UNITED STATES DEPARTMENT OF THE INTERIOR

GEOLOGICAL SURVEY

Transformation of Mapped Data to Grid Systems with Applications to Earthquake Data

By

Bonny Askew

Open-File Report 83-248

1982

Table of Contents

Illustrations

I. INTRODUCTION

This paper documents a computer program that transforms map data from a series of digitized points representing regional boundaries into a grid system in which each cell in the grid contains a value indicating the region within which it falls. An example of the program's use is first presented, then the basic problem of transforming map data to gridded data is examined, and some theoretical results which provide a basis for the program are discussed. A general algorithm is described which expands these specific theoretical results into an approach for a general-purpose program. A Fortran program implementing this algorithm is documented, and, finally, other applications of this program are discussed.

In order to understand the usefulness of this transformation, consider the following example of its application in the study of earthquake intensity data. A measure of the effects of an earthquake is given by the Modified Mercalli (MM) intensity scale. The MM scale has twelve degrees, identified by Roman numerals ranging from I to XII. Each degree of the scale is annotated with a specific set of earthquake effects. The intensity at the epicenter may be anywhere on the scale depending on the size of the earthquake and site conditions. The intensity generally falls off with increasing distance from the epicenter, but not usually at a consistent rate due to site differences. For a given earthquake, data showing the intensities felt at different locations are collected, plotted on a map, and then contoured. An example of a contoured intensity map is shown in figure 1. Contoured intensity maps are available for most moderate and large size earthquakes in the United States.

A given location may have experienced a range of intensities resulting from a number of different earthquakes during recorded history. A useful method for evaluating the earthquake hazard at a given location is to divide the area of interest into a grid of cells and, for each cell, store a history showing the intensity of each earthquake felt at that location. The method used to create this cell history is to process a contoured intensity map for each recorded earthquake by marking each cell affected with the intensity value that was experienced. A value of zero is assumed for cells outside the lowest contour. This information can then be appended to a list of intensities stored for each cell. An example of how cells are assigned values from a contoured intensity map is shown in figure 2. In order to compile such an intensity history a computer program is needed which accepts digitized contours defining an intensity map and produces a grid of cells with each cell assigned a value indicating the intensity felt at that location.

II. THEORY

The basic problem that this program must solve is testing whether a given cell is inside or outside a given region as defined by a boundary. This is known as the point inclusion problem for which several different solutions have been devised. Most commonly, a ray is extended to some point known to be outside the boundary and the intersections with the boundary are counted. An odd number of intersections indicates the point is inside (figure 3, point B) and an even number indicates it is outside (figure 3, point A). An exception to this rule occurs when the ray intersects a boundary tangentially (figure 3,

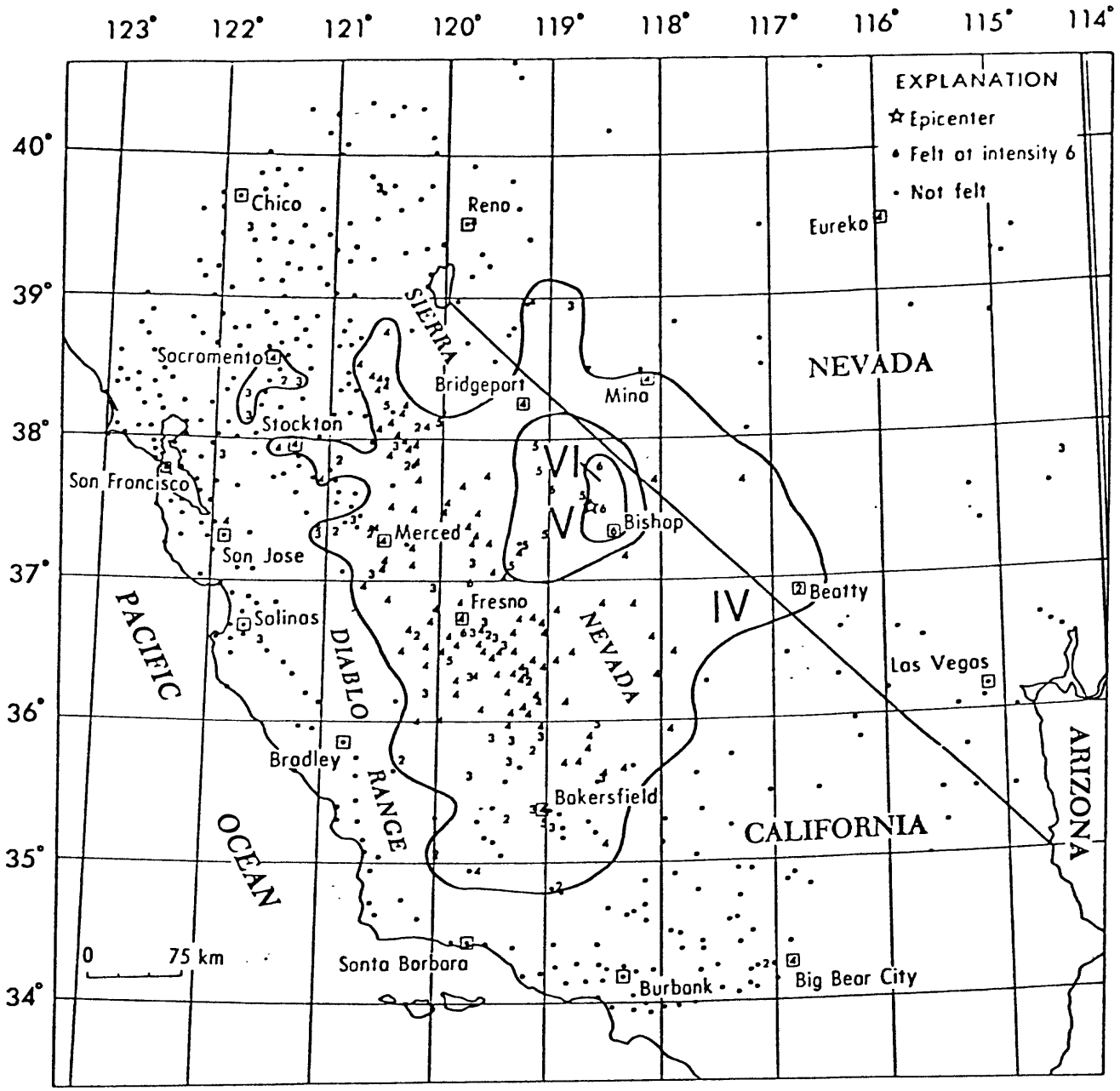


Figure 1.--Contoured Intensity Map taken from Stover, C. W. and C. A. von Hake, U.S. Earthquakes, 1978. 1980.

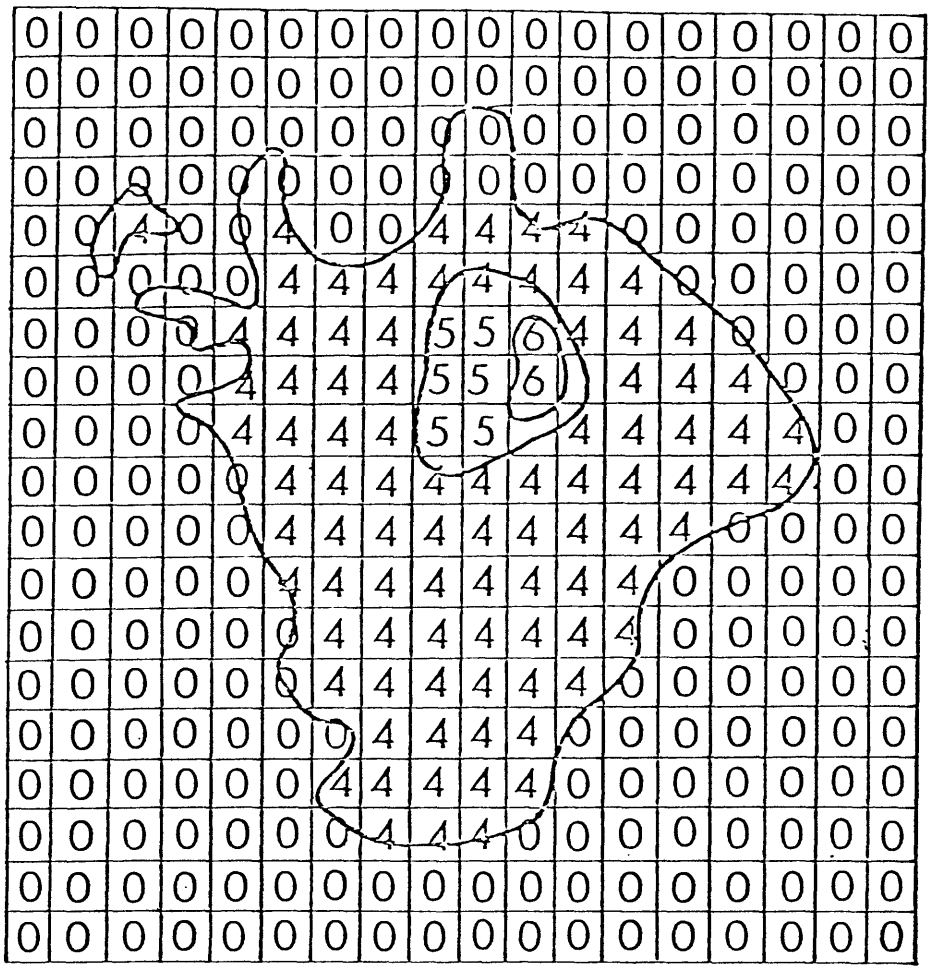


Figure 2.--Example of cell assignment from a contoured intensity map.

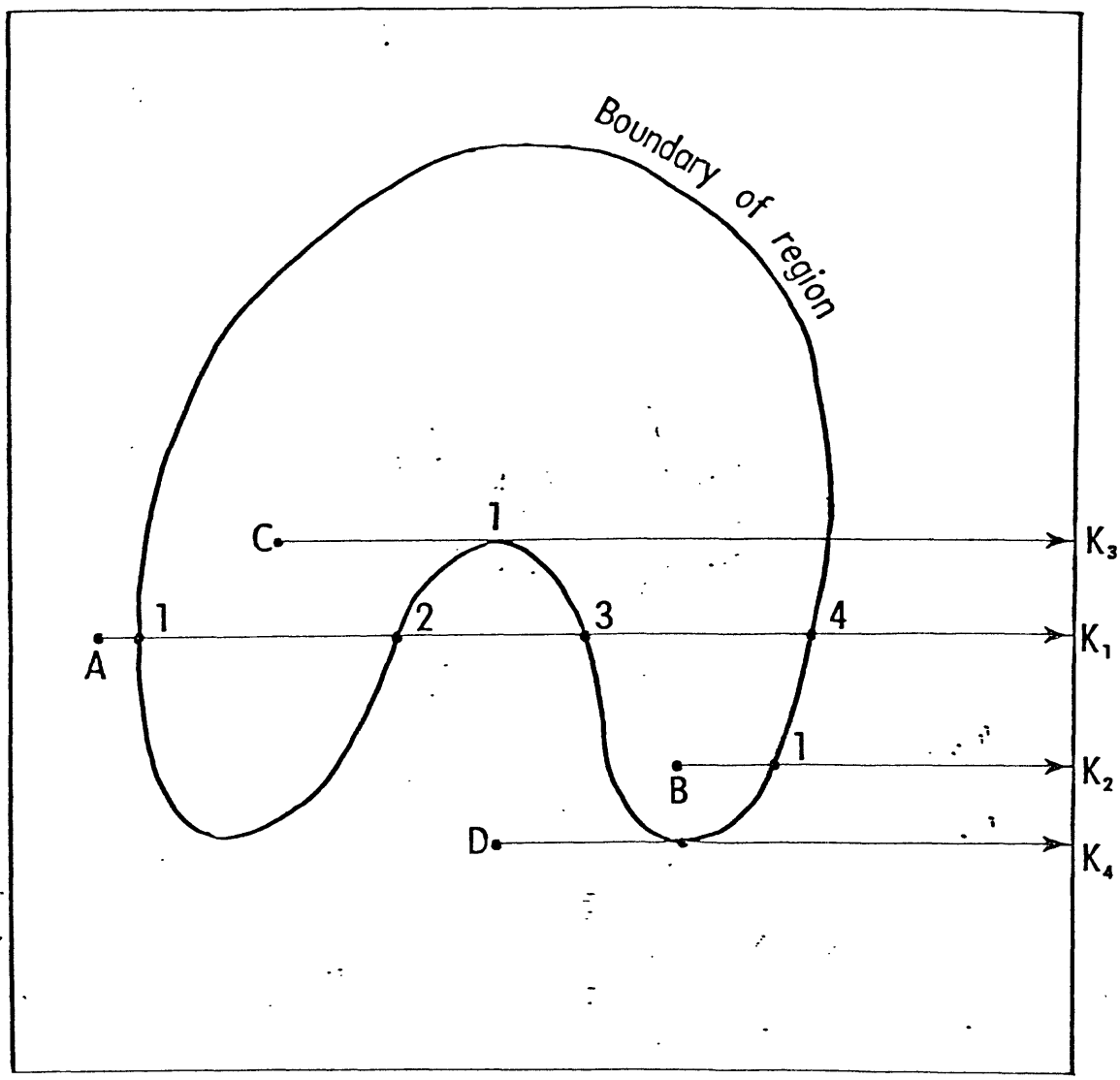


Figure 3.--Counting intersections along a ray from a given point to a point with a known value indicates whether that point is inside or outside a region.

Point A: Four intersections (even) indicate this point is outside.
Point B: One intersection (odd) indicates this point is inside.
Point C: Even number of intersections does not indicate point is outside.
Point D: Odd number of intersections does not indicate point is inside.
Points K_1, K_2, K_3, K_4: Points of known value.

points C and D). In these cases an intersection does not actually indicate a crossing boundary. In order to use this approach, some means must be found to recognize and handle tangential intersections. One method of recognizing these intersections is to check the slope of the boundary line segment in both directions from the point of intersection as described in Newman and Sproull (1979). If the slope changes sign at the intersection (assuming a horizontal ray) a tangent intersection is indicated as seen in figure 4. To keep the counting scheme in order, such an intersection must be counted twice or not counted at all.

In addition, some criteria must be established for assigning a value to a cell that is cut by a boundary such that the cell is partially included and partially excluded from a region. Here, the criterion applied is the percentage of a cell's area in each region.

The next step is to extend this algorithm to determine, for an entire grid of cells, whether each cell is inside or outside a single region, or polygon. It is not necessary to check each cell for inclusion within a region by counting boundary intersections along a ray. A coherence property indicates that two adjacent cells are both included within or excluded from, the polygon unless a boundary line passes between them. One cell can be tested and its value determined, and this value can be propagated in a given direction until a boundary line is encountered (figure 5). In most applications, a significantly larger number of cells can receive values by propagation rather than by testing.

The next step is to compute cell values for a map with multiple boundaries defining different valued zones. One approach is to consider each region separately using the above algorithm. This approach has several drawbacks: (1) A problem arises in the case of one region being contained entirely in a second. The order of processing such regions would affect the final cell assignment. (2) The procedure is inefficient because many cells will be processed more than once. (3) A single line segment would represent the boundary between two regions. If this line is digitized separately in each case, this may introduce some inaccuracies. Storing the boundary twice would also be inefficient. (4) The case when a cell is cut by the boundaries of more than one region may present difficulties. Keeping track of the percentage of the cell's area in each region may become complicated or accuracy may be lost. For these reasons, processing all the regions together becomes desirable. The following technique discussed by Nagy and Wagle (1979) accomplishes this.

When working with map data, this approach is efficient, versatile, and useful with two particular advantages. First, few limitations are set on the characteristics of the input regions. For example, this method can work with a map divided into a large number of regions; the boundaries do not have to form closed figures; boundaries may lie very close to one another; and regions contained entirely within other regions present no problem. The second advantage is that boundaries dividing two regions need to be digitized only once. The concepts used to determine individual cell values are based on the theory outlined above and are integrated into an algorithm that processes an entire map efficiently. Each boundary is read in, the areas of the cells

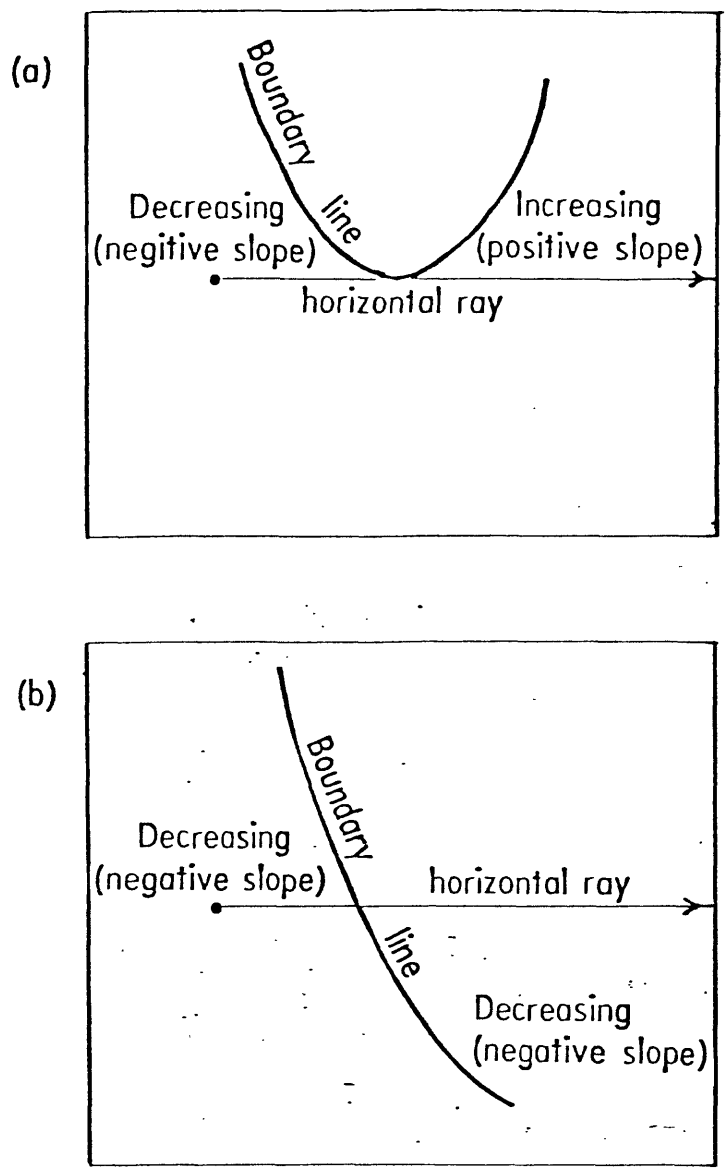


Figure 4.--Recognizing a tangent intersection by checking the sign of the slope on either side of the intersection.

a) Change in the sign of the slope indicates a tangent intersection.
b) No change in the sign of the slope indicates a non-tangent intersection.

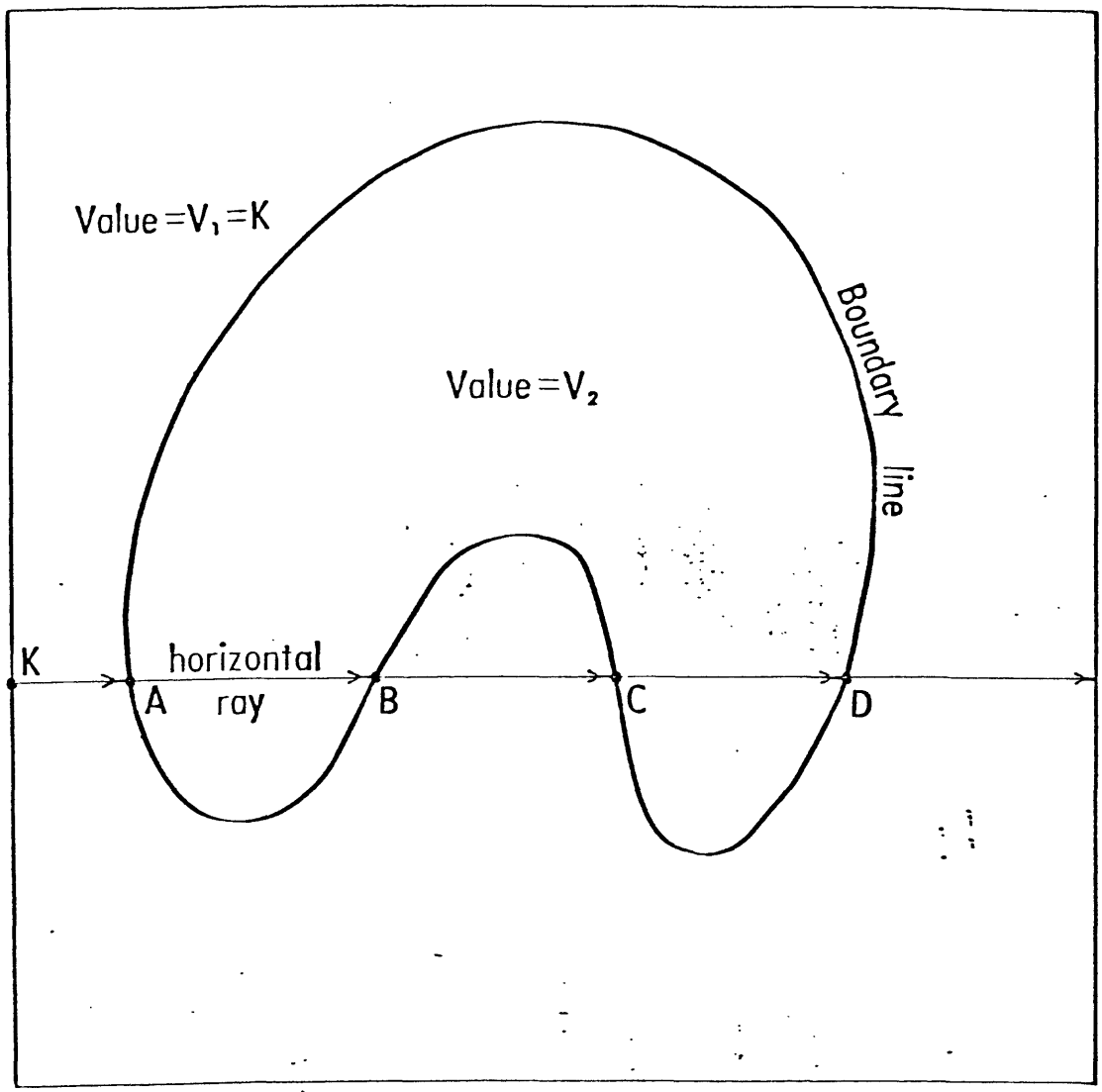


Figure 5.--Propagation in theory: The value at the window boundary (K) is known and is propagated along the horizontal ray to point A. Point A is found to be inside and assigned the value of the area inside the boundary line (V_2). This value is propagated to the right along the horizontal ray until another boundary is reached at B. B is found to be outside the boundary and is assigned the value V_1. This is propagated as far as C. C is found to be inside the boundary and given value V_2. This value is propagated to D. D is outside the boundary and given value V_1. This value is propagated to the window boundary.

crossed by boundaries are incremented, criteria for propagation are established, and then the boundary is discarded. Thus, all the boundaries do not need to be stored in the program. A Fortran implementation of this algorithm is presented here. A summary of the general algorithm follows next.

III. THE ALGORITHM

First, terms will be defined as used in this paper. Figure 6 illustrates the definitions.

Boundary - A series of connected line segments with direction, forming a division between two regions. As the segments are traversed in the specified direction, one region lies always to the right and another always to the left. A change in either adjacent region marks the beginning of a new boundary. A boundary may be a closed polygon.

Edge - One straignt line segment defined by two points forming part or all of a boundary.

Segment - All or part of an edge falling within a single cell.

A map dividing a given area into regions is defined by boundaries. These boundaries are input using a coordinate digitizer, so that a boundary is then defined by a series of points. Each consecutive pair of points defines an edge of the boundary. The values of the regions to the right and left of the boundary are also entered. (Since a boundary has direction, right and left sides are defined). A uniform grid of cells of a specified size is defined to overlay the map. Each edge is broken into its component segments, by computing the points where it crosses grid lines.

Each segment is associated with a cell, which is defined by a column and row in the grid. The percentage of the cell's area lying within the region on each side of the segment is computed. This information is stored for each cell cut by a segment. If additional segments cut the same cell, the information is updated. After processing all segments, cut cells can be assigned a value by determining which region covers the largest proportion of the cell.

In order to find values for interior cells (those uncut by boundaries), values are propagated from a known point. By propagating in a specified direction from each boundary until another boundary is reached, all interior cells will be covered. In the previous examples, rays were shown extending horizontally to the east and propagation took place in that direction. In this algorithm, the ray will be extended vertically to the south, along the west walls of a column of cells. When a boundary segment intersects a west wall, a value is determined for the south side of the intersection and that value is propagated down the west-wall ray until another intersection point is encountered. This propagation is depicted in figure 7a. If a value is known for the west wall of a cell, the value will hold for the entire cell if it is uncut. If the cell is cut, but no segments intersect the west wall, propagation can continue past that cell along the west-wall ray without assigning a

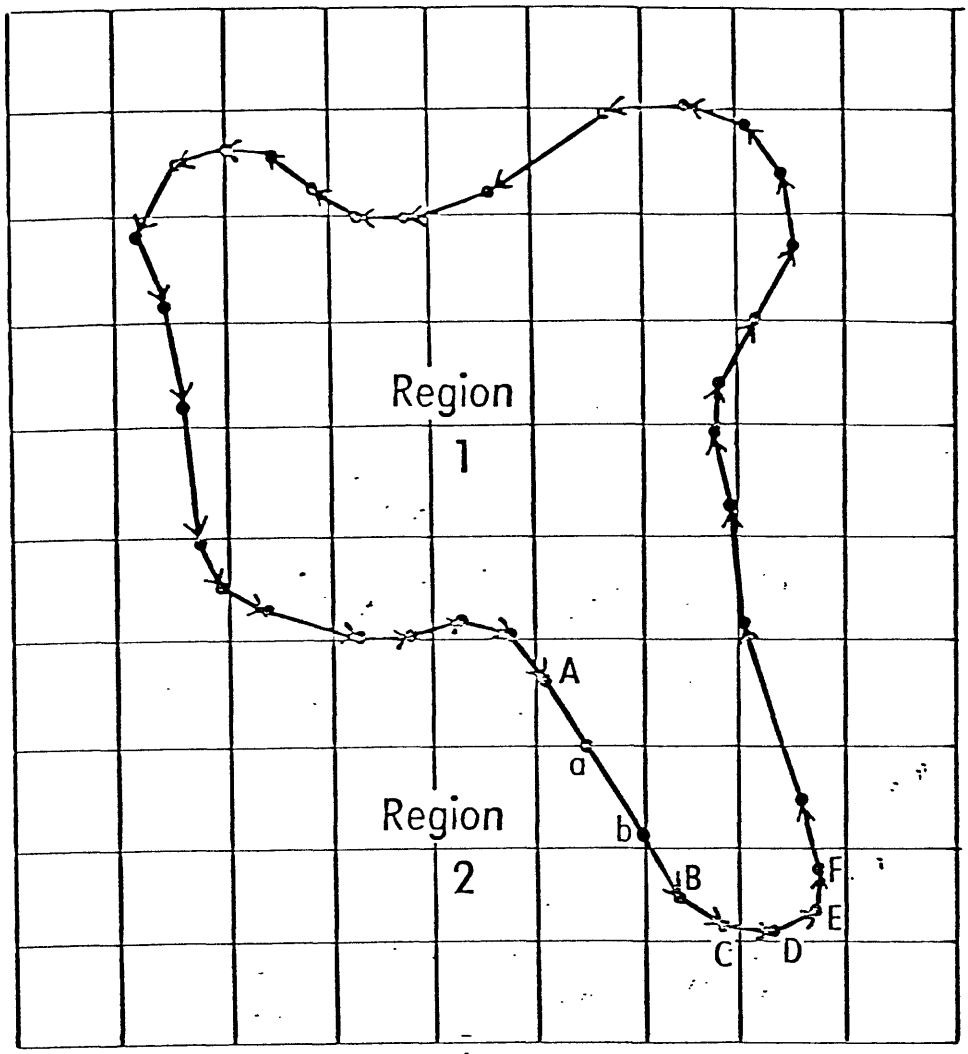
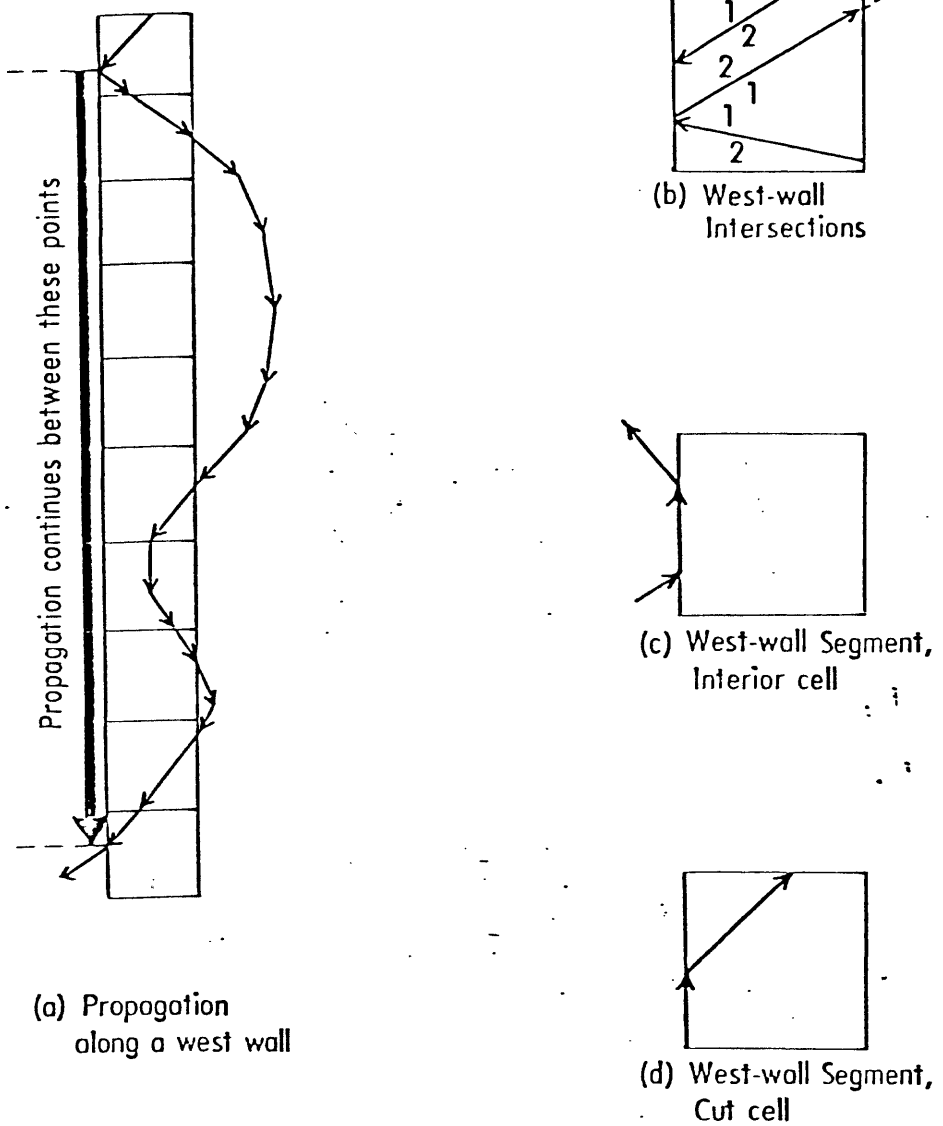


Figure 6.--Definition of Terms:

Boundary	-	The entire perimeter of the polygon dividing region 1 from region 2.
Edge	-	Line segment from A to B or B to C, etc.
Segment	-	Part of an edge that falls within a single cell as in a to b or an entire edge as E to F.



(a) Propogation along a west wall

(b) West-wall Intersections

(c) West-wall Segment, Interior cell

(d) West-wall Segment, Cut cell

Figure 7.--Propagation in actual use. The method of propagation along a west wall is illustrated in A. Figure B illustrates how several intersections on the west wall affect the value in the southwest corner. Figures C and D illustrate how a segment falling entirely on a west wall is not needed to determine propagation.

value to the cut cell. For cut cells with segments intersecting the west wall, a value can be determined for the southernmost point of the west wall and that value can be propagated southward. Such cells will be referred to as propagating cells.

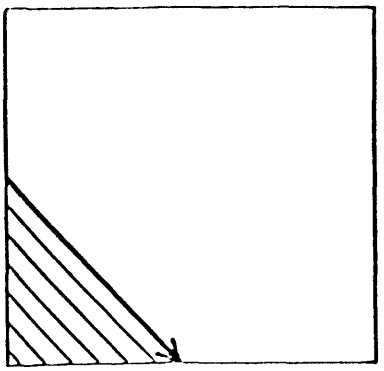
In order to find the value of the southwest point of a cut cell, it is necessary to know all the segments that intersect the west wall. A diagram of a cell with west wall intersections is shown in figure 7b. The values of regions to the right and left of each segment are indicated. In this algorithm, it is not necessary to count intersections along the ray since the regions on each side of a segment are identified. If two or more segments intersect the west wall at different points, the lowest intersection will determine the value at the southwest corner. If two segments intersect at the same point, this is equivalent to the tangent intersection discussed previously. The segment with the lower slope will cross the cell lower and will determine the value at the southwest corner. Refer to figure 7b.

While processing each segment, values are updated which identify if and how propagation will occur. Each segment is tested for intersection with the cell's west wall. If this occurs the cell is marked as a propagating cell, and the point of intersection, the slope, and the region to the south are stored. If another segment intersects the west wall, it is tested to see if it has a lower intersection or an equal intersection and a lower slope. If so the intersection point, the slope, and the region are updated.

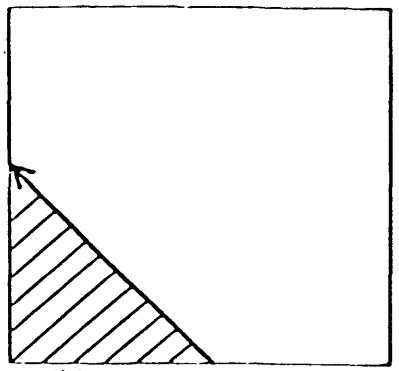
If a segment falls entirely on a west wall, it is ignored because another segment intersecting the west wall will determine propagation. Examples of such segments are shown in figures 7c and 7d. The cell in 7c is treated as an interior cell and receives a value propagated from the cell above. The cell in figure 7d is a cut cell, but the vertical segment has no part in determining propagation.

After all segments are processed, the value of the region in the southwest corner has been stored and that value is propagated south until another propagating cell is reached. Cells that are cut but not propagating are ignored; propagation continues past them, but they receive their values by a comparison of areas of regions overlapping the cell.

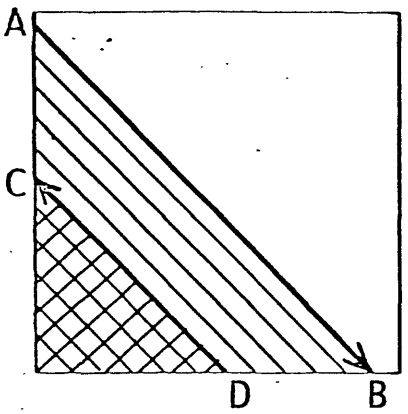
Integration of the segment within the cell has a few complexities (figure 8 and 9). The cell is assumed to be a unit cell having sides of length 1.0 and area of 1.0. An integration is performed to find the area under the segment. This gives a positive result if the segment crosses the cell from left to right and negative if it crosses from right to left. Adjustments must be made to include areas under a north wall when part of the wall acts as a region boundary within that cell. This occurs when one or more segments intersect the cell's north wall. In case 1, the segment lies to the right of the intersection point on the north wall. An additional amount must be added to the integral equal to the X-coordinate of the intersection. In case 2, the segment lies to the left of the intersection. A value of 1 minus the X-



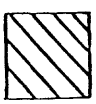
(a) POSITIVE INTEGRAL
(area to right of segment = integral)



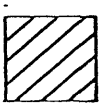
(b) NEGATIVE INTEGRAL
(area to right of segment = 1-integral)



(c) Negative integral subtracted from positive integral (area to the right of both segments = integral of $\overline{AB}$ - integral of $\overline{CD}$)



Positive integral



Negative integral

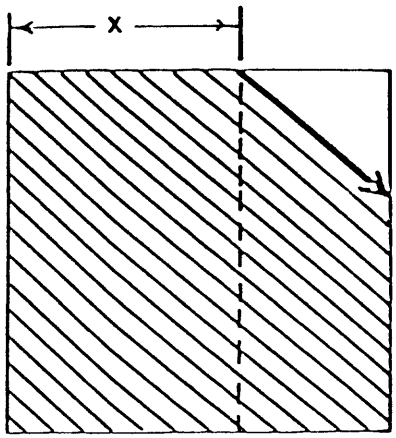
Figure 8.--Integrating under the North Wall.

X: The distance from the northwest corner of the cell to the north wall intersection point.

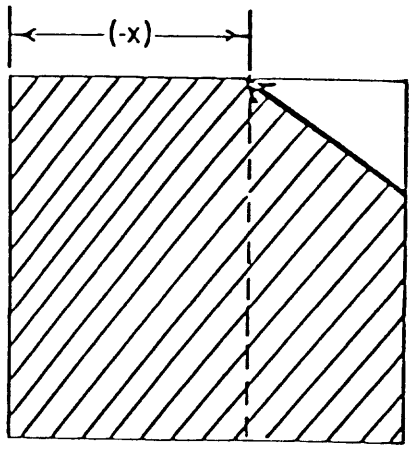
Case I: The segment lies to the right of the north wall intersection and the value of the north wall integral is X.

Case II: The segment lies to the left of the north wall intersection and the value of the north wall integral is (1-X).

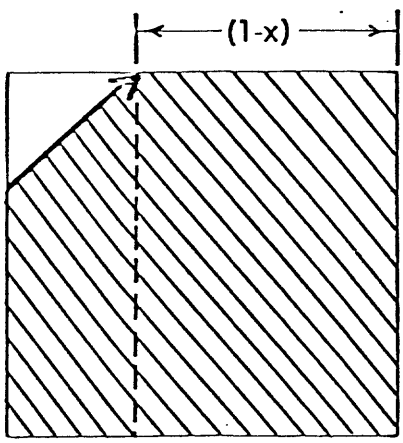
Negative integrals: When the segment crosses the cell from right to left, rather than left to right, the value of the north wall integral is the negative of what it would otherwise have been.



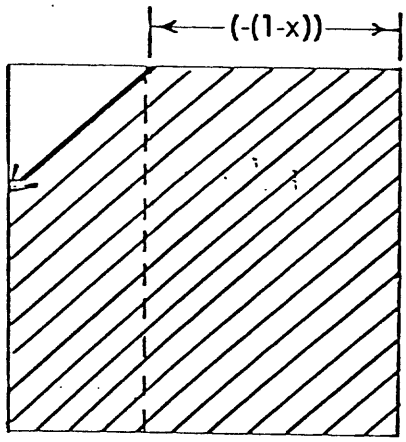
CASE I: POSITIVE INTEGRAL



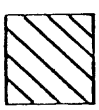
CASE I: NEGATIVE INTEGRAL



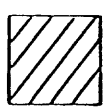
CASE II: POSITIVE INTEGRAL



CASE II: NEGATIVE INTEGRAL



Positive integral



Negative integral

Figure 9.--Negative Integrals. The integral is positive when the segment crosses the cell from left to right and negative when it crosses from right to left. If the final result is negative a value of 1.0 is added to give a positive result. A negative integral may be a subtraction from a positive integral.

coordinate must be added to the integral. If the segment crosses from right to left these north wall integrals are subtracted rather than added. This total integral then gives the area to the right of the segment. The area to the left equals 1 minus this integral.

Figure 9 illustrates positive and negative integrals. A positive area is added to any accumulated area of that region in that cell; a negative area is subtracted. If no other segment has crossed that cell, this leaves a negative area in some cases. This negative area actually represents an area subtracted from the area of the whole cell. Therefore, a value of 1 must be added to negative results (fig. 9b). In some cases, the result will be greater than 1. The true area is this value modulo 1.0. The final result, then, is always greater than or equal to 0 and less than 1, indicating the percentage of the cell's area covered by that region.

Special consideration must be given to the cases when a segment falls exactly on a grid line. Segments on a grid line will always be considered as falling on the west wall of the cell to the right or the south wall of the cell above. Since a vertical line has an integral of 0, west-wall segments can be ignored. A south-wall segment will give an integral of 0 for the area to the right. The area to the left is then 1 minus 0, modulo 1.0, which also equals 0. One of these regions actually covers the entire cell, and this information is stored when such a segment is encountered. During the phase of assigning cell values, a cell which is cut but has all areas equal to zero, is known to be cut only on the south wall, and the value which was stored earlier is used.

IV. THE PROGRAM

Three data structures are primary to the program. These three structures are two-dimensional arrays named CELL, CUT, and PROP which are diagrammed in figure 10. Figure 11 shows sample values after processing the segments and before assigning values to the cells. The CELL array corresponds to the cell grid in which the first subscript designates a grid column and the second a grid row. While processing the segments, each cut cell is assigned a value which points to a row in the CUT array.

The CUT array has enough columns for the maximum number of regions which can cut a single cell plus one additional column. Column two and successive columns are filled with values indicating a region number and the area of the cell in that region. This is given by a single real number. The integer part gives the region number and the fractional part gives the area (which is always greater than or equal to 0 and less than 1). A given region number will be placed in a column only once, then incremented as needed. The first time that region is encountered in that cell, it is assigned to the first unused column in that row. If the cell is also a propagating cell, the first column of the CUT array is used to hold a pointer into the PROP array.

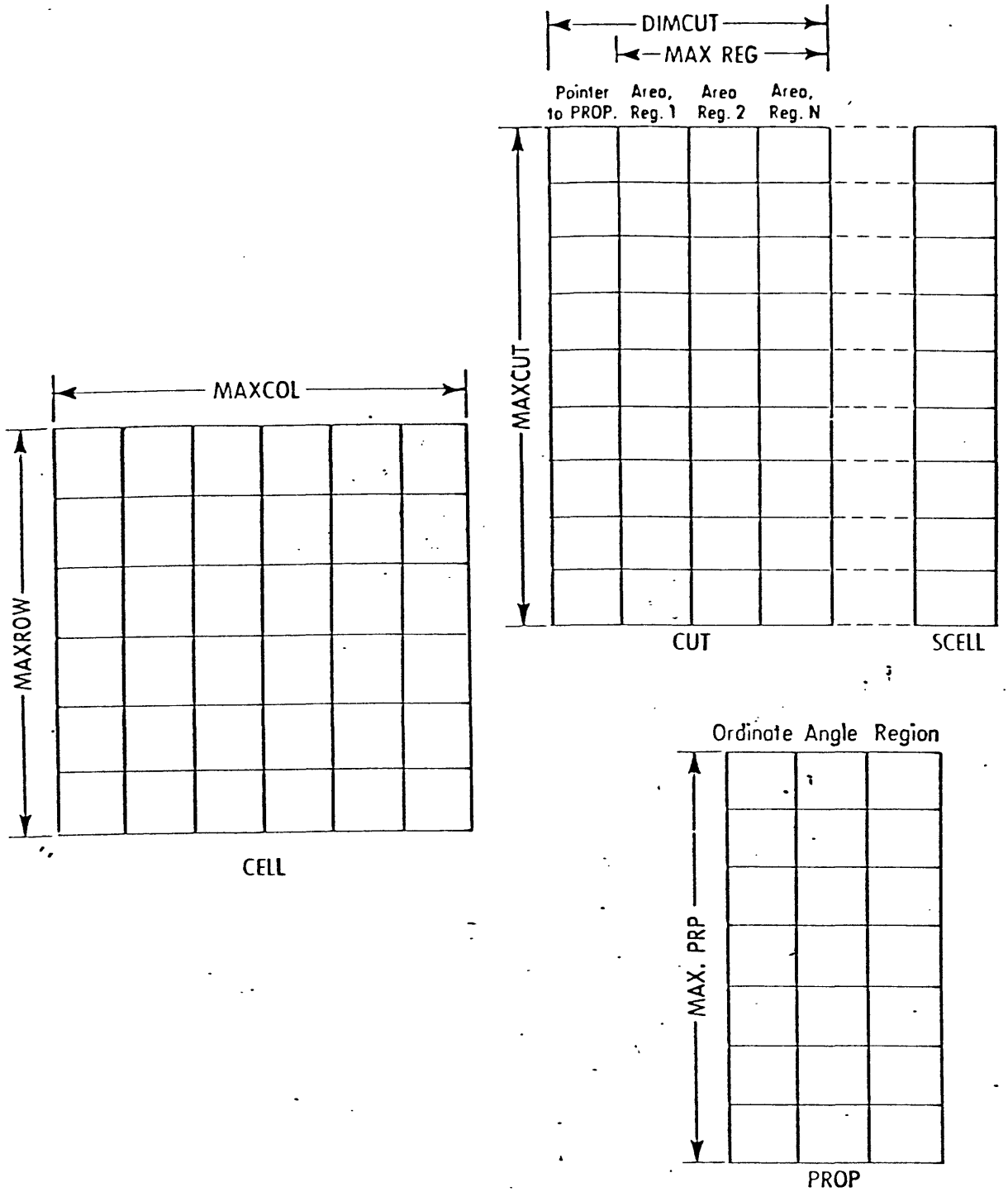


Figure 10.--Data Structures.

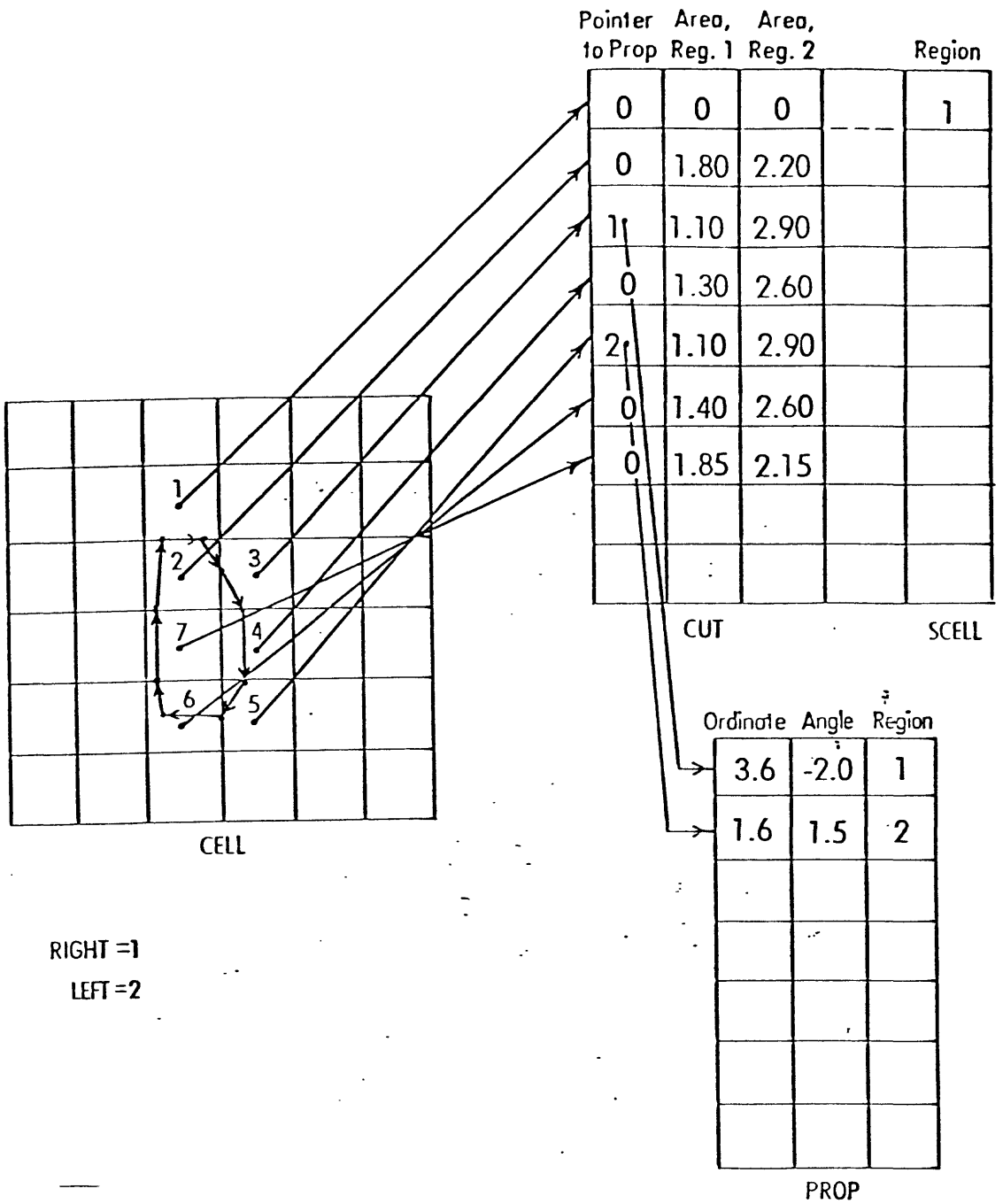


Figure 11.—Example of how values are placed in the data structures.

The PROP array has 3 columns that contain the information needed to find the value of the southwest corner of the cell. These three pieces of information are the ordinate, slope, and region to the south of the last encountered segment which will determine propagation. Any time a segment is encountered in a given cell which has a lower ordinate on the west wall, or an equal ordinate and lower slope, these values are updated to correspond to that segment.

Another array, SCELL, stores the value of a region covering a cell which is cut only by a south-wall segment. It is a one-dimensional array accessed by the same pointer as the CUT array.

The array dimensions are set by parameters. In order to change any dimensions in the program, only these parameters need to be changed. The Fortran "include" statement is used so that only one copy of the parameters is maintained. At compile time this copy is substituted in each routine that calls for it. MAXSEG indicates the maximum number of segments in any edge. MAXROW and MAXCOL are the maximum dimensions of the cell grid. MAXCUT is the limit on the number of cut cells, and MAXPRP limits the number of propagating cells. MAXREG is the maximum number of regions in a single cell. DIMCUT is the column dimension of the CUT array and must be one greater than MAXREG.

These data structures differ from those proposed by Nagy and Wagle (1979) by using pointers into the CUT and PROP arrays. The alternative to the use of pointers is to allot space for each cell to store the information needed for cut and propagating cells. The pointers use additional storage, but will save more storage unless the cells are so large or the boundaries so dense that almost all cells are cut. This is an important consideration since data storage space may be a limiting factor in many applications.

A summary of the input file is shown in figure 12. The first input into the program defines the grid characteristics. The variable, UNIT, defines the spacing between grid lines. The coordinates, (XO, YO) define the southwest corner of the cell in column 1 and row 1. The size of the grid is established by the coordinates (XMAX,YMAX). The value given for MINMOV defines the minimum length of a boundary edge. Points are read and discarded until a minimum length edge is found. A value of 0 for MINMOV will cause the edges to be processed exactly as they are input.

In order to fill the entire cell grid with region values, the north side of the grid must act as a boundary from which values are propagated southward. Otherwise, the area between the north side of the grid and the northernmost boundary will contain whatever value the cells contain initially. This may be undefined or incorrect. This program assumes YMAX is greater than the northernmost boundary by a value at least equal to the value of UNIT. This results in a continuous region along the north side of the grid; no boundary intersects the north side. The value of NORTH indicates the region bordering the north side. After all boundaries are processed, the north side of the grid is treated as another boundary defined by (XO,YMAX) and (XMAX,YMAX) with NORTH as the region to the right and -1 as the region to the left. It is processed the same as all other boundaries.

Unit
XO, YO, XMAX, YMAX
NORTH
MINMOV

RIGHT, LEFT, NPTS
X_1, Y_1
X_2, Y_2
.
.
.
X_{NPTS}, Y_{NPTS}

RIGHT, LEFT, NPTS
X_1, Y_1
X_1, Y_2,
.
.
.
X_{NPTS}, Y_{NPTS}

.
.
.

Figure 12.--Format of Input File for program CELASN

UNIT - Spacing of cell grid
XO, YO - Smallest values of X and Y coordinates
XMAX, YMAX - Largest values of X and Y coordinates
NORTH - value of region adjacent to north boundary of the grid
MINMOV - minimum movement between digitized points to be processed
RIGHT - value of region to the right of a boundary
LEFT - value of region to the left of a boundary
NPTS - number of points in a boundary
X_I, Y_I - X and Y coordinates of the I^{TH} boundary point.

These seven values should appear on the first four lines of the input file as shown in figure 12. This is followed by the input defining a series of boundaries. The boundary definition consists of one line of descriptors defining the region to the right (RIGHT), the region to the left (LEFT), and the number of points in the boundary (NPTS) followed by the series of digitized points (X_I, Y_I). This is repeated for each boundary.

In order for this program to run correctly, all digitized boundaries must intersect exactly without gaps or overlaps. Since at some level of accuracy digitized data are not exact, pre-processing of the data will often be necessary. The type of processing will depend on the nature of the input. In the earthquake-intensity-map example used here, the primary need for pre-processing is to insure that the first and last points of each boundary are the same so that the polygonal contours close. Figure 13 illustrates the effect of an unclosed boundary.

The nesting of the subroutines, as defined by the calling structure, is shown in figure 14. A description of each routine follows given in outline form.

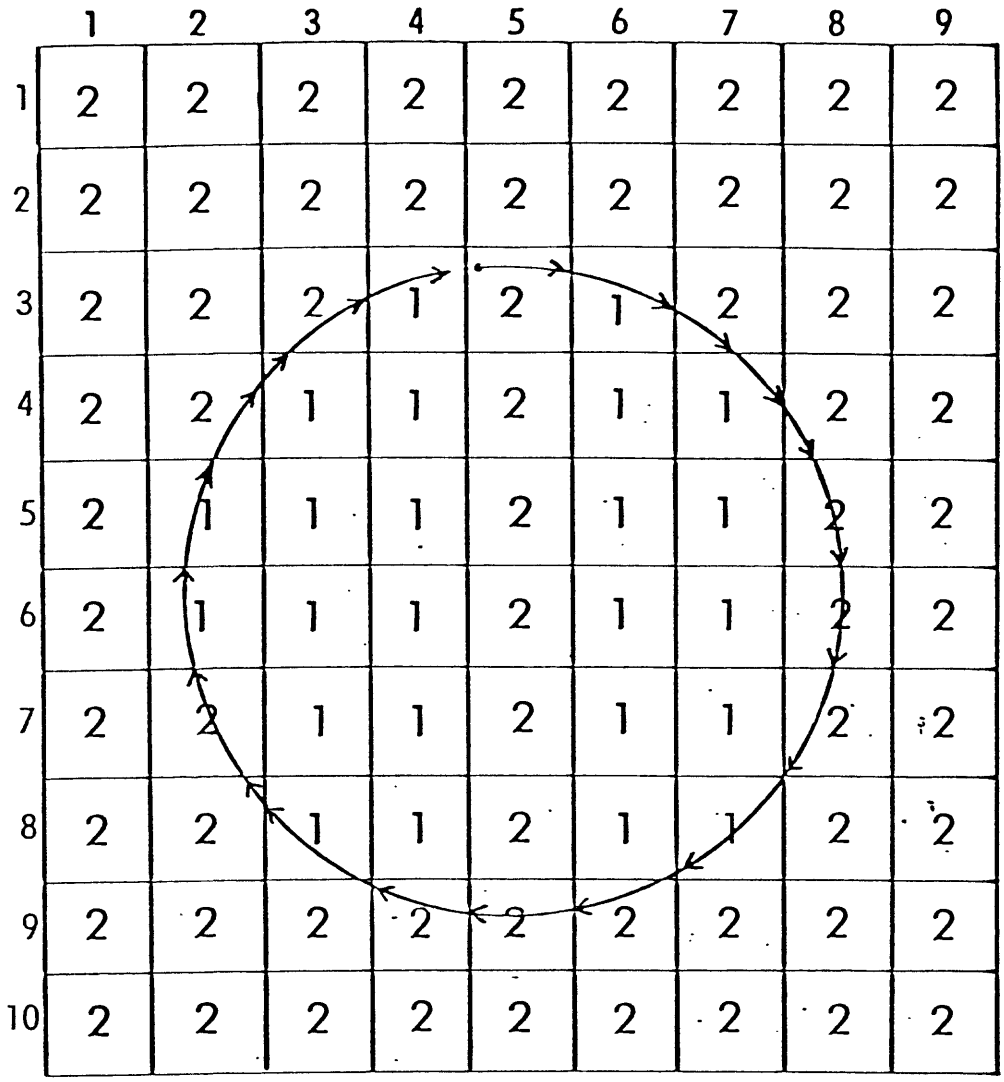


Figure 13.--Effect of digitizing error. As a result of an unclosed boundary in row 3, column 5 propagation is not carried out properly.

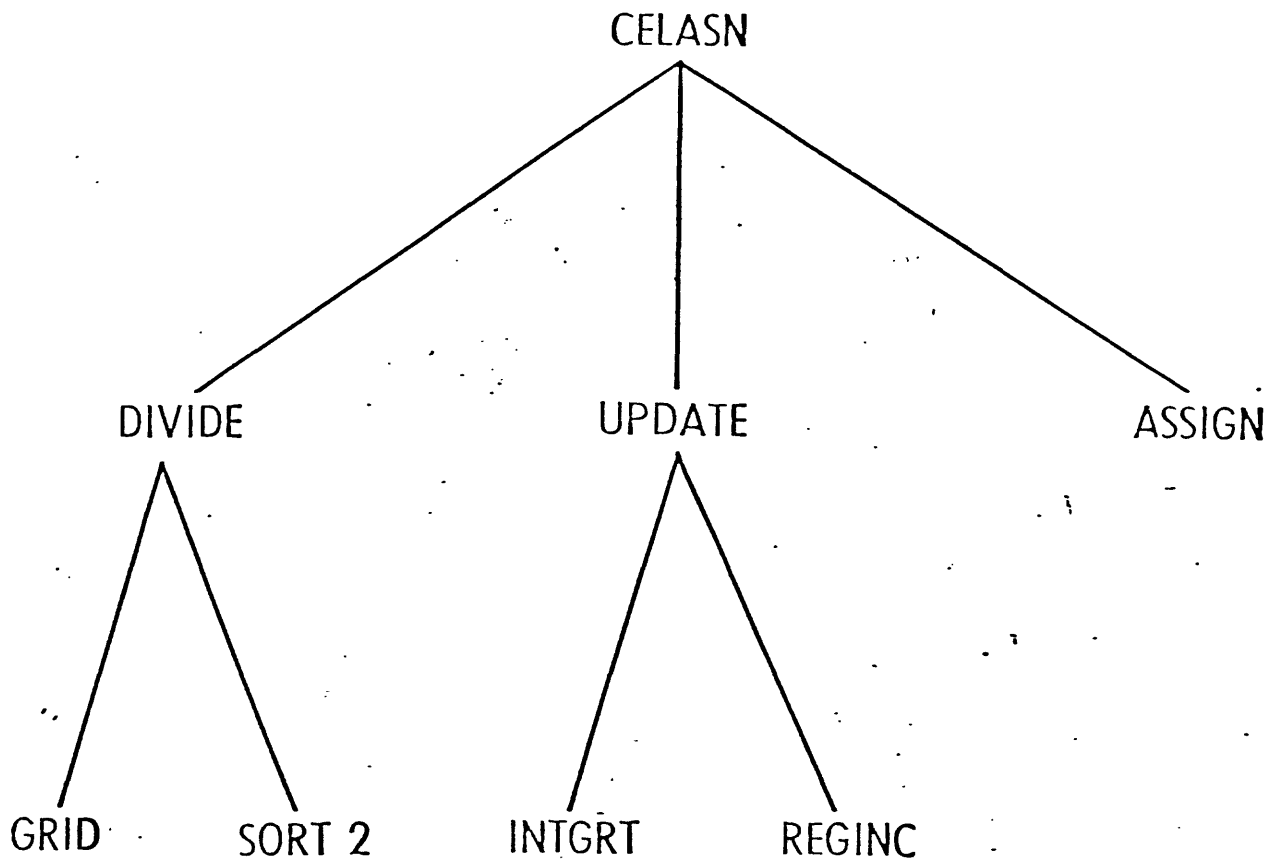


Figure 14.--Nesting structure of subroutines in the program CELASN. See the text for a full description of each routine.

CELASN

This program takes a series of boundaries between regions on a map and produces a grid of cells overlying the map. A value is given to each cell determined by the value of region covering the largest part of the cell's area.

I. Initialize
II. Loop on each boundary:
    A. Loop on each edge:
        Read points until edge is at least MINMOV in length
    B. Divide edge into segments
    C. Update cell descriptions
III. For north wall:
    A. Divide edge into segments
    B. Update cell descriptions
IV. Assign cell values
V. Write results

DIVIDE

This subroutine divides a boundary edge into segments, each falling within a single cell.

I. Place crossings of edge with grid lines in SEG array
    A. Find crossings in X-direction
    B. Find crossings in Y-direction
II. Sort SEG array in direction edge travels

GRID

Given a line defined by two points, this subroutine returns all points where the line crosses grid lines in a specified direction, either parallel to the X-axis or the Y-axis. If the initial point of the line falls on a grid line, this is not placed in the array, but if the terminal point falls on a grid line, it is included.

I. Find the slope and intercept of the line, and initialize the grid increments
II. Find the number of crossings with grid lines and the first crossing
    A. Case 1: increasing direction, terminal point > initial point
    B. Case 2: decreasing direction, initial point > terminal point
III. Generate remaining crossings by adding increments to initial crossing appropriate number of times

SORT2

This is a straight insertion sort on a two-dimensional array. The sorting algorithm is taken from Knuth (1973, p. 81).

I. Pull out each element after the first to test it
    A. Test element against each preceding one to find correct position
    B. Place the element in its correct position

UPDATE

This subroutine examines each segment of an edge, computes the area of the region on either side of the segment within the cell, computes information for determining the propagation region, and then updates the CELL, CUT, and PROP arrays appropriately.

I. Initialize next segment ignoring duplicate segments in the array
II. Update CUT array
    A. Find column and row of the cell
    B. Find or initiate pointer into CUT array
    C. Find integral of segment
    D. Increment region to right by amount of integral
    E. Increment region to left by (1-integral)
III. Check if cell is propagating
    A. Find leftmost point and region to south of segment
    B. Test if leftmost point intersects west wall of cell
    C. If propagating update PROP array
        1. Find or initiate pointer to PROP array
        2. Test if ordinate of west-wall intersection is greater than value stored; if equal test if angle is less than that stored
        3. If no value is stored yet or ordinate is greater or ordinate is equal and angle is less then update ordinate, angle, and region

INTGRT

This subroutine computes the integral under a line segment crossing a cell. The integral is computed assuming a unit cell of area 1, so the value of the integral is always less than or equal to 1. A segment crossing the cell from right to left produces a negative integral.

I. Adjust values for a unit cell
II. Compute integral under segment
III. If north wall intersection, compute integral under north wall

REGINC

This subroutine searches across a given row in the CUT array for a given region, then increments the area of that region according to the value of INTGRL. The area is kept >=0 and <1 by adding 1 to negative numbers and taking the area modulo 1.

I. Find this region in the CUT array row or initiate it
II. Adjust area to be >=0 or <1
III. Increment area of region

ASSIGN

This subroutine loops through each cell in the grid, assigning it a value according to which region covers the largest area of the cell. If the cell is cut by one or more boundaries, the region covering the largest area within the cell is determined. If the cell has a boundary intersecting the west wall, it propagates the value of the southwest corner of the cell through all cells south of it until another propagating cell is encountered. If a cell is cut but all regions have an area of 0.0, this indicates that the cell is cut only by a boundary on the south wall and the value of the cell has been previously stored in the array SCELL.

I. Loop on each cell:
    A. If cell is cut:
        1. Find region covering largest area; if all regions equal 0, find region from SCELL array
    B. If cell is propagating:
        1. Loop through cells to south:
            a. If cell is cut and propagating, stop loop
            b. Assign cell propagating value

V. APPLICATIONS

This program performs a basic function needed when working extensively with geographic data. In some cases, a gridded structure is explicitly needed as for output to a line printer or a raster plotting device. In other cases, transforming data to a gridded format expands data compilation and manipulation capabilities or increases efficiency. In the example used in the introduction, the gridded cell structure gives the capability to store overlays of intensity contour maps. When data are stored in a cell grid, information for a specific location can be easily retrieved. Thus an entire history of earthquake intensities can be stored and easily retrieved for any location, with the resolution controlled by the size of the cell used.

Another example from the study of earthquakes illustrates some of the added capabilities and efficiency achieved by transforming map data to gridded structures. We wish to divide the United States into zones that are geologically similar. We have a file storing the location of the epicenter and other pertinent data about each earthquake recorded in the United States. For each

zone, we wish to compile a file of earthquakes with epicenters located in that zone. The boundaries of the zones are digitized and used as input for this program. The program returns a grid of cells covering the United States, with a value in each cell indicating the zone in which the cell falls. The location for each earthquake in the file is determined to fall in a particular cell and the earthquake is then assigned to the zone indicated by the value for that cell. Other geologic features with known latitude and longitude coordinates may also be associated with zones using the same procedure. The size chosen for the cell grid will determine the accuracy of the results.

REFERENCES

Knuth, D. E., 1973, The art of computer programming, v. III: sorting and searching, Reading, Mass., Addison-Wesley, p. 81.

Nagy, G., and S. G. Wagle, 1979, Approximation of polygonal maps by cellular maps, Comm. ACM 22, 9, p. 518-525.

Newman, W. W. and R. F. Sproull, 1979, Solid area scan conversion, in Principles of interactive computer graphics, New York, McGraw-Hill, p. 2290246.

Stover, C. W. and C. A. von Hake, 1980, United States Earthquakes, 1978, U.S. Geological Survey and National Oceanic and Atmospheric Administration, 31 p.

```
c      program celasn

c-------------------------------------------------------------------------
c-----programmed by bonny askew
c-----
c-----this program takes a series of map boundaries defining regions
c-----and produces a grid of cells overlying the map, where each cell
c-----holds a value indicating the region covering the largest part of
c-----the cell's area.
c-------------------------------------------------------------------------

       include 'param.for'

       integer cell(maxrow,maxcol),scell(maxcut),npt,nseg,ncut,nprop,jpt
       integer k,right,left
       real seg(maxseg,2),cut(maxcut,dimcut),prop(maxprp,3)
       real unit,minmov,x1,y1,x2,y2,move



c-----initialize
       call openfl
       read(5,2000) unit
       read(5,3000) x0,y0,xmax,ymax
       read(5,2000) minmov
       read(5,4000) north
       x0 = x0/unit
       y0 = y0/unit
       xmax = xmax/unit
       ymax = ymax/unit
       minmov = minmov/unit
       ncol = int(xmax-x0)+1
       nrow = int(ymax-y0)+2
       if ((nrow .gt. maxrow) .or. (ncol .gt. maxcol)) call error(3)

c-----for each boundary, process its edges
   10 continue
          read(9,5000,end=100) right,left,npt
          read(9,6000,end=100) x2,y2
          x2 = x2/unit-x0
          y2 = y2/unit-y0
          jpt = 1

c--------for each edge, divide into segments and update cell descriptions
   20     continue
             x1 = x2
             y1 = y2
   40        continue
             jpt = jpt+1
             read(9,6000,end=100) x2,y2
             x2 = x2/unit-x0
             y2 = y2/unit-y0
             move = sqrt((x2-x1)**2+(y2-y1)**2)
             if (move .lt. minmov) go to 40
             call divide(x1,y1,x2,y2,seg,nseg)
             call update(right,left,x0,y0,seg,nseg,
     +                   cell,cut,ncut,prop,nprop,scell)
          if (jpt .lt. npt) go to 20
       go to 10
  100 continue
```

```
c-----process north boundary
      x1 = 0
      y1 = nrow-.5
      x2 = ncol
      y2 = nrow-.5
      call divide(x1,y1,x2,y2,seg,nseg)
      call update(north,0,x0,y0,seg,nseg,
     +            cell,cut,ncut,prop,nprop,scell)

c-----fill cell grid with values for predominant region
      call assign(cell,cut,prop,scell,ncol,nrow)
      do 200 k=1,(nrow-1)
         j = nrow-k+1
         write(10,7000) (cell(i,j),i=1,ncol)
  200 continue
      go to 999

  900 continue
      call error(1)

  999 continue
      stop

 2000 format(f12.6)
 3000 format(4f12.6)
 4000 format(i4)
 5000 format(3i4)
 6000 format(2f12.6)
 7000 format(1x,<ncol>i1)
      end
```

```
      subroutine openfl

c-------------------------------------------------------------
c-----subroutine to open input and output files using
c-----unit 9 for input and unit 10 for output
c-------------------------------------------------------------

      character*30 fname

      print*, 'enter the name of the input file'
      read(5,1000) fname
      open(unit=9,file=fname,status='old')

      print*, 'enter the name of the output file'
      read(5,1000) fname
      open(unit=10,file=fname,status='new')

 1000 format(a30)
      return
      end
```

```
      subroutine divide(x1,y1,x2,y2,unit,seg,nseg)

c-------------------------------------------------------------------------
c-----given a boundary edge, this subroutine divides that edge into
c-----segments falling within a single cell.
c-----
c-----PARAMETERS:
c-----the endpoints of the edge are given by (X1,Y1) and (X2,Y2).
c-----SEG is an array of points breaking the edge into segments with
c-----the x value in column 1, and the y value in column 2.  NSEG
c-----is the number of points in the array.
c-----parameters which are changed: SEG,NSEG.
c-------------------------------------------------------------------------

      include 'param.for'

      real x1,x2,y1,y2,unit,seg(maxseg,2)
      integer nseg
      integer dir,col

c-----find crossings of edge with grid lines
      seg(1,1) = x1
      seg(1,2) = y1
      nseg = 1
      call grid(unit,x1,y1,x2,y2,1,2,seg,nseg)
      call grid(unit,y1,x1,y2,x2,2,1,seg,nseg)
      nseg = nseg+1
      if (nseg .gt. maxseg) call error(2)
      seg(nseg,1) = x2
      seg(nseg,2) = y2

c-----sort segments in proper direction
      dir = 1
      col = 1
      if (x1 .gt. x2) dir = -1
      if (x1 .ne. x2) go to 50
      col = 2
      if (y1 .gt. y2) dir = -1
      if (y1 .eq. y2) call error(7)
   50 continue
      call sort2(seg,nseg,col,dir)

      return
      end
```

```
      subroutine grid(unit,pt1a,pt1b,pt2a,pt2b,acol,bcol,seg,nseg)

c-----------------------------------------------------------------------
c-----given a line defined by 2 points, this subroutine returns all
c-----points where the line crosses grid lines in a specified direction,
c-----either parallel to the x axis or the y axis. if the initial point
c-----of the line falls on a grid line, this is not placed in the array,
c-----but if the terminal point falls on a grid line, it is included.
c-----
c-----PARAMETERS:
c-----the spacing between grid lines is designated by UNIT.  "A" refers
c-----to the direction in which grid line crossings will be generated.
c-----(x or y) and "B" to the remaining direction.  the given line is
c-----defined by the pairs of coordinates, PT1A, PT1B and PT2A, PT2B.
c-----if "A" refers to the x direction, these would be x1,y1 and x2,y2;
c-----if "A" refers to the y direction, these would be y1,x1 and y2,x2.
c-----the grid crossing points are appended in the array SEG, beginning
c-----at row number NSEG+1.  NSEG is updated to point to the last row
c-----used.  the array holds x coordinates in column 1 and y coordinates
c-----in column 2.  if "A" refers to the x direction, ACOL is 1 and BCOL
c-----is 2; if "A" refers to the y direction, ACOL is 2 and BCOL is 1.
c-----parameters which are changed: SEG, NSEG.
c-----------------------------------------------------------------------

      include 'param.for'

      real seg(maxseg,2),unit,pt1a,pt1b,pt2a,pt2b
      real incr,slope,intcpt,cross,temp1,temp2
      integer acol,bcol,ncross,nseg,k

c-----initialize grid increments, and slope and intercept of line
      if ((pt2a-pt1a) .eq. 0) go to 30
      incr = unit
      slope = (pt2b-pt1b)/(pt2a-pt1a)
      intcpt = pt2b-slope*pt2a

c-----find number of crossings with grid lines and initial crossing

c--------increasing direction of line (terminal point > initial point)
         ncross = iabs(int(pt2a/unit)-int(pt1a/unit))
         cross = int(pt1a/unit)*unit
         if (pt1a .le. pt2a)  go to 10

c--------decreasing direction of line (initial point > terminal point)
         incr = -incr
         temp1 = pt1a
         temp2 = pt2a
         if (mod(pt1a,unit) .ne. 0) temp1 = pt1a+unit
         if (mod(pt2a,unit) .ne. 0) temp2 = pt2a+unit
         ncross = iabs(int(temp2/unit)-int(temp1/unit))
         cross = int(temp1/unit)*unit
```

```
c-----generate crossings by adding increments to initial crossing
   10 continue
      if (ncross .eq. 0) go to 30
      do 20 k=1,ncross
         nseg = nseg+1
         if (nseg .gt. maxseg) call error(2)
         cross = cross+incr
         seg(nseg,acol) = cross
         seg(nseg,bcol) = intcpt+slope*seg(nseg,acol)
   20 continue
   30 continue
      return
      end
```

```
      subroutine sort2(arr,n,col,dir)

c-----------------------------------------------------------------------
c-----this is a straight insertion sort on a two dimensional array.  the
c-----sorting algorithm is taken from knuth, vol. 3, p. 81.
c-----
c-----PARAMETERS:
c-----the input parameter, COL, specifies which of the two columns of
c-----the array are to be used as the key for sorting.  the parameter,
c-----DIR, specifies whether the rows of the array are to be sorted in
c-----ascending order (DIR>0) or descending order (DIR<0).  the
c-----parameter ARR is the array and N the number of rows in the array.
c-----parameters which are changed: ARR.
c-----------------------------------------------------------------------

      include 'param.for'

      real arr(maxseg,2),hold(2),key
      integer n,col,dir,j,k

c-----pull out each element after the first to test it
      do 50 j=2,n
         hold(1) = arr(j,1)
         hold(2) = arr(j,2)
         key = hold(col)
         k = j-1

c--------test the element against each preceding element to find its
c---------correct position
   20    continue
            if ((dir .gt. 0) .and. (key .ge. arr(k,col))) go to 30
            if ((dir .lt. 0) .and. (key .le. arr(k,col))) go to 30
            arr(k+1,1) = arr(k,1)
            arr(k+1,2) = arr(k,2)
            k = k-1
         if (k .gt. 0) go to 20

c--------place the element in its correct position
   30    continue
         arr(k+1,1) = hold(1)
         arr(k+1,2) = hold(2)
   50 continue
      return
      end
```

```
      subroutine update(right,left,unit,x0,y0,seg,nseg,
     +                  cell,cut,ncut,prop,nprop,scell)

c-------------------------------------------------------------------
c-----this subroutine looks at each segment of an edge, computes the
c-----areas of the region on either side of the segment within the cell,
c-----computes information for determining the propagation region, and
c-----then updates the CELL, CUT, and PROP arrays appropriately.
c-----
c-----PARAMETERS:
c-----parameters which are changed: CELL, CUT, NCUT, PROP, NPROP, SCELL.
c-------------------------------------------------------------------

      common accu
      include 'param.for'

      integer cell(maxrow,maxcol),scell(maxcut),nseg,ncut,nprop
      integer right,left
      real seg(maxseg,2),cut(maxcut,dimcut),prop(maxprp,3),unit
      real x1,y1,x2,y2
      integer i,j,p1,p2,region
      real intgrl,x,y,angle
      logical first,swall

c-----find next segment which needs to be considered
      k = 1
      x1 = seg(k,1)
      y1 = seg(k,2)
   10 continue
         k = k+1
         if (k .gt. nseg) go to 120
         if ((seg(k,1) .eq. seg(k-1,1))  .and.
     +       (seg(k,2) .eq. seg(k-1,2))) go to 10
      x2 = seg(k,1)
      y2 = seg(k,2)
      err = accu*k
      i = int(((x1+x2)/2)/unit)+1
      j = int(((y1+y2)/2)/unit)+1
      if ((abs(x2-x1) .le. accu) .and. (abs((i-1)*unit-x2) .le. err))
     +   go to 100
      if ((abs(y2-y1) .le. accu) .and. (abs((i-1)*unit-y2) .le. err))
     +   swall = .true.

c-----update CUT array
      if (cell(i,j) .ne. 0) go to 20
         ncut = ncut+1
         if (ncut .gt. maxcut) call error(4)
         cell(i,j) = ncut
   20 continue
      p1 = cell(i,j)
      call intgrt(x1,y1,x2,y2,i,j,unit,intgrl)
      call reginc(intgrl,right,cut,p1)
      intgrl = 1-intgrl
      call reginc(intgrl,left,cut,p1)
```

```
c-----check if cell is propagating
      x = x1
      y = y1
      region = right
      if (swall .eq. .true.) scell(p1) = left
      if (x1 .lt. x2) go to 40
      x = x2
      y = y2
      region = left
      if (swall .eq. .true.) scell(p1) = right
   40 continue
c     print, k,x,(mod(x,unit)),err
      if (abs((i-1)*unit-x) .gt. err) go to 100
c     print, k,x,(mod(x,unit)),err

c--------if propagating, update PROP array
         first = .false.
         if (cut(p1,1) .ne. 0) go to 60
            nprop = nprop+1
            if (nprop .gt. maxprp) call error(5)
            cut(p1,1) = nprop
            first = .true.
   60    continue
         p2 = cut(p1,1)
         if ((y .gt. prop(p2,1)) .and. (first .eq. .false.)) go to 100

c-----------initialize or update the low crossing, angle, and region
            angle = (y2-y1)/(x2-x1)
            if ((first .eq. .true.) .or.
     +          (y .lt. prop(p2,1)) .or.
     +          ((y .eq. prop(p2,1)) .and. (angle .lt. prop(p2,2))) )
     +         prop(p2,2) = angle
            prop(p2,1) = y
            prop(p2,3) = region

c-----reinitialize for next segment
  100 continue
      x1 = x2
      y1 = y2
      go to 10

  120 continue
      return
      end
```

```
      subroutine intgrt (x1,y1,x2,y2,i,j,unit,intgrl)

c-------------------------------------------------------------------------
c-----this subroutine computes the integral under a line segment
c-----crossing a cell.  the integral is computed assuming
c-----a unit cell of area 1, so the value of the integral is
c-----always less than or equal to 1.  a segment crossing the
c-----cell from right to left produces a negative integral.
c-----
c-----PARAMETERS:
c-----the segment is defined by the coordinates (X1,Y1)
c-----and (X2,Y2).  the cell is in the Ith column and Jth row of
c-----the grid (the lower left corner of cell 1,1 is the point(0,0)).
c-----the cell size is defined by UNIT.
c-----parameter which are changed: INTGRL.
c-------------------------------------------------------------------------

      real x1,y1,x2,y2,xx1,yy1,xx2,yy2,intgrl
      integer i,j

c-----adjust to unit cell
      xx1 = x1-(i-1)*unit
      yy1 = y1-(j-1)*unit
      xx2 = x2-(i-1)*unit
      yy2 = y2-(j-1)*unit

c-----compute integral
      intgrl = (xx2-xx1)*(yy2+yy1)/2

c-----check for north wall intersection
      if ((yy1 .eq. 1) .and. (yy2 .eq. 1)) call error(8)

c--------positive direction
         if ((yy1 .eq. 1) .and. (xx1 .le. xx2)) intgrl = intgrl+xx1
         if ((yy2 .eq. 1) .and. (xx2 .ge. xx1)) intgrl = intgrl+(1-xx2)

c--------negative direction
         if ((yy2 .eq. 1) .and. (xx2 .lt. xx1)) intgrl = intgrl-xx2
         if ((yy1 .eq. 1) .and. (xx1 .gt. xx2)) intgrl = intgrl-(1-xx1)

      return
      end
```

```
      subroutine reginc(intgrl,region,cut,ptr)

c-------------------------------------------------------------------------
c-----this subroutine searches across a given row in the CUT
c-----array for a given region, then increments the area of
c-----that region according to the value of INTGRL.  the area is
c-----kept >= 0 and <1 by adding 1 to negative numbers and taking
c-----the area modulo 1
c-----
c-----PARAMETERS:
c-----parameters which are changed: CUT
c-------------------------------------------------------------------------

      include 'param.for'

      real cut(maxcut,dimcut),intgrl
      integer region,ptr
      real area
      integer ireg,regck

c------find position of this region within the row of the CUT array
      ireg = 1
   20 continue
         ireg = ireg+1
         if (ireg .gt. dimcut) go to 50
         regck = int(cut(ptr,ireg))
      if ((regck .ne. region) .and. (regck .ne. 0)) go to 20

c-----increment area for this region in this cell
      area = cut(ptr,ireg)+intgrl
   30 continue
      area = mod(area,1.0)
      if (area .lt. 0.0) area = 1+area
      if ((area .lt. 0.0) .or. (area .ge. 1.0)) go to 30
      cut(ptr,ireg) = region+area
      go to 100

c-----error condition
   50 continue
      call error(6)

  100 continue
      return
      end
```

```
      subroutine assign(cell,cut,prop,scell,ncol,nrow)

c-----------------------------------------------------------------------
c-----this subroutine looks at each cell in the grid, assigning it
c-----a value according to which region covers the largest part of the
c-----cell.  if it is cut by one or more boundaries, the region
c-----covering the largest area within the cell is determined.  if the
c-----cell has a boundary intersecting the west wall, it propagates
c-----the value of its southwest corner through all cells south of it
c-----until another propagatng cell is reached.  if a cell is cut,
c-----but all regions have an area of 0.0, this indicates the cell is
c-----cut only by a boundary on the south wall and the value of the
c-----cell has been stored in the array SCELL.  NCOL and NROW give the
c-----actual dimensions of the CELL array.
c-----
c-----PARAMETERS:
c-----all of the parameters store information which is needed for
c-----assigning values to the cells.
c-----parameters which are changed: CELL.
c-----------------------------------------------------------------------

      include 'param.for'

      integer cell(maxrow,maxcol),scell(maxcut),ncol,nrow
      real cut(maxcut,dimcut),prop(maxprp,3)
      integer p1,p2,p3,i,j,k,jj,hi

c-----loop on each cell
      do 90 i=1,ncol
         do 70 j=1,nrow

c-----------if the cell is not cut go to the next cell
            if (cell(i,j) .eq. 0) go to 50

c-----------find the region covering the highest percentage of the cell's area
            p1 = cell(i,j)
            hi = 1
            do 10 k=1,maxreg
               if (mod(cut(p1,k+1),1.0) .gt. mod(cut(p1,hi),1.0))
     +            hi = k+1
   10       continue
            cell(i,j) = int(cut(p1,hi))
            if (hi .eq. 1) cell(i,j) = scell(p1)
            if (cut(p1,1) .eq. 0) go to 50

c-----------propagate through adjoining cells if necessary
            p2 = cut(p1,1)
            jj = j
   30       continue
               jj = jj-1
               p3 = cell(i,jj)
c--------------if the cell is cut and propagating, stop propagation
               if ((p3 .ne. 0) .and. (cut(p3,1) .ne. 0)) go to 50
                  cell(i,jj) = prop(p2,3)
            if (jj .gt. 1) go to 30

   50       continue
   70    continue
   90 continue
      return
      end
```

```
      subroutine error(n)

      write(6,2000) n
      stop

2000  format(1x,'ERROR - ',i2)
      end
```

```
c------'param.for' used with include statement to set parameters
      parameter maxseg=500
      parameter maxrow=600,maxcol=600,maxcell=360000
      parameter maxcut=36000
      parameter maxprp=36000
      parameter maxreg=6,dimcut=7
```