaunay triangulation is a triangulation in which edges are generated between points that are relatively close to each other. Hence, we can expect that if we insert sufficiently many new points on line segments, either we can see that there is no point of intersection or we can localize the areas in which the points of intersection may exist. So, we concentrate on the following pattern. We say that two edges $e = \{v_i, v_j\}$ and $e' = \{v'_i, v'_j\}$ form a *cross pattern* (see Fig. 4.1) if (i) $\mu(v_i)$, $\mu(v_j)$, $\mu(v'_i)$ form a Delaunay triangle (hence e is a realized edge), (ii) e and e' are not adjacent to each other in G, and (iii) $\mu(v'_j)$ is in the angle $\angle\mu(v_i)\mu(v'_i)\mu(v_j)$ and $\mu(v'_i)$ and $\mu(v'_j)$ are mutually in the other sides of $\mu(e)$ (note that the condition (iii) is equivalent to that $l(v_i, v_j)$ and $l(v'_i, v'_j)$ have a point of intersection). Thus, the cross pattern is a pair of line segments that are mutually intersecting and that are incident to a common Delaunay triangle. If the conditions (i), (ii), (iii) are checked in this order, all cross patterns can be found in time linear in the number of vertices, because the number of Delaunay triangles is of the same order as that of the vertices.

Suppose that in $\mathcal{T}(P)$ there is a pair of realized edge $e = \{v_i, v_j\}$ and unrealized edge $e' = \{v'_i, v'_j\}$ forming a cross pattern. Then, we change the structure of G by adding new vertex v to V, deleting the two edges e and e' from E, and adding four new edges $\{v_i, v\}$, $\{v_j, v\}$, $\{v'_i, v\}$, $\{v'_j, v\}$ to E. Also we define $\mu(v)$ as the point of intersection between $\mu(e)$ and $\mu(e')$. Thus, we represent the point of intersection explicitly in the resultant diagram (G,μ). We do the same processing for all pairs of realized and unrealized edges forming a cross pattern.

Summarizing this idea, we get the next algorithm.

Algorithm 1.

```
Input: diagram $(G,\mu)$, where $G = (V,E)$ is a finite graph with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$.
Output: coarsest planar refinement of $(G,\mu)$.
Procedure:
— Initialization —
1.1. $P \leftarrow \{\mu(v) \mid v \in V\}$, and $k \leftarrow n$.
1.2. Construct the Delaunay triangulation $\mathcal{T}(P)$.
— Extraction of points of intersection —
2. While there exists a pair of realized edge $e = \{v_i, v_j\}$ and unrealized edge $e' = \{v'_i, v'_j\}$
   forming a cross pattern, do:
     begin
     $k \leftarrow k + 1$;
     $V \leftarrow V \cup \{v_k\}$;
     $E \leftarrow (E - \{e, e'\}) \cup \{\{v_i, v_k\}, \{v_j, v_k\}, \{v'_i, v_k\}, \{v'_j, v_k\}\}$;
     $\mu(v_k) \leftarrow$ the point of intersection between $\mu(e)$ and $\mu(e')$;
     $P \leftarrow P \cup \{\mu(v_k)\}$;
     construct the Delaunay triangulation $\mathcal{T}(P)$;
     end.
— Midpoint insertion —
3. If all the edges in $E$ are realized, go to Step 4. Otherwise, choose the longest unrealized
   edge $e = \{v_i, v_j\}$ and do:
     begin
     $k \leftarrow k + 1$;
     $V \leftarrow V \cup \{v_k\}$;
     $E \leftarrow (E - \{e\}) \cup \{\{v_i, v_k\}, \{v_k, v_j\}\}$;
     $\mu(v_k) \leftarrow$ midpoint of $\mu(e)$;
     $P \leftarrow P \cup \{\mu(v_k)\}$;
     construct the Delaunay triangulation $\mathcal{T}(P)$ and go to Step 2;
     end.
— Deletion of the midpoints —
4. For each vertex $v_k$ that is not in the original diagram and that is incident to exactly two
   edges $e = \{v_i, v_k\}$ and $e' = \{v_j, v_k\}$, do:
     begin
     $V \leftarrow V - \{v_k\}$;
     $E \leftarrow (E - \{e, e'\}) \cup \{\{v_i, v_j\}\}$;
     end.
5. Return $(G,\mu)$.                                                        □
```

An example is shown in Fig. 3.2: (a) shows a diagram composed of 60 vertices and 30 line segments; (b) is the Delaunay triangulation spanning the 60 vertices; (c) is the Delaunay triangulation at the final stage of the processing, where the points of intersection are represented by dots surrounded by circles; (d) represents the points of intersection found by the procedure. In this example, the number of points of intersection is 97, and the number of midpoints inserted is 446.

From this example, we can see several good properties of our new method. First, the density of inserted midpoints is not uniform. Midpoints are inserted more densely around the areas where line segments come closer to each other. This means that our method can change computational cost adaptively; higher computational cost is paid to places where the judgment of intersection is not easy. Secondly, the pairwise check is not done for two line segments which are far away from each other, because Delaunay triangles involving the end points of these line segments usually are not adjacent to each other. Thus, we are likely to be able to avoid $O(n^2)$ naive pairwise checks if the number of intersections is not large. Thirdly, the output of our method not only includes a collection of points of intersection but also includes the explicit description of the topological structure composed of the line segments and their intersections; this kind of structure is usually called the arrangement of the line segments. Moreover, even if numerical error takes place, the topological structure thus obtained is always consistent in the sense that it is embeddable in the plane without any extra points of intersection if we use curved line segments. This is because the output diagram (G, μ) is a subgraph of the Delaunay triangulation and we have a robust algorithm for constructing the Delaunay triangulation whose output is guaranteed at least to be embeddable in the plane [Sugihara and Iri, 1989c].

5. Conditions for the Finite Termination of Algorithm 1

In this section we study the behavior of Algorithm 1 in more detail, and clarify the conditions necessary for Algorithm 1 to terminate in a finite number of repetitions of Steps 2 and 3. Steps 2 and 3 are executed as many times as the number of midpoints inserted in Step 3. Hence, we need to check how many midpoints are inserted in the algorithm.

Let (G, μ) be the diagram given to Algorithm 1 as the input. We call edges in G *original edges*. In Algorithm 1, midpoints and points of intersection are inserted. If edge $e = \{v_i, v_j\}$ is divided into two edges $e' = \{v_i, v_k\}$ and $e'' = \{v_j, v_k\}$ in Step 2 or Step 3, we say that e' and e'' are *children* of e and e is the *parent* of e' and e''. If there is sequence $(e_1, e_2, \ldots, e_k)$ such that e_i is the parent of e_{i+1} for all $i = 1, 2, \ldots, k-1$, we say that e_1 is an *ancestor* of e_k and e_k is a *descendant* of e_1.

Let (G, μ) be a diagram, which may be the input diagram to Algorithm 1 or may be any diagram that appears in Step 2 or 3 as the result of modification of the original diagram by inserting points of intersection and/or midpoints. Edge e is said to be *stable* in diagram (G, μ), if a new vertex (midpoint or point of intersection) is not inserted in e any more in Step 2 or 3. Hence, if all the original edges are divided into stable descendants, Algorithm 1 terminates. Thus, what we want to show is that every edge is divided into stable descendants in a finite number of repetitions of Steps 2 and 3.

Vertex v of G is said to be *obtuse* either if v is of degree 1 or if any pair of line segments incident to v form the smaller angle equal to or greater than $\pi/2$, and *acute* otherwise. Fig. 5.1 shows examples of obtuse vertices and acute vertices.

Edge $e = \{v_i, v_j\}$ of G is said to be *clear*, if for any edge e' that is not adjacent to e, $\mu(e)$ and $\mu(e')$ do not intersect. For clear edge e, we define the *clearance* of e as the minimum distance from $\mu(e)$ to $\mu(e')$ over all edges e' not adjacent to e.

Similarly, a vertex v is called *clear* if for any edge e that is not adjacent to v, $\mu(e)$ does not pass through $\mu(v)$. For clear vertex v, the minimum distance from $\mu(v)$ to $\mu(e)$ for any edge e that is not incident to v is called the *clearance* of v.

Lemma 1. Suppose that e is a clear edge with clearance δ and that both of the endpoints of e are obtuse. Then, every descendant of e with length smaller than δ is stable.

Proof. A direct consequence from Property 1. □

For example, suppose that the diagram (G, μ) is a collection of line segments that do not have any point of intersection. Then, every edge is clear. Moreover, by a finite number of midpoint insertions, we can make every edge shorter than its clearance. Hence, after a finite number of repetitions of Steps 2 and 3, Algorithm 1 terminates and tells us that there is no point of intersection.

Thus, we need to consider only those edges e such that $\mu(e)$ has a point of intersection with other line segments in $L(G,\mu)$ or such that $\mu(e)$ has an adjacent edge meeting with $\mu(e)$ in an acute angle.

Lemma 2. Suppose that vertex v is of degree-two, acute and clear with clearance δ. After a finite number of midpoint insertions on edges incident to v, both of the edges incident to v become stable.

Proof. Let e_1 and e_2 be the two edges incident to v, and θ be the smaller angle formed by e_1 and e_2. As shown in Fig. 5.2, let C be the circle with radius $\delta/2$ that is tangent to e_2 at v and that has another point of intersection, say p, with e_1. Let q be the center of C. Then, we get $\angle pqv = 2\theta$, and hence

$$|\bar{l}(v,p)| = \delta \tan\theta. \tag{5.1}$$

C does not intersect any edge other than e_1 and e_2, and it intersects e_2 at v only. Hence, any descendant $\{v, v'\}$ of the edge e_1 whose length is less than $\delta\tan\theta$ is realized. Thus, the descendant of e_1 that is incident to v becomes stable at latest when its length becomes smaller than $\delta\tan\theta$. Interchanging the roles of e_1 and e_2, we can prove the other half of the lemma similarly. □

For two real-value functions $f(x)$ and $g(x)$, we write $f(x) = \mathrm{o}(g(x))$ if there exists constant C such that

$$\limsup_{x\to 0} \left|\frac{f(x)}{g(x)}\right| \le C. \tag{5.2}$$

Lemma 3. Suppose that v is a degree-two clear acute vertex with angle θ and clearance δ. The number of midpoints inserted within distance δ from $\mu(v)$ is at most of $\mathrm{o}(-\log\theta)$.

Proof. From the argument in the proof of Lemma 2, we see that the edge incident to v becomes stable at latest when its length becomes smaller than $\delta\tan\theta$. Hence, the number of midpoints that should be inserted within distance δ from $\mu(v)$ to make the edge stable is bounded by the smallest integer $\gamma = \gamma^*$ that satisfy

$$2\delta\left(\frac{1}{2}\right)^{\gamma} < \delta\tan\theta. \tag{5.3}$$

For small $|\theta|$ we get $\tan\theta \approx \sin\theta \approx \theta$. Hence, $\gamma^* = \mathrm{o}(-\log\theta)$. □

We say that two edges e and e' form an *implicit intersection pair* if e and e' are not adjacent to each other in G and $\mu(e)$ and $\mu(e')$ have a point of intersection. The point of intersection of $\mu(e)$ and $\mu(e')$ is said to be *clear* if (i) $\mu(e)$ and $\mu(e')$ are not collinear, (ii) the point of intersection does not coincide with an endpoint of $\mu(e)$ or $\mu(e')$, (iii) there is no other edge e'' such that $\mu(e'')$ passes through the point of intersection. For a clear point of intersection between $\mu(e)$ and $\mu(e')$, the minimum distance from the point of intersection to $\mu(e'')$ for $e'' \in E - \{e, e'\}$ is called the *clearance* of the point of intersection.

Lemma 4. Suppose that e and e' are two edges forming an implicit intersection pair and that their point of intersection is clear. After a finite number of inserting midpoints on the edges containing the point of intersection, Algorithm 1 creates vertex v corresponding to the point of intersection. Moreover, all the four edges adjacent to v becomes stable in a finite number of further repetitions of Steps 2 and 3.

Proof. Suppose that e and e' represent the original edge given in Lemma 4 or their descendants in the current diagram (G, μ) containing the point of intersection. Let $e = \{v_1, v_2\}$ and $e' = \{v_3, v_4\}$, and let θ be the smaller angle formed by two line segments $\mu(e)$ and $\mu(e')$.

Without loss of generality, let us assume that $\mu(v_1) = (0,0)$, $\mu(v_2) = (a,0)$, $\mu(v_3) = (b + c\cos\theta, c\sin\theta)$ and $\mu(v_4) = ((b - (d-c)\cos\theta, -(d-c)\sin\theta)$, as shown in Fig. 5.3, where $a = |\mu(e)|$, $d = |\mu(e')|$, $0 \le b \le a$ and $0 \le c \le d$. Let C_{ijk} denote the circle passing through $\mu(v_i)$, $\mu(v_j)$ and $\mu(v_k)$ for $1 \le i < j < k \le 4$. We show that at least one of the four circles $C_{123}, C_{124}, C_{134}, C_{234}$ will become an empty circle in a finite number of midpoint insertions. The circle C_{123} is represented by

$$H_{123}(x,y) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 1 & a & 0 & a^2 \\ 1 & b + c\cos\theta & c\sin\theta & (b + c\cos\theta)^2 + (c\sin\theta)^2 \\ 1 & x & y & x^2 + y^2 \end{vmatrix} = 0. \tag{5.4}$$

Rewriting this, we get

$$\begin{aligned} H_{123}(x,y) &= (ac\sin\theta)(x^2 + y^2) - (a^2 c\sin\theta)x \\ &\quad + (-a(b + c\cos\theta)^2 - (c\sin\theta)^2 + a^2(b + c\cos\theta))y \\ &= ac\sin\theta\Big[\left(x - \frac{a}{2}\right)^2 + \left(y - \frac{b^2 + c^2 + 2bc\cos\theta - ab - ac\cos\theta}{2c\sin\theta}\right)^2 \\ &\quad - \frac{a^2}{4} - \frac{(b^2 + c^2 + 2bc\cos\theta - ab - ac\cos\theta)^2}{4c^2\sin^2\theta}\Big]. \end{aligned}$$

Hence, the radius, say r_{123}, of the circle $C_{123}$is represented by

$$r_{123}^2 = \frac{a^2}{4} + \frac{1}{4\sin^2\theta}(c + \frac{b^2 - ab}{c} + (2b - a)\cos\theta)^2. \tag{5.5}$$

By replacing c by $d - c$ and θ by $-\theta$, we get the radius r_{124} of the circle C_{124}:

$$r_{124}^2 = \frac{a^2}{4} + \frac{1}{4\sin^2\theta}(d - c + \frac{b^2 - ab}{d - c} - (2b - a)\cos\theta)^2. \tag{5.6}$$

We want to find an upper bound, say α, of the smaller value of r_{123} and r_{124}. However, equation $r_{123} = r_{124}$ gives a degree-three equation in c, which is difficult to solve. Hence, we get an upper bound using the principle that the smaller of the two values is not greater than a weighted average of the two values. Let us define

$$A = c + \frac{b^2 - ab}{c} + (2b - a)\cos\theta, \tag{5.7}$$

$$B = d - c + \frac{b^2 - ab}{d - c} - (2b - a)\cos\theta. \tag{5.8}$$

We get the weighted average with weights c/d and $(d-c)/d$ by

$$\frac{c}{d}A + \frac{d-c}{d}B = \frac{1}{d}\left(c^2 + (d-c)^2 + 2(b^2 - ab) + (2c-d)(2b-a)\cos\theta\right).$$

The right-hand expression is quadratic in b and in c, and hence the maximum value is attained at $b = 0$ or $b = a$ and $c = 0$ or $c = d$. Comparing these values, we get

$$\max_{0\le b\le a}\ \max_{0\le c\le d}\left(\frac{c}{d}A + \frac{d-c}{d}B\right) = d + a\cos\theta, \tag{5.9}$$

and the right-hand value is attained when $b = a$ and $c = d$, or when $b = 0$ and $c = 0$. Thus, we get

$$\begin{aligned}\alpha^2 &= \max_{0\le b\le a}\ \max_{0\le c\le d}\ \min\{r_{123}^2, r_{124}^2\}\\ &= \max_{0\le b\le a}\ \max_{0\le c\le d}\left(\frac{a^2}{4} + \frac{1}{4\sin^2\theta}\min\{A^2,B^2\}\right)\\ &\le \max_{0\le b\le a}\ \max_{0\le c\le d}\left(\frac{a^2}{4} + \frac{1}{4\sin^2\theta}\left(\frac{c}{d}A + \frac{d-c}{d}B\right)^2\right)\\ &= \frac{a^2}{4} + \frac{(d+a\cos\theta)^2}{4\sin^2\theta}.\end{aligned} \tag{5.10}$$

Recall that θ is fixed, and that a and d become half of their previous values when the midpoints are inserted on $\mu(e)$ and $\mu(e')$ respectively. Hence, after a finite number of midpoint insertions, α becomes smaller than $\delta/2$ and hence one of the circles C_{123} and C_{124} contains no points in $P - \{\mu(v_1), \mu(v_2), \mu(v_3), \mu(v_4)\}$. By a similar argument we can see that by a finite number of midpoint insertions, one of the circles C_{134} and C_{234} contains no points in $P - \{\mu(v_1), \mu(v_2), \mu(v_3), \mu(v_4)\}$. Since $\mu(v_3)$ and $\mu(v_4)$ are in mutually opposite sides of $\mu(e)$ and $\mu(v_1)$ and $\mu(v_2)$ are in mutually opposite sides of $\mu(e')$, we see the following: if $\mu(v_4)$ is inside C_{123}, then $\mu(v_3)$ is inside C_{124} and $\mu(v_1)$ is outside C_{234} and $\mu(v_2)$ is outside C_{134}; on the other hand, if $\mu(v_4)$ is outside C_{123}, then $\mu(v_3)$ is outside C_{124} and $\mu(v_1)$ is inside C_{234} and $\mu(v_2)$ is inside C_{134}. In any case, the smaller circle of C_{123} and C_{124} or the smaller circle of C_{134} and C_{234} does not contain any point in its interior, and hence one of the four triangles is a Delaunay triangle, which means that e and e' form a cross pattern and consequently the point of intersection is detected and inserted by Algorithm 1.

The latter half of the lemma can be proved in the same manner as Lemma 2, because once the point of intersection is inserted, the four edges incident to this vertex can be considered as a compound of two acute vertices and the proof of Lemma 2 can be applied to each of them separately. □

Lemma 5. Suppose that e and e' are two edges forming an implicit intersection pair and their point of intersection is clear, with clearance δ. Let θ be the smaller angle formed by $\mu(e)$ and $\mu(e')$. The number of midpoints inserted to the descendants of e and e' containing the point of intersection is of $o(-\log\theta)$.

Proof. Let γ be the smallest integer satisfying

$$\frac{|\mu(e)|^2}{4}\left(\frac{1}{2}\right)^{2\gamma} + \frac{1}{4\sin^2\theta}\left(|\mu(e')|\left(\frac{1}{2}\right)^{\gamma} + |\mu(e)|\cos\theta\left(\frac{1}{2}\right)^{\gamma}\right)^2 \le \left(\frac{\delta}{2}\right)^2. \tag{5.11}$$

In the proof of Lemma 4, we saw that the number of points inserted by the time when the point of intersection is detected is not larger than γ. Thus, we get

$$\left(\frac{1}{2}\right)^{2\gamma}\left(\frac{|\mu(e)|^2}{4}+\frac{(|\mu(e')|+|\mu(e)|\cos\theta)^2}{4\sin^2\theta}\right)\leq\left(\frac{\delta}{2}\right)^2, \tag{5.12}$$

$$\gamma \geq \frac{1}{2\log 2}\left(\log\left(\text{const.}+\frac{\text{const.}}{\sin^2\theta}\right)-2\log\delta+2\log 2\right)=\mathrm{o}(-\log\theta). \tag{5.13}$$

From this inequality and Lemma 3 we get Lemma 5. □

From Lemmas 1, 2 and 4 we get the next theorem.

Theorem 1. Algorithm 1 terminates in finite steps if the following conditions are satisfied: (i) in the original diagram (G,μ) every vertex is obtuse or is incident to at most two edges, (ii) no two line segments are collinear, (iii) every point of intersection lies on exactly two line segments, and (iv) no numerical error takes place in the course of computation.

However, these conditions are not practical, firstly because we usually do not know how many line segments lie on each point of intersection before we find the point of intersection, secondly because as shown by Lemmas 3 and 5, the time complexity is not so small if two lines come close to be collinear, and thirdly because it is very difficult to avoid numerical errors in actual computation. Hence, in the next section we will revise the algorithm into a robust one in the sense that it is valid even if all the above conditions are removed.

6. Robust Method

We will revise Algorithm 1 into a robust one. As stated in Theorem 1, many conditions are required for Algorithm 1 to be valid. Here we assume that the input does not necessarily satisfy those conditions. The only assumption we pose on the diagram (G,μ) is

(A1) G is a finite graph without selfloops or parallel edges.

We place no assumption on the mapping μ, and hence various kinds of degeneracy are allowed; for example, any number of vertices in G may fall on the same point in the plane, and any number of line segments may have a common implicit point of intersection.

By numerical computation we mean computation involving floating-point numbers. Let $\hat{+}$, $\hat{-}$, $\hat{\times}$, $\hat{\div}$ be imprecise operations corresponding to precise operations $+$, $-$, $\times$, $\div$. We place the following assumptions on numerical computation.

(A2) The results of imprecise operations $\hat{+}$, $\hat{-}$, $\hat{\times}$, $\hat{\div}$ are in general different from the results of precise operations $+$, $-$, $\times$, $\div$.

(A3) The comparison $x=y$ or $x<y$ can be done precisely.

(A4) The imprecise addition $\hat{+}$ is monotone in the sense that for any $x,y\geq 0$, $x\hat{+}y\geq x$ and $x\hat{+}y\geq y$.

(A5) Distance $|\bar{l}(p_i,p_j)|$ computed in imprecise arithmetic is nonnegative.

Actually (A2) is not an assumption; it simply states explicitly that we do not expect correctness of these operations. We do not assume any bound of numerical error; hence, for example, the difference between $x+y$ and $x\hat{+}y$ can be arbitrarily large. In actual computation we can of course expect some bound of numerical error, and consequently can expect a meaningful output up to the precision at hand. However, we do not explicitly assume any bound because such a bound is not necessary for the proof of the robustness of our algorithm. On the other hand, we assume the correctness of comparison, monotonicity of addition of positive numbers and nonnegativity of the distance, as stated in (A3), (A4) and (A5). Note that these properties are usually satisfied by actual floating-point computation. Our goal is to design an algorithm for finding points of intersection that always terminates and gives a topologically consistent output in the world with assumptions (A1) $\sim$ (A5).

First of all there already exists a robust algorithm for constructing the Delaunay triangulation [Sugihara and Iri, 1989b, 1989c]. In that algorithm, the basic procedure is described in terms of combinatorial computation and numerical values are employed only in order to select the most promising branch of the procedure. Consequently, the algorithm is robust in the sense that no matter how poor the precision in computation may be, it terminates in finite steps and gives an output whose topological structure is at least a planar triangular graph. So we use this algorithm as a tool for constructing our robust intersection algorithm.

In order to guarantee termination of our algorithm, we need to guarantee finiteness of the number of points of intersection and the number of midpoints inserted in the processing. To restrict the number of points of intersection being finite, we place the next constraint.

(C1) The point of intersection is generated at most once for each pair of line segments.

Hence, if the descendants of two edges form cross patterns twice or more, we consider that they are due to numerical errors and ignore the second and later appearances of the cross patterns.

To guarantee finiteness of the number of midpoints, we intuitively place the following constraint. We prespecify a small positive real number, say ε, and once a descendant of an edge becomes shorter than ε, we do not insert the midpoint any more. It might seem that the finiteness of the number of midpoints is obviously guaranteed by this constraint. However, we must be a little more careful because the coordinates of the midpoints and the lengths of the resultant edges cannot necessarily be computed precisely.

For each edge e of the original diagram (G, μ), we associate number $\beta(e)$ which is defined as the smallest integer satisfying

$$|\mu(e)| \left(\frac{1}{2}\right)^{\beta(e)} < \varepsilon. \tag{6.1}$$

$\beta(e)$ is the minimum number such that the $\beta(e)$-th descendant of e has length smaller than ε. An edge is said to be *saturated* if its ancestors suffer $\beta(e)$ times of midpoint insertions, and *unsaturated* otherwise. An edge in general is created as a result of insertion of midpoints and points of intersection, among which we count only the number of midpoint insertions; if this number reaches $\beta(e)$, we do not insert the midpoint on the edge any more. In other words, we place the following constraint:

(C2) The midpoints are inserted only to unsaturated edges.

Note that (C2) guarantees the finiteness of the number of midpoints inserted on original edge e. Since numerical error takes place, the computed value of $\beta(e)$ may not be correct. However, once we get an integer $\beta(e)$ (even if it is not correct), we can count the depth of descendants by integer computation and hence can bound the number of midpoints inserted in the procedure.

Thus, employing (C1) and (C2), we can guarantee the finite termination. However, as the cost of this, we have the possibility of having unrealized edges when the algorithm terminates.

So the next question is how to deal with these unrealized edges.

When the algorithm terminates, we have refinement (G', μ') of the original diagram (G, μ), and the associated triangulation T. Because of numerical errors T is not necessarily the Delaunay triangulation spanning vertices in (G', μ'), but at least the topological structure of T is a planar triangular graph. So, for each unrealized edge $e = \{v_i, v_j\}$ (if it exists) we find the shortest path in T connecting $\mu(v_i)$ and $\mu(v_j)$; let us call this the *shortest Delaunay path* connecting $\mu(v_i)$ and $\mu(v_j)$. We replace e by this shortest Delaunay path.

For example, suppose that three line segments have a common point of intersection and that the Delaunay triangulation at the final stage of the midpoint insertion is as shown in Fig. 6.1(a), where bold lines represent realized descendants of the line segments and thin lines represent other Delaunay edges. All the three line segments have gaps near the point of intersection, because the descendants containing the point of intersection are not realized. Replacing such unrealized edges by the Delaunay shortest paths, we get the planar diagram as shown in (b).

It is known that the shortest Delaunay path is not so bad as an approximation to the direct route connecting two end points [Keil and Gutwin, 1989] [Dobkin, Friedman and Supowit, 1990]. Hence, we can expect that this replacement gives a not-so-bad approximation of the original line segment. As a result of this, we have the diagram whose edges are all realized in the triangulation T. Thus, the final diagram is a substructure of T, and consequently is a planar diagram.

Summarizing all the above consideration, we get the next algorithm, where the descriptions of Steps 2, 3 and 5 are simplified because they are almost the same as Steps 2, 3 and 4, respectively, of Algorithm 1.

Algorithm 2 (robust version)

```
Input: diagram $(G,\mu)$, where $G = (V, E)$ is a finite graph with vertex set $V = \{v_1, v_2, \ldots,$
        $v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$, and positive number $\varepsilon$.
Output: approximation of the coarsest planar refinement of $(G, \mu)$.
Procedure:
— Initialization —
1.1. For every pair of vertices $v_i$ and $v_j$ such that $\mu(v_i) = \mu(v_j)$, merge them into one vertex;
   that is, do:
      begin
      $V \leftarrow V - \{v_j\}$;
      replace every appearance of $v_j$ in the list of edges by $v_i$;
      if there is an edge $e = \{v_i, v_j\}$, $E \leftarrow E - \{e\}$;
      end.
1.2. Construct the Delaunay triangulation spanning $\{\mu(v) \mid v \in V\}$.
— Extraction of points of intersection —
2. For each pair of realized edge $e$ and unrealized edge $e'$ forming a cross pattern such that the
   point of intersection between the ancestor of $e$ and that of $e'$ has not yet been generated,
   do:
      generate new vertex $v$ representing the point of intersection, and
      modify the Delaunay triangulation by adding $v$.
— Midpoint insertion —
3. If all the unrealized edges are saturated, go to Step 4. Otherwise, choose an unsaturated
   unrealized edge, say $e$, and do:
      insert the midpoint $v$ to $e$, and
      modify the Delaunay triangulation by adding $v$, and go to Step 2.
```

— Approximation of unrealized edges —
4. For each unrealized edge $e = \{v_i, v_j\}$, replace it by the shortest Delaunay path connecting $\mu(v_i)$ and $\mu(v_j)$.
— Deletion of the midpoints —
5. Delete all the vertices that are incident to exactly two edges and that are not in the original diagram, and return the resultant diagram. □

Step 1.1 need comparison between the coordinates of vertices, which can be done correctly because of (A3). Step 1.2 can be done by the robust method given by Sugihara and Iri (1989b, 1989c). In Steps 2 and 3, we employ the constraints (C1) and (C2), respectively, so that the finiteness of the number of repetitions of these steps is guaranteed. Note that unlike Algorithm 1 we need not insert the midpoint to the longest edge first, because the termination is guaranteed by the constraint (C2).

In Step 4 we can apply any existing shortest-path algorithm. However, in order to see the finite termination, let us explicitly state the algorithm. The next is the simplest algorithm for finding the shortest path between two specific points v_s and v_t. In this algorithm, $c(v)$ denotes the length of the shortest path from v_s to v among the paths traced so far, Q is a stack to store vertices from which we expand the shortest paths, and back(v) denotes the last vertex lying on the path from v_s to v.

Algorithm 3 (shortest path)

Input: diagram (G,μ), where $G = (V,E)$ is a finite graph having vertex set V and edge set E, and two specific vertices $v_s, v_t \in V$
Output: shortest path between v_s and v_t.
Procedure:

```
1. $c(v_s) \leftarrow 0$; $c(v) \leftarrow \infty$ for all $v \in V - \{v_s\}$; $Q \leftarrow \{v_s\}$.
2. Repeat 2.1, 2.2 and 2.3.
2.1. Choose and delete from $Q$ vertex $v$ with the smallest $c(v)$.
2.2. If $v = v_t$, go to Step 3.
2.3. For every edge $e = \{v, v_j\}$ incident to $v$, if $c(v_j) > c(v) + |\mu(e)|$, then
       begin
       $c(v_j) \leftarrow c(v) + |\mu(e)|$; back($v_j$) $\leftarrow v$; $Q \leftarrow Q \cup \{v_j\}$;
       end
3. $i \leftarrow 0$; $v_i \leftarrow v_t$;
   while back($v_i$) $\neq v_s$ do
     $v_{i+1} \leftarrow$ back($v_i$); $i \leftarrow i + 1$;
   return $(v_s, v_k, v_{k-1}, \dots, v_1, v_t)$ (where back($v_k$) $= v_s$).
```
□

Step 1 is for initiation, in which the length from v_s to itself is set 0, the lengths from v_s to other vertices v are set ∞, and Q is initialized as the set consisting of the start vertex v_s. In Step 2, the vertex v with the smallest $c(v)$ is chosen (Step 2.1), and if it is the terminal vertex v_t, we have found the shortest path and so go to Step 3 (Step 2.2); otherwise we expand the path from v to the adjacent vertices (Step 2.3). Step 3 is for postprocessing to get the list of vertices on the shortest path. For the correctness of this algorithm, see standard textbooks [Knuth, 1973].

We can guarantee the finite termination of Algorithm 3 by showing that any vertex which is once put in Q in Step 1 or Step 2.3 and is deleted from Q in Step 2.1 will never be put in Q again. Indeed, from (A3) the vertex v with the smallest $c(v)$ is correctly chosen and deleted from Q in Step 2.1, and from (A4) and (A5) the computed value of $c(v_j)\hat{+}|\mu(e)|$ in Step 2.3 is not smaller than $c(v_j)$. Hence, after each repetition of Steps 2.1 $\sim$ 2.3, no element v' in Q has the value of $c(v')$ smaller than $c(v)$, which means that the minimum value of $c(v)$ over all elements v in Q never decreases in the execution of Algorithm 3. Therefore, any vertex which is once deleted from Q will never be put in Q again. Thus, the finite termination of Algorithm 3 is guaranteed in the assumption (A3) $\sim$ (A5).

On the other hand, if these assumptions do not hold, the finite termination of Algorithm 3 is not guaranteed. Indeed, if (A4) or (A5) does not hold, the length of a path may decrease as we walk farther along a path, and hence the shortest-path finding procedure may fall into an endless loop.

The assumptions (A3), (A4) and (A5) are satisfied by most of actual computers. Hence, we need not worry about unrobustness due to numerical error when we implement Algorithm 3; the straightforward implementation usually gives a valid computer program.

Let us go back to Algorithm 2. Step 5 of Algorithm 2 does not employ numerical computation, so that it can be done always in finite steps.

Thus, Algorithm 2 terminates in finite steps. Also the output diagram is topologically consistent in the sense that it can be embedded in the plane if we use curved line segments because the output diagram is a subgraph of the final triangulation which is planer.

In Step 2 we have to find cross patterns. For this purpose we use a queue, say Q. When the initial Delaunay triangulation is constructed in Step 1.2, all the Delaunay edges are stored in Q. In Step 2 we choose and delete an edge e from Q one by one and examine the (at most two) Delaunay triangles containing e to check whether the other vertex of the triangle has an edge which together with e forms a cross pattern. New Delaunay edges generated in Steps 2 and 3 are added to Q, and old Delaunay edges which are in Q and which disappear as the result of modification of the Delaunay triangulation in Steps 2 and 3 are deleted from Q.

Let m be the number of edges in the original diagram (G, μ), and let l and k be the number of points of intersection and that of midpoints generated in Algorithm 2. If we can regard that the vertices are located almost at random, we may expect the following average time complexity. Step 1.1 can be done in $O(m)$ time by the bucketing technique [Ohya, Iri and Murota, 1984]. Step 1.2 is done in $O(n)$. If points are located almost at random, we can expect that the modification of the Delaunay triangulation for addition of one new point requires only constant time. Hence, the time required for Step 2 is proportional to the number of Delaunay edges ever stored in Q, which is of $O(n+l+k)$. Step 3 is also done in $O(n+l+k)$. Step 4 is for the exceptional case, so that we can expect that Step 4 is usually skipped. Step 5 is done in $O(k)$ time. Thus, the average time complexity can be expected to be of $O(N)$, where $N = n + m + l + k$.

The time complexity of Algorithm 2 depends not only on the input and output size (i.e., $n + m + l$) but also on k, the number of midpoints inserted in Step 3. We can bound k by $k \leq \sum_{e \in E}(|\mu(e)|/\varepsilon)$, where E is the edge set of the original diagram, but this bound is usually an awful overestimation. An actual value of k depends on the distances between mutually nonintersecting line segments and the angles between mutually intersecting line segments; k becomes large if the distances and the angles become small. Thus, the time complexity of the present method depends on "degree of difficulty" of the problem. This property seems reasonable particularly when we design an algorithm that is robust and stable in imprecise arithmetic.

It may be difficult to know an appropriate value of ε. In that case we can take the following alternative. Instead of giving the minimum length ε of edges to be divided, we give the maximum number, say K, of midpoints to be inserted, and instead of choosing an arbitrary edge in Step 3 we choose the longest edge. If the number of midpoints generated in Step 3 becomes K, we stop the repetition of Steps 2 and 3 and go to Step 4. By this modification we can bound the average time complexity in terms of the input and output size and K; if we use a priority queue to extract the longest unrealized edge, the time complexity will be of $O(M \log M)$ where $M = n + m + l + K$.

In this section we placed emphasis on maintaining topological consistency. However, this does not mean that numerical computation is unimportant. Careful choice of the ways of computing numerical values will improve the performance of the algorithm. For this reason, we need to be a little careful in the computation of points of intersection and midpoints. The locations of the original line segments are specified by the coordinates of the endpoints. The original line segments are decomposed into smaller segments, and they are likewise specified by the coordinates of the endpoints. However, the points of intersection and the midpoints should be computed from the coordinates of the endpoints of the original line segments. Otherwise, numerical errors will be accumulated. The point of intersection of two line segments can be computed as the point of intersection of their original ancestors. For the computation of the midpoint of a line segment, we keep the history of how the line segment is generated from the original ancestor and compute the point on the original segment that corresponds the midpoint of the present descendant. Numerical computation related with the construction of the Delaunay triangulation was discussed in [Sugihara and Iri, 1989b].

7. Computational Experiments

Algorithm 2 was implemented in a FORTRAN program, in which VORONOI2 [Sugihara and Iri, 1989c], a robust incremental method for constructing Voronoi diagrams, was employed for the construction of the Delaunay triangulation. The FORTRAN code consisted of about 4500 lines, including routines for generating input data and those for drawing the output. The experiments were done in Sequent Symmetry Multiprocessor System with UNIX operating system and with FORTRAN compiler FORTRAN 77 V3.2 by Silicon Valley Software Inc. All numerical computations were done in single-precision floating-point arithmetic. The following are examples of the behavior of this computer program.

In the first example, a collection of ten line segments shown in Fig. 7.1(a) was given as an input. The ten line segments were first generated in such a way that they had a common point of intersection at the center, and next they were perturbed slightly by random numbers. The line segments are generated in the region $\{(x, y) \mid 0 \leq x, y \leq 1\}$, and the square drawn in the figure represents the region $\{(x, y) \mid -0.1 \leq x, y \leq 1.1\}$.

First, we saw the effect of the value of input ε, the threshold such that unrealized line segments are divided only when their lengths are greater than ε. Fig. 7.1(b) shows the Delaunay triangulation obtained at the final stage of the algorithm for $\varepsilon = 0.1$; the length of ε is represented by the horizontal line segment near the right lower corner of the figure. In this figure, the solid lines represent Delaunay edges and the broken lines represent unrealized descendants of the input line segments. Fig. 7.1(b') shows the final output, where each point of intersection is represented by a dot surrounded by a circle.

Fig. 7.1(c) and (c') show the Delaunay triangulation at the final stage and the output, respectively, for $\varepsilon = 0.01$.

If we magnify the central region in (a) by 100, we get the diagram as shown in (d). The final Delaunay triangulations and the outputs for $\varepsilon = 0.002$, $\varepsilon = 0.001$ and $\varepsilon = 0.0001$ magnified by the same factor are shown in (e) and (e'), (f) and (f'), and (g) and (g'), respectively. For $\varepsilon = 0.0001$, all the descendants of the input line segments were realized in the Delaunay triangulation in (g) and hence all the points of intersection were found in (g').

The number of points generated in the algorithm (i.e., the total number of midpoints and points of intersection) were 103 for $\varepsilon = 0.1$, 194 for $\varepsilon = 0.01$, 256 for $\varepsilon = 0.002$, 281 for $\varepsilon = 0.001$, and 300 for $\varepsilon = 0.0001$.

From this example, we can see that the value of ε can control the resolution of the output; larger ε gives lower resolution and smaller ϵ gives higher resolution.

In the next example, we saw the effect of numerical errors. In the computer program many floating-point computations were done. Using random numbers, we added artificial errors to the result of floating-point computation. Let a be the result of a certain floating-point computation. We replaced a by

$$a' = (1 + \frac{\delta}{100} r)a, \tag{7.1}$$

where r is a random number uniformly distributed in $[-1, 1)$ and $\delta/100$ is a constant representing the maximum of the relative error. When a is replaced by a' according to equation (7.1), we say that δ % numerical error is added to a.

The input to the computer program is the same set of line segments as in the previous example, i.e., the ten line segments in Fig. 7.1(a). Fig. 7.2(a) and (a') show the Delaunay triangulation and the output obtained when 5% numerical errors were added to the computation of midpoints and to the computation of the points of intersection. In this example, Step 5 of Algorithm 2 was not executed, and hence the midpoints inserted by the algorithm remain in the figure. This is because we can see the effect of numerical error more clearly. In this case, the Delaunay triangulation was computed without any artificial error, so that the resultant diagram (a') has no implicit point of intersection.

Fig. 7.2(b) and (b') show the result obtained when 5% numerical errors were added to the computation of the Delaunay triangulation (in which the floating-point computation was used for two purposes; one is to compute the distance between two points and the other is to judge whether a circle passing through three points includes the fourth point in its interior). Because the Delaunay triangulation was not computed correctly, the output diagram (b') has many implicit points of intersection. However, it is assured that the graph structure of the triangulation in (b) is planar [Sugihara and Iri, 1989b], and hence the diagram in (b'), which is a subgraph of (b), can be embedded in the plane if we use curved line segments. In this sense the output is topologically consistent.

Fig. 7.2(c) and (c') show the result obtained when 100% numerical errors were added to both the computation of midpoint and intersection and the computation of Delaunay triangulation. 100% numerical errors mean that numerical computation were almost meaningless, but still our computer program gave some output. Even when 500% or 1000% numerical errors were added, the program carried out its task and gave some output, though the output was almost nonsense.

What should be noted in this example is that no matter how large numerical errors take place, the algorithm does not come across any inconsistency and hence it always gives some output that is at least topologically consistent.

In this experiment, numerical errors were not added in the stage of finding the shortest paths, i.e., in Step 4 in Algorithm 2. This is because the termination of the shortest path algorithm is not guaranteed if the sum of the lengths of two edges becomes shorter than the lengths of the original edges. However, this does not mean that the shortest path cannot be found robustly. As we saw in the previous section, the shortest path can be found robustly in the assumptions (A3), (A4) and (A5). We skipped adding numerical error in this step simply because the error generated by equation (7.1) does not satisfy (A4) and (A5).

Another example is shown in Fig. 7.3. The input was a collection of 30 line segments shown in (a). This set of line segments was generated in such a way that first 30 line segments meeting at the center were generated at random and next their endpoints were perturbed at random. The outputs for $\varepsilon = 0.1$, for $\varepsilon = 0.02$ and for $\varepsilon = 0.0001$ are shown in (b), (c) and (d), respectively. In (b) and (c) the extracted points of intersection are not shown explicitly (they are represented by dots with circles in other figures); this is because we want to show the microstructure more clearly. The number of unrealized edges was 236 in (b), 181 in (c) and 0 in (d), and the number of generated points was 359 in (b), 1123 in (c) and 1537 in (d). The number of points of intersection found in (d) was 375.

Fig. 7.4(a) shows a polygonal line connecting 50 points (and hence the number of line segments is 49). This diagram was generated by locating 50 points at random and connecting them sequentially. The output for $\varepsilon = 0.3$, for $\varepsilon = 0.05$ and for $\varepsilon = 0.0001$ are shown in (b), (c) and (d), respectively. The number of unrealized edges was 120 in (b), 77 in (c) and 19 in (d), and the number of generated points was 247 in (b), 810 in (c) and 1376 in (d).

The last example is shown in Fig. 7.5. The input was a collection of ten line segments; they were generated so that they met at the center up to numerical error due to single-precision floating-point expression. The output for $\varepsilon = 2 \times 10^{-6}$ is shown in (a). If we magnify the central region of this figure by 10^5, we get the figure shown in (b). The output at the same central region for $\varepsilon = 10^{-6}$ and for $\varepsilon = 10^{-8}$ are shown in (c) and (d), respectively. In this example, Step 5 of Algorithm 2 was skipped, and hence the midpoints inserted in the algorithm remain in the figure. The number of unrealized edges was 41 in (a), 40 in (c) and 39 in (d), and the number of generated points was 814 in (a), 890 in (c) and 1065 in (d). Even if smaller value of ε (i.e., $\varepsilon = 10^{-9}$ or $\varepsilon = 10^{-10}$) was used, no new midpoint or point of intersection was generated. This is because the midpoint of any unrealized edge computed in single precision coincided with one of the endpoints of the edge, and hence no new point was created in the stage of the midpoint insertion. In this sense, the result shown in (d) seems the best possible that we can obtain in singe-precision arithmetic.

8. Concluding Remarks

We have constructed a new algorithm for finding points of intersection of line segments in the plane. The algorithm first subdivides the plane into the Delaunay triangulation spanning the end points of the line segments, and next refines the subdivision by inserting midpoints recursively until the areas containing points of intersection are sufficiently localized.

This algorithm has many good properties. First, the algorithm is robust; no matter how poor the precision in computation may be, the algorithm executes the task without facing inconsistency and gives an output that is topologically consistent in the sense that the output diagram is always embeddable in the plane. Secondly, the algorithm is stable; it does not miss pairs of line segments that can be detected easily by pairwise check in the precision at hand. Thirdly, the algorithm is simple; it does not have any exceptional branch of processing for degenerate or nearly degenerate cases. Forthly, the algorithm can pay its const adaptively. The algorithm terminates quickly if the input is simple in the sense that no two line segments are near to collinear, no mutually nonintersecting line segments come close to each other, and no two points of intersection come close to each other. If the input is not simple, on the other hand, the algorithm spends much time in the areas where the situations are not simple. This property seems reasonable as the behavior of an algorithm in the world with numerical errors.

We cannot bound the time complexity of the present algorithm in terms of the number of the input line segments and the number of points of intersection, because the number of midpoints inserted in the algorithm depends on the distances between nonintersecting line segments and the angles between mutually intersecting line segments. However, it is not fair to conclude that the present algorithm is less efficient than the existing "efficient" algorithms, because the correctness of those algorithms is guaranteed only in precise arithmetic whereas the present algorithm is valid in the world with numerical errors. We need some new concept to measure the complexity of algorithms that take numerical errors into account.

The basic idea presented in the paper also has some alternative ways of implementation.

First, in our algorithm new vertices are placed at the midpoints of unrealized edges. This is because this strategy can be easily converted into a numerically robust one. Other strategies for placing new points are studied for the domain Delaunay triangulation problem [Boissonnat, 1988] [Schroeder and Shephard, 1988] [Sapidis and Perucchio, 1990]. They might be useful for our problem in order to decrease the number of new points. However, these strategies are designed basically for precise arithmetic, and how to implement them in a numerically robust manner is one of our future problems.

Second, we consider all of the given line segments from the beginning. Another method may be an incremental one, in which we start with a single line segment and add other line segments one by one; at each addition of a line segment, we find or localize all the points of intersection between the line segments added so far. The incremental method may have advantage for degeneracy; for example, even if three lines have a common point of intersection, the degeneracy does not appear before the third line is added.

Third, we use the Delaunay triangulation, but other triangulation can be used similarly as long as it is a "good" triangulation, where "good" means that the triangulation does not contain relatively narrow and elongated triangles. The triangulation obtained as a dual of the L_1-metric Voronoi diagram [Chew, 1988] seems one of such candidates. To search for this possibility is also one of our future work.

We are now extending the present idea to the three-dimensional intersection problem.

The author would like to express his thanks to Prof. C. M. Hoffmann, Prof. C. Bajaj and T. K. Dey of Department of Computer Science, Purdue University, for their valuable comments. This work is partly supported by the Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan (Grant No. 01550279) and by the Computing-About-Physical-Object Project at Purdue University.

Reference

J. L. Bentley and T. A. Ottmann, 1979: Algorithms for reporting and counting geometric intersections, *IEEE Transactions on Computers*, vol. 28, pp. 643–647.

J. D. Boissonnat, 1988: Shape reconstruction from planar cross sections. *Computer Vision, graphics, and Image Processing*, vol. 44, pp. 1–29.

P. Chew, 1988: There is a planar graph almost as good as the complete graph. *Proceedings of the 2nd ACM Annual Symposium on Computational Geometry*, pp. 169–177.

D. P. Dobkin, S. J. Friedman and K. J. Supowit, 1990: Delaunay graphs are almost as good as complete graphs, *Discrete and Computational Geometry*, vol. 5, pp. 399–407.

D. Dobkin and D. Silver, 1988: Recipes for geometry and numerical analysis — Part I, An empirical study. *Proceedings of the 4th ACM Annual Symposium on Computational Geometry*, Urbana-Champaign, pp. 93–105.

S. Fortune, 1987: A sweepline algorithm for Voronoi diagrams. *Algorithmica*, vol. 2, pp. 153-174.

S. Fortune, 1989: Stable maintenance of point-set triangulations in two dimensions. Preprint, AT&T Bell Laboratories, Murray Hill.

D. H. Greene and F. Yao, 1986: Finite-resolution computational geometry, *Proceedings of the 27th IEEE Annual Symposium on Foundations of Computer Science*, Toronto, pp. 143–152.

L. Guibas, D. Salesin and J. Stolfi, 1989: Epsilon geometry — Building robust algorithms from imprecise computations. *Proceedings of the 5th ACM Annual Symposium on Computational Geometry*, Saarbruchen, pp. 208–217.

L. Guibas and J. Stolfi, 1985: Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, vol. 4, pp. 74–123.

C. M. Hoffmann, 1989: *Geometric & Solid Modeling*. Morgan Kaufmann Publishers, Inc., San Mateo.

C. M. Hoffmann, J. Hopcroft and M. Karasick, 1988: Towards implementing robust geometric computations. *Proceedings of the 4th ACM Annual Symposium on Computational Geometry*, Urbana-Champaign, pp. 106–117.

Y. Kanazawa, 1990: A robust method for finding points of intersection (in Japanese). Master's Thesis submitted to the Department of Mathematical Engineering, Faculty of Engineering, University of Tokyo, Tokyo, Japan.

M. Karasick, 1989: On the representation and manipulation of rigid solids. Technical Report, TR 89-976, Department of Computer Science, Cornell University, Ithaca.

M. Karasick, D. Lieber and L. R. Nackman, 1989: Efficient Delaunay triangulation using rational arithmetic. IBM Report, RC 14455, IBM Thomas J. Watson Research Center, Yorktown Heights.

J. Katajainen and M. Koppinen, 1988: Constructing Delaunay triangulations by merging buckets in quadtree order. *Fundamental Informatica*, vol. XI, pp. 275–288.

J. M. Keil and C. A. Gutwin, 1989: The Delaunay triangulation closely approximates the complete Euclidean graph. *Proceedings of the First Workshop on Algorithms and Data Structures*, pp. 47–56.

D. E. Knuth, 1973: *The Art of Computer Programming, Volume III — Sorting and Searching*. Addison-Wesley, Reading.

D. T. Lee and B. J. Schachter, 1980: Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, vol. 9, pp. 219–242.

V. Milenkovic, 1988: Verifiable implementations of geometric algorithms using finite precision arithmetic. Technical Report, CMU-CS-88-168, Computer Science Department, Carnegie-Mellon University.

V. Milenkovic, 1989: Double precision geometry — A general technique for calculating line and segment intersections using rounded arithmetic. *Proceedings of the 30th IEEE Annual Symposium on the Foundations of Computer Science*, pp. 500–505.

T. Ohya, M. Iri and K. Murota, 1984: Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms. *Journal of the Operations Research Society of Japan*, vol. 27, pp. 306–336.

A. Ohsawa, 1986: Pointer-based 2d-space modeling method allowing local geometric processing (in Japanese). *Transactions of Information Processing Society of Japan*, vol. 27, pp. 1174–1185.

A. Ohsawa, 1990: Failure free geometrical algorithm against calculation error — Realization by means of space model (in Japanese). *Transactions of Information Processing Society of Japan*, vol. 31, pp. 42–55.

T. Ottmann, G. Thiemt and C. Ullrich, 1987: Numerical stability of geometric algorithms. *Proceedings of the 3rd ACM Annual Conference on Computational Geometry*, Waterloo, pp. 119–125.

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

H. Samet, 1990: *Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA.

N. Sapidis and R. Perucchio, 1990: Delaunay triangulation of arbitrarily shaped planar domains. Preprint, Department of Mechanical Engineering, University of Rochester.

W. J. Schroeder and M. S. Shephard, 1988: Geometry-based full automatic mesh generation and the Delaunay triangulation. *International Journal for Numerical Methods in Engineering*, vol. 26, pp. 2503–2515.

M. I. Shamos and D. Hoey, 1975: Closest-point problems. *Proceedings of thr 16th IEEE Annual Symposium on Foundation of Computer Science*, pp. 151–162.

K. Sugihara and M. Iri, 1989a: A solid modelling system free from topological inconsistency. *Journal of Information Processing*, vol. 12 (1989), pp. 380–393.

K. Sugihara and M. Iri, 1989b: Construction of the Voronoi diagram for one million generators in single-precision arithmetic. Paper presented at the First Canadian Conference on Computational Geometry, August 21–25, 1989, Montreal, Canada, and submitted for publication.

K. Sugihara and M. Iri, 1989c: VORONOI2 reference manual. *Research Memorandum* RMI 89-04, Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, 1989.

K. Sugihara, Y. Ooishi and T. Imai, 1990: Topology-oriented approach to robustness and its applications to several Voronoi-diagram algorithms. *Abstracts of the Second Canadian Conference on Computational Geometry*, Ottawa, August 1990, pp. 36-39.

G. Vaněček, Jr., 1989: Set operation on polyhedra using decomposition method. Ph.D Dissertation, Department of Computer Science, University of Maryland.

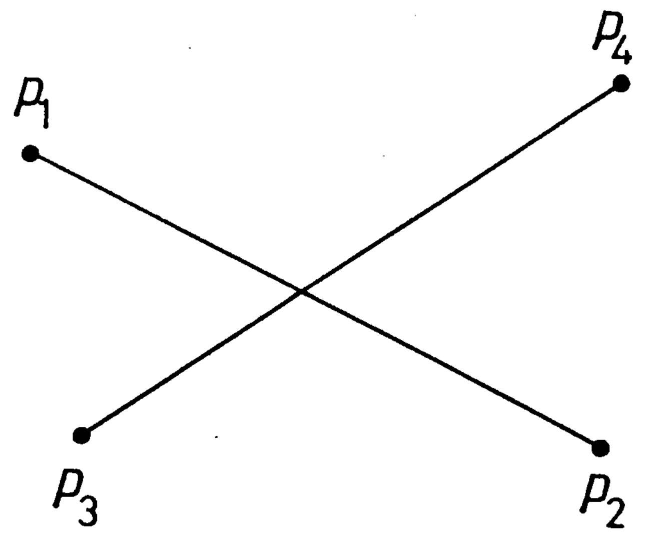


Fig. 2.1. Two line segments and their point of intersection.

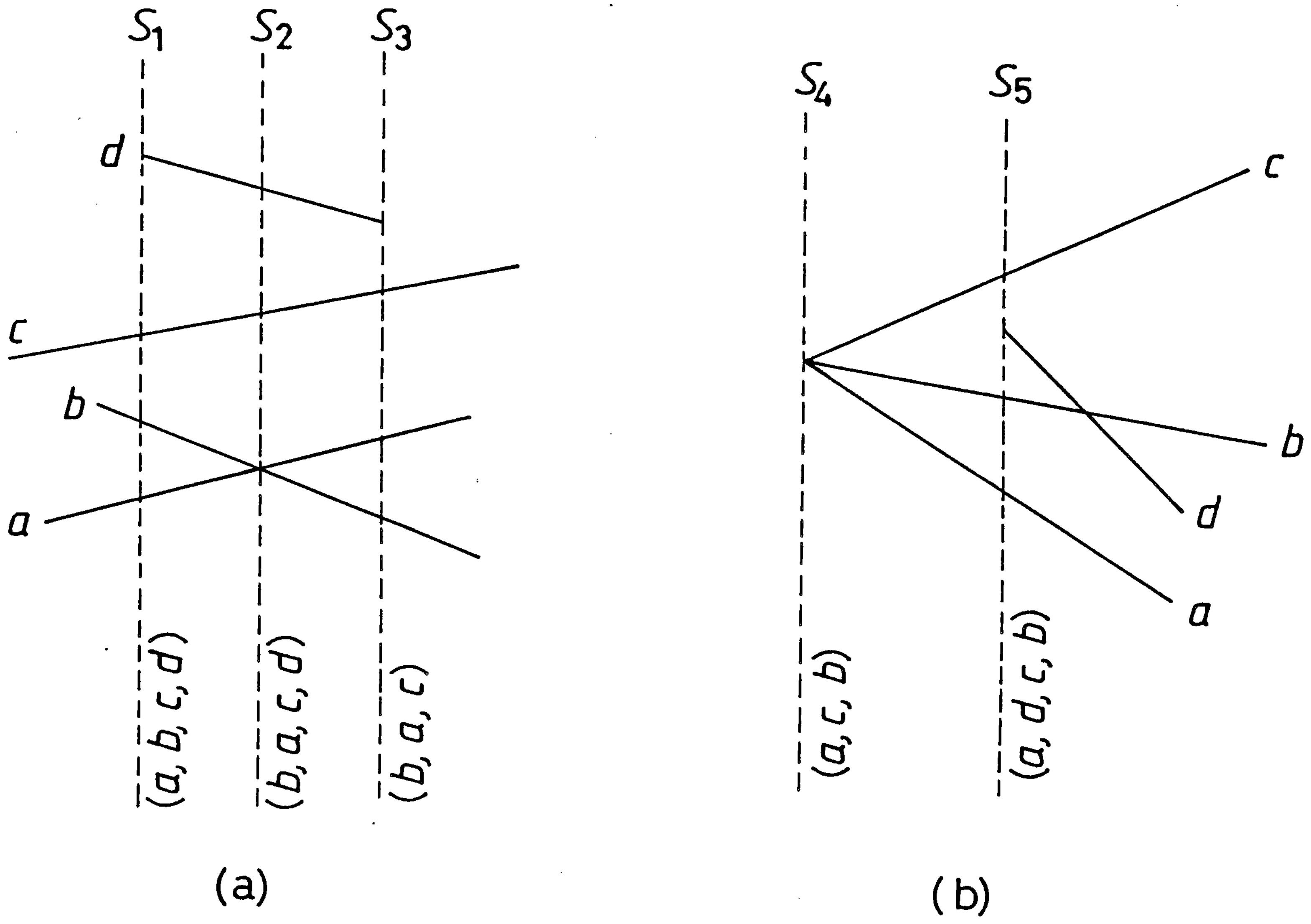


Fig. 2.2. Plane sweep method and its instability: (a) basic idea of the plane sweep method; (b) example in which the plane sweep method misses a point of intersection that is easily detectable by the naive pairwise check.

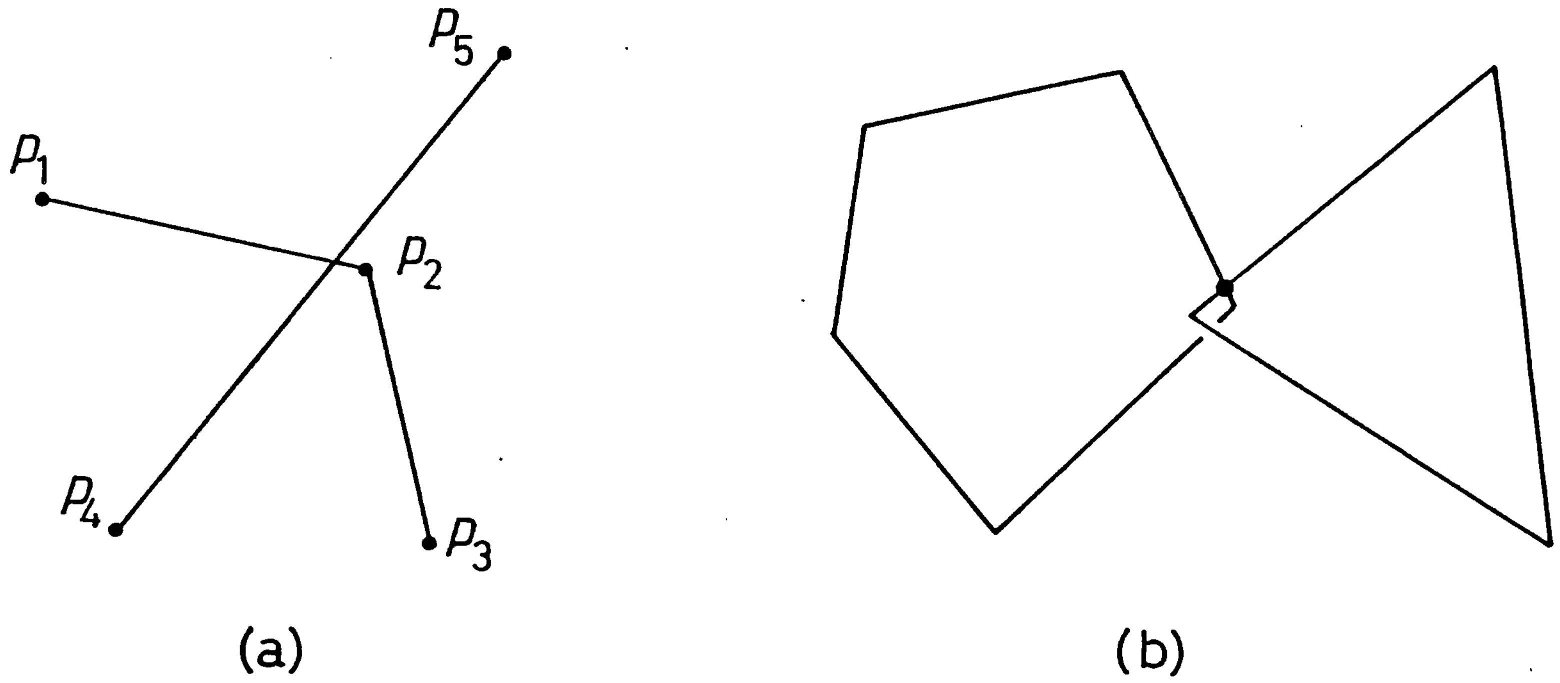


Fig. 2.3. Configurations for which the exhaustive pairwise check is not stable: (a) intersection of two polygonal lines; (b) intersection of two polygons.

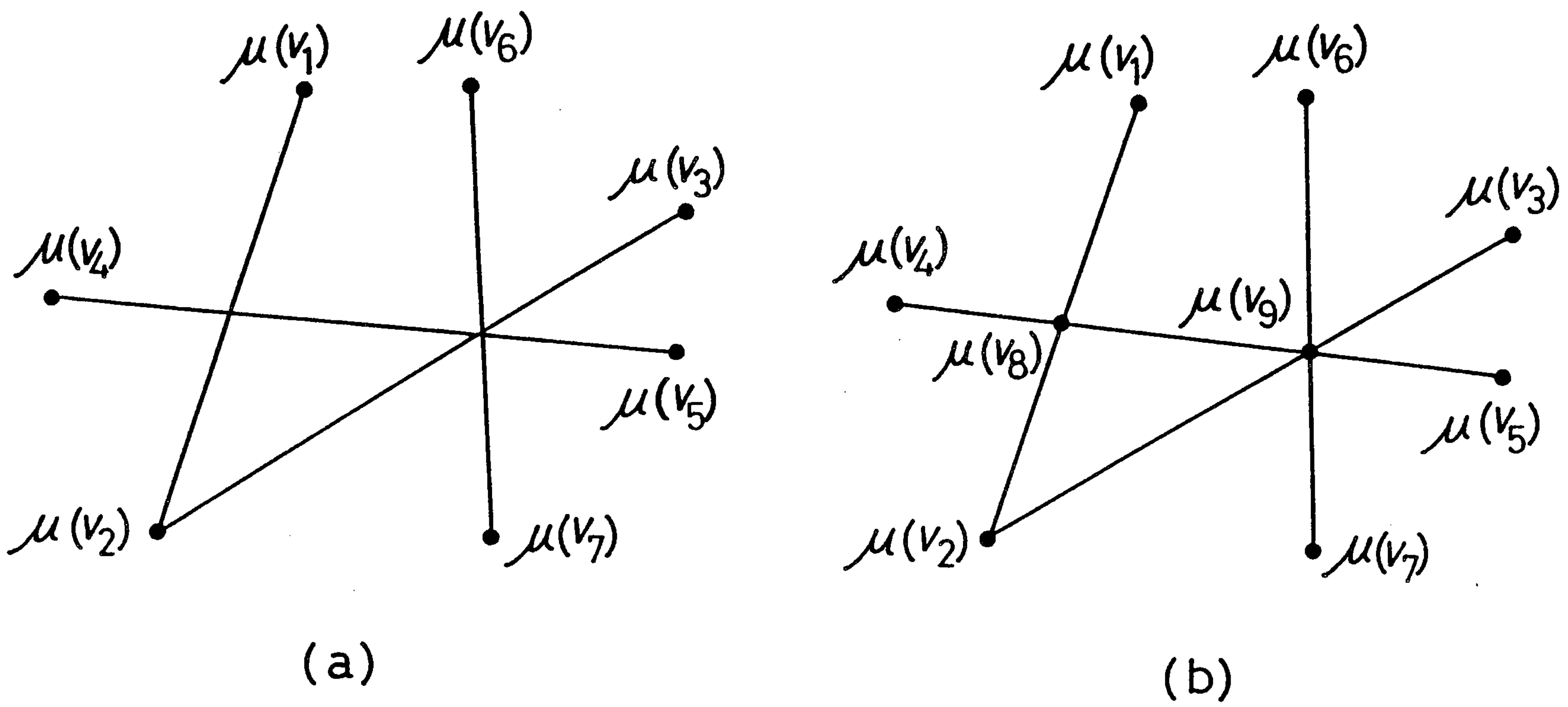


Fig. 3.1. Diagram and its coarsest planar refinement: (a) diagram having implicit points of intersection; (b) coarsest planar refinement of the diagram in (a).

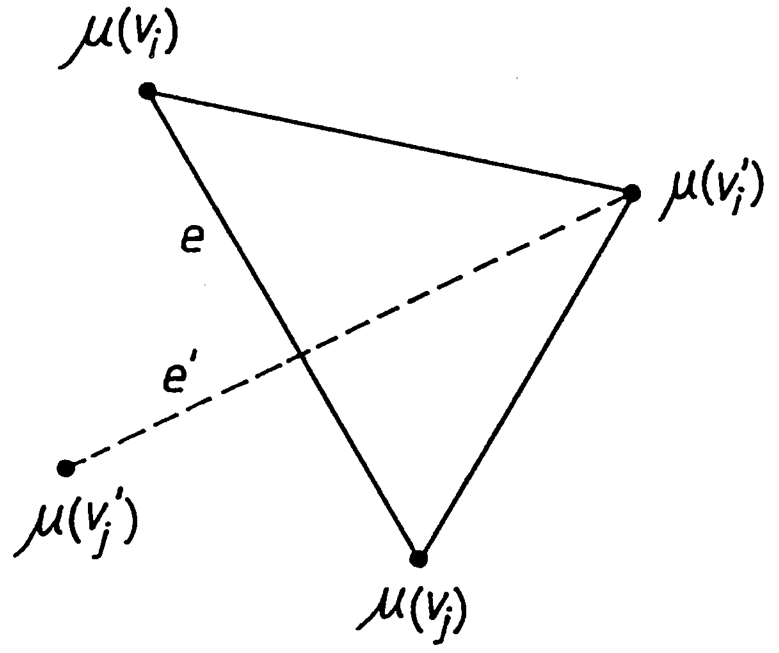


Fig. 4.1. Cross patter.

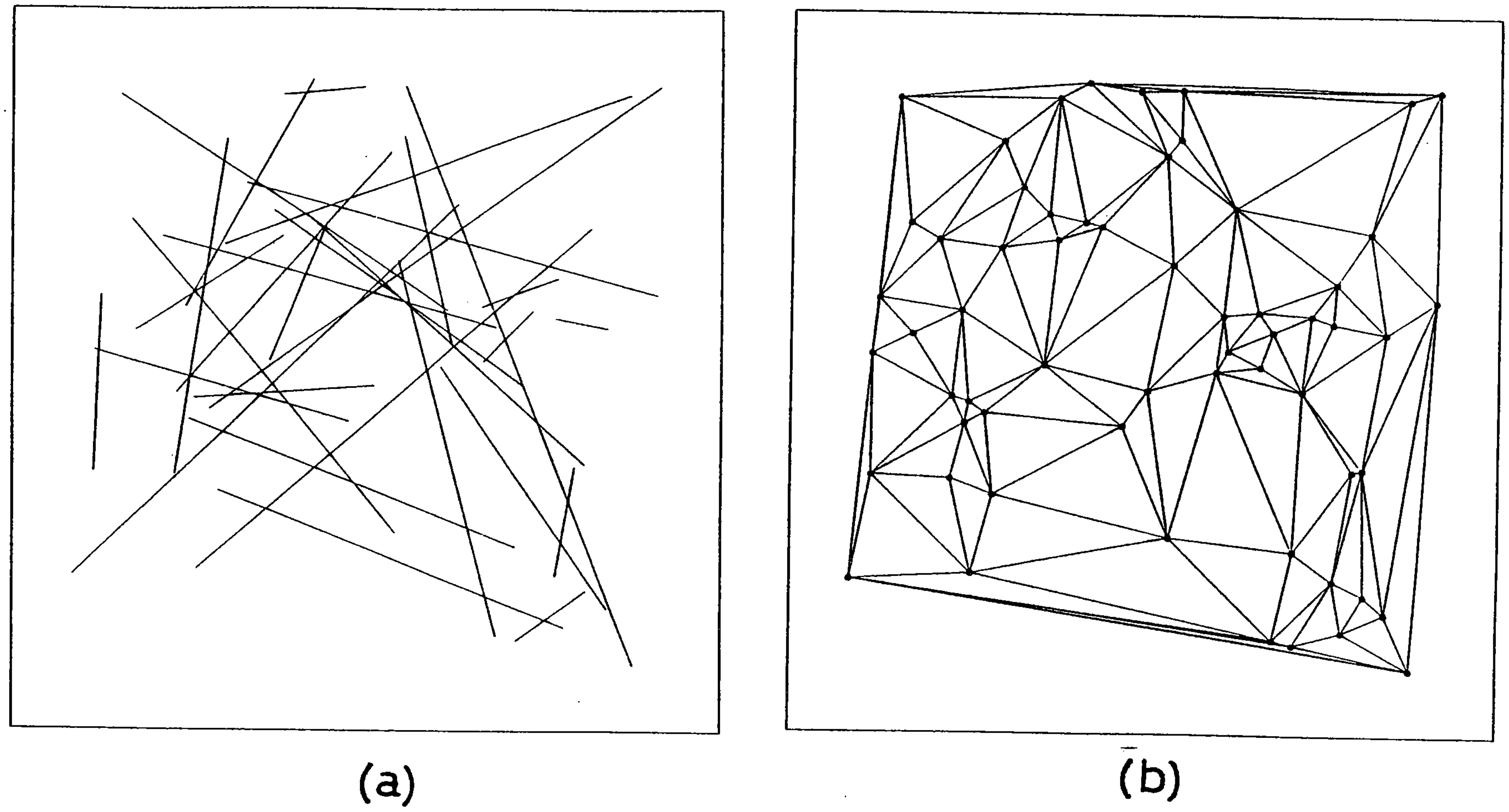


Fig. 4.2. Behavior of Algorithm 1: (a) 30 line segments generated at random; (b) Delaunay triangulation spanning the end points of the line segments; (c) Delaunay triangulation at the time when all the original line segments are realized; (d) 97 points of intersection found by the algorithm.

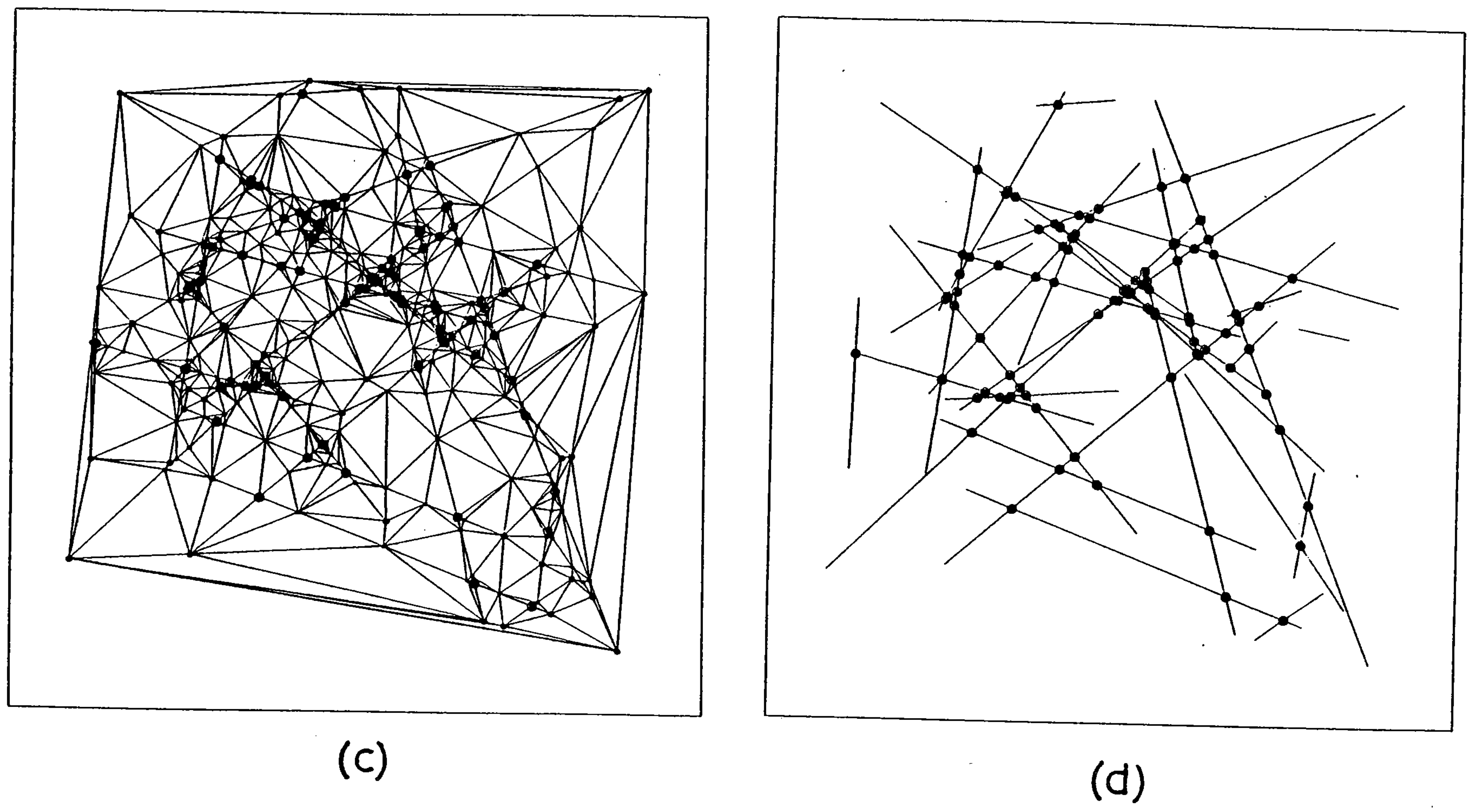


Fig. 4.2 (continued)

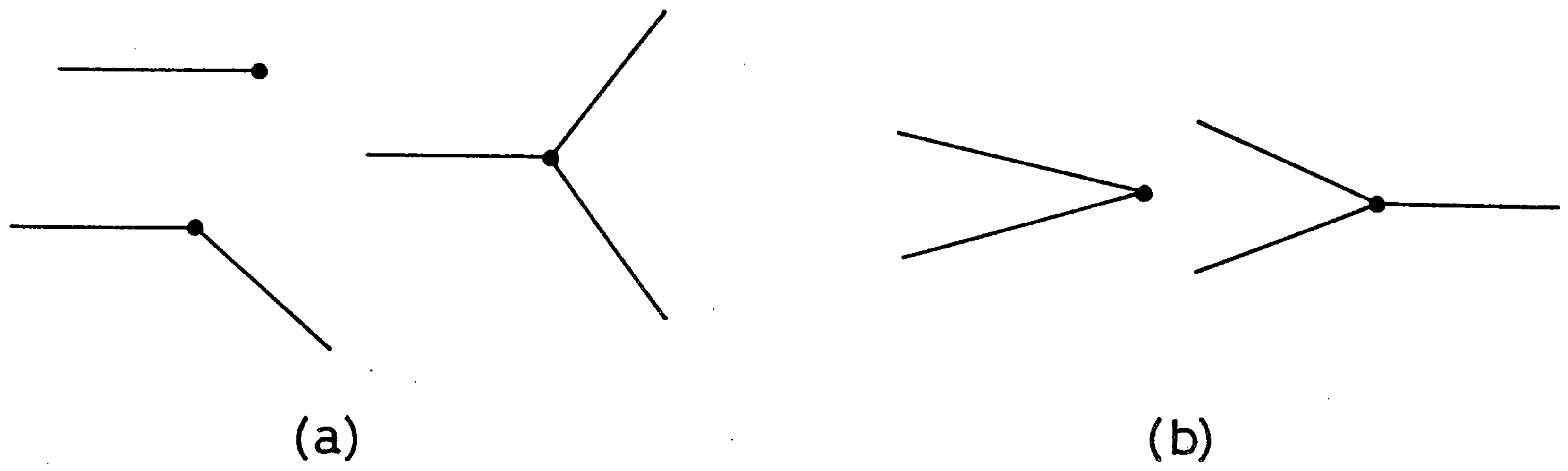


Fig. 5.1. Obtuse vertices and acute vertices: (a) obtuse vertices; (b) acute vertices.

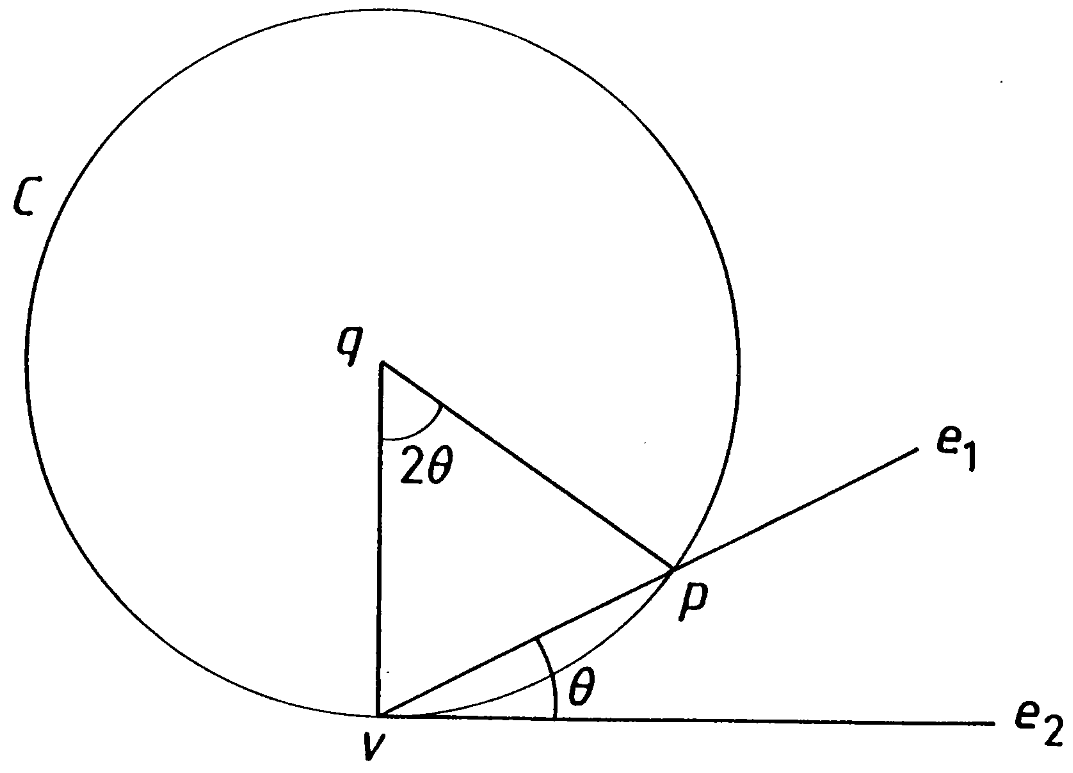


Fig. 5.2. Empty circle at a clear vertex.

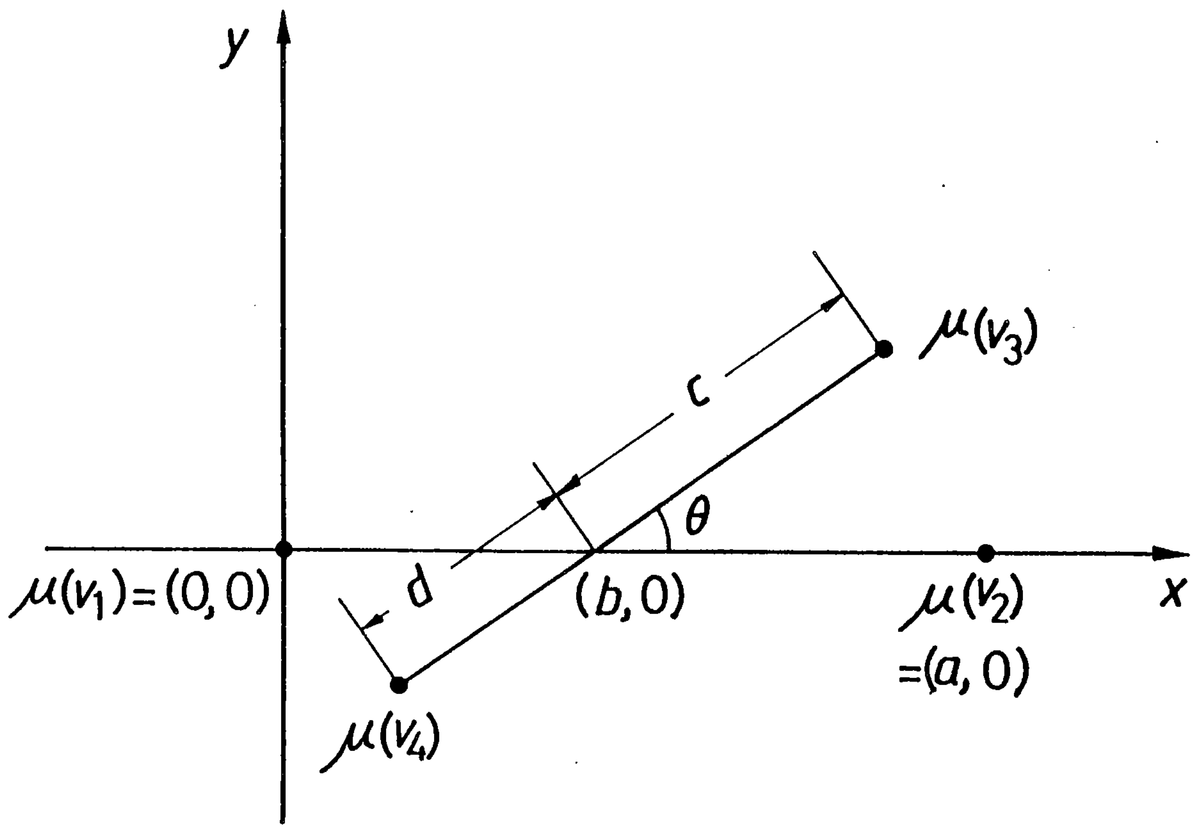


Fig. 5.3. Two line segments containing a common point of intersection.

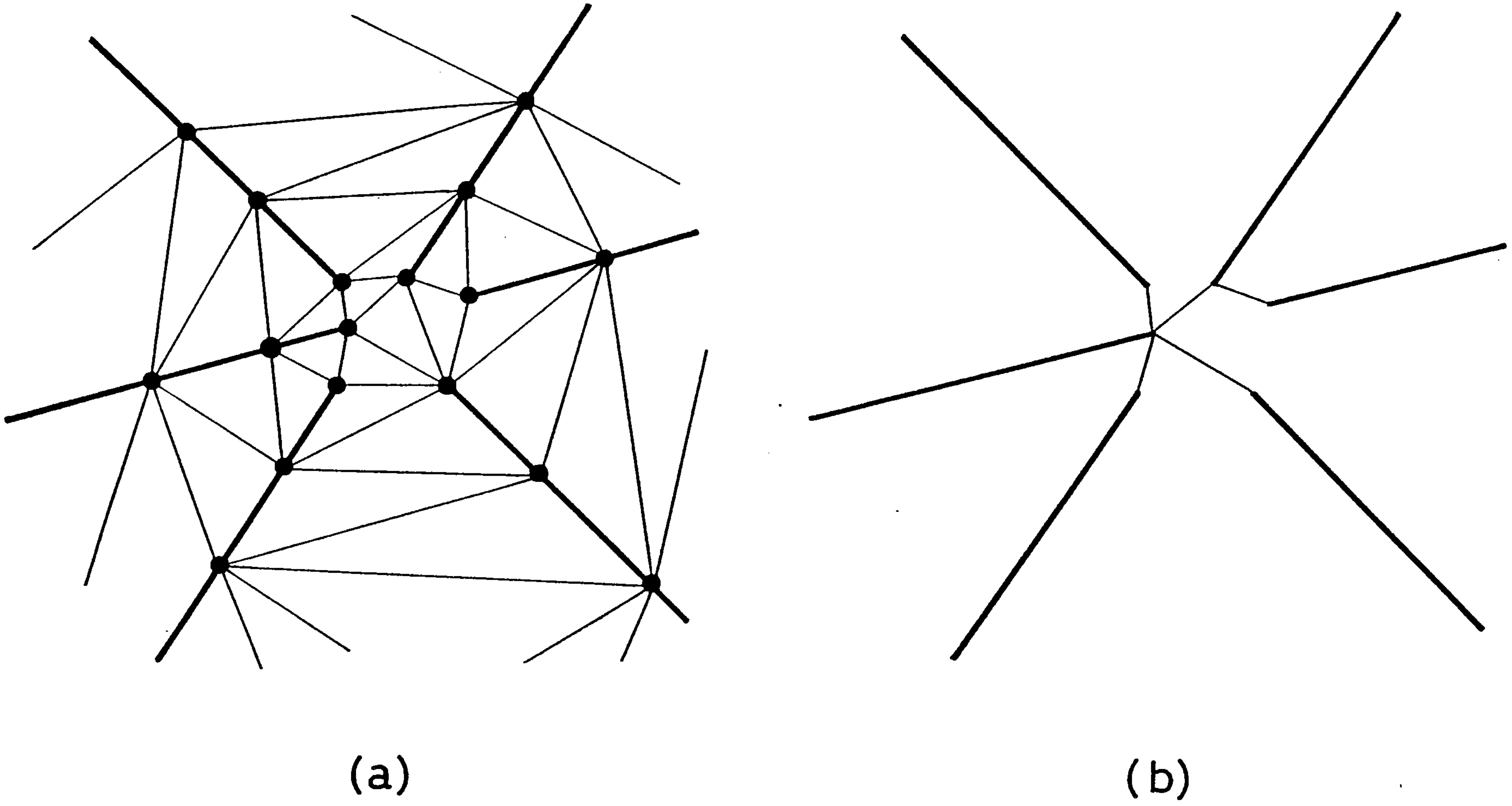
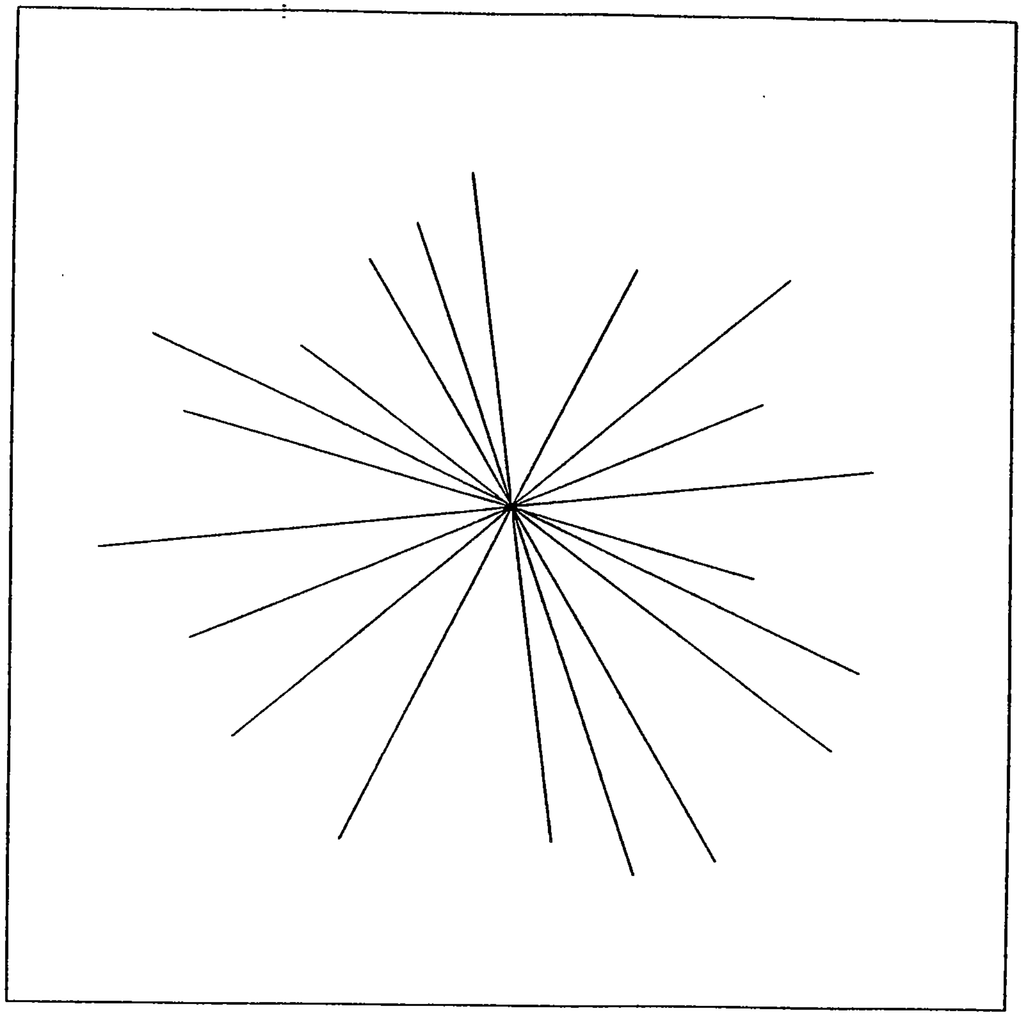
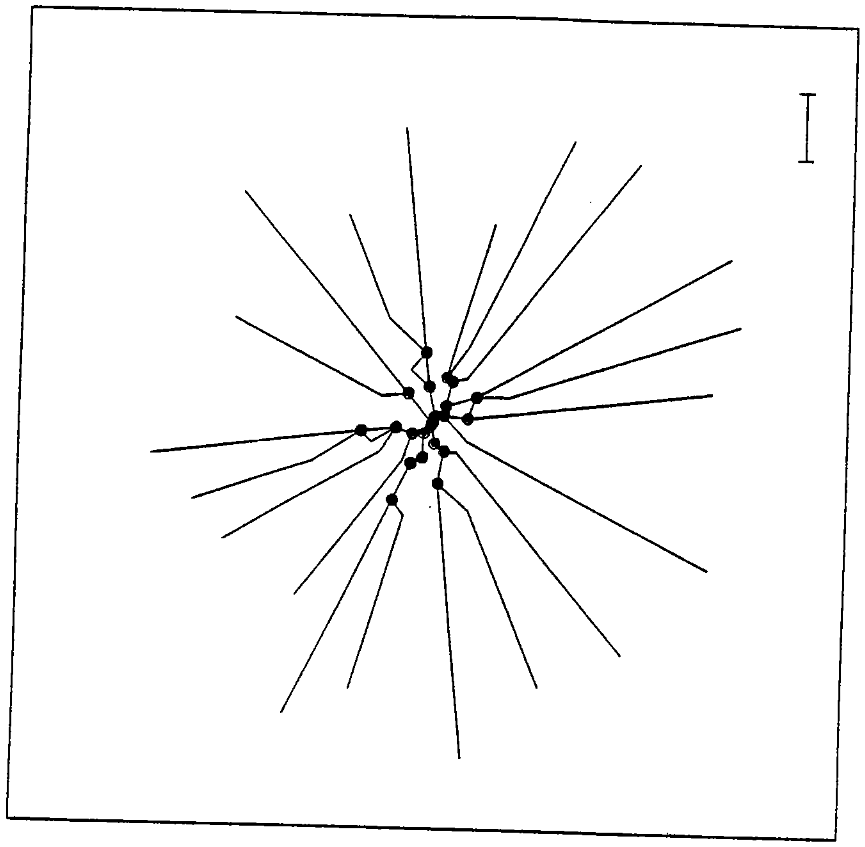


Fig. 6.1. Unrealized edges and Delaunay shortest paths: (a) triangulation obtained at the end of the midpoint insertion; (b) approximation of unrealized edges by Delaunay shortest paths.



(a)

Fig. 7.1. Example 1 — Effect of the threshold ε: (a) input diagram; (b) and (b') $\varepsilon = 0.1$; (c) and (c') $\varepsilon = 0.01$; (d) central region of (a) magnified by 100; (e) and (e') $\varepsilon = 0.002$; (f) and (f') $\varepsilon = 0.001$; (g) and (g') $\varepsilon = 0.0001$.


(b')

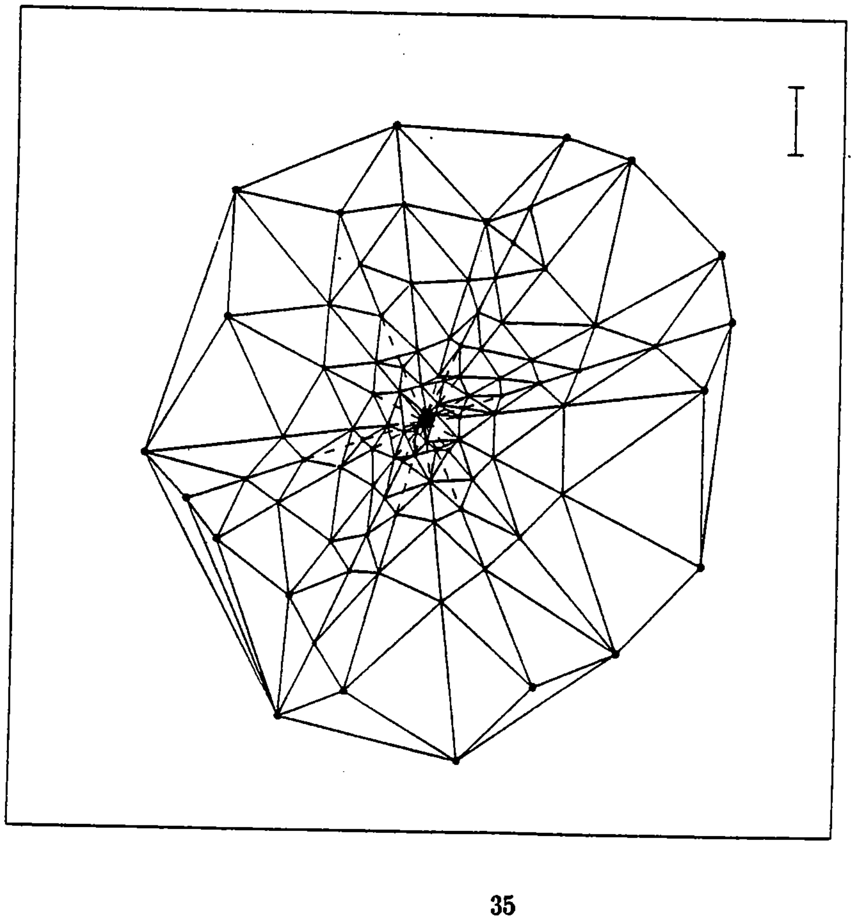
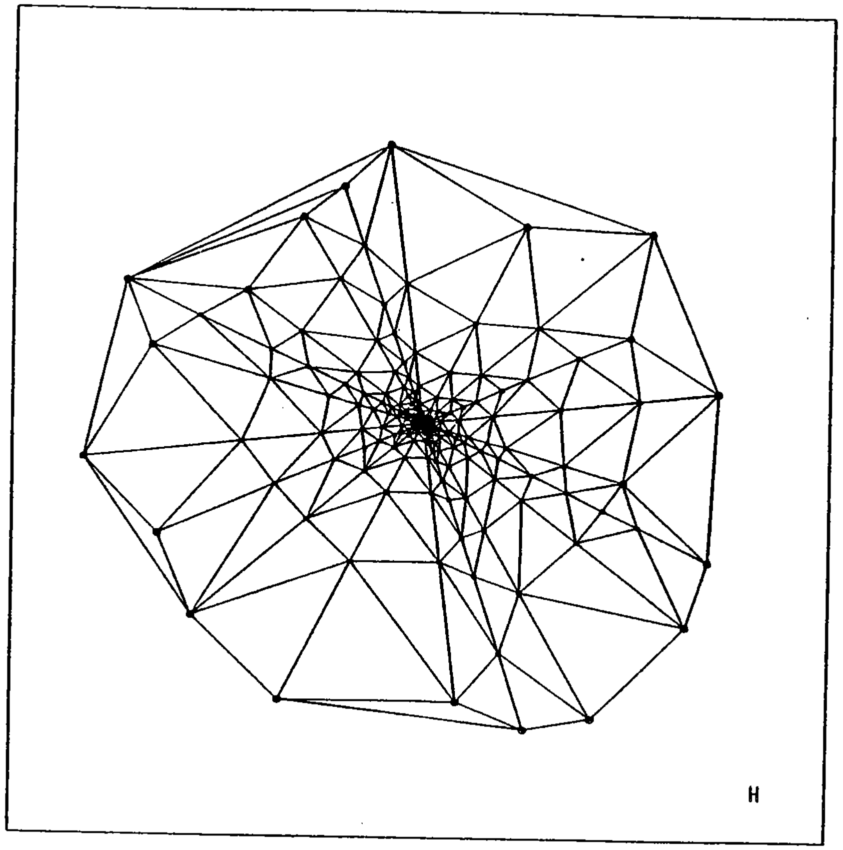
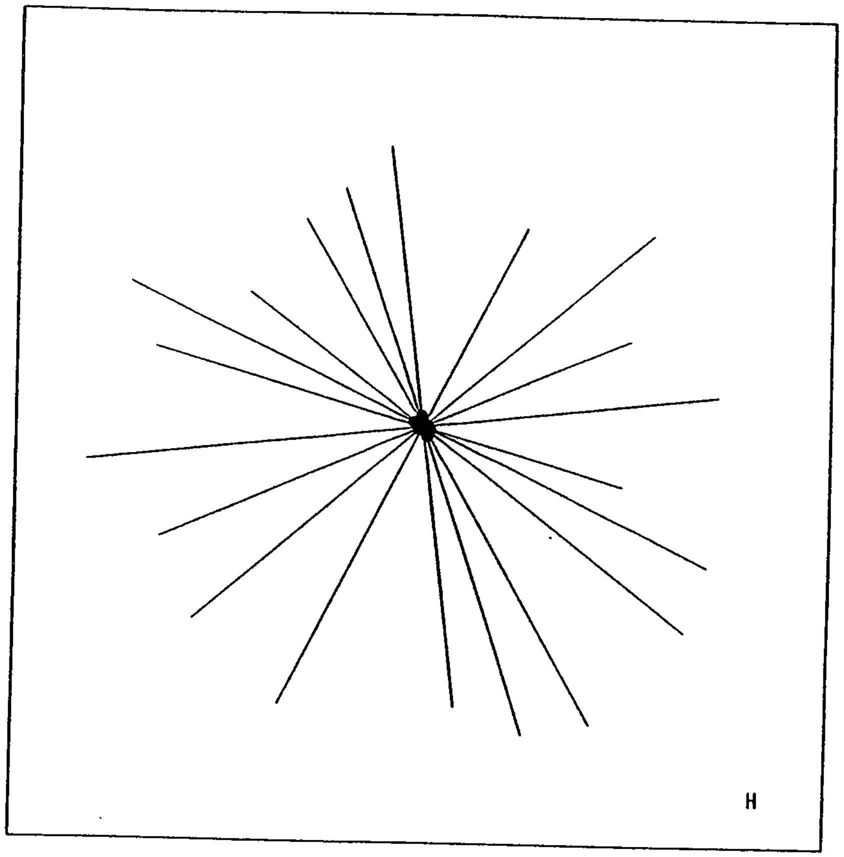

(b)

Fig. 7.1 (continued)

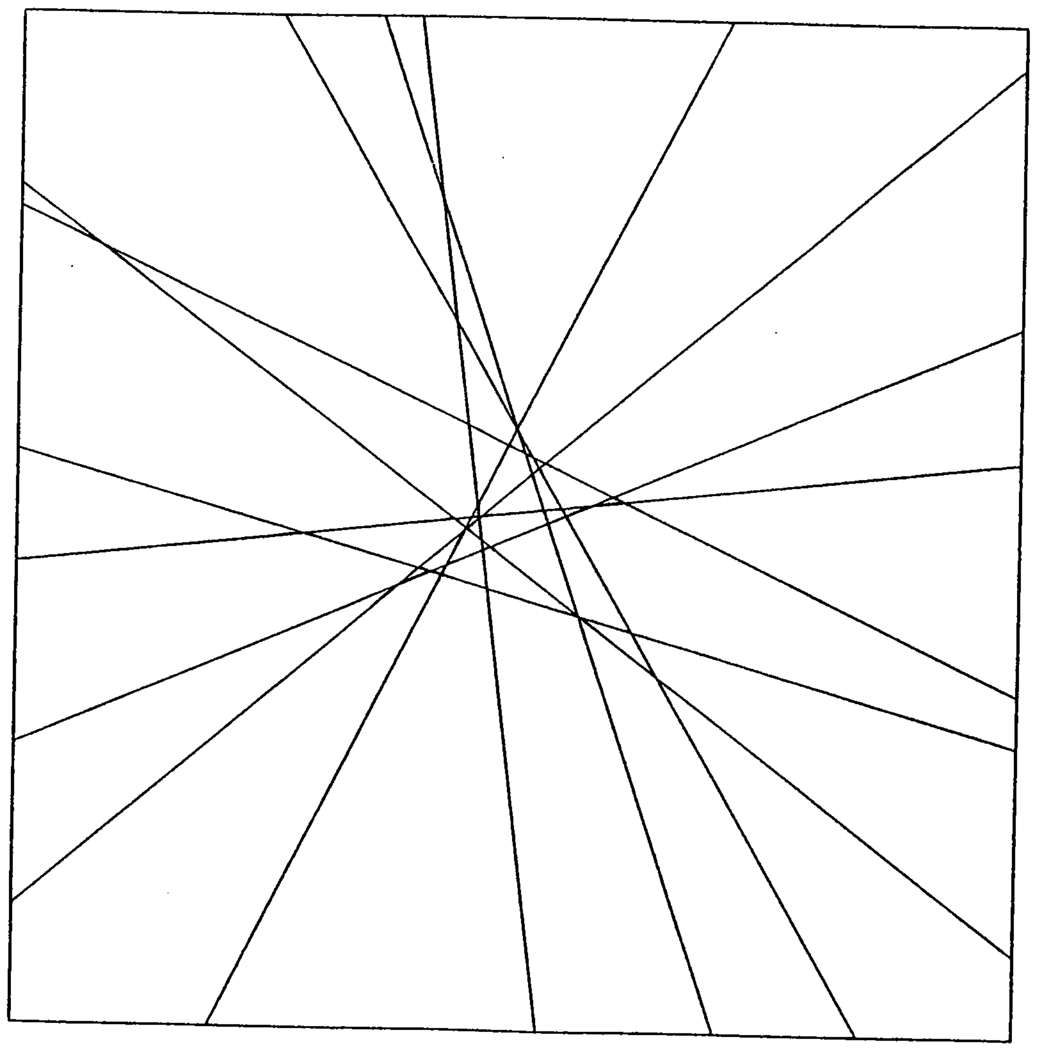


(c)



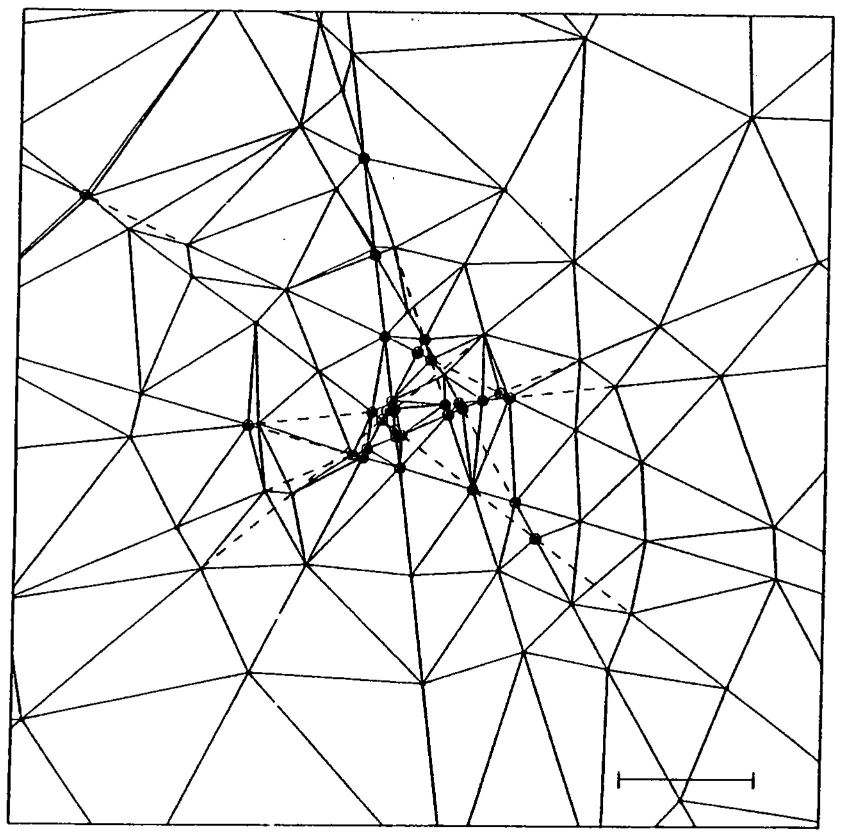
(c')

Fig. 7.1 (continued)

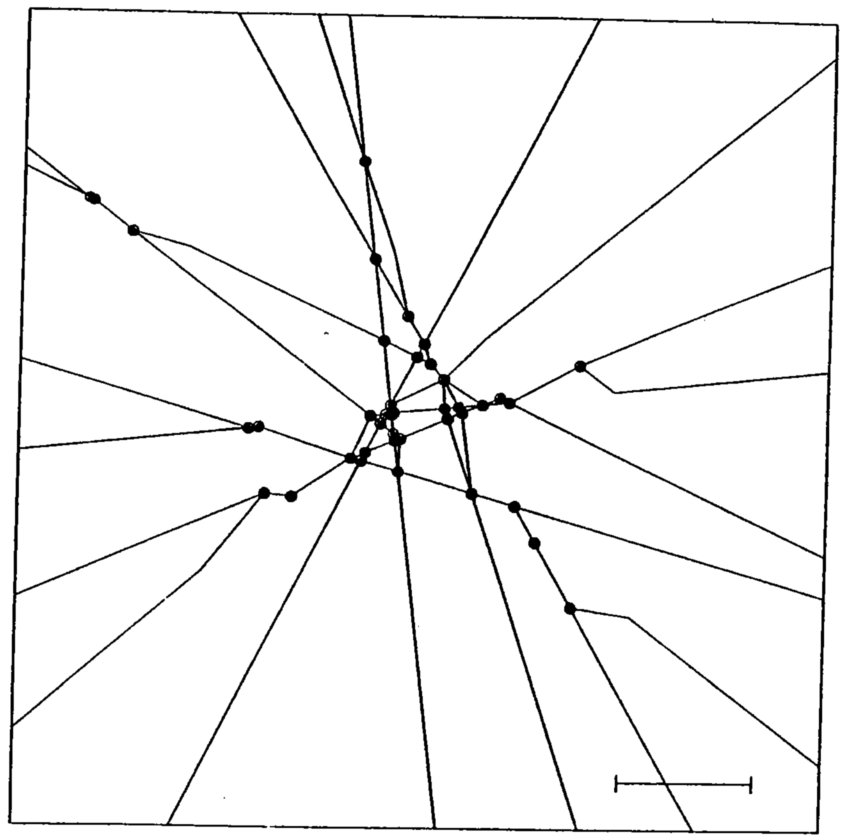


(d)

Fig. 7.1 (continued)

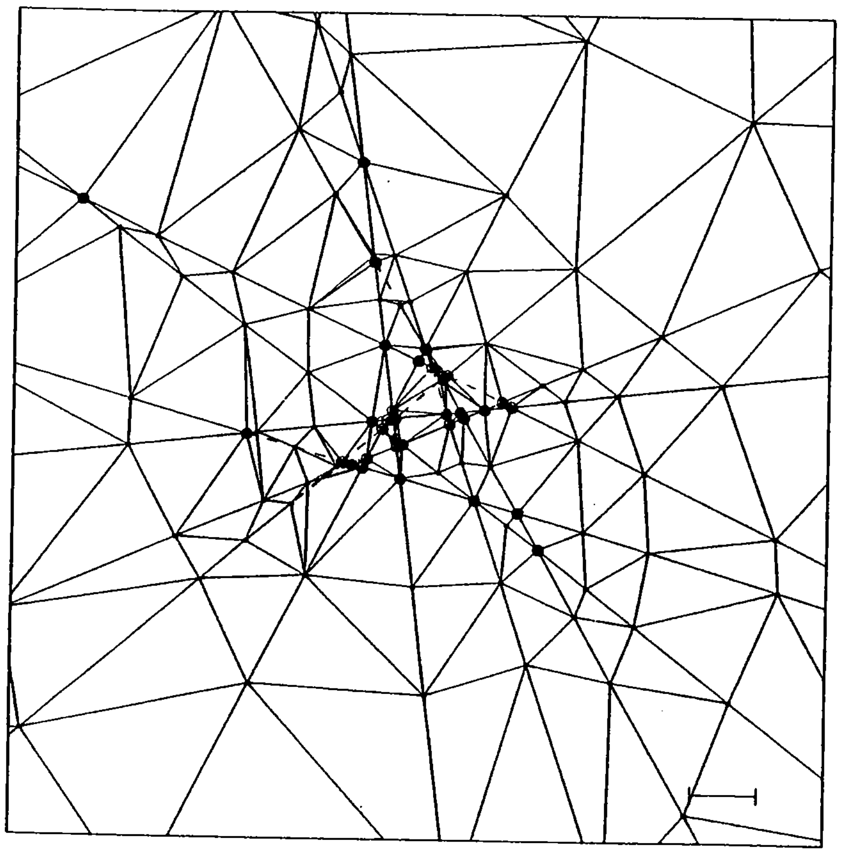


(e)

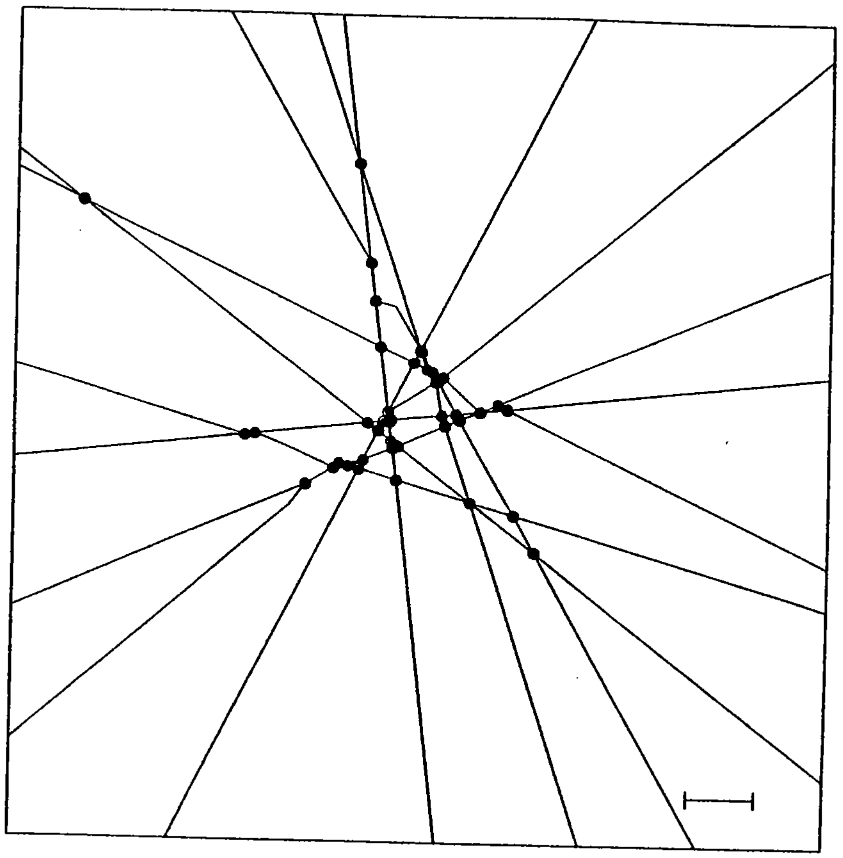


(e')

Fig. 7.1 (continued)

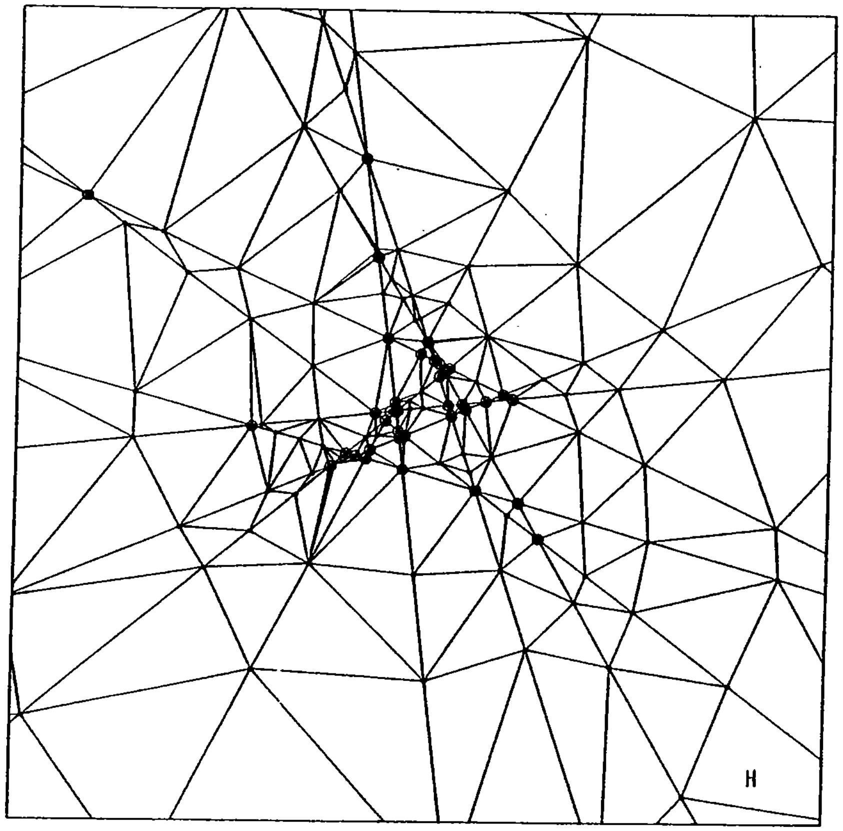


(f)

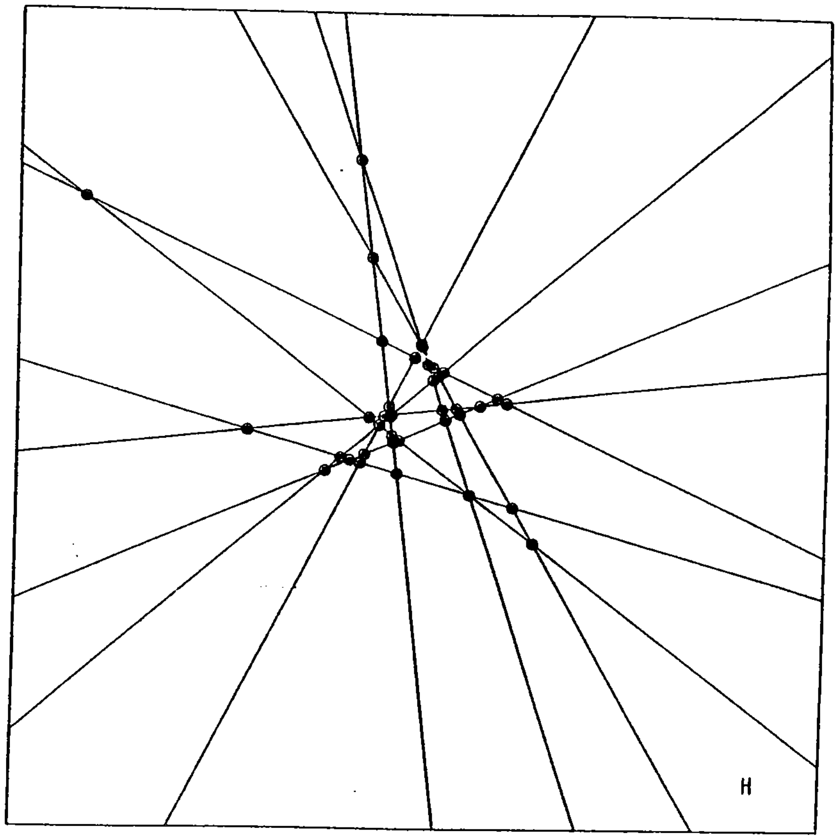


(f′)

Fig. 7.1 (continued)

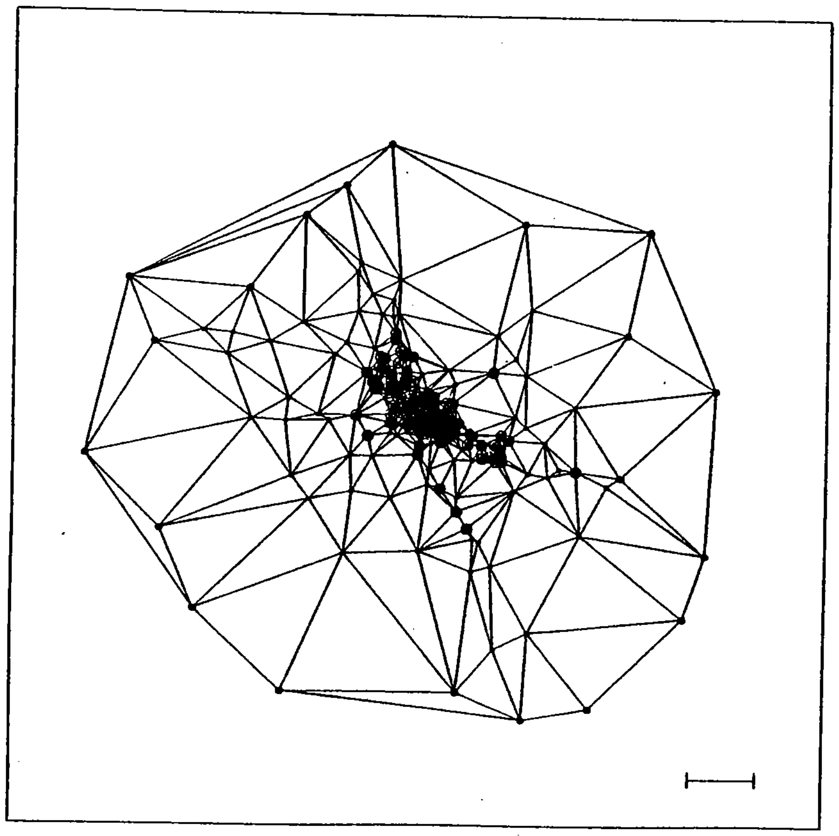


(g')

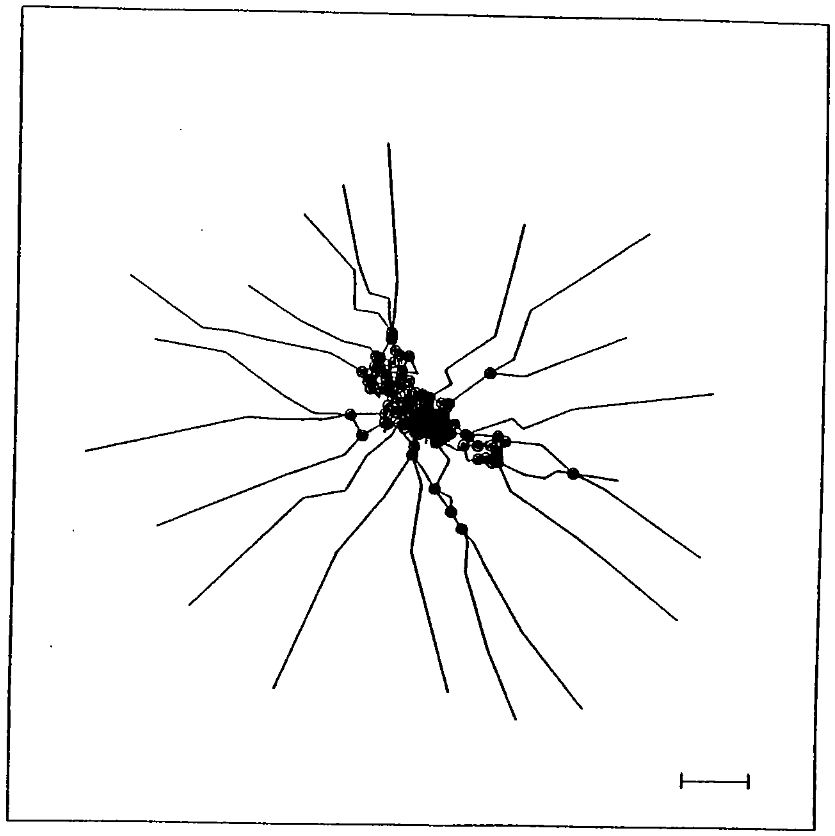


(g)

Fig. 7.1 (continued)

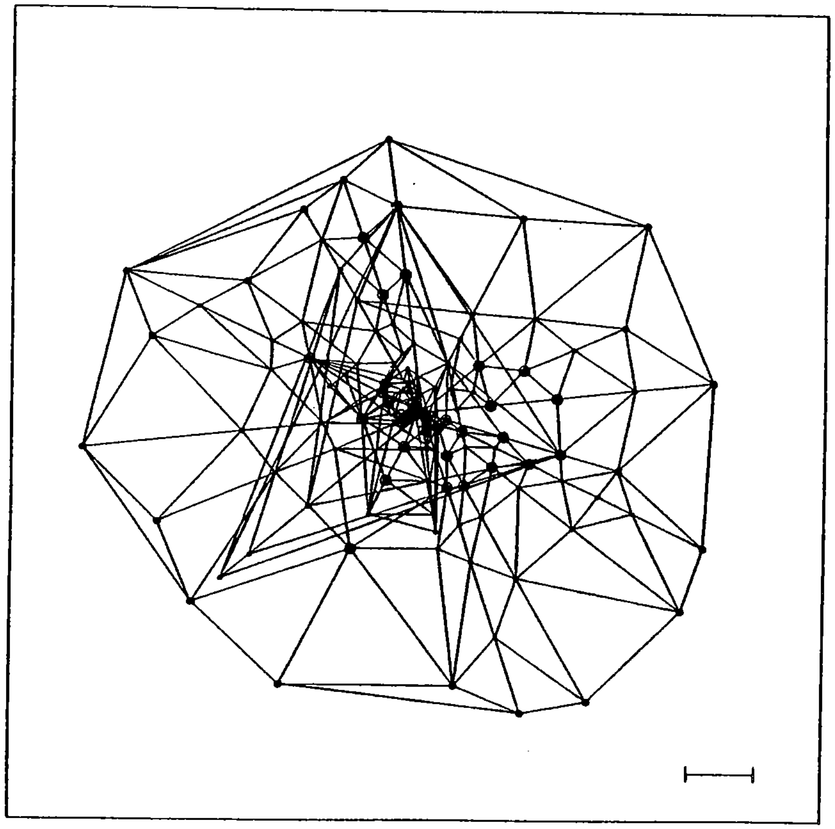


(a)

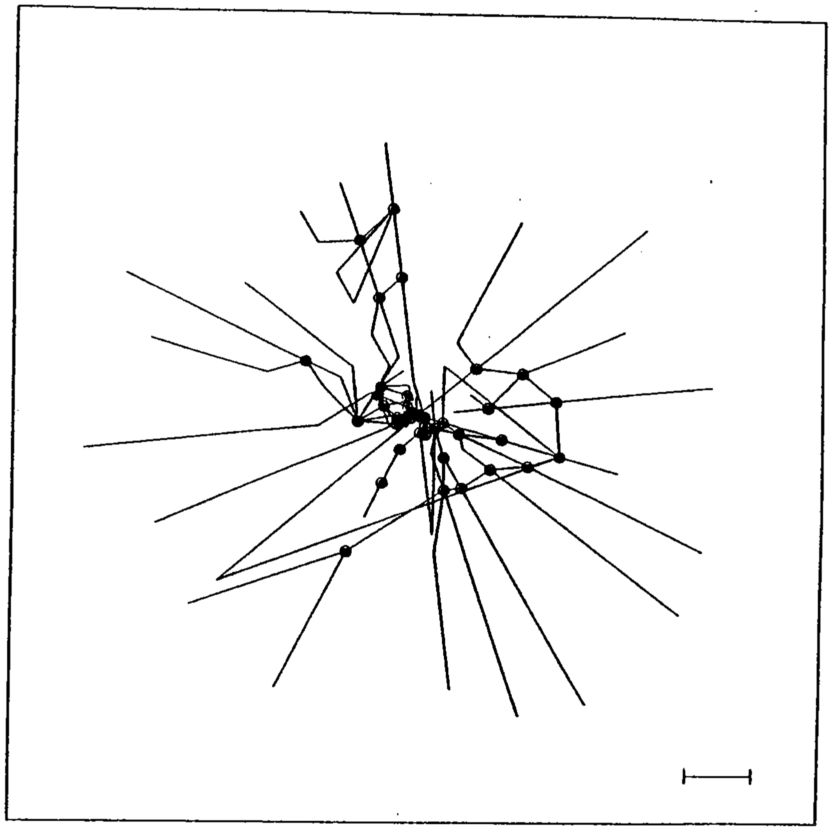


(a')

Fig. 7.2. Example 2 — Effect of numerical errors ($\varepsilon = 0.1$): (a) and (a') 5% errors to the computation of midpoints and intersection; (b) and (b') 5% errors to the computation of the Delaunay triangulation; (c) and (c') 100% errors to both the computation of midpoints and intersection and the computation of the Delaunay triangulation.

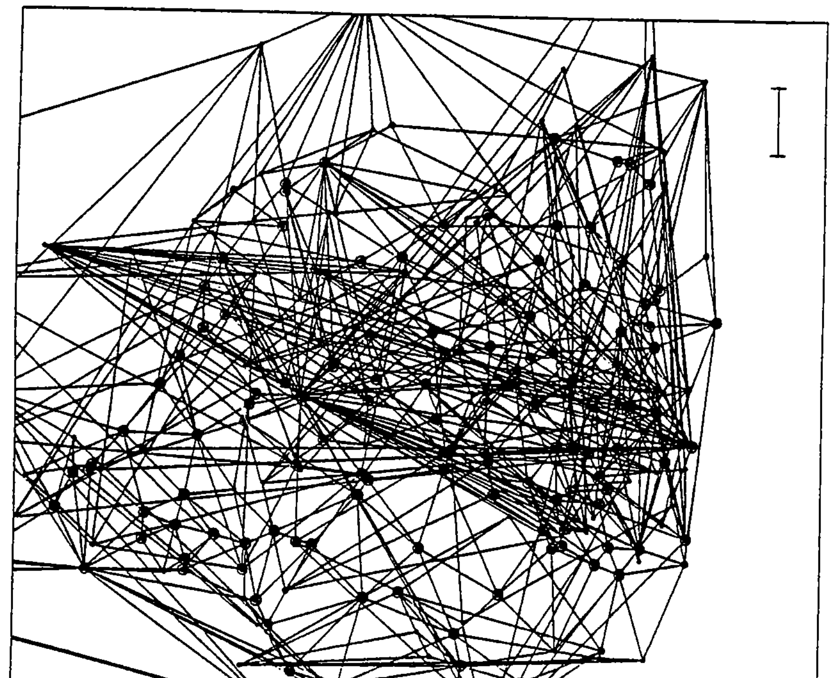


(b)

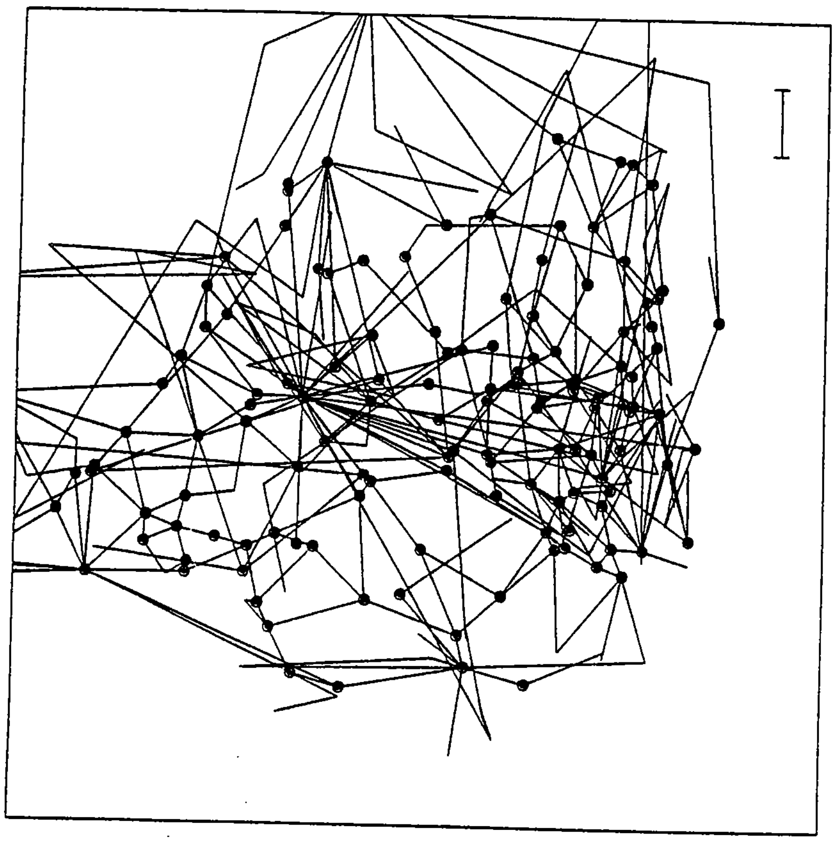


(b')

Fig. 7.2 (continued)



(c)



(c')

Fig. 7.2 (continued)

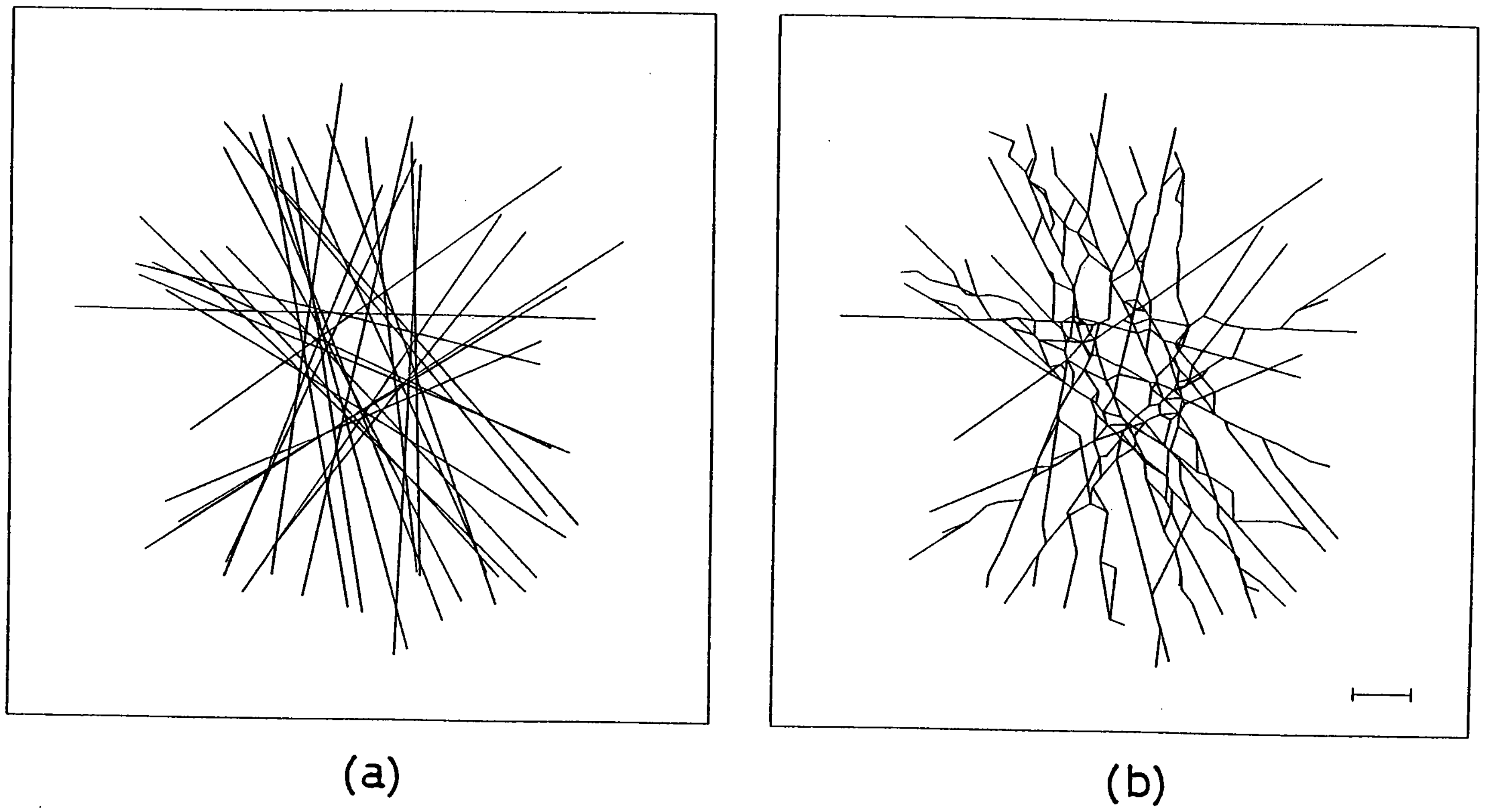
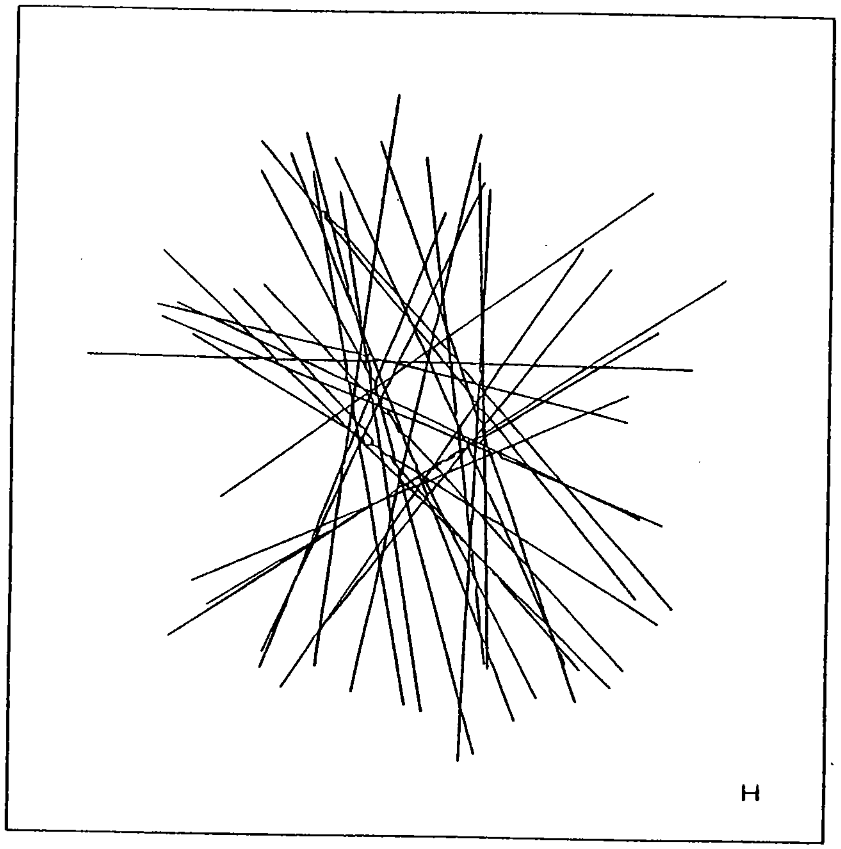
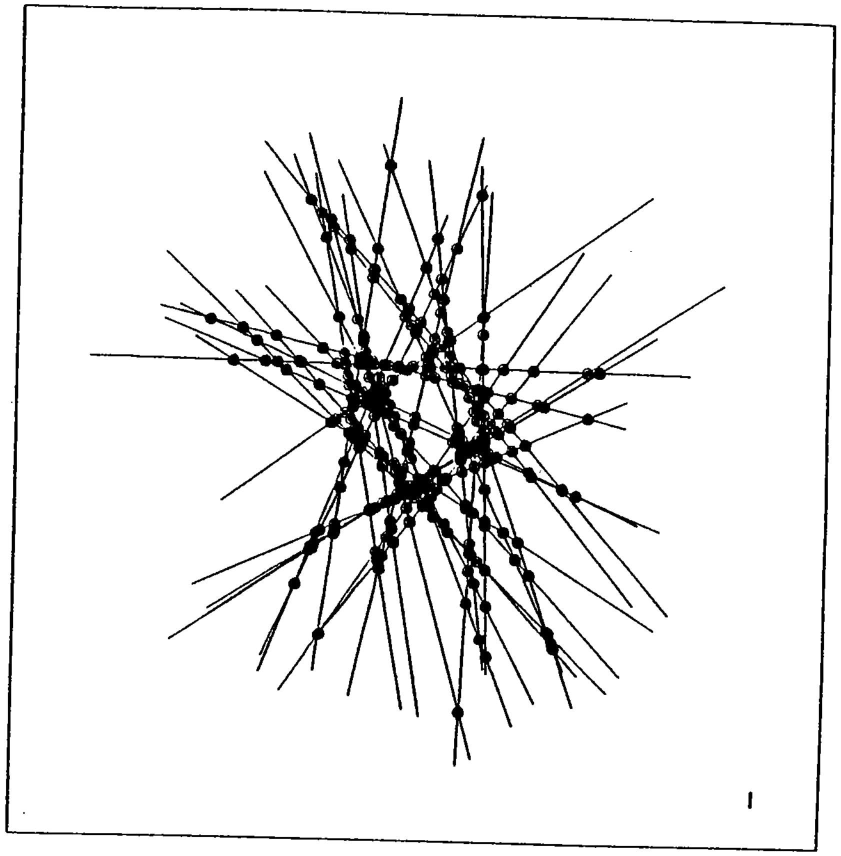


Fig. 7.3. Example 3 — 30 line segments: (a) input diagram; (b) $\varepsilon = 0.1$; (c) $\varepsilon = 0.02$; (d) $\varepsilon = 0.0001$.



(c)



(d)

Fig. 7.3 (continued)

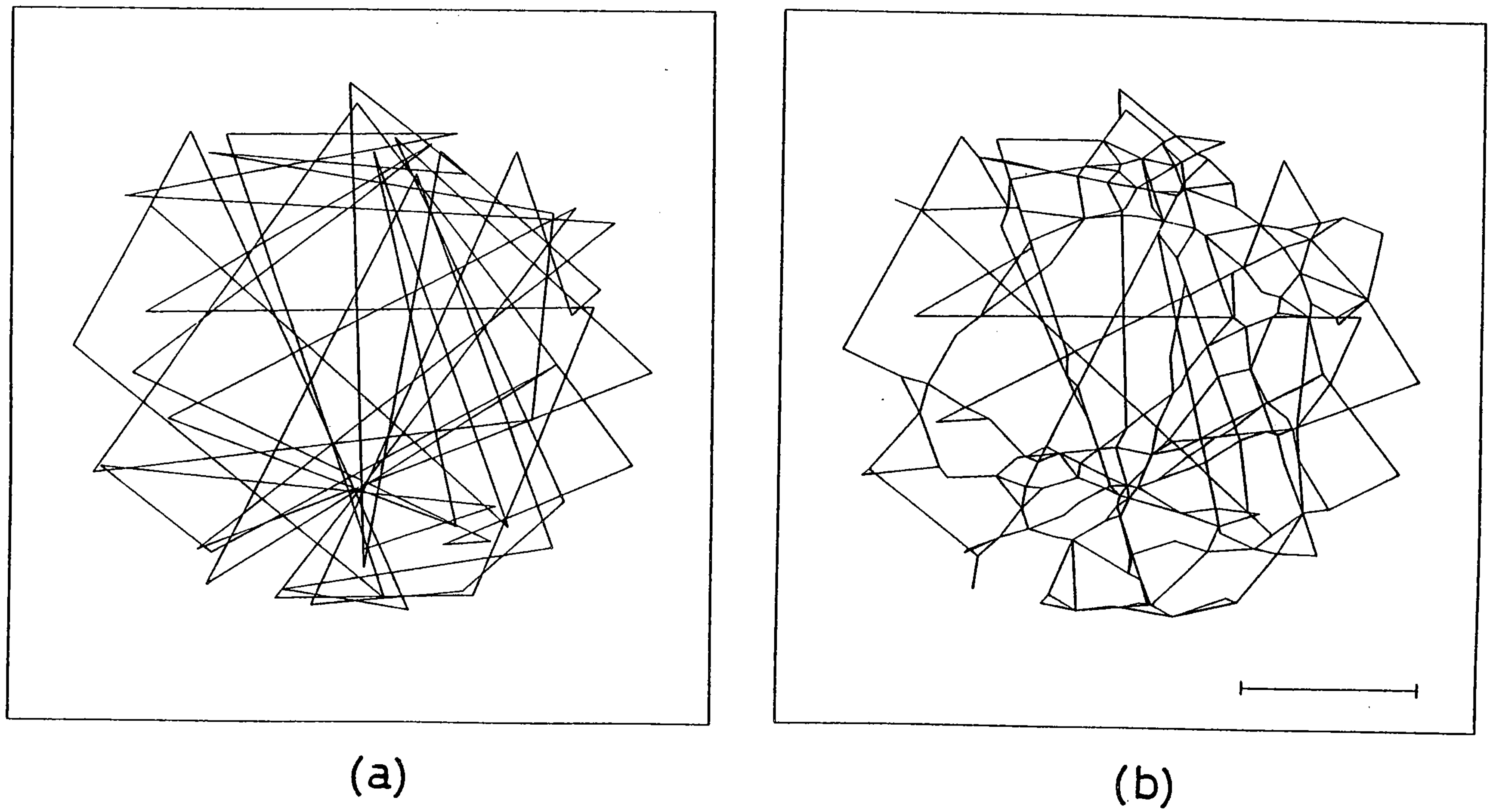
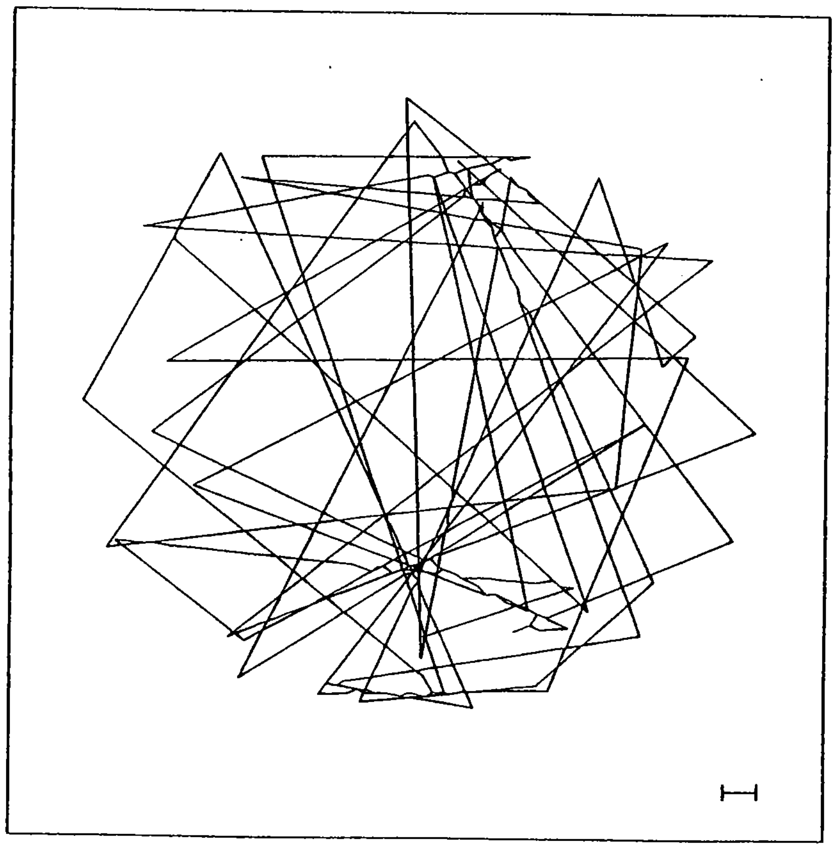


Fig. 7.4. Example 4 — Polygonal line having 50 vertices: (a) input diagram; (b) $\varepsilon = 0.3$; (c) $\varepsilon = 0.05$; (d) $\varepsilon = 0.0001$.


(c)

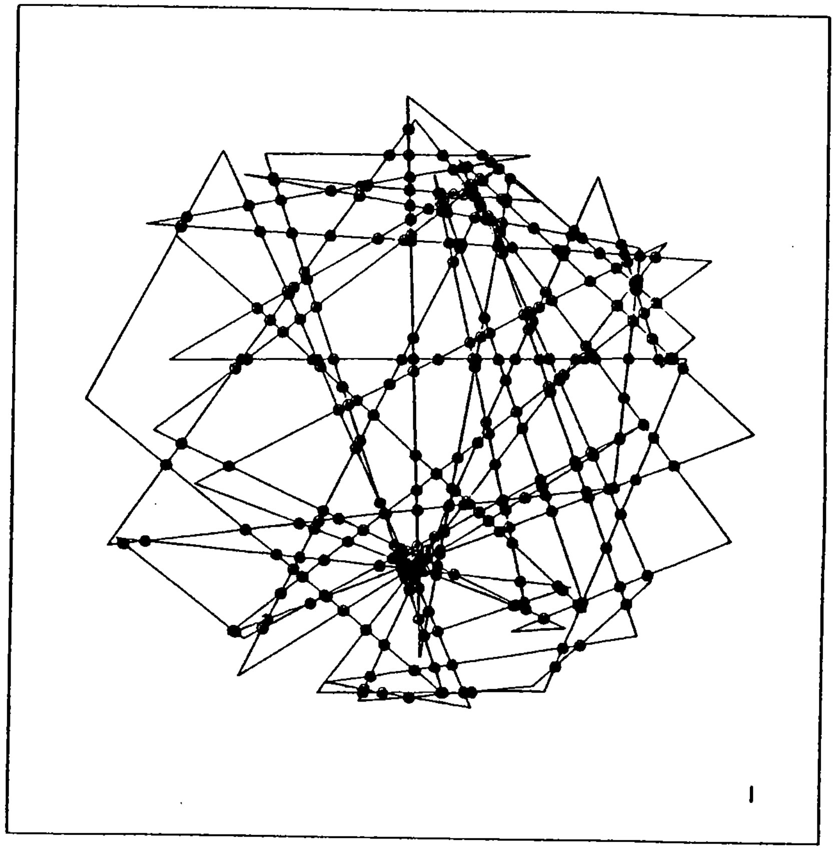

(d)

Fig. 7.4 (continued)

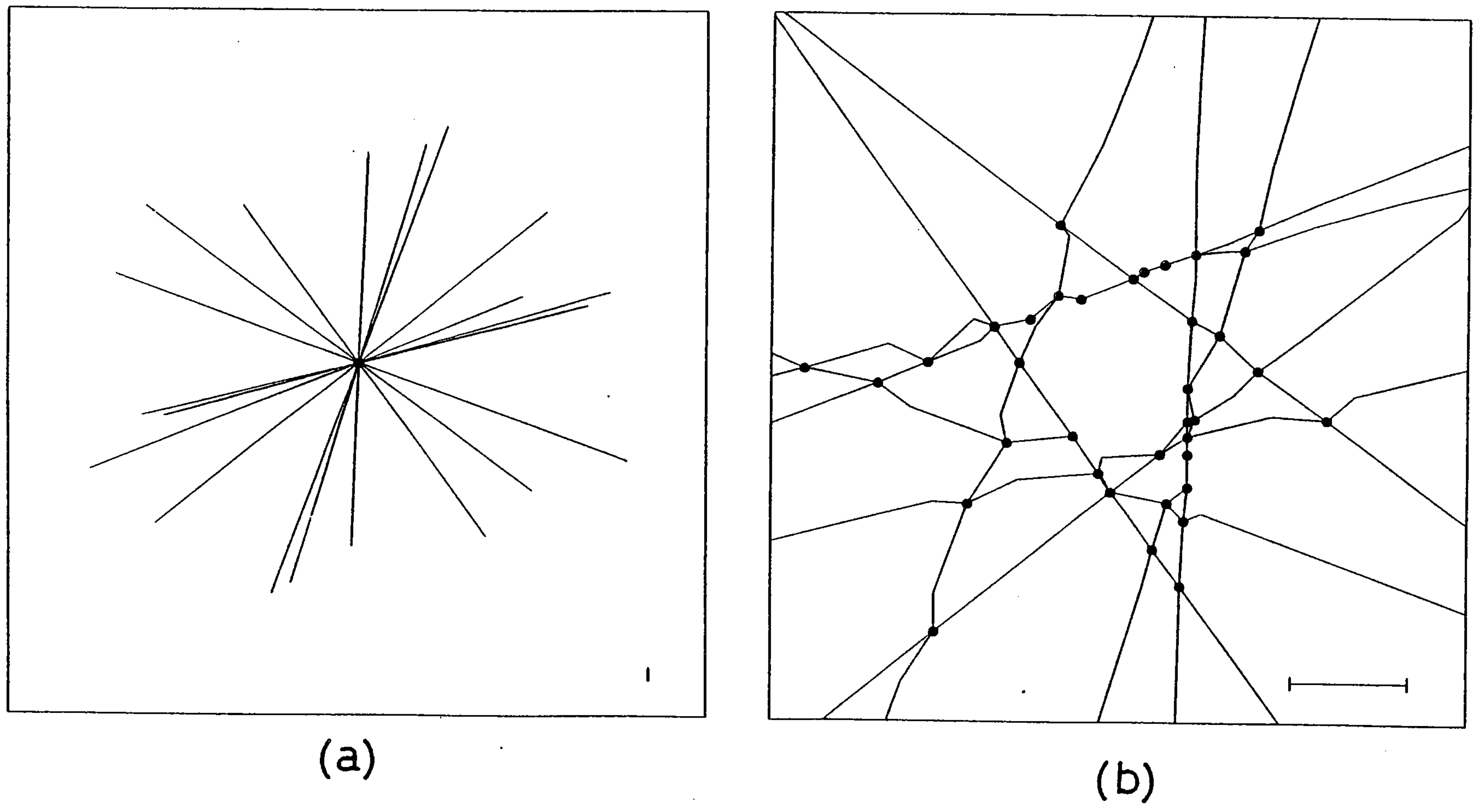


Fig. 7.5. Example 5 — Ten line segments having a common point of intersection: (a) output for $\varepsilon = 2 \times 10^{-6}$; (b) central region of (a) magnified by 10^5; (c) output for $\varepsilon = 10^{-6}$; (d) output for $\varepsilon = 10^{-8}$.

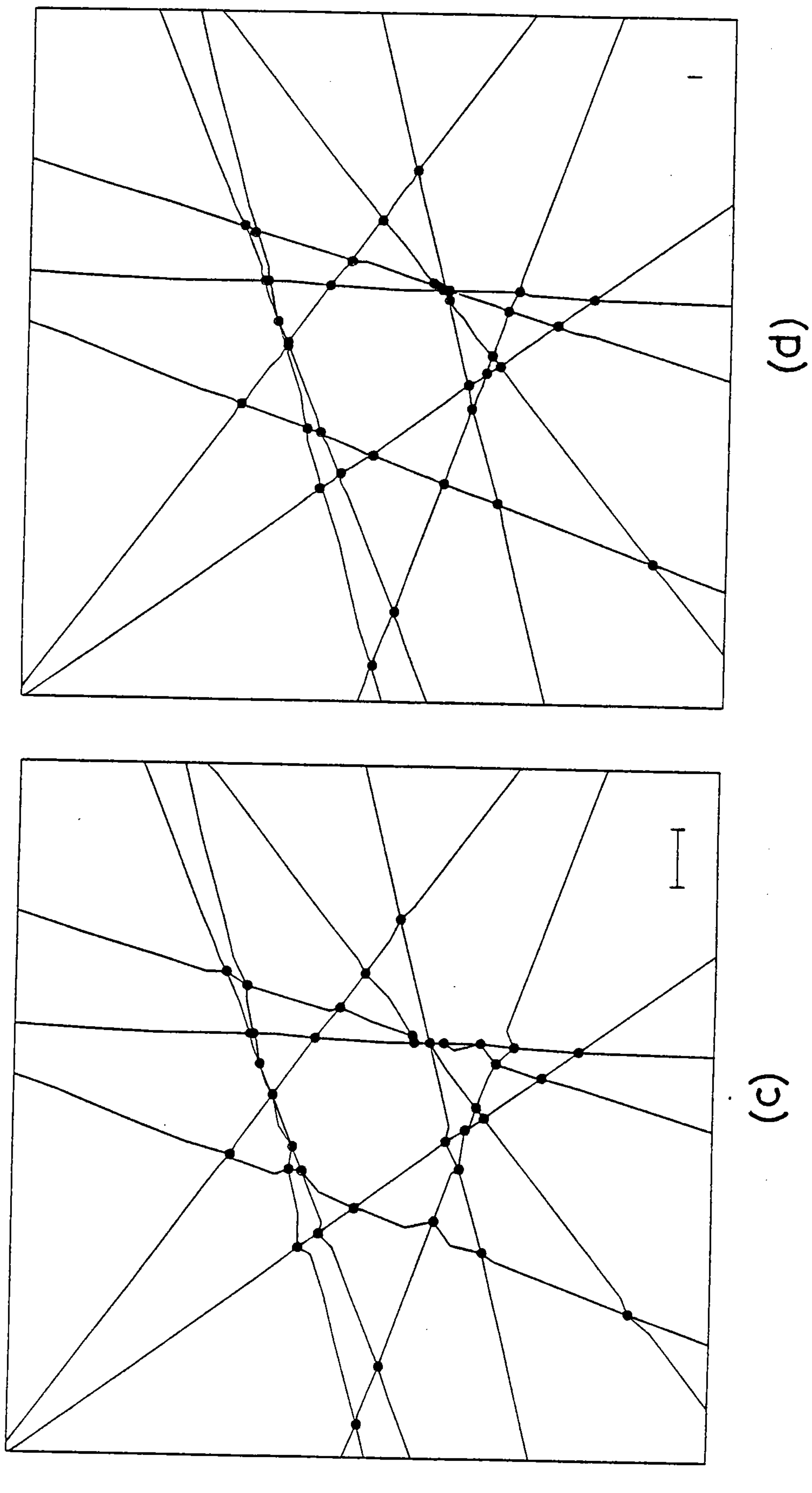


Fig. 7.5. (continued)