01 Aug 1987

Medial Axis Transform using Ridge Following

Richard Mark Volkmann

Daniel C. St. Clair
Missouri University of Science and Technology

MEDIAL AXIS TRANSFORM USING RIDGE FOLLOWING

R. M. Volkmann* and D. C. St. Clair

CSc 87-17

*This report is substantially the M.S. thesis of the first author, completed, August 1987

ABSTRACT

The intent of this investigation has been to find a robust algorithm for generation of the medial axis transform (MAT). The MAT is an invertible, object centered, shape representation defined as the collection of the centers of disks contained in the shape but not in any other such disk. Its uses include feature extraction, shape smoothing, and data compression. MAT generating algorithms include brushfire, Voronoi diagrams, and ridge following. An improved implementation of the ridge following algorithm is given. Orders of the MAT generating algorithms are compared. The effects of the number of edges in the polygonal approximation, shape area, number of holes, and number/distribution of concave vertices are shown from test results. Finally, a set of useful extensions to the ridge following algorithm are discussed.

ACKNOWLEDGEMENT

I am indebted to Karl G. Kempf of INTEL Corporation for his guidance in this study. In addition, I would like to thank Daniel F. Mullen of Washington University, John A. Zammit of Digital Equipment Corporation, and Joseph L. Epplin, Kenneth F. Gerke, and Erwin W. Baumann of McDonnell Douglas Corporation for their useful insights into this work.

CONTENTS

LIST OF ILLUSTRATIONS

Figures Page

LIST OF TABLES

I INTRODUCTION

I.A Definition Of The Medial Axis Transform

The medial axis transform (MAT) is an invertible, object centered, shape representation. It is defined as the locus of points (called medial points) that do not have a unique closest point on the shape boundary. Each of these medial points is annotated with the shortest distance to the shape boundary. In Fig. 1, the triangles represent holes in the shape and the thick curves represent the MAT of the shape. Points B and C are the non-unique closest points on the shape boundary to the medial point A. The dashed circle represents the maximal disk of the medial point A.

Alternately the MAT can be defined in 2-D as the collection of the centers of maximal disks. Maximal disks are disks that cannot be fully contained by any other disk inside the shape. In Fig. 2, the MAT of a rectangle is shown by dashed lines. Some of the maximal disks on the left side of the rectangle are also shown. The shortest distance to the shape boundary from the center of each maximal disk is its radius. The definition of the MAT for 3-D objects is obtained by replacing disks with spheres [23, 26, 27].

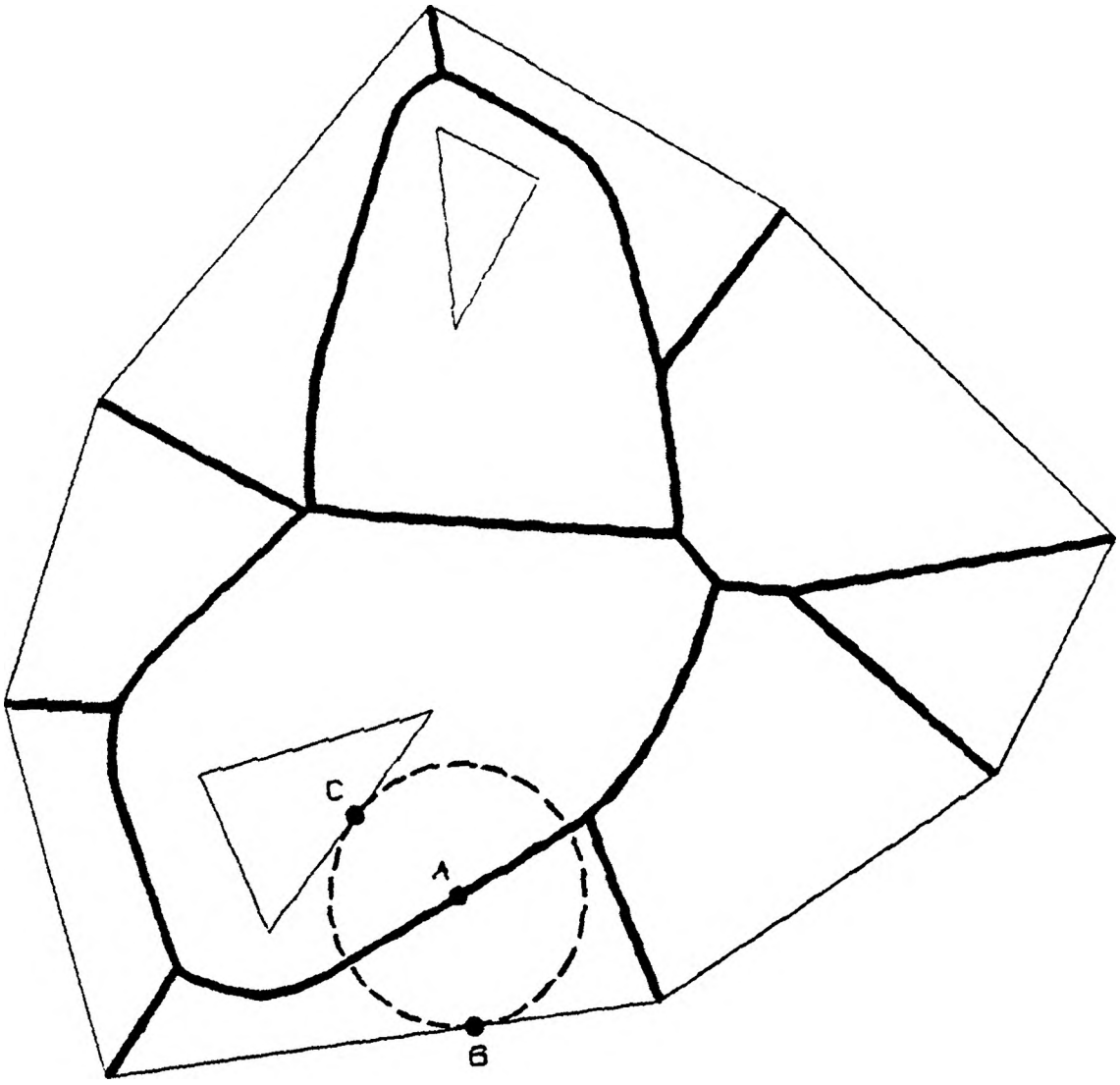

Fig. 1. Non-unique closest points on the shape.

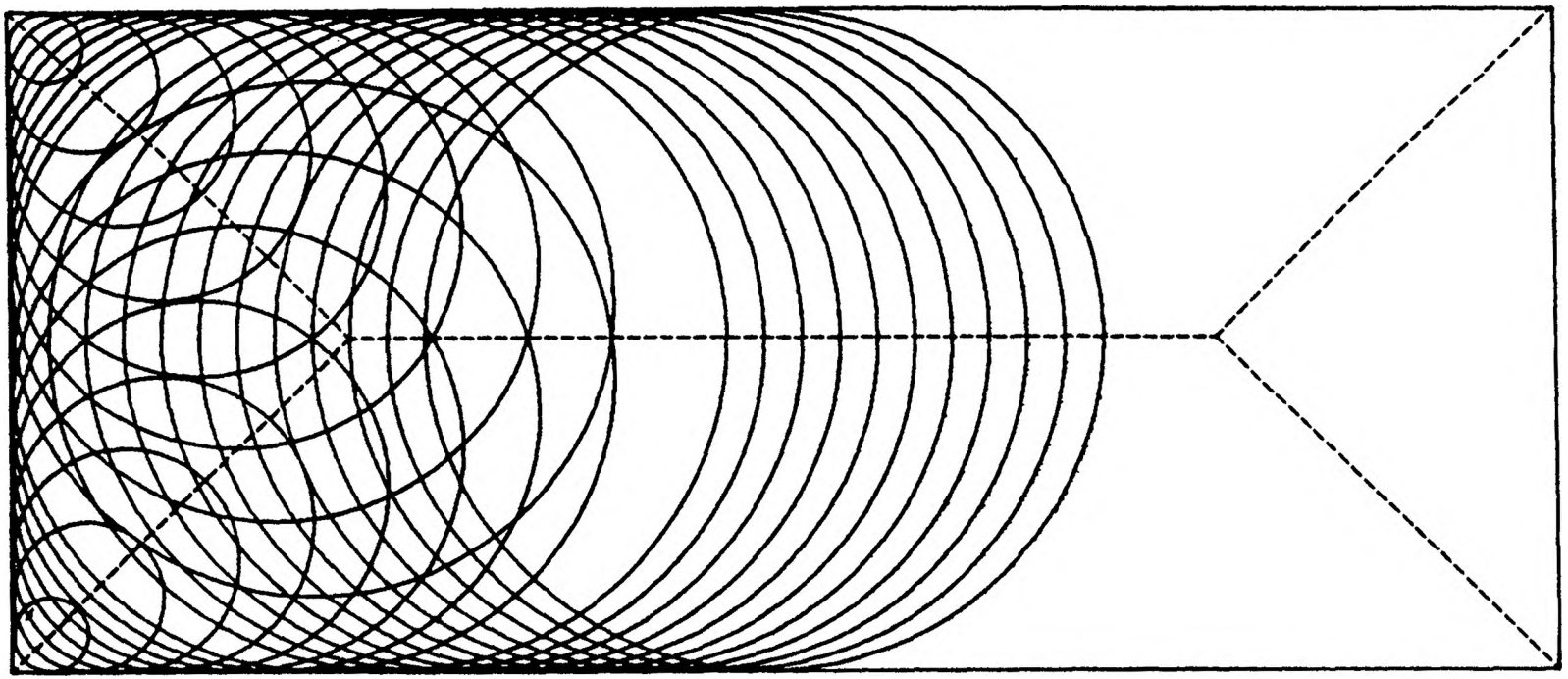

Fig. 2. Maximal disks of a shape.

For shapes that are not convex the MAT exists on the outside of the shape as well. This provides a way of representing internal holes and external concavities. Fig. 3 shows the external MAT, represented by dashed lines, of a shape containing two holes. The MAT of a shape interior is guaranteed to be connected [12].

Other names used in the literature to refer to methods of obtaining the MAT include symmetric axis transform (SAT), Blum transform (for its inventor H. Blum) [8, 9, 10, 11]. Skeletonization and thinning are also used but are less common.

I.B Work Motivation

The motive for this study of MATs has been to find a shape representation that lends itself to automated nesting of irregular two dimensional shapes. Nesting is a space allocation problem in which a reasoner is given a piece of stock, a set of shapes (or templates) to be cut from the stock, and a set of constraints. The goal is to cut the shapes from the stock in such a way that waste is minimized and the constraints are not violated.

Nesting has been proven to be an NP-complete problem [19]. Due to the combinatorics involved, most practical algorithms for problem solution are based on search. Template placement is often conceptualized as a double search approach. In each cycle of placement, the templates

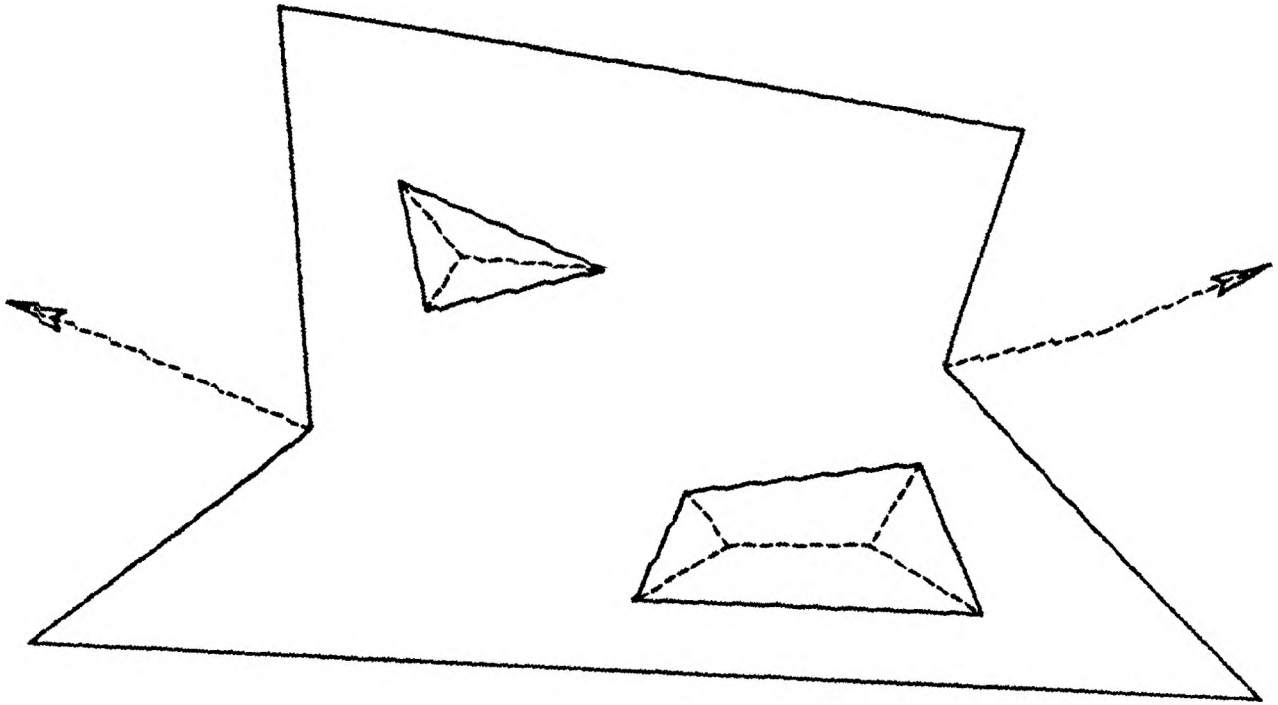

Fig. 3. External medial axis transform.

remaining are searched to select the single template to place next. Once selected, the stock remaining is searched to determine the precise location at which to place the template. This second search step is especially difficult because the template has three continuous degrees of positional freedom; two translational (along X and Y in the plane of the stock) and one rotational (around Z normal to the plane of the stock).

The branches in the MAT of a shape identify the features of the shape. These features provide a level of abstraction that can be used to classify "peninsulas" and "bays" of shapes. If one can assign direction to them and identify compatible peninsula-bay pairings then MATs have the potential to ease the double search nature of the nesting problem. Given a feature in the unused stock, the selection and positioning of the next template to be placed might be determined by matching this feature to the features in the set of unplaced templates.

II HISTORICAL

II.A Applications Of The Medial Axis Transform

The applications for which MATs have been utilized include feature extraction, shape smoothing, and data compression. These uses are described in the following three sections.

II.A.1 Feature Extraction

Feature extraction is the process of identifying the characteristic components of a shape. Once extracted, they can be used to measure the similarity between shapes or portions of shapes. The extraction process is comprised of two basic steps; location of features and description of features. For many shapes, the determination of where one feature ends and another begins can be a difficult problem.

The most extensive work on the use of MATs for 2-D feature extraction has been done by Blum and Nagel [8, 9, 10, 11, 25]. To extract features from the MAT, the medial axis is segmented based on three types of MAT points. The resulting medial axis segments are then classified into seven primitive feature types.

MAT points can be classified as normal points, branch points, and end points. This distinction is based on the number of contiguous sets of points on the shape boundary touched by their associated maximal disks. Points with only one set are end points. Points with two sets are normal points. Points with three or more sets are branch points. Contiguous sets of normal points bounded by branch points and/or end points are called medial branches. These medial branches represent shape features [28].

In Fig. 4, the triangles represent holes in the shape and the thick curves represent the MAT of the shape. Points A, B, C, D, E, F, and G are all medial points. A and G are end points. C and F are normal points. B, D, and E are branch points. The set of medial points between the following pairs of branch and/or end points form features of the shape: AB, BD, DE, EG.

For some shapes, such as a 2-D torus, it is possible for a contiguous set of normal points to connect to itself. In this case, the shape has only one branch and the branch is not bounded by branch points or end points (see Fig. 5).

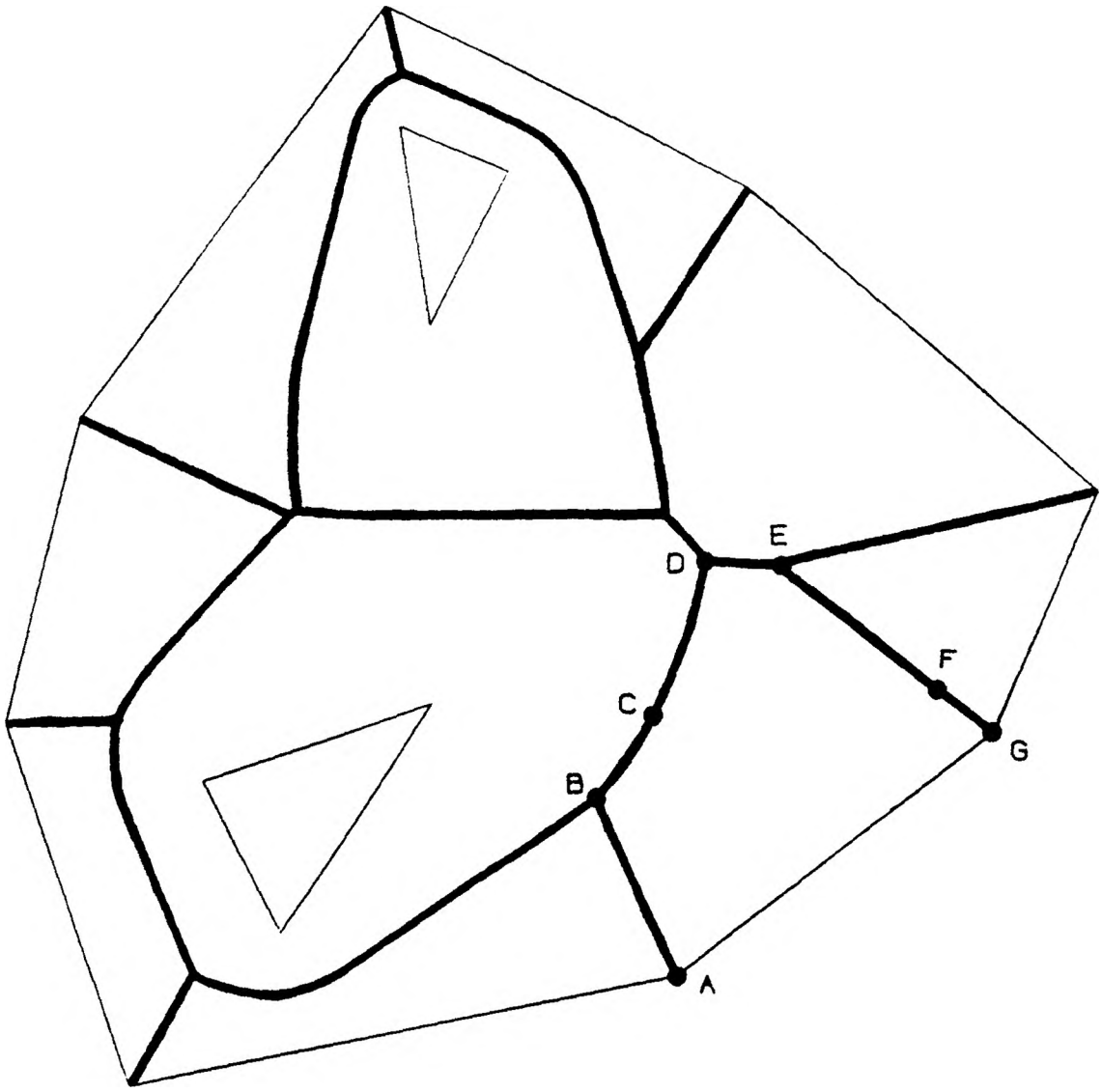

Fig. 4. Medial point types.

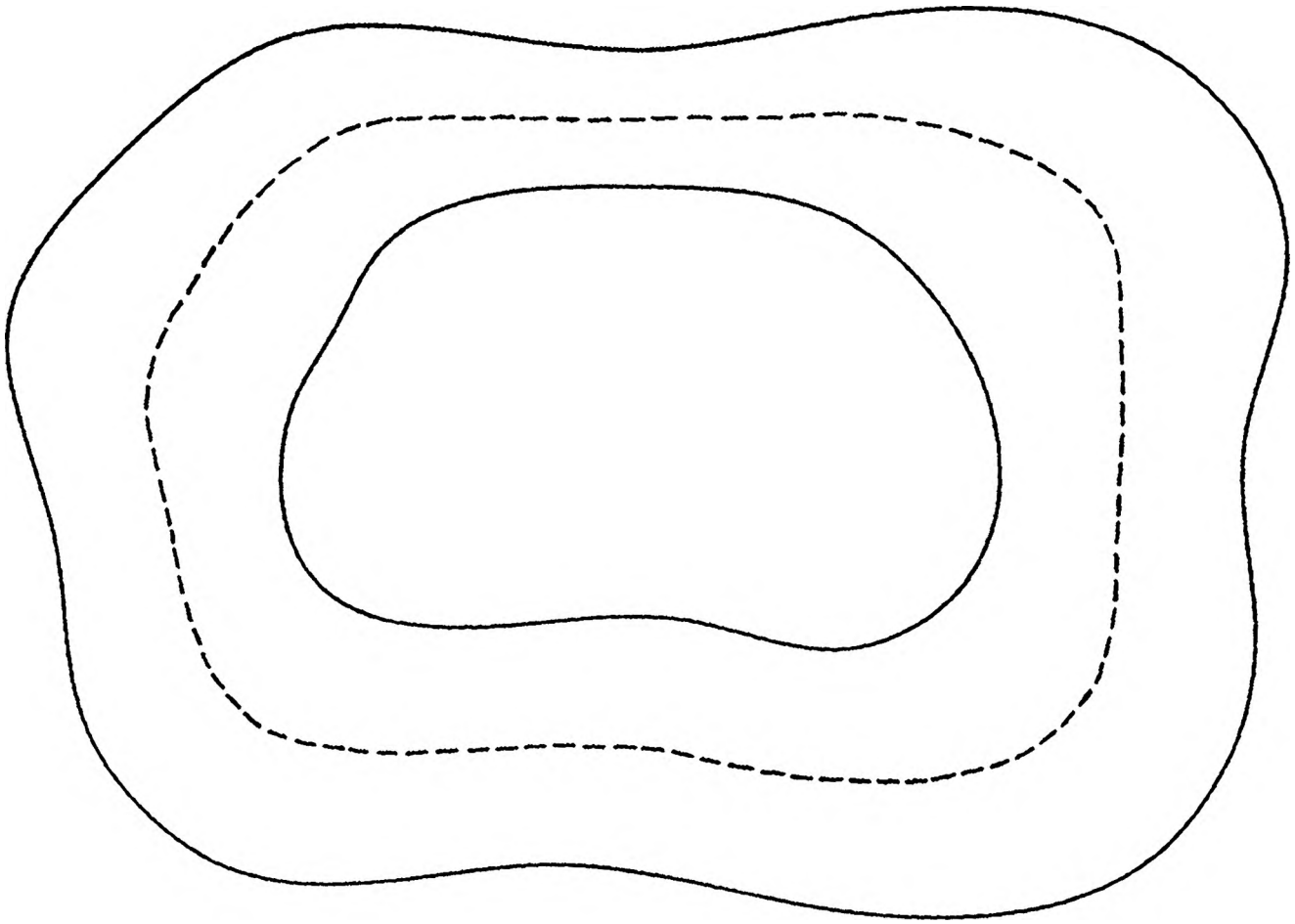

Fig. 5. Medial axis transform of a deformed torus.

To facilitate classification of the medial branches, Blum and Nagel define two functions. The axis function gives the location of the MAT points. The radius function gives the radius of the maximal disk at each MAT point. According to Blum and Nagel, a reasonable shape language should have a small set of primitives. Seven primitives that are characterized by the behavior of the radius function along the medial branch are identified. These primitives are named worm, opening wedge, closing wedge, opening cup, closing cup, opening flare, and closing flare (see Fig. 6). Variations of the primitives are characterized by the curvature of the axis function along the medial branch. The names given to the primitive variations are spiral in left, left circular, spiral out left, straight, spiral out right, right circular, and spiral in right (see Fig. 7).

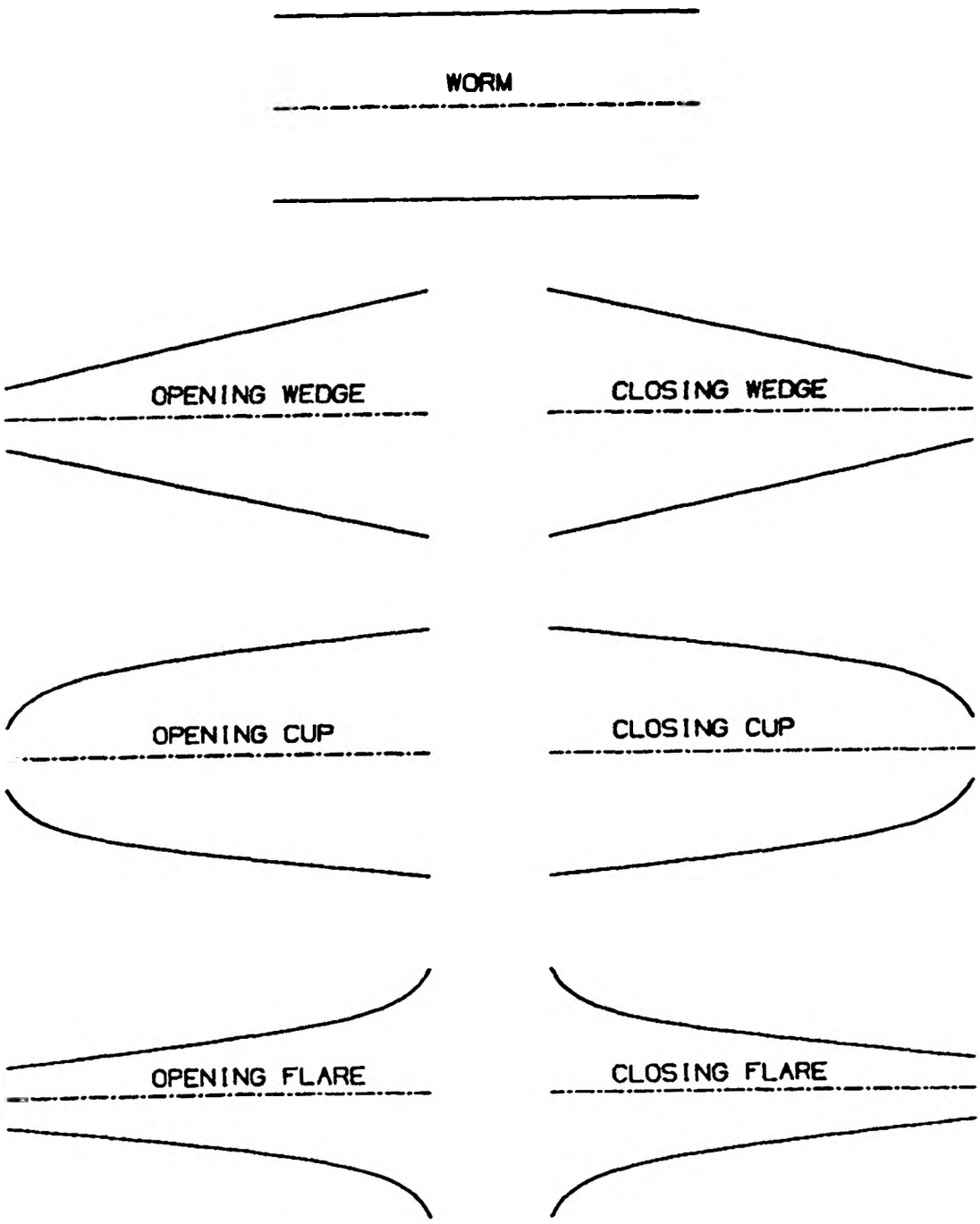

Fig. 6. Medial axis branch primitive types [11].

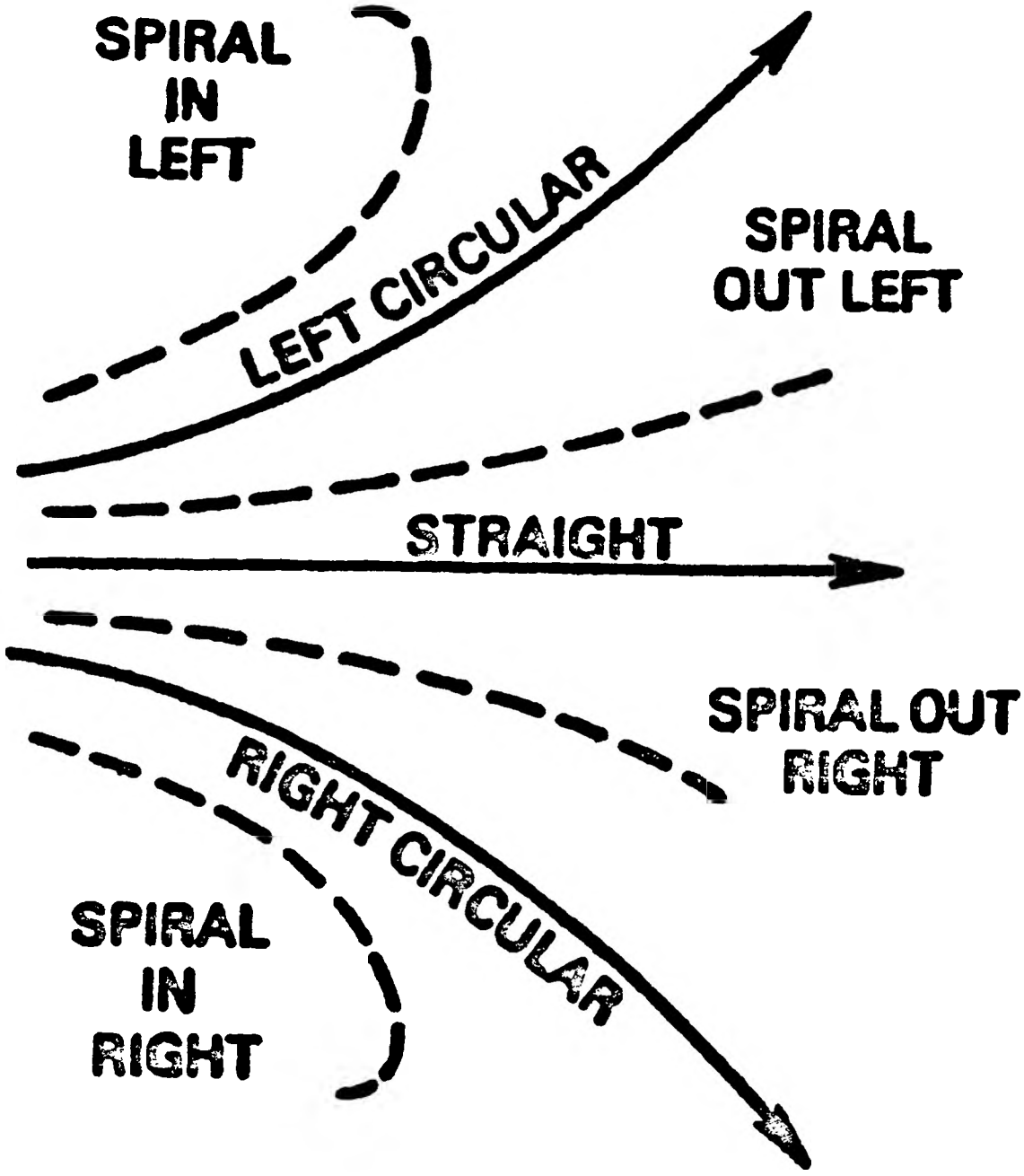

Fig. 7. Variations of the primitive types [11].

II.A.2 Shape Smoothing

Shape smoothing simplifies shape descriptions by eliminating insignificant shape features. The determination of whether a shape feature is significant has, in the past, usually been based on measurements that are relatively local and scale dependent. These measurements do not allow for the fact that a shape feature that is significant on one shape may not be significant on another shape or in a different area of the same shape. In Fig. 8, a shape with two perturbations of the same size is shown. Although they are the same size, the perturbation near the center of this shape is more significant than the perturbation on the left side.

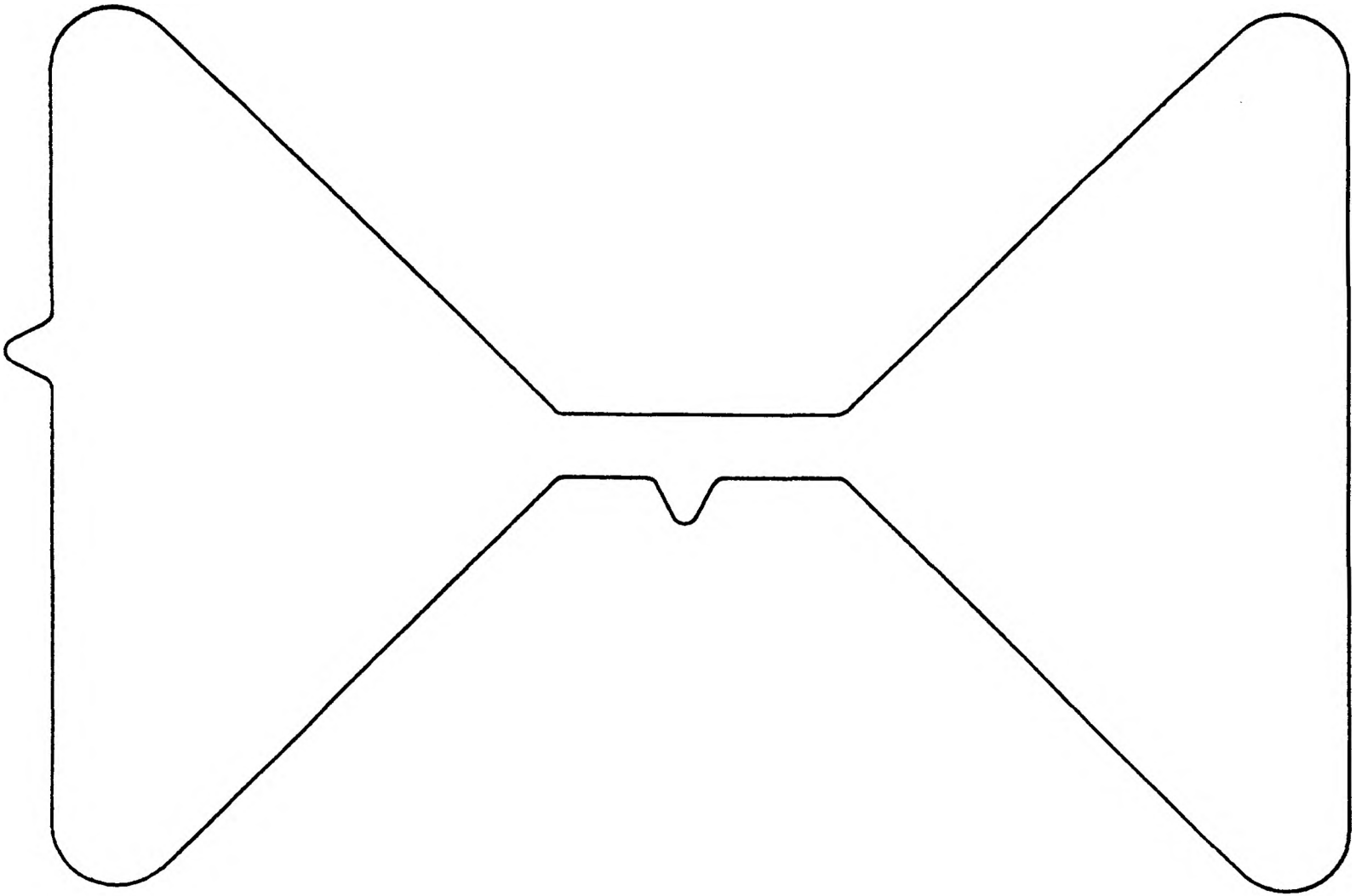

Fig. 8. Significance of shape features [18].

Ho and Dyer [18] have proposed a method of shape smoothing that utilizes the object centered nature of MATs. The advantage of object centered representations over boundary representations is that they make the global properties of shapes more apparent. Their basic approach includes: finding the MAT of the shape, removing insignificant medial branches or portions of medial branches, and reconstructing the shape from the remaining MAT. The top shape in Fig. 9 shows the MAT of a rectangle in dashed lines along with some of the maximal disks near the end of each branch that is bounded by an end point. If it is determined that the centers of the larger disks are the last significant MAT points on their branches then the result of shape smoothing will be the shape that is shown at the bottom of Fig. 9.

II.A.3 Data Compression

A spatial occupancy array is an array in which the element (or pixel) values are determined by a membership predicate p(x,y) whose value is true when the pixel (x,y) is in the shape and false otherwise. A shape that is approximated by a spatial occupancy array can only be recovered as a smoothed version of the original shape. This digitization also affects the properties of invariance to translation, rotation, and scaling [6, 24].

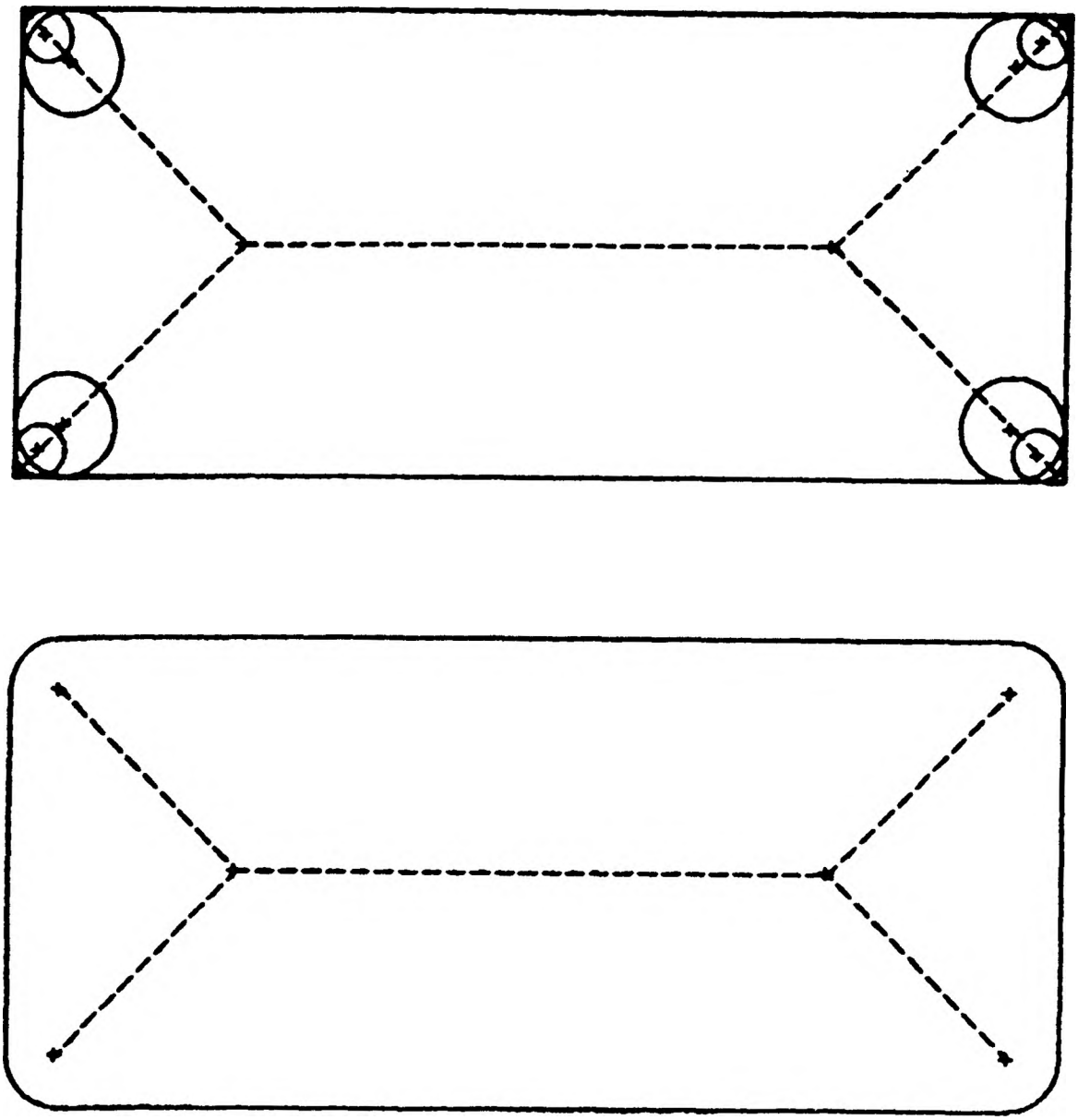

Fig. 9. Ho and Dyer shape smoothing example.

Use of spatial occupancy arrays for generating the MAT of a shape has similar drawbacks. If the medial points of a shape are selected from the elements of a spatial occupancy array then the result of inverting the MAT will be a smoothed version of the original shape. This is true because the medial elements will only be approximations of medial point locations and only a finite number of them will be available for inversion [5, 6].

Ahuja, et al. [2] have suggested that, for applications where some loss of accuracy in retrieved shapes is acceptable, a variation of the MAT could be used to store shape information in a more compact form. Their modification of the MAT replaces disks with squares. The loss of accuracy is controlled by the resolution of the spatial occupancy array.

II.B Methods Of Obtaining The Medial Axis

Methods that have been used to obtain MATs fall into three main categories, namely brushfire, Voronoi diagrams, and ridge following. Generators other than disks, such as squares [1, 6, 32, 33] and rectangles [37], have been used in brushfire algorithms but will not be discussed in this paper. When non-disk generators are used the connectivity of the MAT is no longer guaranteed [1].

II.B.1 Brushfire

The brushfire method operates by eroding the shape boundary in a way that is analogous to a brushfire. A brushfire burns by each point once and only once. If a shape boundary is ignited simultaneously at all points along the boundary then the points where the fire is quenched form the MAT. In Fig. 10, the contour lines in the shape illustrate the spread of a brushfire into the shape at equal time intervals. The MAT of the shape is represented by dashed curves [8, 24].

The heart of a typical brushfire algorithm is shown in the following pseudo-code:

```
Represent the shape by a spatial occupancy array.

Repeat
  For each boundary pixel in the spatial occupancy array
    If the pixel is not medial Then
      delete it (set to FALSE).
Until all remaining pixels are medial
Label remaining pixels with their distance
  from the original shape boundary.
```

The determination of whether a pixel is medial can be made on the basis of local properties. Patterns of the pixels in a 3x3 neighborhood centered at a pixel can be used to test its mediality [16, 29, 35].

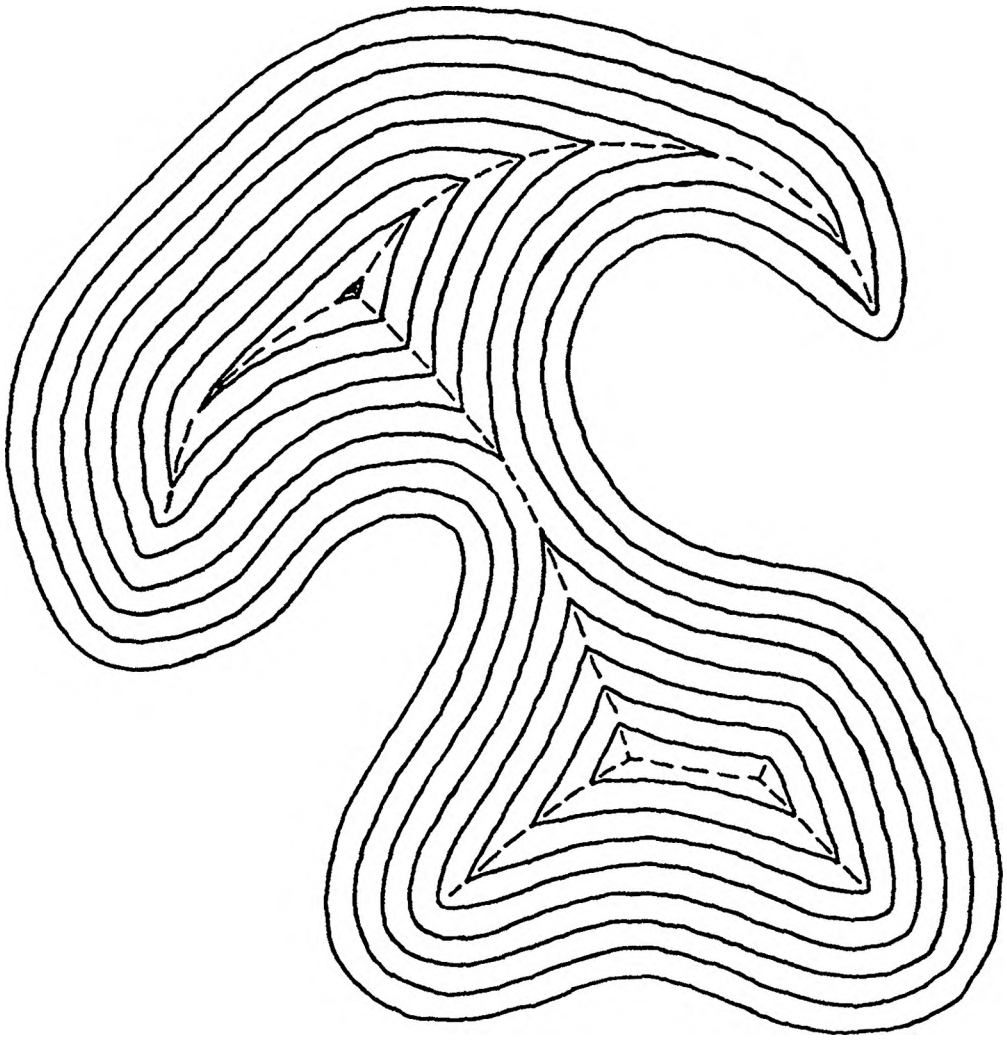

Fig. 10. Spread of brushfire in a shape.

The order in which boundary pixels are evaluated can affect whether the resulting MAT will be connected. Variations of the basic brushfire algorithm devised to address this problem can be catagorized into four types. The first is to check each pixel that is to be deleted to insure that the deletion will not locally disconnect pixels in its neighborhood [14, 32]. The second is to alternate sides (north, east, south, and west) of the shape where deletion is to take place [14, 29, 31, 35]. The third is to evaluate all boundary pixels, marking those to be deleted, before any deletion from the current boundary is performed [4, 7, 32]. The fourth is to propagate deletion simultaneously from four sides (north, east, south, and west) to achieve an isotropic transformation [3, 4, 5, 16, 29, 38]. This approach requires a parallel processor with at least four CPUs that uses shared memory for the spatial occupancy array. Hilditch's method [17] combines the second and fourth variations.

Shape information is retained more accurately by the last two brushfire variations than the first two since the last two variations are more isotropic. The first two variations can produce slightly different MATs depending on the order in which the pixels are evaluated. A drawback of the last two variations is that they must handle the additional problem of insuring that the MAT does not vanish or retain a width of two pixels along some branches.

A disadvantage common to all brushfire algorithms is that the shape boundary is forced to erode inward in the four compass directions instead of eroding normal to the actual shape boundary. This is due to the fact that the spatial occupancy array is composed of square pixels.

The lowest order claimed for brushfire algorithms is $O(n^2)$ where the spatial occupancy has a resolution of n x n [7]. This is an upper bound on the order since the time required actually depends on the thickness of the shape along its medial axis. To see why this is the case, consider a square in which each element of an n x n spatial occupancy array is set to TRUE. Let a diagonal that is one element thick extend across another n x n spatial occupancy array. Several passes, roughly n / 2, will be required to thin the square but only one pass will be required to determine that the diagonal line is already thin.

II.B.2 Voronoi Diagrams

The Voronoi diagram (also known as Thiessen or Dirichlet tessellation) of a point set containing n points in 2-D divides the space into n regions that are bounded by line segments. Each region contains one of the points from the point set and all of the points in the space that are closer to that point than to any other point in the set [15, 34]. Fig. 11 illustrates this concept.

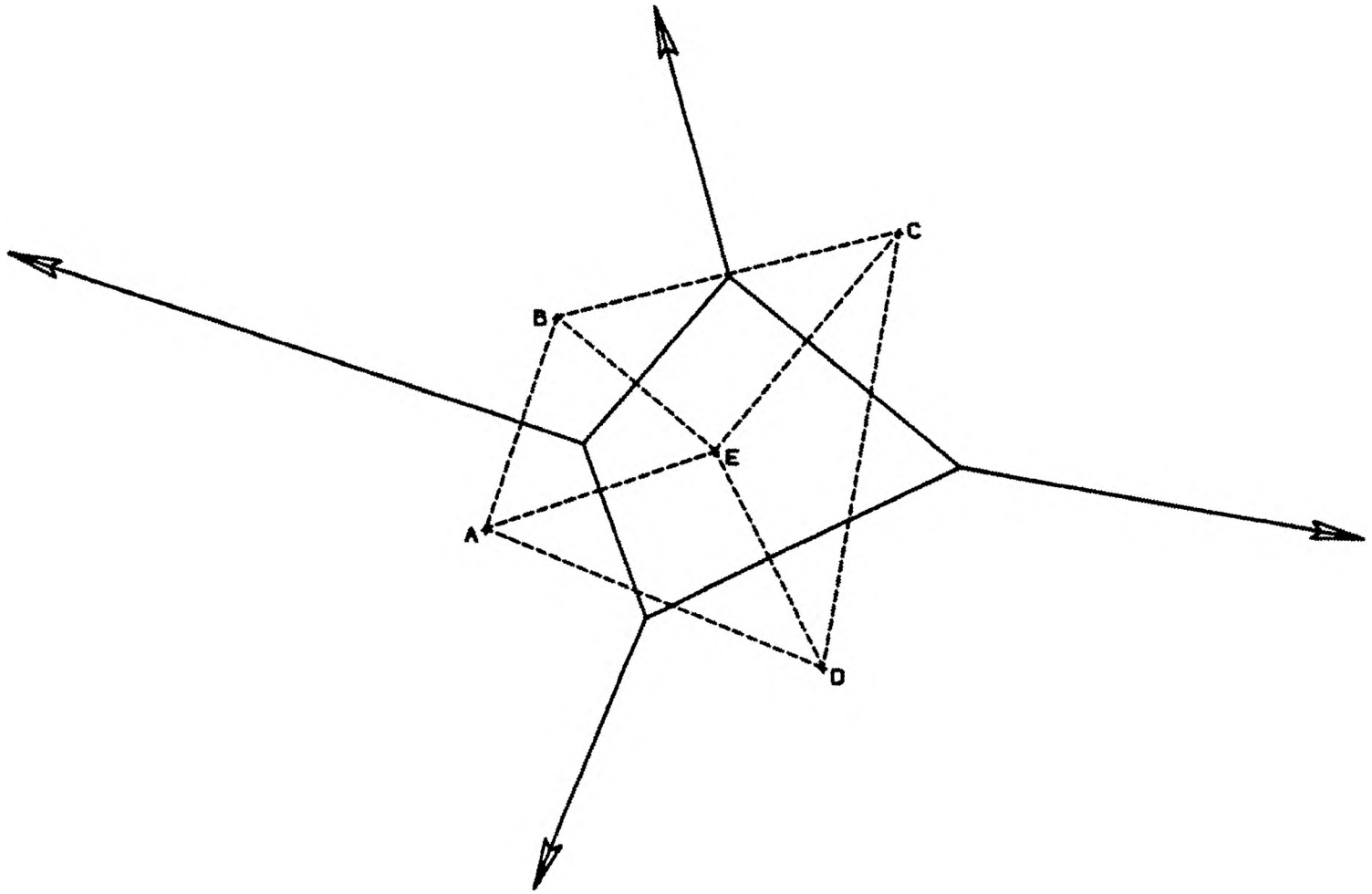

Fig. 11. Voronoi diagram of a point set.

A variation of this idea is to use a set of line segments instead of a point set to generate the regions. When this is done the regions in the Voronoi diagram are bounded by linear and parabolic edges (see Fig. 12).

For a polygon, the MAT is a subset of the Voronoi diagram obtained using the edges of the polygon as the set of line segments. The MAT is acquired by removing the two region bounding edges incident with each concave vertex of the polygon and augmenting the remaining region bounding edges with radius functions (see Fig. 13).

The method for construction of Voronoi diagrams of polygons proposed by Lee [22] is based on the divide-and-conquer technique. The basic steps are shown below:

1) Create an ordered set of elements consisting of all edges and concave vertices.
2) Divide the set into two contiguous sets.
3) Find the Voronoi diagram of each set recursively.
4) Merge the two Voronoi diagrams.

In step 4, a "merge curve" that is the bisector of two sets of elements is constructed. The merge curve is composed of line segments and parabolic segments. The final merge curve is the output of the algorithm.

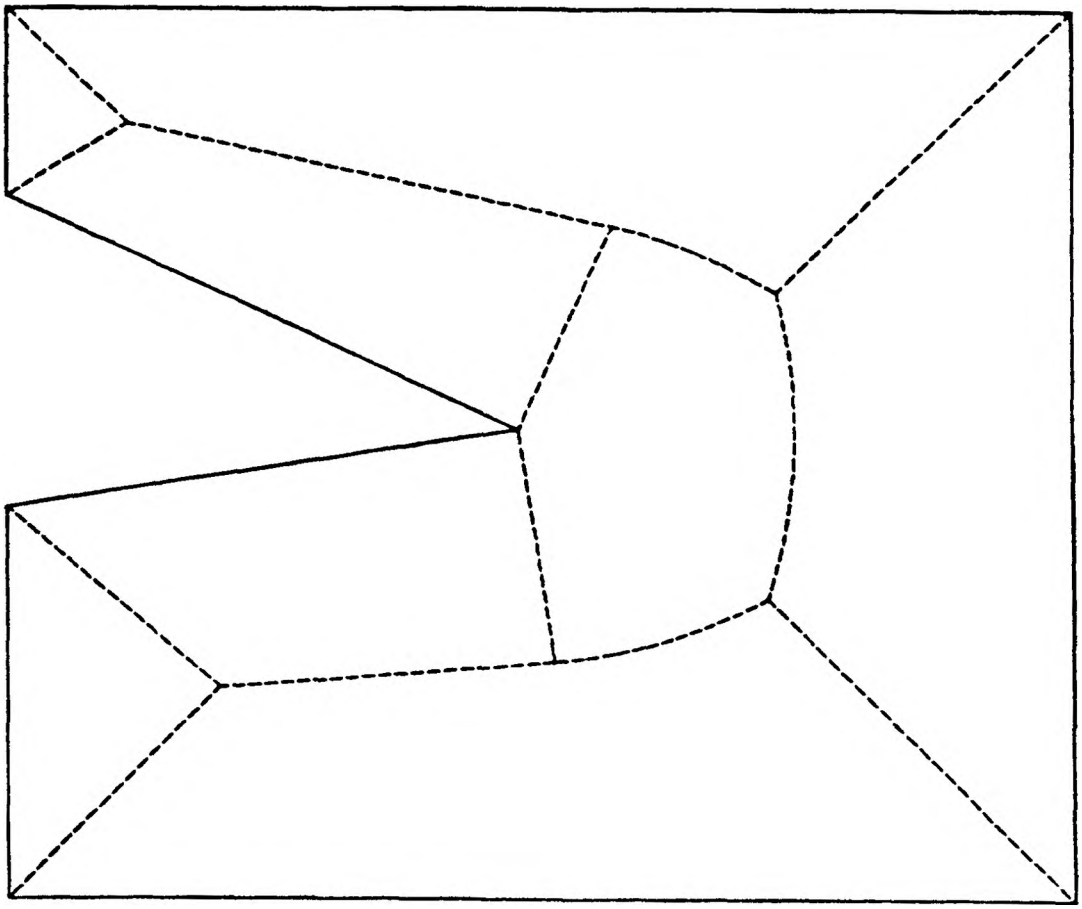

Fig. 12. Voronoi diagram of the interior of a polygon.

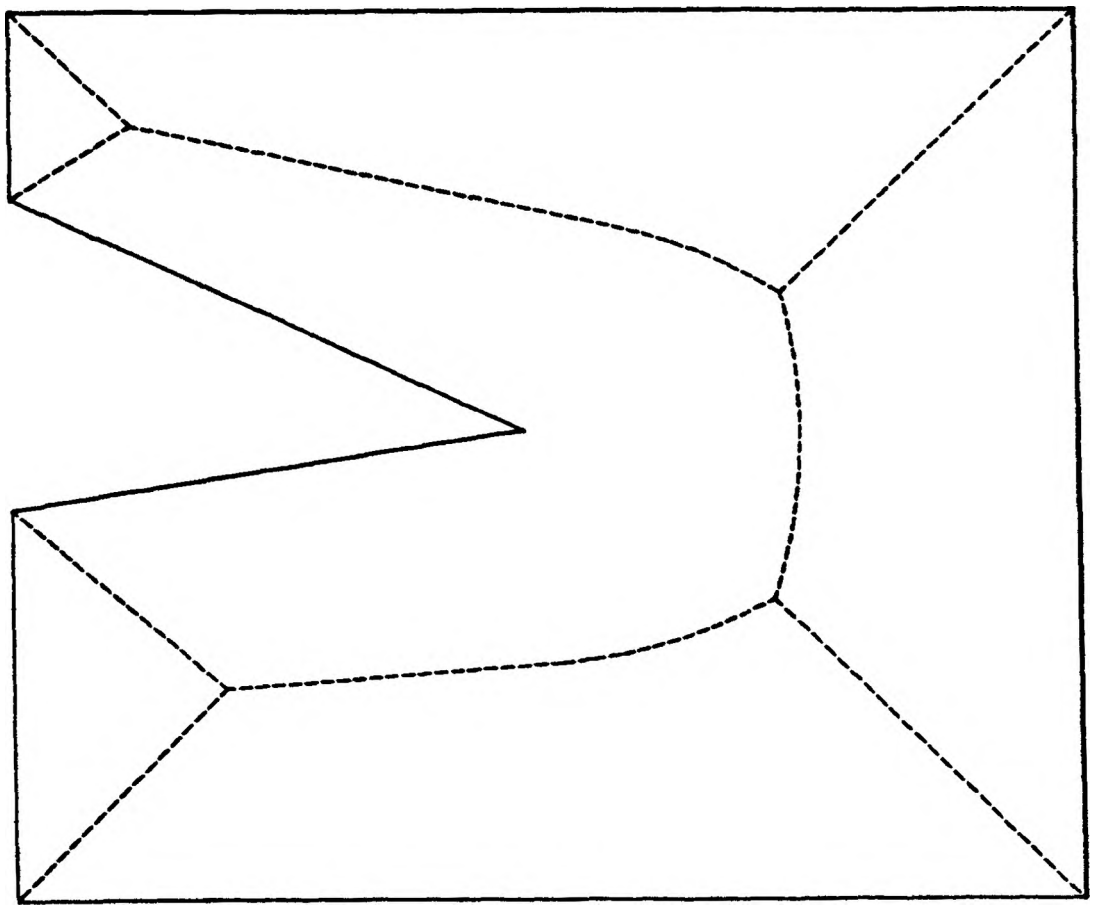

Fig. 13. Medial axis transform of the polygon in Fig. 12.

Using the equations of the line segments and parabolic segments, any number of medial points can be computed along them. Details of this algorithm can be found in [20, 21, 30]. The lowest order claimed for algorithms that compute the Voronoi diagram of a polygon is O(n log n) where n is the number of edges plus the number of concave vertices in the polygon [20, 22].

II.B.3 Ridge Following

Ridge following is a method of locating medial pixels in a spatial occupancy array that attempts to minimize the number of pixels that must be tested for mediality while guaranteeing that all of them will be found [13]. To obtain the MAT of the inside of a digitized shape, interior pixels are examined to find a starting medial pixel. The remainder of the medial pixels are found by searching along ridges (or branches) from the known medial pixel. Searching along ridges is accomplished by examining the neighboring pixels of known medial points. Neighboring pixels are the pixels in a 3x3 neighborhood centered at a given pixel. Due to the connectivity of MATs on the inside of shapes one can be assured that all of the interior medial pixels will be reached from the starting medial pixel.

As in the brushfire method, the resulting MAT is composed of a set of pixels. However, in ridge following the test for mediality of pixels is taken directly from the definition of medial points using distances to the shape boundary as opposed to examining patterns of pixel neighborhoods. For this reason, the ridge following method is more accurate than the brushfire method.

Accuracy can be measured by examining the difference between the original digitized shape and the digitization of the inverted MAT. In Fig. 14, (a) shows the pixels that are turned on in a spatial occupancy array, (b) shows the pixels that would be selected as medial pixels by a typical brushfire algorithm, and (c) shows the pixels that would be turned on as a result of inverting the MAT in (b). The *'s in Fig. 14 (c) also represent the pixels that would be selected as medial in the inverted shape. It is clear that a considerable loss of accuracy can occur during the inversion process. This is further evidenced by the difference between the two MATs.

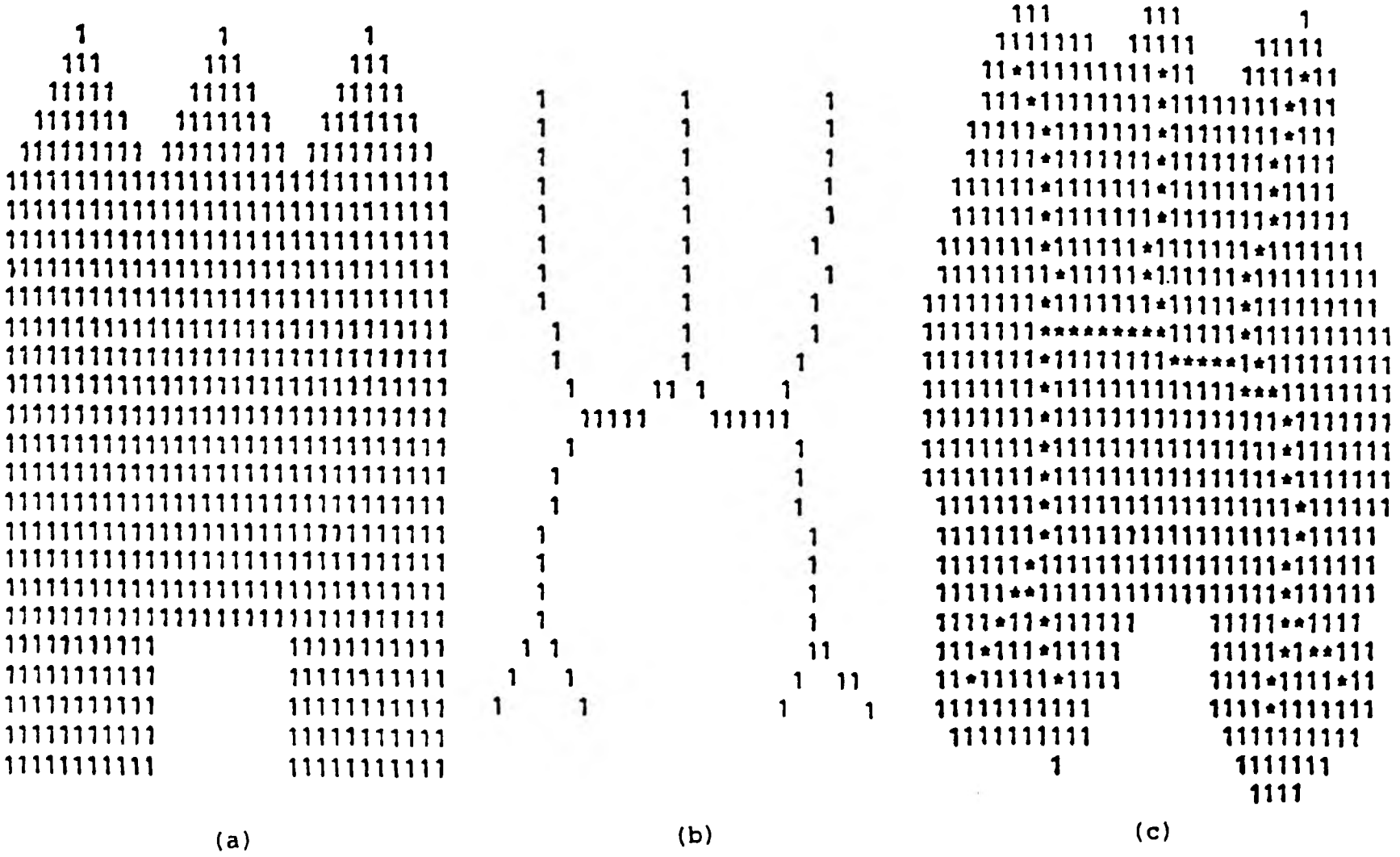

Fig. 14. Inversion of a brushfire MAT [36].

The ridge following algorithm is O(n log n) where n is the number of edges in the polygon. This order can be obtained by examining the details of the improved algorithm described in section III. However, this order is misleading since several factors other than the number of edges affect the time required. Other factors include shape area, proximity of edges, angle measures, and concavities. The effects of these factors are also discussed in Section III.

II.C Reconstruction Of Shape From MAT

The spatial occupancy array of a shape corresponding to its MAT is easily obtained by selecting a finite number of points along the MAT and recording the spatial occupancy of their maximal disks.

The literature contains few references to generating a polygonal approximation for a shape from its MAT. However, it seems that the algorithm described below might work well. To obtain an approximation using splines one could use this procedure to obtain a polygonal approximation and then fit splines to sections of the polygonal approximation.

Select a finite number of points along the MAT.
(The number of medial points selected and the distance between them along the medial axis will affect the quality of the polygonal approximation.)

```
For each selected MAT point A
  If A is a normal point Then
    Find the MAT points B and C that
      immediately precede and follow A.
    Find the two lines m and n (one on each side of the
      medial axis) that are tangent to the maximal disks
      of B and C.
    Find the two points D and E on the maximal disk of A
      (one on each side of the medial axis) where the
      tangent to the maximal disk of A is parallel to
      lines m and n on the same side of the medial axis.
    Include points D and E in the polygonal
      approximation.
  Else
    If A is an end point Then
      Find the MAT point B that immediately precedes A.
      Find the two lines m and n (one on each side of
        the medial axis) that are tangent to the maximal
        disks of A and B.
      Find the points of tangency, C and D, of lines m
        and n (one on each side of the medial axis) on
        the maximal disk of point A.
      Include points C and D along with several other
        points between C and D on the side of the maximal
        disk opposite from point B in the polygonal
        approximation.
```

```
Else
  If A is a branch point Then
    Do nothing.
    If enough closely spaced medial points are
      selected then sufficient polygonal
      approximation points will be obtained
      from the medial points that are near A.
```

III THEORETICAL APPROACH

The MAT generating algorithm being proposed is an improved version of the ridge following algorithm given by DeSouza and Houghton [13]. After presenting the basic components of the algorithm, three sub-algorithms will be discussed. For each of these, the DeSouza and Houghton approach will be presented, its shortfalls will be discussed, and methods to overcome these will be given.

III.A Ridge Following Algorithm

The algorithm operates on polygons. It can be modified to handle shapes containing curves as will be described in Section V.B.1.

The input data consists of the coordinates of the vertices in the shape boundary (including holes) and the grid size to be used for the spatial occupancy array. The output of the algorithm is a set of medial point coordinates together with the radius value (or closest distance to the shape boundary) for each medial point. The medial points returned are ordered by increasing x within increasing y, which is the same order that they are stored in the spatial occupancy array.

The basic steps of the algorithm are shown below:

Digitize the shape boundary, not including the shape interior, in a spatial occupancy array.

Find any pixel that is inside the shape (seed pixel).
Search vertically up and down from this interior pixel to find the first medial pixel (existence is guaranteed since the MAT is connected).
Find the rest of the medial pixels by recursively examining all of its neighboring pixels.

III.B Finding A Point Inside A Polygon

In the DeSouza and Houghton method an interior point of the polygon is found by performing the following steps:

Find the minimum x and y values of the polygon vertices.
Find the first two intersections encountered when moving from the point (minimum x, minimum y) along a ray making an angle of 45 degrees with the positive x axis.
Find the midpoint of the line segment connecting these two points.

For many shapes, such a ray will not intersect the shape at all. This approach is therefore not sufficient for all polygons.

The problem of finding an interior point of a convex polygon can be solved by finding the centroid of the polygon. For a non-convex polygon, the centroid may lie outside the polygon. It is possible to find a convex region of a non-convex polygon. Once this is found, one

can find the centroid of the convex region to obtain an interior point. This method is outlined below:

Let A be the highest concave vertex in the polygon.

Let B be the highest vertex in the polygon.

Let C be the next vertex in the
counter-clockwise direction from B.

Let D be the next vertex in the
clockwise direction from B.

If C is below A Then
  Change C to be the point on the line
  segment BC with the same y value as A.

If D is below A Then
  Change D to be the point on the line
  segment BD with the same y value as A.

The centroid of B, C, and D is
in the interior of the polygon.

Note that the concavity of each vertex must be determined anyway for the purpose of avoiding incorrect mediality tests (described in section III.C). This approach works without modification on polygons with polygonal holes. If the digitized shape boundary has more than one bounded set of interior pixels then an interior seed pixel will be required for each bounded set. To use the above procedure

for finding each seed pixel the polygon must be subdivided into several polygons such that the digitized shape boundary of each has only one bounded set of interior pixels.

III.C Determining Whether A Point Is Medial

By definition, a point is medial if it does not have a unique closest boundary point. The accuracy of shape represention when using spatial occupancy arrays is dependent on the grid size. To allow for this, the definition must be relaxed to consider pixels as medial if the difference between the distance to the closest and the second closest boundary points is less than a tolerance of one grid size.

When testing points for mediality, distances from points to line segments in the polygonal approximation of the shape are measured without finding the coordinates of the closest points on the line segments. Remember that the distance from a point to a line segment can be greater than the distance to the continuous line. In Fig. 15, consider the distances from the points C and D to the line segment AB. For C the desired distance is the perpendicular distance to the line AB. The foot of the perpendicular through D to the line AB, point E, is not on the line segment AB. The desired distance for D is therefore the distance between D and the closest line segment endpoint,

namely B. This distance is greater that the distance from C to E.

If the two closest line segments are adjacent and form a concave angle then the closest point on each line segment will be the endpoint that is common to the line segments. To avoid an incorrect mediality test, this case requires that one of the line segments be replaced by the next closest line segment. In Fig. 16, all of the points in the cross-hatched region would be incorrectly classified as medial if this specal case were not observed.

In the DeSouza and Houghton method, the two closest edges are found by performing the following steps:

```
Find the six closest vertices to the point.
For each of the six vertices
  Calculate the distance to the two line
    segments incident at the vertex.
Select the two edges with the shortest distances.
```

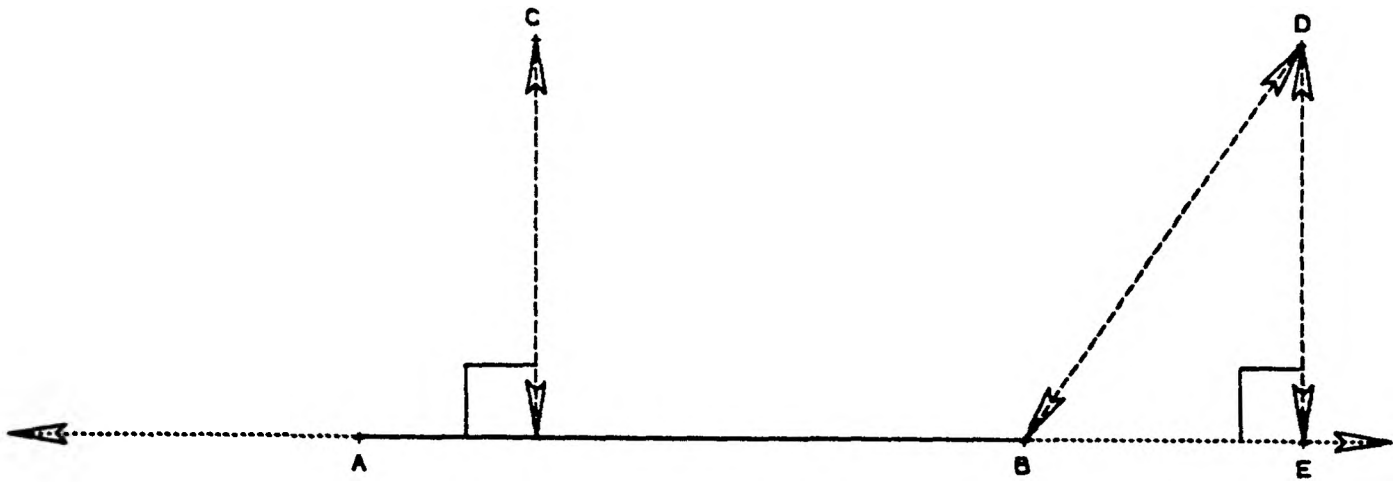

Fig. 15. Distance from a point to a line segment.

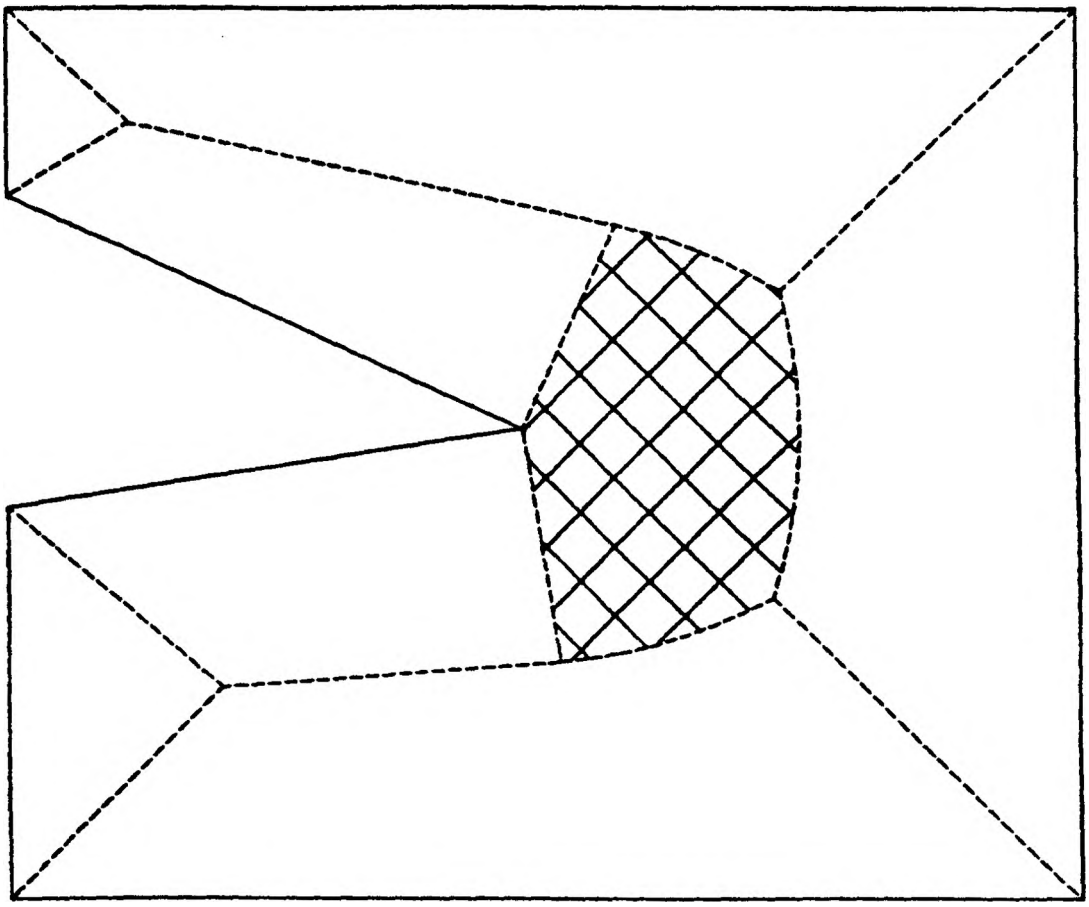

Fig. 16. Concave angle special case.

The number six was chosen rather arbitrarily. Fig. 17 shows a shape where this approach would fail. The six closest vertices to point P are A, H, I, C, D, and E. The two closest edges that are adjacent to these vertices are HI and DE. By using these edges, point P would appear to be medial when in fact it is not. Therefore, their method is not sufficient for all polygons. The only way to guarantee that the two closest line segments will be found is to compute the distance to all of them or find a way of insuring that any ignored line segment cannot possibly be one of the closest two.

Calculating the distance from a point to every line segment in a polygon becomes a very time consuming task as the number of line segments in the shape increases. To reduce the number of distance computations a scheme involving a lower bound on the distance to each line segment has been devised (see Fig. 18). This lower bound is the greater of the lower bound on the x distance and the lower bound on the y distance. The lower bound on the x distance is zero if the x values of the line segment endpoints are on opposite sides of the x value of the point. Otherwise, it is the smallest x distance from the point to one of the endpoints. The lower bound on the y distance is defined similarly.

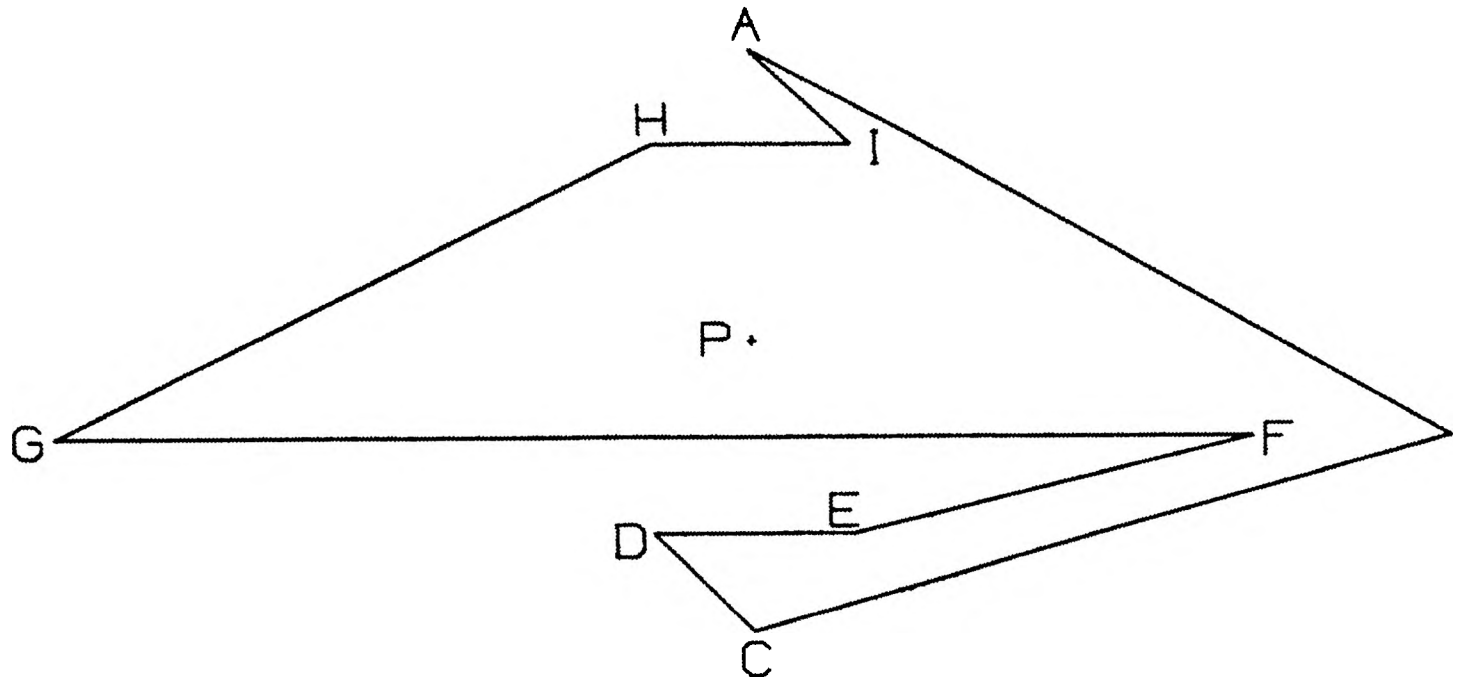

Fig. 17. A shape that the DeSouza and Houghton ridge following algorithm cannot handle.

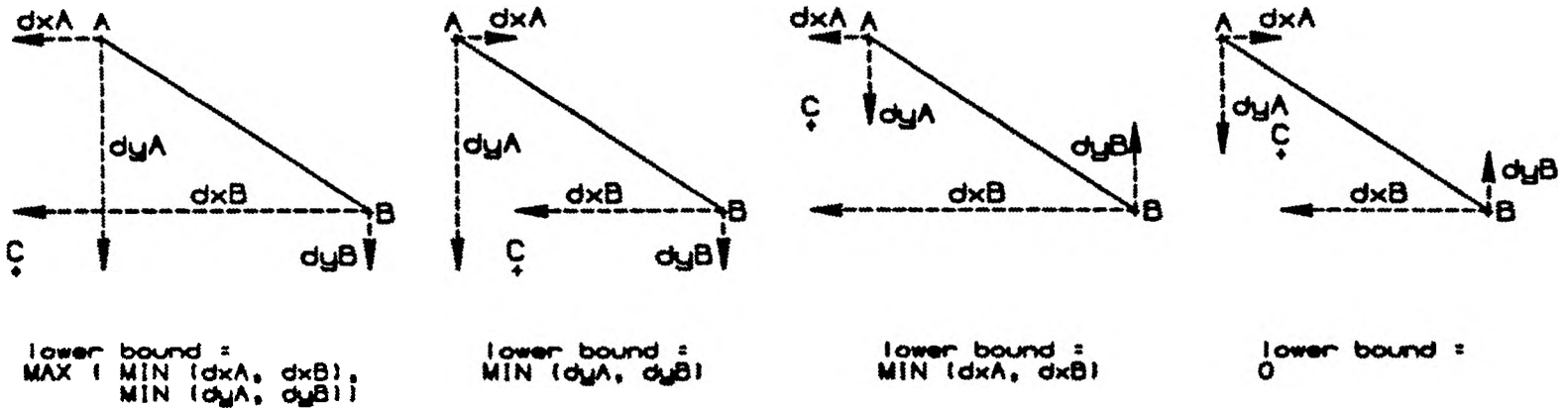

Fig. 18. Lower bound on the distance from point C to line segment AB.

To determine the mediality of a pixel, this lower bound is computed for each line segment in the polygon. The line segments are then sorted on ascending lower bound (recall that the sort can be O(n log n) [34]). Distances to the line segments are computed evaluating them in this order. Whenever the lower bound on the distance to the next line segment is greater than the shortest distance calculated so far, the distance calculations for the rest of the line segments can be disregarded.

The benefits of this scheme depend on the proximity of polygon edges to the pixels being tested for mediality. If relatively few edges have small lower bounds on their distance to the pixel, then a significant number of distance calculations will be avoided.

III.D Elimination Of Excess Medial Pixels

Due to the tolerance used in determining the mediality of the pixels in the spatial occupancy array, the resulting MAT is likely to have a thickness of more than one pixel along some branches. This is especially noticeable at branch points. Whether this is of any concern depends upon the application.

If there is a need to further thin the MAT, the quality of the retained pixels can be examined to determine which ones to eliminate. The quality of a medial pixel can be measured by the difference between the distance to the

closest and the second closest boundary points. Low values of this measure indicate better medial pixels. This is the same value that is compared to the tolerance to determine whether a pixel is medial, so it can be saved for this purpose.

In the DeSouza and Houghton method, excessive medial pixels are eliminated by performing the following steps:

For each medial pixel whose quality is better than the quality of any other medial pixel in the surrounding 5 x 5 neighborhood
  Check the mediality of 16 evenly spaced points inside the grid of this medial pixel to find a medial point of better quality.
  Save this medial point.
  Eliminate all medial pixels in the surrounding 5 x 5 neighborhood.

This approach certainly eliminates excessive medial pixels and raises the quality of the remaining medial points. Its downfall, however, is that it reduces the resolution of the resulting MAT and makes the connectivity of the medial points unclear.

Another approach is to use the notion of simple pixels combined with the quality measure. A simple pixel is defined as a pixel that can be removed without damaging the connectivity of the medial pixels. The following steps

illustrate this method:

```
Repeat
  For each medial pixel
    If its quality is worse than the quality
       of each of its medial neighbors
       AND
       it is a simple pixel Then
      Eliminate it
Until no more medial pixels can be removed
```

The steps used to determine whether a pixel is simple are shown below:

Create an 8 element array of Boolean values to correspond to the 8 neighbors of the pixel.

Use the following direction numbers

```
3   2   1
  \ | /
4 - * - 0
  / | \
5   6   7
```

where diagonal neighbors are those with an odd direction number and non-diagonal neighbors are those with an even direction number.

Set the array elements corresponding to medial

pixels to TRUE.

For each diagonal neighbor
  If both of the adjacent non-diagonal
     neighbors are set to TRUE Then
    set the diagonal neighbor array element to TRUE

Count the number of TRUE elements in the array.

Count the number of changes from TRUE to FALSE in the
  array.

The pixel is simple if
  (TRUE elements > 2) AND (Changes = 1).

In this approach the resolution of the resulting MAT is the same as the resolution of the unthinned MAT. Also, the connectivity of the medial pixels can still be determined by pixel adjacency in the spatial occupancy array.

IV EXPERIMENTAL RESULTS

To test the performance of the improved ridge following algorithm, the four shape characteristics that were expected to have the most impact on run time were selected: number of edges, area, number of holes, and number/distribution of concave vertices. The shapes that were used for testing were chosen so that one of the four characteristics could be varied while the other three remained constant.

CPU times were gathered for four different phases of the algorithm: initializing, finding the first medial pixel, finding the remainder of the medial pixels, and eliminating excess medial pixels. The initialization phase includes digitizing the shape boundary, calculating the type (convex or concave) of each vertex, combining adjacent colinear line segments, and computing the coefficients in the line equations for each line segment. The phase to find the first medial pixel also includes finding an internal seed point for the polygon.

IV.A Varying Number Of Edges

The shapes used for this set of tests were regular polygons. The number of edges was varied from 3 to 70.

CPU times for initialization and finding the first medial pixel were expected to increase linearly with the number of edges. The plots shown in Fig. 19 and Fig. 20 are rather erratic but CPU time appears to increase as the number of edges increases. The linear correlation coefficients are approximately 0.64 for Fig. 19 and 0.74 for Fig. 20. CPU time required for these phases was extremely small compared to the total CPU time required.

CPU time to find the remaining medial pixels was expected to be O(n log n) with respect to number of edges but is obviously higher as shown in Fig. 21. This was the first indication of the role that angle measure plays in the order of the algorithm. As the number of edges in the regular polygons increases, the angle measures increase. This allows many more pixels to be counted as medial due to the tolerance check on the difference between the distances to the closest and second closest edges.

The same comments apply to the CPU time required to eliminate excess medial pixels (also referred to as thinning the MAT). All of the regular polygons were created with their centers at the origin and their first vertices on the positive x axis. The upward turn that occurs in Fig. 21 and Fig. 22 at around 16 edges can be explained by noting that the branch from the shape center to the first vertex is trivial in comparison to the other branches. Since it is oriented in one of the four compass

directions with respect to the spatial occupancy array, fewer excess medial pixels are generated. This is similar to the stair stepping effect that is seen when non-horizontal, non-vertical lines are drawn in raster graphics. At around 16 edges, the effects of this trivial branch dissipate. The levelling off that occurs in the plot for the thinning phase can be explained by noting that at around 40 edges nearly every pixel in the interior of the shapes is considered medial. Additional edges no longer translate into a large increase in excess medial pixels so the time required to eliminate them does not increase significantly. Fig. 23 shows a plot of the total CPU time required.

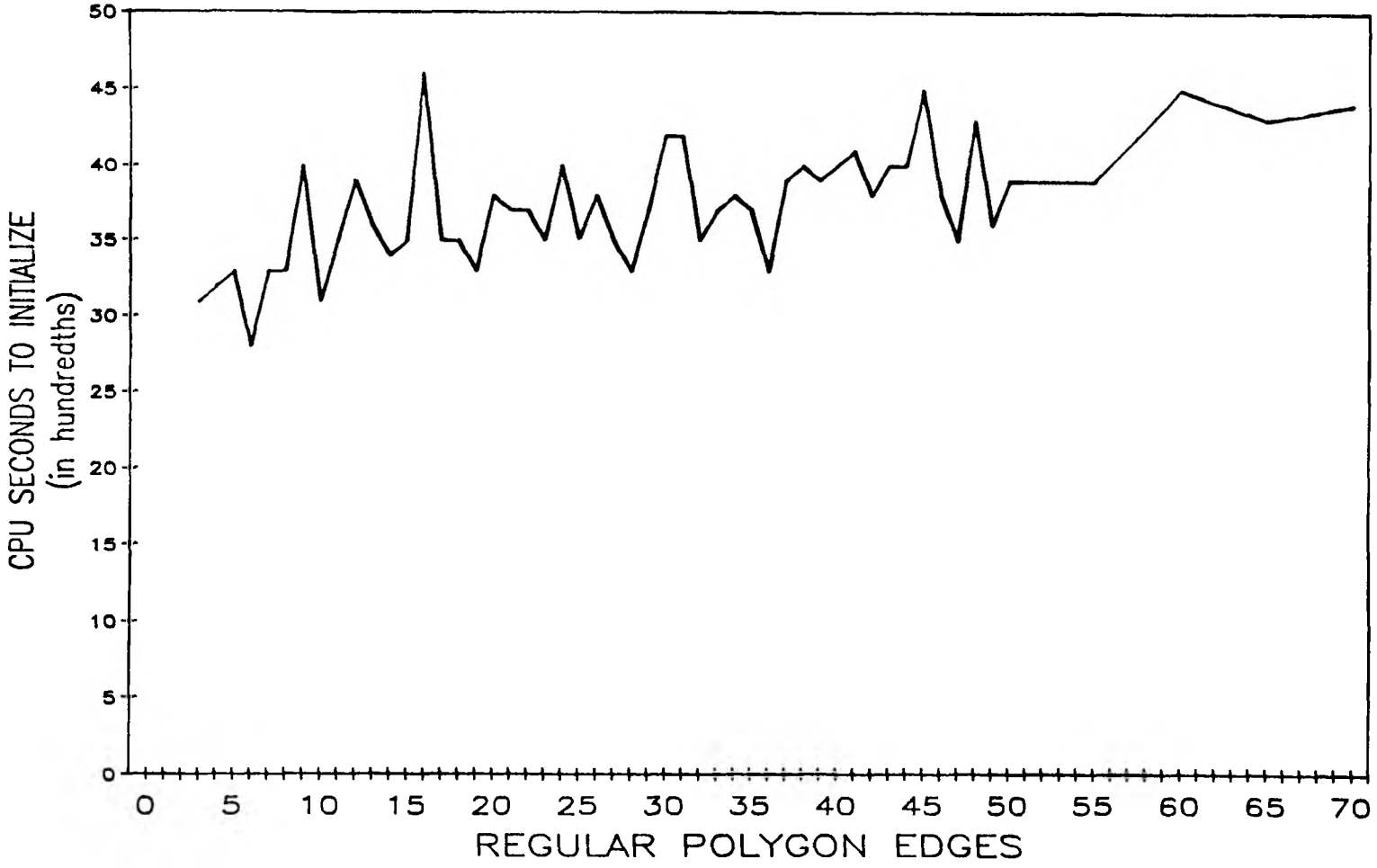

Fig. 19. Plot of CPU time for initialization when the number of edges is varied.

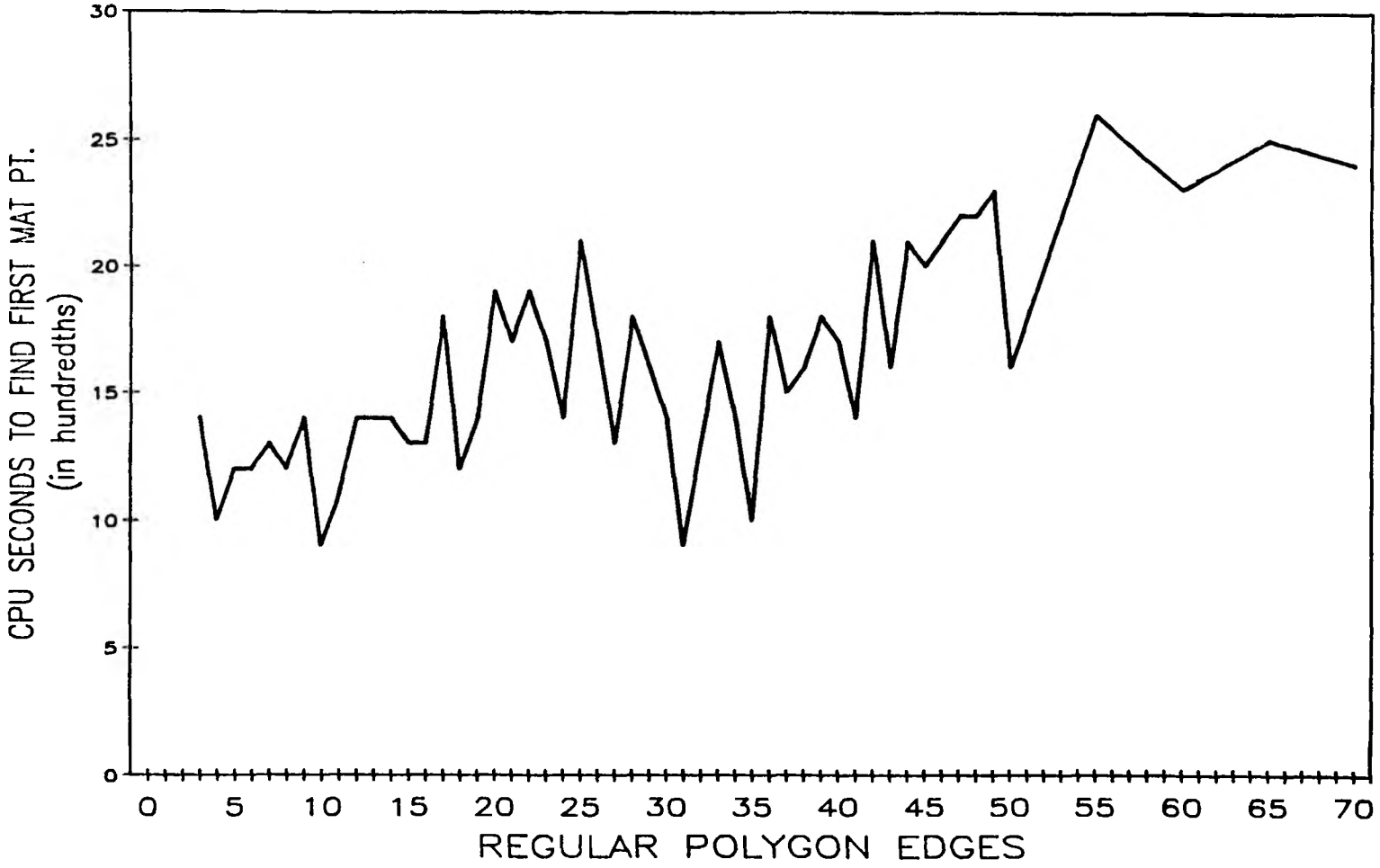

Fig. 20. Plot of CPU time to find the first medial point when the number of edges is varied.

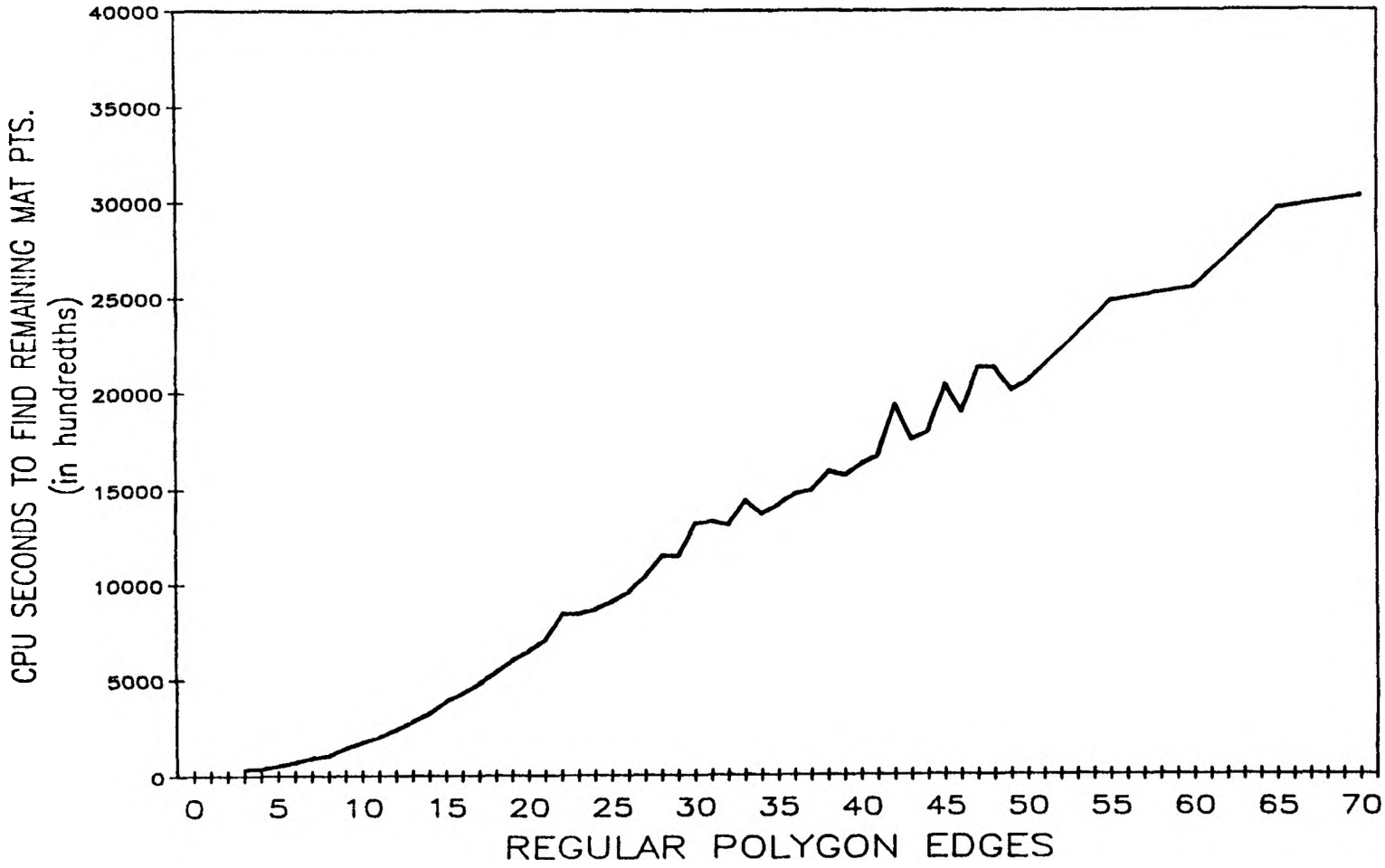

Fig. 21. Plot of CPU time to find the remaining medial points when the number of edges is varied.

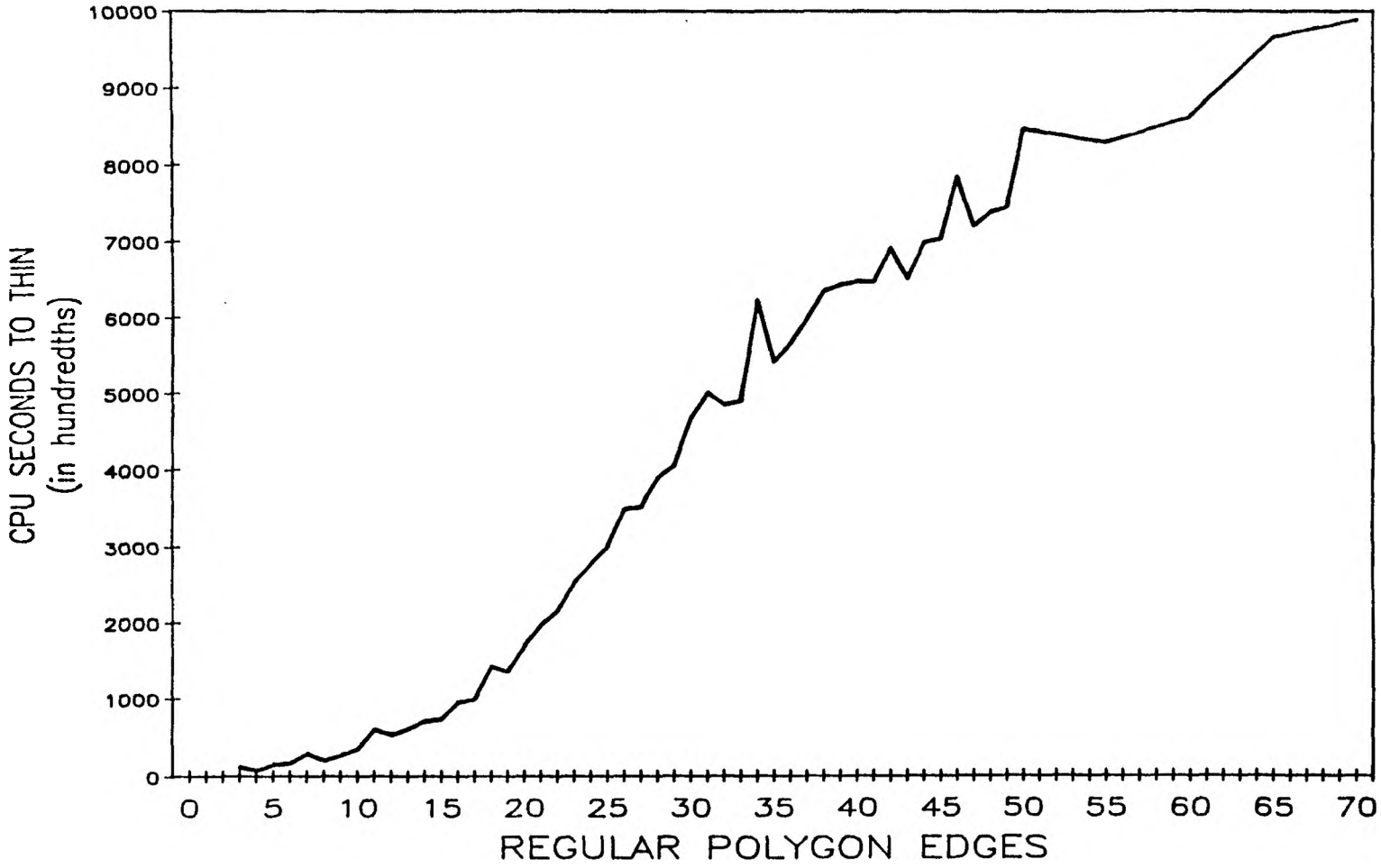

Fig. 22. Plot of CPU time to thin when the number of edges is varied.

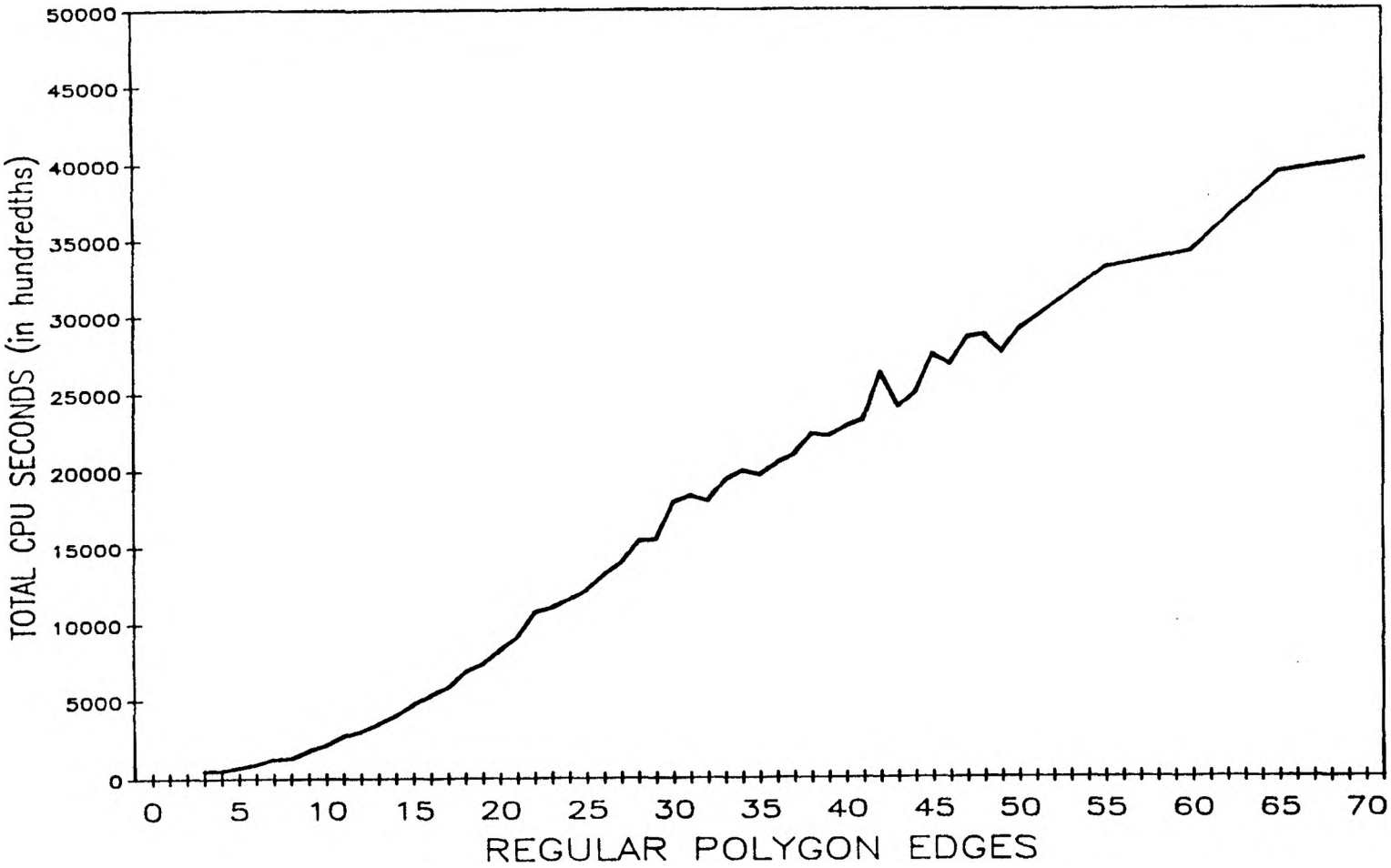

Fig. 23. Plot of total CPU time when the number of edges is varied.

IV.B Varying Area

The shapes used for this set of tests were equilateral triangles. The area was varied from 3 to 70 square units.

CPU time for initialization was expected to be O(SQRT(n)) with respect to area since the digitization process should be linear with respect to the grid size of the spatial occupancy array and all other portions of the initialization phase should be linear. Fig. 24 plot bears this out.

CPU time to find the first medial pixel was expected to be constant for two reasons. The time required to find an interior pixel depends only on the number of edges. Also, the interior pixel that is found for an equilateral triangle is always medial. In fact, it is a medial branch point since it is at the centroid of the equilateral triangle. The plot shown in Fig. 25 is rather erratic but appears to be constant in general. The linear correlation coefficient is approximately 0.30. CPU time required for this phase is small compared to the total CPU time required.

CPU time to find the remaining medial pixels was expected to be O(SQRT(n)) with respect to area. Fig. 26 supports this.

CPU time to thin the MAT was expected to be linear with respect to area. This is due to the fact that, in general, the number of excess medial pixels increases linearly with respect to area. Fig. 27 confirms this. Fig. 28 shows a plot of the total CPU time required.

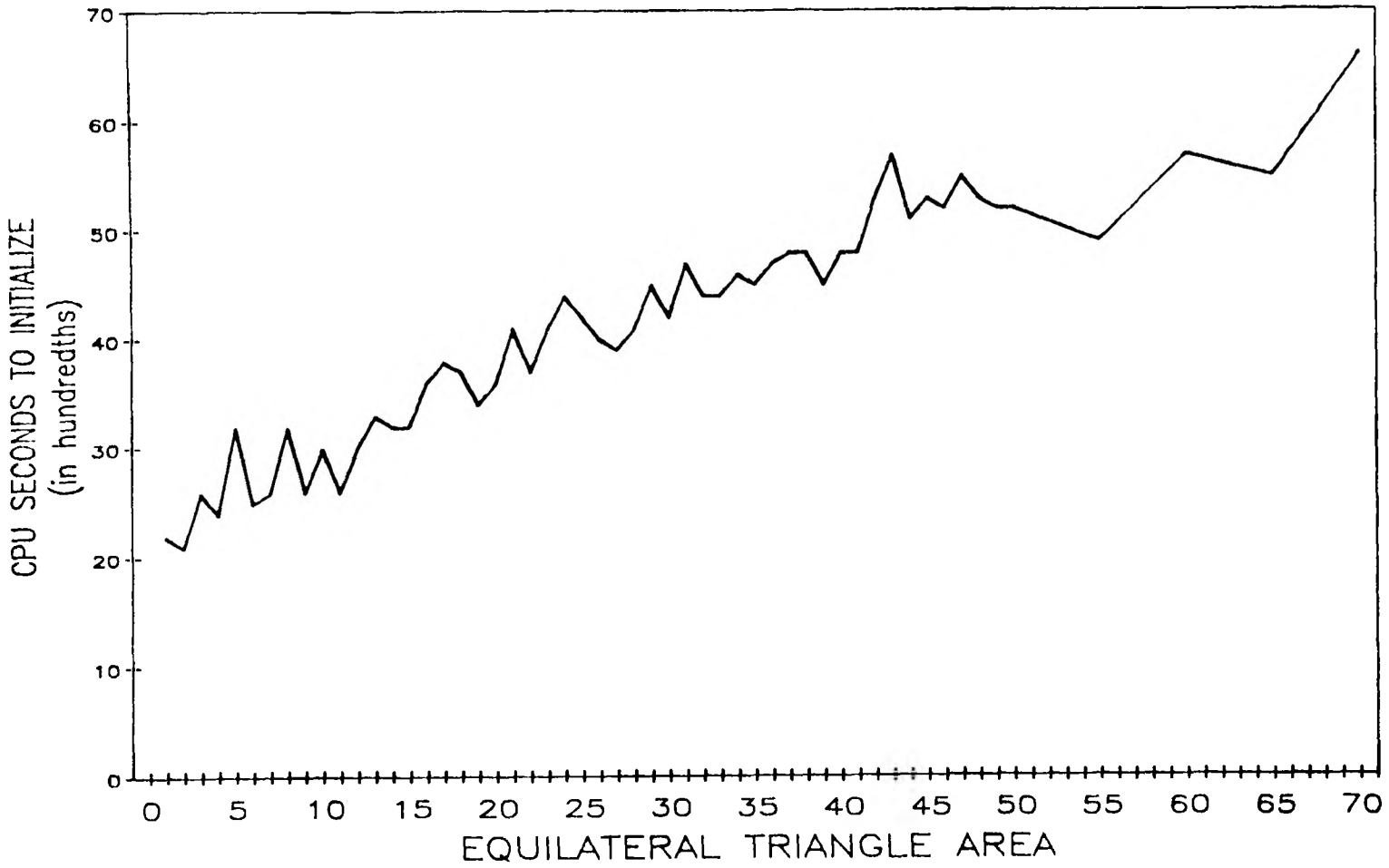

Fig. 24. Plot of CPU time for initialization when the shape area is varied.

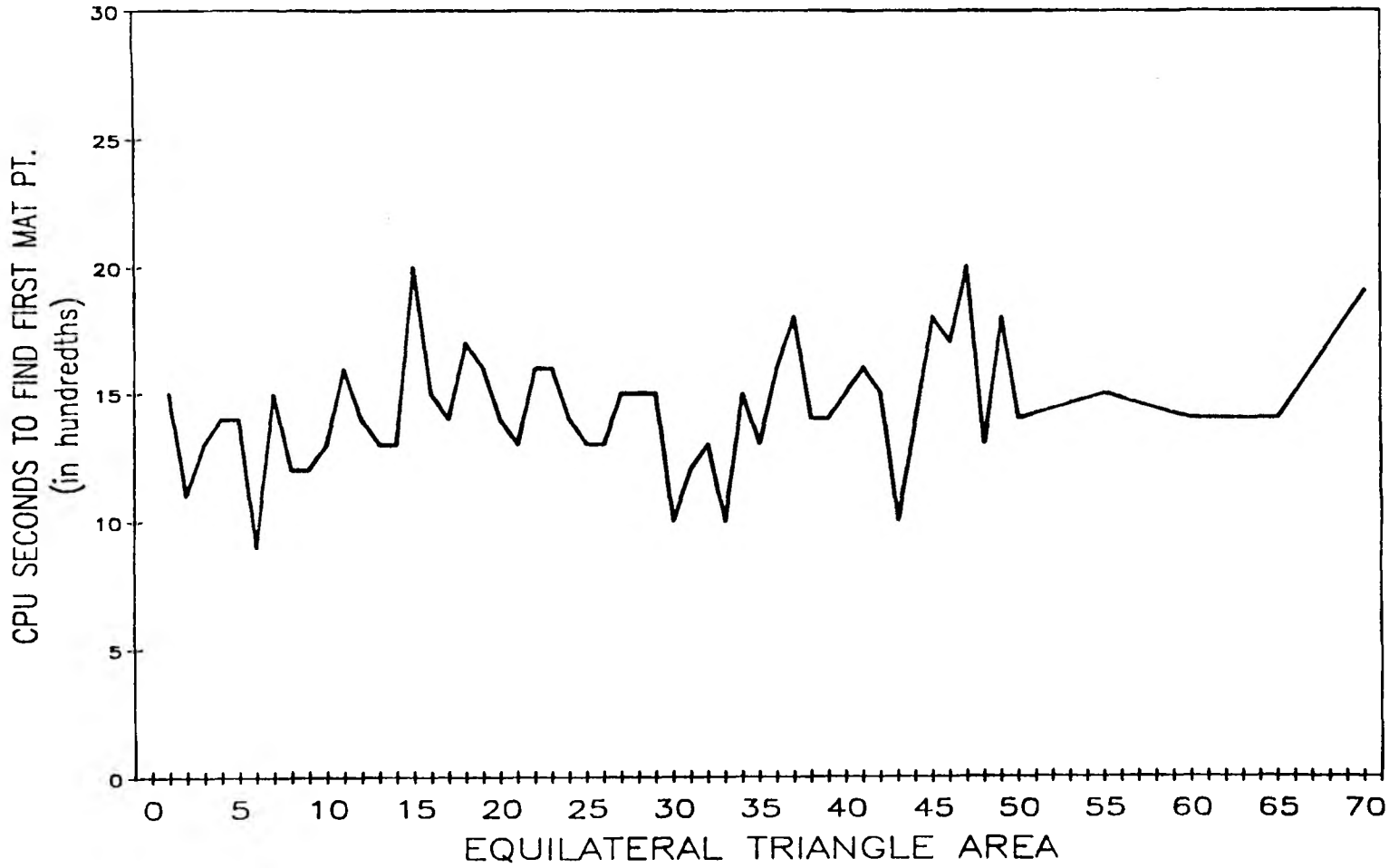

Fig. 25. Plot of CPU time to find the first medial point when the shape area is varied.

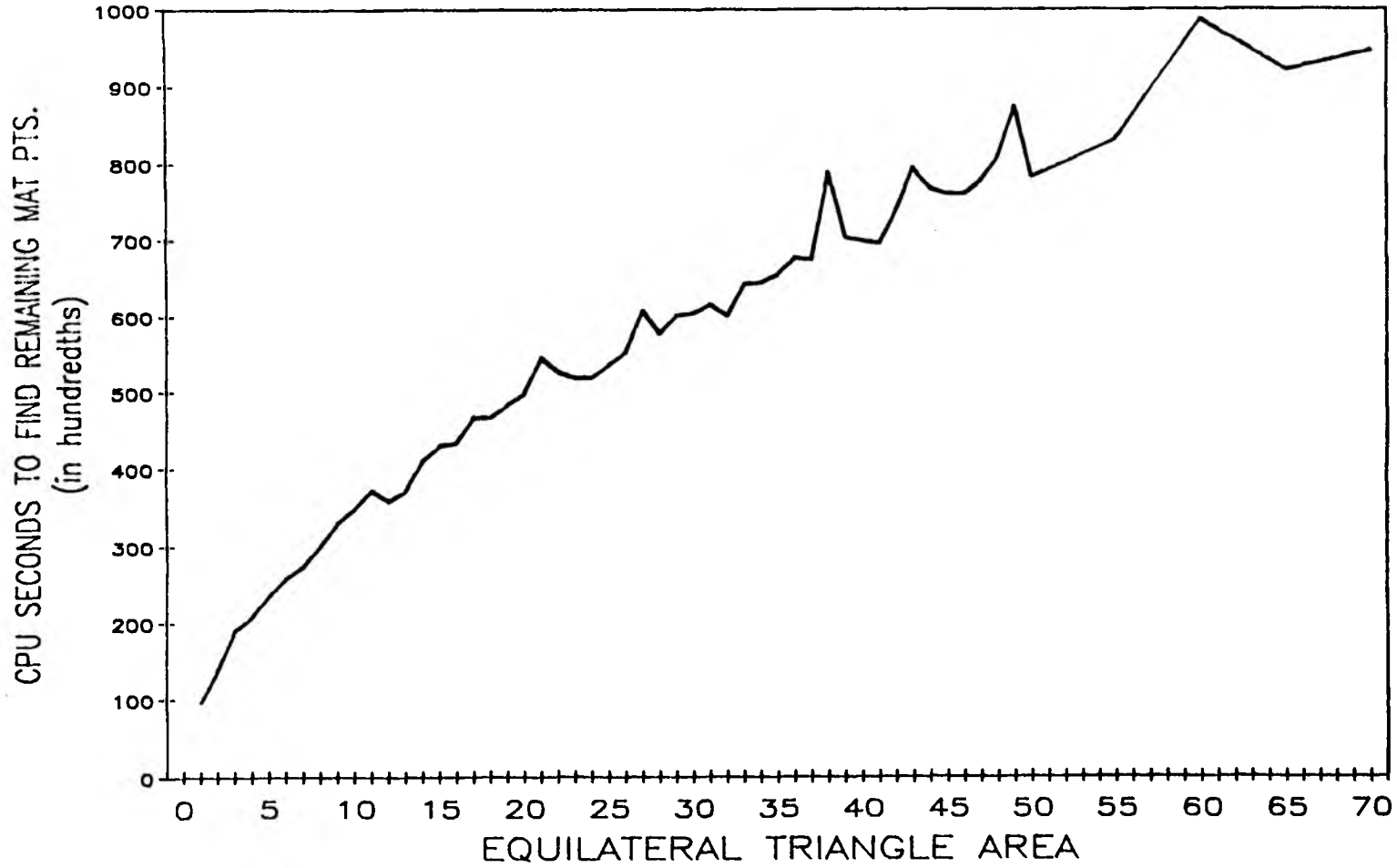

Fig. 26. Plot of CPU time to find the remaining medial points when the shape area is varied.

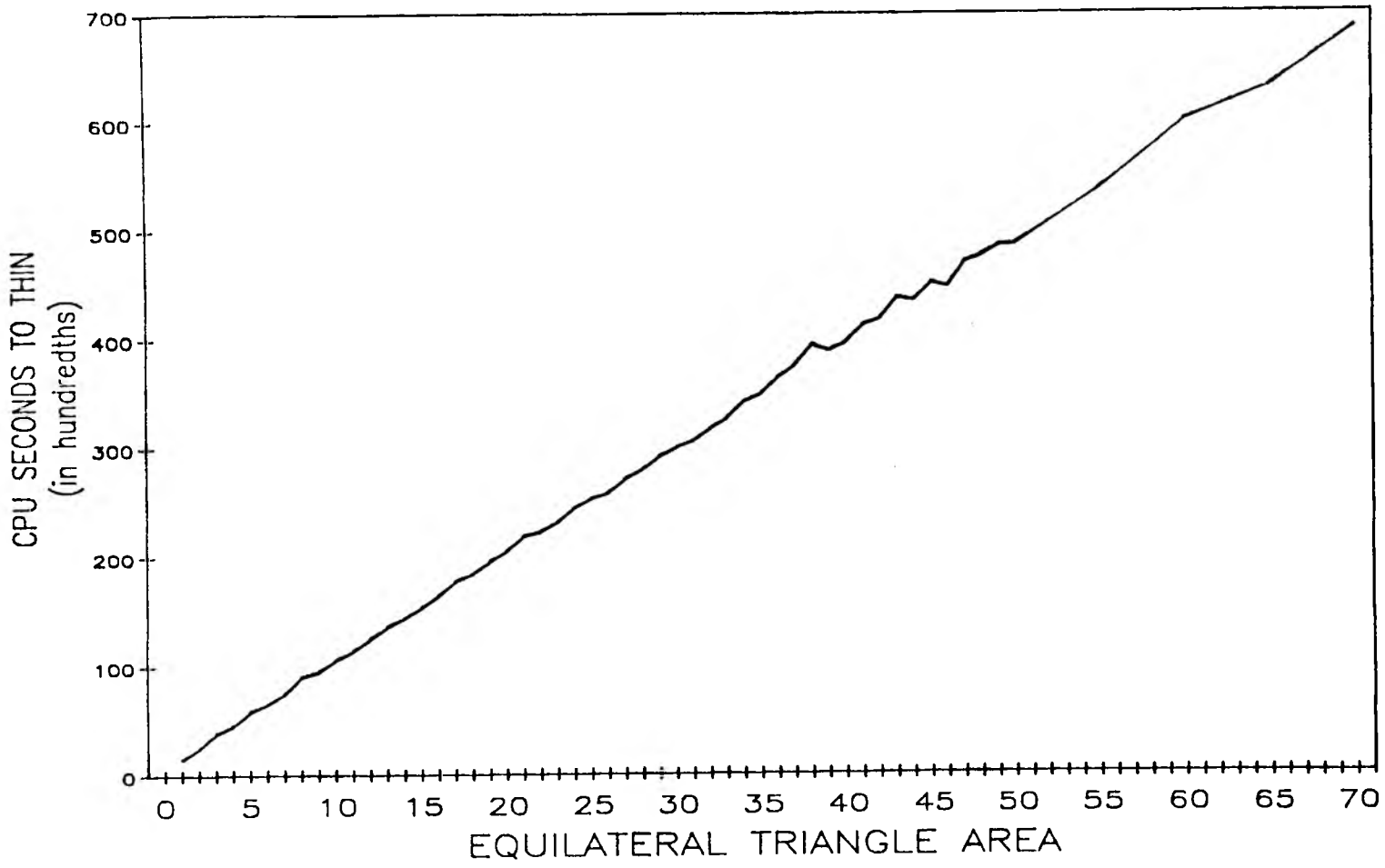

Fig. 27. Plot of CPU time to thin when the shape area is varied.

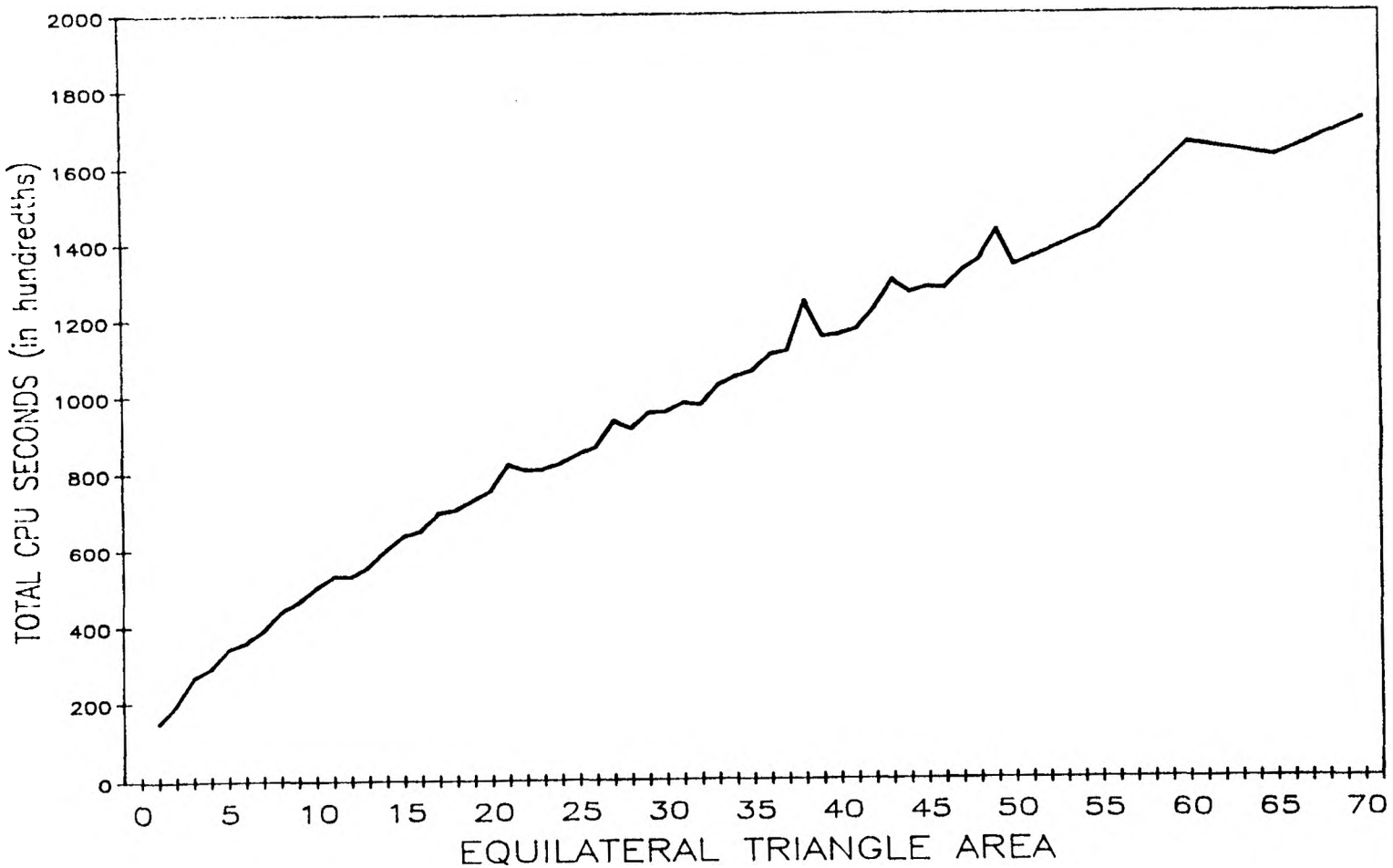

Fig. 28. Plot of total CPU time when the shape area is varied.

IV.C Varying Number Of Holes

Suitable shapes to be used for this set were difficult to find. While it is easy to come up with a shape in which edges can be removed and used to create holes, it is difficult to keep the number of convex and concave vertices constant. Note that the sense of convex and concave is reversed for holes. A triangle shaped hole has three concave vertices, not three convex vertices. Fig. 29 shows two of the shapes from this test set. Each hole is created by removing six "teeth" edges from the shape boundary, replacing the removed edges with a longer edge, and using the remaining five edges to create a triangle that is dented on two sides. Three concave and two convex vertices are removed from the shape boundary and the new hole that is created contains three concave and two convex vertices. The number of holes was varied from 0 to 20.

All of the holes used were the same shape and size. In addition, they were positioned in similar locations with respect to the shape border. This was done so that the effects of adding holes would not be tainted by introducing these three characteristics of the holes.

This set of tests provided the first indication of the role that edge proximity plays in the order of the algorithm. As mentioned in section 3.3, when many edges are close to a pixel that is being tested for mediality the benefits of the lower bound scheme are reduced. Consider

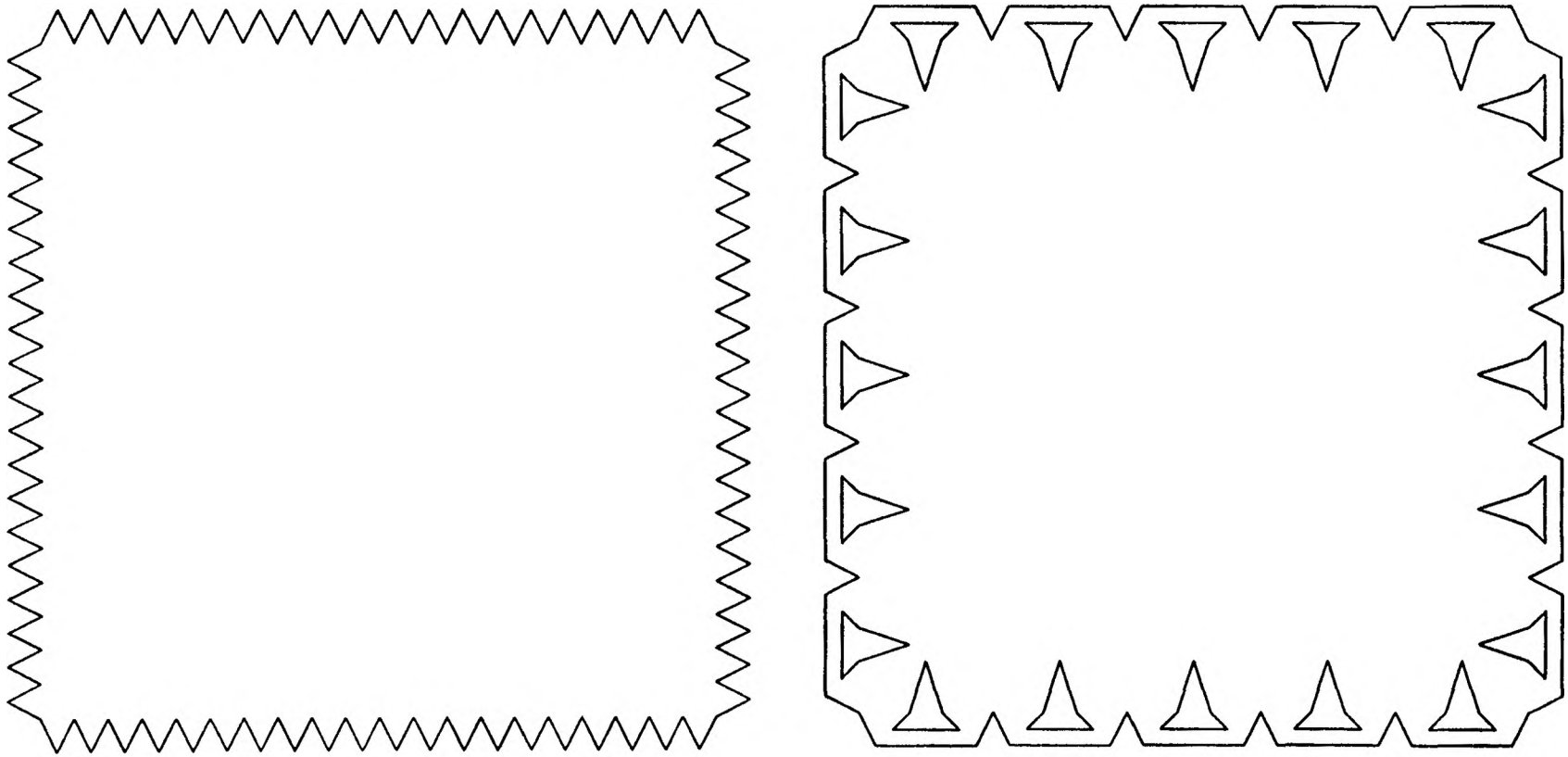

Fig. 29. Two of the shapes used to test the effect of varying the number of holes on the run time of the ridge following algorithm.

the pixels near the center of the test shape containing no holes. Many of the "teeth" edges are nearly equidistant from these pixels so a high number of distances are calculated. In contrast, for the test shape containing 20 holes far fewer edges are nearly equidistant from pixels near the center of the shape. The plot in Fig. 30 shows that as the number of holes increased, average edge proximity decreased causing the total CPU time required to decrease.

IV.D Varying Distribution Of Vertex Types

The shapes used for this test set were obtained by using a regular polygon with 20 edges and "flipping in" various vertices. The number of edges in the regular polygon was selected in such a way that we would obtain a sufficient number of concavities in the test shapes generated from the original regular polygon.

The process of generating test shapes from the original 20-gon is illustrated in the following examples. To obtain shapes in which no two concave vertices are adjacent one simply steps around the 20-gon reflecting every other vertex about a line through its previous and next vertices. This generates 10 different shapes. Returning to the original 20-gon and continuing in this manner one steps around the 20-gon reflecting consecutive pairs of vertices about the previous and next vertices. The next vertex

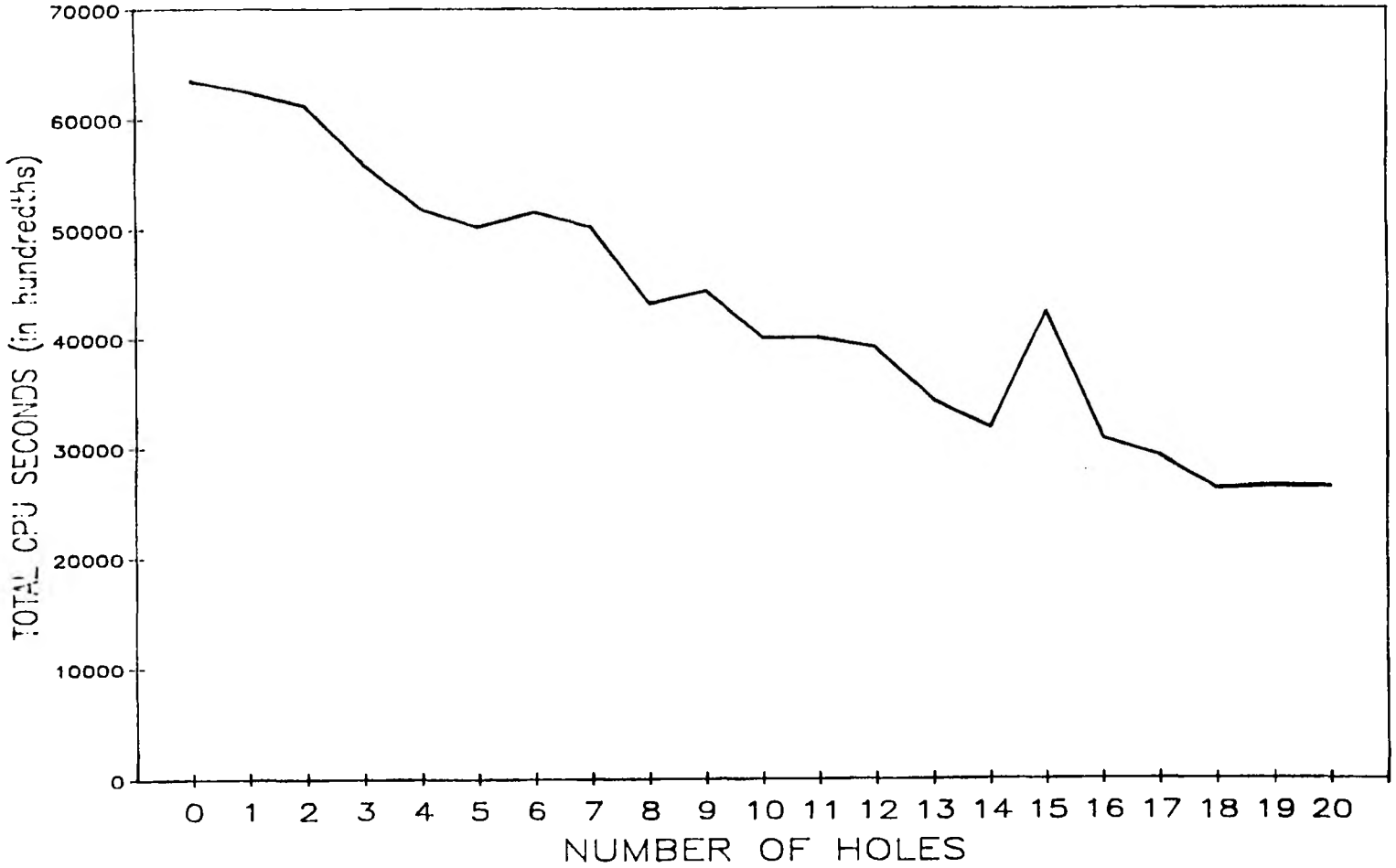

Fig. 30. Plot of total CPU time when the number of holes is varied.

becomes the previous vertex for the next pair of vertices to be flipped. This procedure yields the following set of shapes, three of which are shown in Fig. 31.

KEY A / B where
A = total # of concave vertices in shape
B = # of concave vertices in each set of adjacent concave vertices

0/0

1/1 2/1 3/1 4/1 5/1
6/1 7/1 8/1 9/1 10/1

2/2 4/2 6/2 8/2 10/2 12/2

3/3 6/3 9/3 12/3 15/3

4/4 8/4 12/4 16/4

5/5 10/5* 15/5*

6/6 12/6*

7/7 14/7*

8/8 16/8*

9/9* 18/9*

Some of these shapes (followed by *) are not simply connected. At least one edge crosses another edge in the shape so they cannot be processed by the algorithm. This

is the case for the rightmost shape in Fig. 31.

This test set was the most difficult from which to extract a pattern. The characteristics that vary in this set are the total number of concave vertices and the number of concave vertices within each set of adjacent concave vertices.

It was expected that in general CPU time would decrease as the total number of concave vertices increased since concave vertices do not have medial branches leading into them. The plot in Fig. 32 supports this conjecture. For shapes with the same number of concave vertices, varying the number of concave vertices in each adjacent set of concave vertices did not have a consistent effect on the CPU time required to generate the MAT.

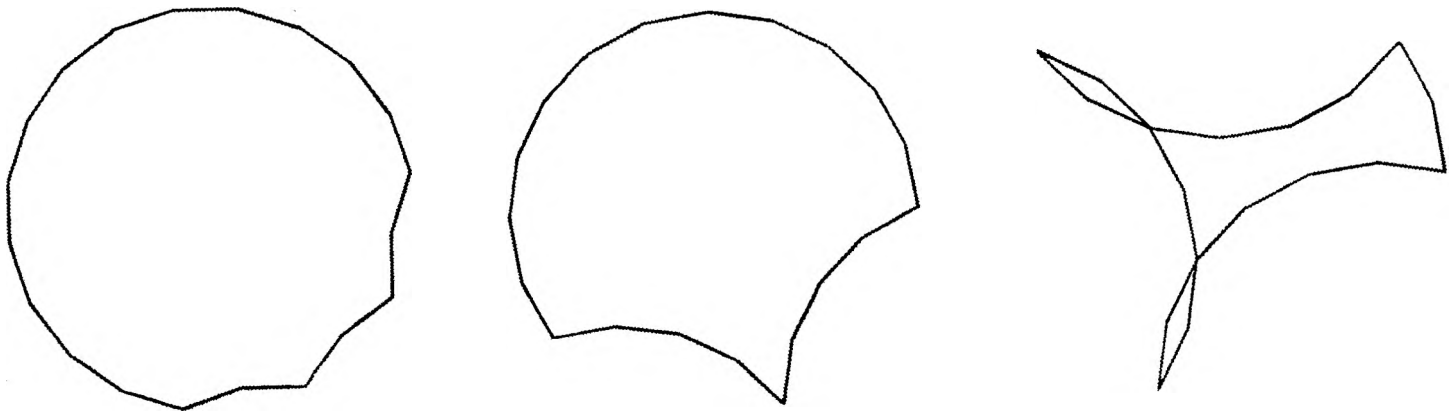

Fig. 31. Three of the shapes used to test the effect of varying the distribution of vertex types on the run time of the ridge following algorithm.

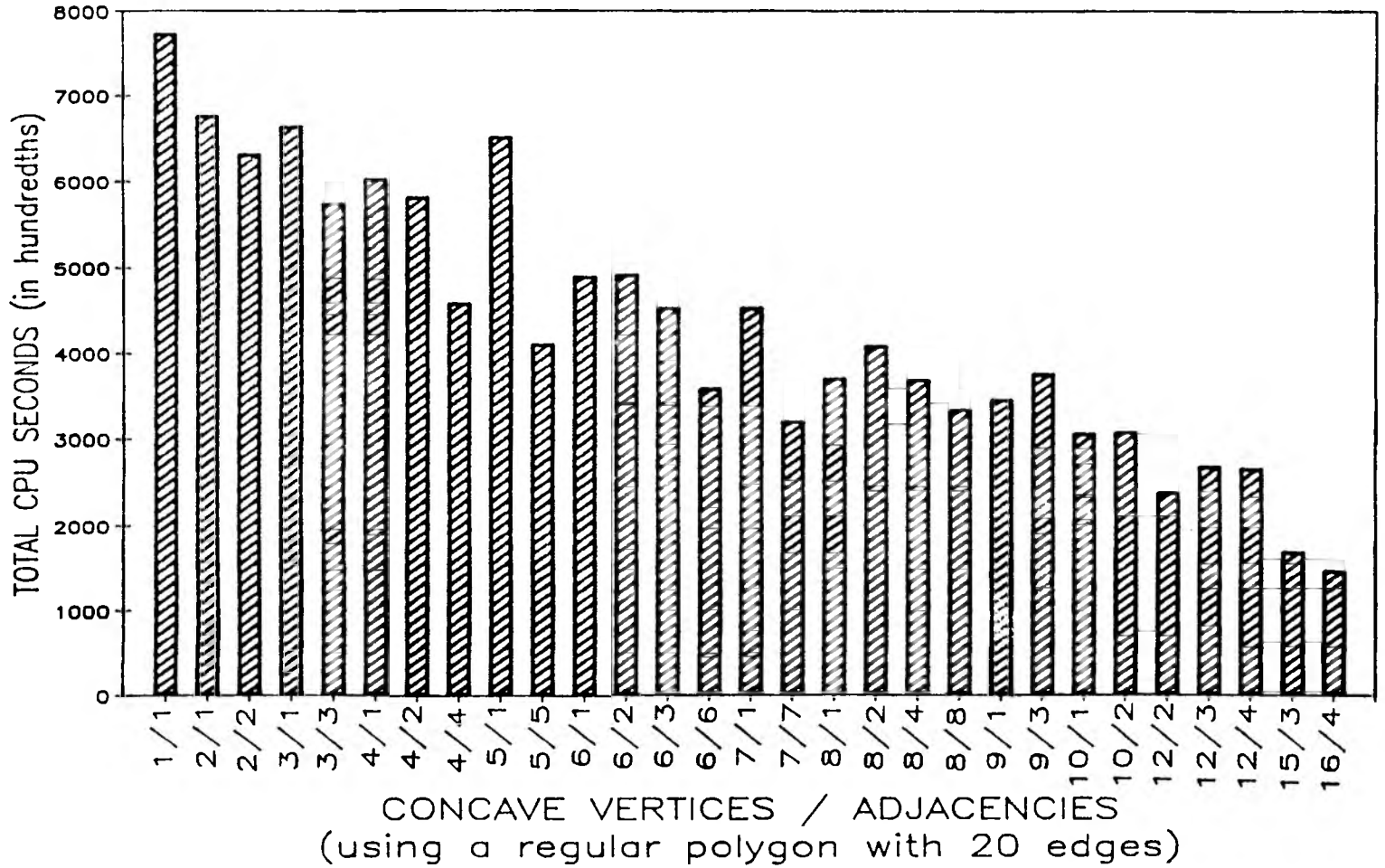

Fig. 32. Plot of total CPU time when the number/distribution of concave vertices is varied.

V CONCLUSION

Applications of the MAT and methods of obtaining it have been presented. The ridge following method has been presented in depth. Improvements to a previous ridge following algorithm have been proposed. Test results that show the effects of certain shape characterisics on the run time of the improved ridge following algorithm have been presented and interpreted.

V.A Effect Of Shape Characteristics On Run Time

The brushfire algorithm runs fastest for shapes that have small area and are nearly "thin" (small radius values for the maximal disks). This greatly restricts the class of shapes that it can process efficiently. The Voronoi algorithm runs fastest for shapes whose polygonal approximations have few edges and few concave vertices.

Many more shape characteristics affect the run time of the ridge following algorithm. The same shape characteristics that are good for the Voronoi algorithm are also good for ridge following. Other shape characteristics that decrease the run time of the ridge following algorithm are small area (see section IV.B), small angles between edges in the polygonal approximation (see section IV.A), and low edge proximity (see section IV.C).

Table I shows the shape characteristics that affect each section in the ridge following algorithm. It also shows the order of each section in terms of number of edges and shape area. Plots of combined CPU times for steps 1 thru 6 are in Fig. 19 and Fig. 24. Plots of combined CPU times for steps 7 and 8 are in Fig. 20 and Fig. 25. Plots of CPU times for step 9 are in Fig. 21 and Fig. 26. Plots of CPU times for step 10 are in Fig. 22 and Fig. 27. Plots of total CPU times for all steps are in Fig. 23, Fig. 28, Fig. 30, and Fig. 32.

V.B Extensions To The Ridge Following Algorithm

In the interest of making the ridge following algorithm more generally applicable it would be beneficial to extend its capabilities in three ways. First, the requirement that the starting represention be a polygon should be removed. Second, the ability to find the MAT of a shape's exterior should be added. This is especially important for feature extraction. Third, the algorithm should be extended to handle 3-D shapes.

V.B.1 Medial Axis Of Shapes Containing Curves

The ridge following algorithm can be modified to find the medial axis of a non-polygonal shape. A polygonal approximation of the shape is still required, but the algorithm can be modified to prevent medial branches from

Table I
Detailed Information on Sections
of the Ridge Following Algorithm

		Shape Characteristic					O(?) where n is	
	Section Description	1	2	3	4	5	Edges	Area
1	find polygon extrema		x				n	1
2	allocate and initialize maps	x					1	n
3	draw polygons in map	x	x				n	sqrt n
4	get vertex types		x				n	1
5	combine colinear edges		x				n	1
6	compute line equations		x				n	1
7	find internal point		x				n	1
8	find 1st medial point		x	x		x	n log n	sqrt n
9	find rest of medial points	x	x	x		x	n log n	n
10	thin medial points	x			x	x	1	n
s	determine if 1 pt. is medial		x	x		x	n log n	1

Section s is a subroutine that is called by steps 8 and 9.

Key for shape characteristic columns:
1 - area
2 - number of edges
3 - proximity of edges
4 - angle measures
5 - number of concavities

extending into convex vertices created by the polygonal approximation of curves. During polygonal approximation of curves the average radius of curvature of the original shape boundary in the area of each approximated edge must be computed and saved. The check for pixel mediality must be modified to disqualify the pixel if the distance to one of its two closest edges is less than the radius of curvature for the edge.

A robust algorithm for generation of the MAT should have the ability to process curved shapes. Along with investigation into extending the ridge following algorithm to handle curved shapes, the possibility of extending the Voronoi algorithm should be explored.

V.B.2 Medial Axis Of The Exterior Of Shapes

The MAT of the exterior of a shape exists provided that the shape contains at least one concavity. To find the MAT of the exterior of a shape using ridge following, a seed pixel is required for each bounded set of exterior pixels. Since the MAT extends outward from the shape indefinitely, the spatial occupancy array must be dimensioned to include all medial pixels of interest. This dimensioning provides a stopping point for the recursive process of finding neighboring medial pixels.

V.B.3 Three Dimensional Ridge Following

The ridge following method can be extended to operate on 3-D shapes. Pixels must be replaced by volume elements (or voxels). The basic steps of the revised algorithm are shown below:

Digitize the shape surface, not including the shape interior, in a 3-D spatial occupancy array.
Find any voxel that is inside the shape (seed voxel).
Search a plane of voxels containing that voxel to find the first medial voxel (existence is guaranteed since the MAT is connected).
Find the rest of the medial voxels by recursively examining all the neighboring voxels.

The concepts that must be altered for the 3-D case are listed below:

2-D	3-D
8 neighbors per pixel	26 neighbors per voxel
distances are measured from pixels to edges	distances are measured from voxels to polygonal faces
check is made for a concave vertex shared by adjacent edges	check is made for a concave vertex shared by adjacent faces
lower bound on distance is based on edge endpoints	lower bound on distance is based on face vertices

The same tolerance can be used for evaluating the difference between the closest and second closest face. Also, the same steps for thinning the MAT can be used. The procedure for identifying simple voxels might be similar but will be somewhat more complicated than identifying simple pixels since the ordering of voxels in a neighborhood is not clear.

REFERENCES

[1] N. Ahuja and W. Hoff, "Augmented Medial Axis Transform", Proc. Workshop on Computer Vision: Representation and Control", pp. 251-256, 1984.

[2] N. Ahuja, B. An, and B. Schachter, "Image Representation Using Voronoi Tessellation", Computer Vision, Graphics, and Image Proc., vol. 29, pp. 286-295, 1985.

[3] C. Arcelli and L. Cordella and S. Levialdi, "A Grassfire Transformation for Binary Digital Images", Proc. Second Int. Joint Conf. on Pattern Rec., pp. 152-153, 1974.

[4] C. Arcelli and G. Sannita di Baja, "On the Sequential Approach to Medial Line Transformation", IEEE Trans. Sys. Man. Cybernet., vol. 8, pp. 133-144, 1978.

[5] C. Arcelli and G. Sanniti, "A Thinning Algorithm Based on Prominence Detection", Pattern Recogn., vol. 13, pp. 225-235, 1981.

[6] C. Arcelli and G. Sanniti, "An Approach to Figure Decomposition Using Width Information", Computer Vision, Graphics, and Image Proc., vol. 26, pp. 61-72, 1984.

[7] C. Arcelli and G. Sanniti, "A Width-Independent Fast Thinning Algorithm", IEEE Trans. Pattern Anal. Machine Intell., vol. 7, no. 4, pp. 463-474, 1985.

[8] H. Blum, "A Transformation for Extracting New Descriptions of Shape", Models for the Perception of Speech and Visual Form, W. Dunn, Ed. Cambridge, MA:M.I.T. Press, pp. 153-171, 1967.

[9] H. Blum, "Biological Shape and Visual Science (Part I)", Journal of Theoretical Biology, vol. 38, pp. 205-287, 1973.

[10] H. Blum, "A Geometry for Biology", Conf. on Mathematical Analysis of Fundamental Biological Phenomena, O. Gurel, Ed., Ann. N.Y.A.S., vol. 231, pp. 19-30, 1974.

[11] H. Blum and R. Nagel, "Shape Description Using Weighted Symmetric Axis Features", Pattern Recognition, vol. 10, pp. 167-180, 1978.

[12] L. Calabi, "A Study of the Skeleton of Plane Figures", Parke Mathematical Laboratories, Scientific Report no. 2, 1965.

[13] P. DeSouza and P. Houghton, "Computer Location of Medial Axes", Computers and Biomedical Research, vol. 10, no. 4, pp. 333-343, 1977.

[14] C. Dyer and A. Rosenfeld, "Thinning Algorithms for Gray-Scale Pictures", IEEE Trans. Pattern Anal. Machine Intell., vol. 1, no. 1, pp. 88-89, 1979.

[15] J. Fairfield, "Segmenting Dot Patterns by Voronoi Diagram Concavity", IEEE Trans. Pattern Anal. Machine Intell., vol. 5, no. 1, pp. 104-110, 1983.

[16] A. Favre and H. Keller, "Parallel Syntactic Thinning by Recoding of Binary Pictures", Computer Vision, Graphics, and Image Proc., vol. 23, pp. 99-112, 1983.

[17] C. Hilditch, "Comparison of Thinning Algorithms on a Parallel Processor", Image Vis. Comput., vol. 1, no. 3, pp. 115-132, 1983.

[18] S. Ho and C. Dyer, "Shape Smoothing Using Medial Axis Properties", IEEE Trans. Pattern Anal. Machine Intell., vol. 8, no. 4, pp. 512-519, 1986.

[19] S. Israni and J. Sanders, "Two-dimensional Cutting Stock Problem Research: A Review and a New Rectangular Layout Algorithm" Journal of Manufacturing Systems, vol. 1, no. 2, pp. 169-181, 1982.

[20] D. Kirkpatrick, "Efficient Computation of Continuous Skeletons", Proc. 20th Annu. Symp. Found. Computer Sci., pp. 18-27, Oct. 1979.

[21] D. Lee and R. Drysdale, "Generalization of Voronoi Diagrams in the Plane", SIAM J. Comput., vol. 10, pp. 73-87, Feb. 1981.

[22] D. Lee, "Medial Axis Transformation of a Planar Shape", IEEE Trans. Pattern Anal. Machine Intell., vol. 4, no. 4, pp. 363-369, 1982.

[23] R. Mohr and R. Bajcsy, "Packing Volumes by Spheres", IEEE Trans. Pattern Anal. Machine Intell., vol. 5, no. 1, pp. 111-116, 1983.

[24] J. Mott-Smith, "Medial Axis Transformations", Picture Processing and Psychopictorics, Lipkin and Rosenfeld, Eds. New York:Academic Press, pp. 267-283, 1970.

[25] R. Nagel, "A Symmetric Axis Basis for Object Recognition and Description", IEEE Proc. of the Conference on Decision and Control Including the 15th Symposium on Adaptive Processes, pp. 168-170, 1976.

[26] J. O'Rourke and N. Badler, "Decomposition of Three-Dimensional Objects into Spheres", IEEE Trans. Pattern Anal. Machine Intell., vol. 1, no. 3, pp. 295-305, 1979.

[27] J. O'Rourke and N. Badler, Correction to "Decomposition of Three-Dimensional Objects into Spheres", IEEE Trans. Pattern Anal. Machine Intell., vol. 1, no. 4, p. 417, 1979.

[28] T. Pavlidis, "Algorithms for Shape Analysis of Contours and Waveforms", IEEE Trans. Pattern Anal. Machine Intell., vol. 2, no. 4, pp. 301-312, 1980.

[29] T. Pavlidis, "Algorithms for Graphics and Image Processing", Computer Science Press, 1982.

[30] F. Preparata, "The Medial Axis of a Simple Polygon", Proc. Sixth Symp. on Math. Found. of Computer Science, 1977.

[31] A. Rosenfeld, "A Characterization of Parallel Thinning Algorithms", Information and Control, vol. 29, pp. 286-291, 1975.

[32] A. Rosenfeld and A. Kak, "Digital Picture Processing", 2nd Edition, Academic Press, 1982.

[33] H. Samet, "Quadtrees and Medial Axis Transforms", IEEE Proc. Int. Conf. Pattern Recognition, pp. 184-187, 1982.

[34] R. Sedgewick, "Algorithms", Addison-Wesley Publishing Co., 1983.

[35] H. Tamura, "A Comparison of Line Thinning Algorithms from Digital Geometry View Point", IEEE Proc. Int. Conf. Pattern Recognition, pp. 715-719, 1978.

[36] Y. Tsao and K. Fu, "Stochastic Skeleton Modeling of Objects", Computer Vision, Graphics, and Image Proc., vol. 25, pp. 348-370, 1984.

[37] A. Wu and S. Bhaskar and A. Rosenfeld, "Computation of Geometric Properties from the Medial Axis Transform in O(n log n) Time", Computer Vision, Graphics, and Image Proc., vol. 34, no. 1, pp. 76-92, 1986.

[38] B. Zvolanek and C. Lee, "Image Skeletonization for Object Position Measurement", Proc. of SPIE Applications of Digital Image Processing IV, vol. 359, pp. 92-97, 1982.

VITA

Richard Mark Volkmann was born on April 16, 1961 in St. Louis, Missouri. He received his primary and secondary education in Maryland Heights, Missouri. He has received his college education from the University of Missouri-St. Louis and the University of Missouri-Rolla Graduate Engineering Center in St. Louis, Missouri. He received a Bachelor of Science degree in Computer Science from the University of Missouri-St. Louis in May 1983. He has been enrolled in the Graduate School of the University of Missouri-Rolla since September 1983.

Since April of 1986 he has been working at McDonnell Douglas. His efforts there have been focused on automatic nesting of sheet metal and composite aircraft parts. He has also worked on automatic placement of rivets used to hold down scrap areas created during the cutting of sheet metal parts.

APPENDICES

A IMPLEMENTATION

The improved ridge following algorithm was implemented on a DEC VAX 8300 computer using Pascal. The details of the algorithm are presented in the following sections.

A.1 Definitions

The following terms are used to describe the steps of the algorithm:

A.1.a Boundary_Map

The Boundary_Map is a 2-D array of Boolean values, the TRUE elements of which represent pixels that are on a boundary of the shape.

A.1.b Checked_Map

The Checked_Map is a 2-D array of Boolean values, the TRUE elements of which represent pixels that have been checked for mediality.

A.1.c Contours

Contours refer to the polygons that are used to represent the shape, i.e. the boundary and the holes (if any).

A.1.d Medial_Map

The Medial_Map is a 2-D array of Medial_Nodes corresponding to pixels.

A.1.e Medial_Node

A Medial_Node is an element in the Medial_Map array with the following fields:

Medial
- = a Boolean value indicating whether the corresponding pixel is medial

Quality
- = the difference between the distance from the pixel to the closest contour polygon edge and the second closest contour polygon edge that does not form a concave angle with the closest edge (The lower the Quality value, the closer the pixel center is to being on the actual medial axis.)

Radius_Squared

- the distance squared to the closest contour polygon edge

A.1.f Vertex List

The vertex list contains the (X,Y) points in the polygonal approximation of each contour of the shape. The points are stored in clockwise order for the boundary and in counter-clockwise order for holes.

A.1.g Vertex Type

Vertex type refers to the angle between the two contour polygon edges that are incident at a vertex. It can be Concave, Colinear (angle = Pi), or Convex. For vertices in the contour polygons of holes, the meaning of Concave and Convex is reversed.

A.1.h Segment Linked List (SLL)

The segment linked list is a linked list of Segment_Nodes.

A.1.i Segment_Node

A Segment_Node is a node in the segment linked list (SLL) that represents a contour polygon edge. It contains the following fields:

Owner_Contour =

contour number of which this edge is a member

(1 for the boundary, 1 + hole number for holes)

Begin_Point =

index of the first endpoint of the edge

in the vertex list of the Owner_Contour

End_Point =

index of the second endpoint of the edge

in the vertex list of the Owner_Contour

Begin_VT =

the vertex type of the Begin_Point

Line_Coefficients =

A, B, & C where Ax + By + C = 0 for the

line through Begin_Point and End_Point

Lower_Bound =

lower bound on the distance from the current

pixel being tested for mediality to this edge

Distance_Squared =

distance squared from the current pixel

being tested for mediality to this edge

Next =

pointer to the next Segment_Node

in the segment linked list (SLL)

A.2 Input Required

The input required by the algorithm is:

1) number of contours in the shape
   (1 for the boundary + number of holes in shape)

2) 1-D array containing the number of points
   in the vertex list of each contour

3) 1-D array containing a vertex list for each contour

4) resolution (the grid size of the pixels that the
   medial pixels will be chosen from will be
   1 / resolution)

A.3 The Algorithm

The basic steps in the algorithm are:

1) Allocate the Boundary_Map, Checked_Map, and
   Medial_Map dynamic arrays based on the minimum
   and maximum X and Y vertex values in the boundary
   contour and the Grid_Size.

2) "Draw" the boundary of the shape and
   its holes in the Boundary_Map.

3) Create the SLL.

4) Combine colinear segments in the SLL by deleting those that have a Colinear Begin_VT and setting the End_Point of their previous Segment_Node to the End_Point of the deleted Segment_Node.

5) Find any pixel that is inside the shape boundary but not inside one of its holes.

6) Set Tolerance to the Grid_Size plus a very small number. (If the quality of a pixel is close to the Grid_Size it may appear to be slightly greater than the Grid_Size due to the precision of the computer. If a small number were not added to the tolerance, the pixel would not be marked as medial and, due to the recursive nature of the algorithm, much of the MAT could be missed.)

7) Find the first medial pixel by examining all internal pixels above and below the known internal pixel until one is found (existence is guaranteed since the MAT is connected). (see the section A.6)

8) Find the rest of the medial pixels by recursively examining all the neighboring pixels (8 for each pixel) of the known medial pixel that are:
   1) not turned on in the Checked_Map
   2) not turned on in the Boundary_Map
   3) not obtained by crossing a shape contour in the boundary map in a diagonal direction

(see the section A.6)

9) "Thin" excess pixels that were marked as medial.

A.3.a Detail Of Step 2

The Bresenham line drawing algorithm was used to turn on pixels in the Boundary_Map along the contour polygon edges.

A.3.b Detail Of Step 3

```
For each contour
  For each edge in the current contour
    Create a new Segment_Node and set the following
    fields:
      Owner_Contour,
      Begin_Point,
      End_Point,
      Begin_VT,
      Line_Coefficients.
      { see the sections A.4 and A.5 }
    Set the Next field of the previous Segment_Node
    to point to the newly created Segment_Node.
```

A.3.c Detail Of Step 5

The concave vertices of the contour polygons will have been marked already in the SLL.

Let A be the highest concave vertex in the SLL.

Let B be the highest vertex in the boundary contour polygon.

Let C be the next vertex in the counter-clockwise direction from B in the boundary contour polygon.

Let D be the next vertex in the clockwise direction from B in the boundary contour polygon.

If C is below A in the sense of its y coordinate then let C be the point on the line segment BC with the same y value as A.

If D is below A in the sense of its y coordinate then let D be the point on the line segment BD with the same y value as A.

Find the centroid of B, C, and D.

This point is in the interior of the shape as is the pixel that contains it.

A.3.d Detail Of Step 9

```
Repeat
  For each Medial_Node
    If the Medial field is set to TRUE Then
      If the Quality values of all of its neighboring
         medial pixels are lower than its Quality Then
        If the pixel can be removed without damaging the
           connectivity of the medial pixels
           (simple pixel) Then
          Set the Medial field to FALSE
            to remove the medial pixel.
Until no more medial pixels can
      be removed by this process
```

A.4 Computing Vertex Types

In order to compute the vertex type of a vertex in a contour polygon one considers the two edges incident at the vertex. Label the vertex of interest B and label vertices A and C such that A immediately precedes B and C immediately follows B when the vertices are traversed in clockwise order. Let V1 be the vector from B to A and let V2 be the vector from B to C.

When the dot product of two vectors is positive then the smallest angle between the vectors is less than Pi / 2. This could be either the angle from the first vector to the

second or the second to the first.

When the cross product of two vectors is positive then the counter-clockwise angle from the first vector to the second vector is less than PI. When the cross product of two vectors is negative then the counter-clockwise angle from the first vector to the second vector is greater than PI (right-hand rule).

The process of determining the vertex type of B is shown below:

```
Normalize V1 and V2 (divide by vector length) so that
their cross product gives the sine of the angle between
them.

Let DP be the dot product of V1 and V2.

Let CP be the cross product of V1 and V2.

If DP > 0 Then
  If CP > 0 Then
    Begin_VT is Convex
  Else
    If CP < 0 Then
      vertex type of B is Concave
    Else
      { CP = 0, the line segments
        lie on top of each other }
      vertex type of B is Colinear
```

```
Else
  If CP > SIN (5 degrees) Then
    vertex type of B is Convex
  Else
    If CP < - SIN (5 degrees) Then
      vertex type of B is Concave
    Else { - SIN (5 degrees) <= CP <= SIN (5 degrees) }
      vertex type of B is Colinear
```

Checking the CP against SIN (5 degrees) allows points that are nearly colinear to be labeled as colinear. This may not be desirable for line segments that are used to approximate curves.

A.5 Computing Line Coefficients

The coefficients of the equation Ax + By + C = 0 for a line specified by two points (x1, y1) and (x2, y2) are computed as follows:

```
A = y1 - y2
B = x2 - x1
C = (x1 * y2) - (y1 * x2)
```

A.6 Determining Pixel Mediality

The following steps are performed when a particular pixel is being tested for mediality:

1) Set the Checked_Map element corresponding to the pixel to TRUE.

2) For each edge in the SLL, set Lower_Bound to a lower bound on the distance from the pixel center to the edge.

3) Sort the nodes in the SLL on ascending lower bound. The order of the SLL remains fairly constant while evaluating pixels that are neighbors of the previously evaluated pixel. This reduces the amount of sorting required for most pixels.

4) Find the closest edge to the pixel in terms of distance squared.

5) Find the next closest edge to the pixel in terms of distance squared that does not form a concave angle with the closest edge.

6) Set the Quality of the pixel to the difference between the second closest and the closest distances to edges.

7) If Quality < Tolerance Then
   Set Medial to TRUE for the pixel and save the distance squared to the closest edge (from step 4) in Radius_Squared.

A.6.a Detail Of Step 2

The lower bound on the distance from a point to an edge (or line segment) that was used can be computed quickly since it involves only subtraction, MIN, and MAX operations. The following steps are performed to obtain a lower bound on the distance from a point (x0, y0) to a line segment from (x1, y1) to (x2, y2):

```
Delta_X_1 = x1 - x0
Delta_Y_1 = y1 - y0
Delta_X_2 = x2 - x0
Delta_Y_2 = y2 - y0

If Sign (Delta_X_1) = Sign (Delta_X_2) Then
  { endpoints are on the same horizontal side of x0 }
  Lower_Bound = MIN (ABS (Delta_X_1), ABS (Delta_X_2))
  If Sign (Delta_Y_1) = Sign (Delta_Y_2) Then
    { endpoints are on the same vertical side of y0 }
    Lower_Bound =
      MAX (Lower_Bound,
           MIN (ABS (Delta_Y_1), ABS (Delta_Y_2)))
```

```
  Else
    { endpoints are on different horizontal sides of x0 }
    If Sign (Delta_Y_1) = Sign (Delta_Y_2) Then
      { endpoints are on the same vertical side of y0 }
      Lower_Bound = MIN (ABS (Delta_Y_1), ABS (Delta_Y_2))
    Else
      { endpoints are on different vertical sides of y0 }
      Lower_Bound = 0
```

A.6.b Detail Of Step 4

The following steps are performed to find the closest edge to a pixel:

```
  Loop through sorted SLL from beginning

    Compute D, the distance squared from
    the pixel center to the current edge.
    { see section A.7 }

    If D < shortest distance squared computed so far Then
      save D and the current edge

  Until the end of the SLL is reached
                    OR
        the Lower_Bound of the next edge is greater than
        or equal to the shortest distance found so far
        { since all remaining edges must be at least that
          far away }
```

A.6.c Detail Of Step 5

The following steps are performed to find the second closest edge to a pixel:

```
Loop through sorted SLL from beginning

  If the current segment is not the closest segment Then
    If the current segment is not incident with
       the closest edge at a concave angle Then
       { see section A.8 }
      If the distance squared from the pixel center to
         the current edge has already been computed Then
        Retrieve D from Distance_Squared.
      Else
        Compute D, the distance squared from
        the pixel center to the current edge.
        { see section A.7 }

      If D < second shortest distance squared
         computed so far Then
        save D and the current edge

Until the end of the SLL is reached
                    OR
      the Lower_Bound of the next edge is greater than
      or equal to the second shortest distance found so
      far { since all remaining edges must be at least
      that far away }
```

A.7 Computing Distance Squared From A Pixel To An Edge

The following steps are performed to obtain the distance squared from a pixel center (x0, y0) to a line segment from (x1, y1) to (x2, y2):

```
Let V1 be the vector from (x1, y1) to (x0, y0).

Let V2 be the vector from (x1, y1) to (x2, y2).

Let DP be the dot product of V1 and V2.

If DP <= 0 Then
  compute the distance squared
  from (x0, y0) to (x1, y1),
  (x0 - x1)**2 + (y0 - y1)**2
Else
  Let V1 be the vector from (x2, y2) to (x0, y0).

  Let DP be the dot product of V1 and V2.

  If DP >= 0 Then
    compute the distance squared
    from (x0, y0) to (x2, y2),
    (x0 - x2)**2 + (y0 - y2)**2
  Else
    compute the perpendicular distance
    from (x0, y0) to the edge
```

using the line equation coefficients for the edge that were computed earlier, the distance squared is

```
(A * x0 + B * y0 + C)**2 / (A**2 + B**2)
```

A.8 Determining Whether Two Edges Form A Concave Angle

The following steps are performed to determine whether two edges are incident at a concave vertex:

```
If the edges are in the same contour Then
  If the second endpoint of the first
     edge is the same point as the first
     endpoint of the second edge Then
    The edges are incident at a common vertex
    so check the vertex type of this vertex.
  Else
    If the second endpoint of the second
       edge is the same point as the first
       endpoint of the first edge Then
      The edges are incident at a common vertex
      so check the vertex type of this vertex.
    Else
      The edges are not incident.
Else
  The edges are not incident.
```

Index Terms - medial axis transform, Blum transform, symmetric axis transform, skeleton, shape representation, feature extraction, shape smoothing, brushfire, Voronoi diagram, ridge following

FIGURE CAPTIONS

Fig. 1. Non-unique closest points on the shape.

Fig. 2. Maximal disks of a shape.

Fig. 3. External medial axis transform.

Fig. 4. Medial point types.

Fig. 5. Medial axis transform of a deformed torus.

Fig. 6. Medial axis branch primitive types.

Fig. 7. Variations of the primitive types.

Fig. 8. Significance of shape features.

Fig. 9. Ho and Dyer shape smoothing example.

Fig. 10. Spread of brushfire in a shape.

Fig. 11. Voronoi diagram of a point set.

Fig. 12. Voronoi diagram of the interior of a polygon.

Fig. 13. Medial axis transform of the polygon in Fig. 12.

Fig. 14. Inversion of a brushfire MAT.

Fig. 15. Distance from a point to a line segment.

Fig. 16. Concave angle special case.

Fig. 17. A shape that the DeSouza and Houghton ridge following algorithm cannot handle.

Fig. 18. Lower bound on the distance from point C to line segment AB.

Fig. 19. Plot of CPU time for initialization when the number of edges is varied.

Fig. 20. Plot of CPU time to find the first medial point when the number of edges is varied.

Fig. 21. Plot of CPU time to find the remaining medial points when the number of edges is varied.

Fig. 22. Plot of CPU time to thin when the number of edges is varied.

Fig. 23. Plot of total CPU time when the number of edges is varied.

Fig. 24. Plot of CPU time for initialization when the shape area is varied.

Fig. 25. Plot of CPU time to find the first medial point when the shape area is varied.

Fig. 26. Plot of CPU time to find the remaining medial points when the shape area is varied.

Fig. 27. Plot of CPU time to thin when the shape area is varied.

Fig. 28. Plot of total CPU time when the shape area is varied.

Fig. 29. Two of the shapes used to test the effect of varying the number of holes on the run time of the ridge following algorithm.

Fig. 30. Plot of total CPU time when the number of holes is varied.

Fig. 31. Three of the shapes used to test the effect of varying the distribution of vertex types on the run time of the ridge following algorithm.

Fig. 32. Plot of total CPU time when the number/distribution of concave vertices is varied.