12

Data Structure Transformations

12.1 WHY TRANSFORM BETWEEN STRUCTURES?

In virtually all mapping applications it becomes necessary to convert from one cartographic data structure to another. The ability to perform these object-to-object transformations often is the single most critical determinant of a mapping system's flexibility. Why is this the case? A number of reasons dictate that the mapping process involve data structure conversions, and these are related to data input and geocoding, data storage and representation, the suitability of particular structures for different analytical and modeling demands, and finally the demands of a particular symbolization transformation.

Data input has already been noted as a primary determinant of data structure. Particular data capture devices, such as scanners and semiautomatic digitizers, generate data in a specific form. Scanners, for example, generate grids, and semiautomated digitizers produce strings of (x, y) coordinates. Geocoding stamps a particular coordinate system, resolution, and map projection on the data. In virtually every case, the cartographer will find that the available digital cartographic data are in the wrong structure for the required type of cartographic object, are on the wrong map projection, or have the wrong resolution for mapping or that the map rectangle needs to be rotated, scaled, or translated to produce the map.

Analytical and computer cartography, unlike many other disciplines, have available large reserves of common, generically digitized cartographic data. As a result, cartographers must change data structures simply to move the data into the correct geographic extent and from an input format into the format required by a particular piece of mapping software.

In Chapter 5, consideration was given to the different mechanisms by which digital cartographic data can be stored within a computer's memory and to how different means of data representation can save storage or improve data accessibility. Because one of the distinguishing characteristics of cartographic and geographic data is sheer volume, the conversion of data between structures, or between representations of a single structure, can save considerable storage space and processing time. The time and space limitations become more apparent as cartographic software moves from larger to smaller computers.

Although the amount of RAM and disk storage available to microcomputers has increased and mass storage technology is making significant breakthroughs in volume capabilities, processing the millions of data points necessary for high-accuracy cartography really strains the microcomputer. Precision or scale are usually sacrificed, with cartographically unacceptable results. Transforming data between cartographic data structures means that the application can optimize storage and processing time as is appropriate. Often the best data structure for a map depends on the demands placed on the data for analytical or display purposes. Good cartographic software does not force data into a single structure, but retains the option to flip between structures on demand.

Analysis and modeling also require different data structures. As an example, the skeleton or medial axis transform of a polygon can be performed in both grid or polygon entity-by-entity mode. In entity-by-entity mode, the locus of the largest enclosed circles must be traced through the polygon. In grid mode, the polygons are simply eroded away step by step from the edges while connectivity is maintained, until the medial axis is formed. Each operation is fairly fast. Inverting the transformation in grid mode is simple. Rebuilding the polygon from the medial axis, however, is difficult in entity-by-entity mode. The grid data structure is suitable for modeling and analytical operations such as Fourier and principal component analysis, filtering, and edge detection.

The TIN structure is good for modeling overland and stream hydrology and for simulating erosion. Entity-by-entity definitions are good for high-precision output, with multiple-weight lines and elaborate shading. Continuous patterns are more suited to the grid. The relative advantages and disadvantages of the various structures are many and even depend on the implementation characteristics, such as language, computer, and operating system.

For symbolization, actually producing the map, again certain structures meet different sets of cartographic demands, which implies that different types of map, different map scales, and so forth, all have their different optimal cartographic data structure. Often, the characteristics of the output device determine the best data structure. For example, raster devices favor grid and quad tree structures, while plotters, laser-jet printers, and automatic scribers require data in line or polygon entity-by-entity format.

The type of representation is also important. Choropleth maps can be produced in any structure, but hill-shading and perspective views are best using grids or TINs. As far as symbolization is concerned, nowhere is the influence of data structure more obvious than in map text labeling. Vector displays usually use the Hershey fonts and are capable of some quite attractive precision lettering.

Raster devices use bit-mapped graphics and as such produce blocky text (which looks worse with enlargement). In addition, the bit-mapped graphics are restrained by their angle on the screen. Just as many systems now support both raster and vector, the now common PostScript graphics merge vector draw commands with the raster capabilities of laser-jet printers. Some paint programs support bit-mapped (raster) or “object” (vector) modes within a single structure. It is with the text, again, that the differences are most apparent, especially with enlargement.

Clearly, there are many reasons to transform digital cartographic data between data structures. As such, we move from one type of cartographic object to another. Changing data structure can result in a loss of spatial information. These data structure transformations may not be fully invertible. Information loss can come as the result of changing scale, termed here *resampling*, as a direct consequence of changing data structure, or as a consequence of the data structure transformational process or the algorithm itself. The study of data structure transformations, and of course their perfection as a consequence of their study, is an important goal for analytical cartography. In this chapter we will first consider scale or resampling transformations, then the specific transformation between vector and raster data structures. A classification of cartographic data structure transformations and their inverses will then be considered. To conclude, we will discuss the role of error in data structure transformation.

12.2 GENERALIZATION TRANSFORMATIONS

Generalization transformations are those in which the scale or equivalent scale of cartographic objects is changed. Cartographic generalization is usually undertaken for one of two reasons: first, so that a set of cartographic objects can be symbolized or used analytically at a scale different (usually smaller) from that at which geocoding took place; and second, to generalize a map either for clarity of symbolization or for the reduction of the data set size. It is important to realize that this transformation applies only to the map data, rather than to the attribute data involved.

At several stages in the discussion of cartographic data structures it has been necessary to consider separately point, line, area, and volume data. Resampling transformations are within their own dimensional type, such as point to point, rather than between types, such as area to point.

12.2.1 Point-to-Point Transformations

A point-to-point transformation involves selecting one point to represent multiple points. One such point is the centroid, discussed in Chapter 11. The centroid, perhaps with a scatter parameter, is a summary of the locational characteristics of a point distribution. A primary use for the centroid is in the conversion of irregularly spaced data, or data collected throughout a set of regions, into a continuous representation such as a grid. Population density data, for example, are often computed for census tracts in a city and converted to a grid by interpolation from point centroids selected in some manner. The point-to-grid transformation will be covered in detail in Chapter 13.

Many grid-to-grid transformations are, in fact, point to point, as are changes between map projections and coordinate systems. A transformational process that uses as input a global data set of latitudes, longitudes, and elevations, usually organized at regular intervals of degrees, minutes, or seconds, does not produce a regular grid after a map projection transformation. After the transformation into a map projection, the data must be resampled into a grid based on the axes of the new coordinate system. Because point-to-point transformations are so common, they are usually thought of as part of the geocoding process. They are, however, important examples of cartographic transformations.

12.2.2 Line-to-Line Transformations

Line-to-line transformations have received considerable attention. A general statement of the problem would be to take a cartographic line, as represented as an object in a particular data structure, and to reduce the number of elements required to store the line in such a way that the line symbolization produced carries the spatial properties of the line to the map reader. Line character is important to preserve, yet difficult to define, and involves a complex set of related approaches to generalization (Buttenfield, 1985).

Function 12.1 performs the *n*th point retention generalization on a string in the STRING data structure from Chapter 5. The function `read_strings()` was introduced in Chapter 7. These functions are used with the main program in Function 12.2. The GKS function `draw_strings()` shown in Function 12.3 is used to draw the results of the generalization, and produced the output in Figure 12.1.

Function 12.1

```
/* generalize a line by N pt retention kcc 6-93 funct 12.1 *
#include “cart_obj.h”
#include “extern.h”
void generalize(number_of_strings)
int number_of_strings;
{
int n, k = 0, str, pt;
STRING gen_line;
printf(“Enter n, where every n-th point will be selected:”)
scanf(“%d%*1c”, &n);
for (str = 0; str < number_of_strings; str++) {
  for (pt = 0; pt < line[str].number_of_points; pt++) {
    if ((pt % n) == 0) {
       gen_line.point[k].x = Line[str].Point[pt].x;
       gen_line.point[k].y = Line[str].Point[pt].y;
       k++;
    }}
/* Add the last point */
  gen_line.point[k].x =
  Line[str].Point[Line[str].number_of_points - 1].x;
  gen_line.Point[k].y =
  line[str].point[Line[str].number_of_points - 1].y;
  k++;gen_line.number_of_points = k; k = 0;
  for (pt = 1; pt < gen_line.number_of_points; pt++) {
    Line[str].Point[pt].x = gen_line.Point[pt].x;
    Line[str].Point[pt].y = gen_line.Point[pt].y; }
  Line[str].number_of_points = gen_line.number_of_points;
  } return;
}
```

Function 12.2

```
/*
/* Main program for generalize & draw strings */
* kcc 6-93 function 12.02 */
#include “cart_obj.h”
extern STRING Line[MAXSTRINGS];
main()
{
 int read_strings(), number_of_strings;
 void generalize(), draw_strings();
 number_of_strings = read_strings();
 (void) generalize(number_of_strings);
 (void) draw_strings(number_of_strings);
}
```

Definitive treatments of the line generalization problem are available elsewhere (Buttenfield and McMaster, 1993; McMaster and Shea, 1992). The simplest techniques for reducing the number of points necessary to represent a line are *n*th-point retention and equidistant resampling. In the first case, the nodes at the end of a vector line representation in entity-by-entity mode are retained as pivots, while for the remainder of the line only every *n*th point is kept. Similarly, for equidistant resampling, the line is followed by a distance tracking algorithm, and a point is retained at multiples of some key distance along the line.

A number of techniques treat a line as adequately represented by the original data points selected for its representation as a cartographic object, and then fit a smooth curve through the points to make symbolization more attractive or easier to interpret. Splines, polynomials, and Bezier curves are mathematical functions that have been used to perform smoothing. Buttenfield (1985) provided a list of references to the algorithms behind these functions. Many automated contouring packages use these methods to smooth the lines fed though a grid during contouring.

Among the various line generalization methods, one of the most long-standing is the Douglas-Peucker method (Douglas and Peucker, 1973). This method uses the worst-case generalization of a line as the starting approximation,that is, the line segment formed by simply connecting the endpoints. Each point along the line has an orthogonal distance from the line, that can be computed by simple trigonometry. The Douglas-Peucker algorithm selects the point with the largest distance, breaks the line at this point, and then recursively applies this criterion to the resulting segments.

Recursion continues until either only a minimum number of points remain in the string segment, or a tolerance level is reached, perhaps a proportion of the initial orthogonal distance (Figure 12.2).

A number of researchers have suggested and used different criteria for evaluating line generalization methods both subjectively and objectively. McMaster (1986) used a quantitative measure based on the areas of the triangles formed between triplets of points in a line and in its generalization. Muller (1986) has suggested as a criterion the ability of an

Function 12.3

```
/* draw a set of strings using GKS kcc 6-93 funct 12.03 */
#include <gks/ansicgks.h>
#include “cart_obj.h”
void draw_strings(number_of_strings)
  int number_of_strings;
{
Gchar *conn = NULL; Gchar *wstype = “postscript”;
Gws ws = 1; Gwlimit win; Gnlimit viewport;
int transformation = 1, i, j;
float aspect_ratio;
/* Open up GKS */ if (gopengks(stdout, GMEMORY)) exit();
/* Open the workstation */ gopenws(ws, conn, wstype);
/* Activate the workstation */ gactivatews(ws);
/* Find the bounding reactangle of the points */
win.xmin=line[0].point[0].x; win.xmax=line[0].point[0].x;
win.ymin=line[0].point[0].y; win.ymax=line[0].point[0].y;
for (i = 0; i < number_of_strings; i++) {
  for (j = 0; j < line[i].number_of_points; j++) {
if(Line[i].Point[j].x<win.xmin)win.xmin=Line[i].Point[j].x;
if(Line[i].Point[j].y<win.ymin)win.ymin=Line[i].Point[j].y;
if(Line[i].Point[j].x>win.xmax)win.xmax=Line[i].Point[j].x;
if(Line[i].Point[j].y win.ymax)win.ymax=Line[i].Point[j].y;
} }
  gsetwindow(transformation, &win);
  aspect_ratio = (win.ymax-win.ymin)/(win.xmax-win.xmin);
  viewport.xmin = 0.0;viewport.ymin = 0.0;
  if (aspect_ratio < 1.0) {/* Landscape */
    viewport.ymax = aspect_ratio;viewport.xmax = 1.0;
  } else {/* Portrait */
    viewport.xmax = aspect_ratio;viewport.ymax = 1.0;};
  gsetviewport(transformation, &viewport);
  gselnormtran(transformation);
  for (i = 0; i < number_of_strings; i++)
    gpolyline(Line[i].number_of_points, Line[i].point)
  gmessage(ws, “quits after 35 seconds”);sleep(35);
  /* Deactivate the workstation */gdeactivatews(ws);
  /* Close the workstation */ gclosews(ws);
  /* Close down GKS */ gclosegks();
}
```

North America *n = 2*

North America *n = 15*

North America *n = 45*

Figure 12.1 Line generalization by *n*-point retention.

algorithm to preserve the fractal dimension of the line. The fractal dimension, a value reflecting the degree of scale invariance of a line, seems related to line complexity.

Although line generalization by resampling reduces the number of points in the cartographic object representing the line or in its symbolization, several authors have devised methods to actually increase the number of points. Dutton (1981) proposed an algorithm that computes midpoints for segments and then moves the midpoints using the values of four controlling parameters. Dutton pointed out that fractalization allows lines to have features exaggerated and allows the introduction of smaller scale features with enlargement. The introduced features, being self-similar, have similar properties to the existing line.

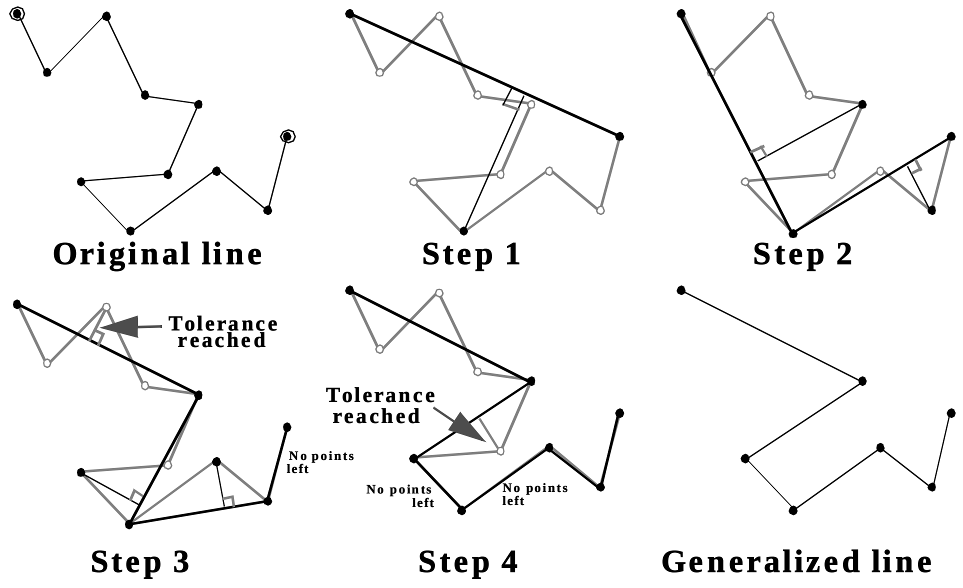

Figure 12.2 Douglas-Peucker line generalization.

12.2.3 Area-to-Area Transformations

Resampling areas to yield areas is a resampling transformation of considerable value to analytical and computer cartography and to GIS. A general statement of the goal is to merge multiple data sets into a set of regions such that the merged data allow comparison between maps. Usually, at least for sets of geographic areas, this means computing a set of greatest common geographic units, or areas that need no longer be partitioned to represent spatial data as cartographic objects.

As a practical expression, consider nonnested regions such as census tracts, school districts or police districts. A crime study may wish to collate population characteristics by police district, only to find that the boundaries do not coincide with census tracts. Similarly, when data have to be compared between different time periods, invariably change in the geographic extent of regions has taken place. How analytical and computer cartography deals with this problem is largely a function of the data structure used to store the various map layers and the data structure in which the map comparison is to be conducted. Clearly, for two maps in two different structures there are two strategies.

First, we can transform both maps into a common data structure that allows comparison directly. Often this is done by converting to a set of topological or entity by entity most common geographic units or to a grid. In the case of the grid, we actually perform

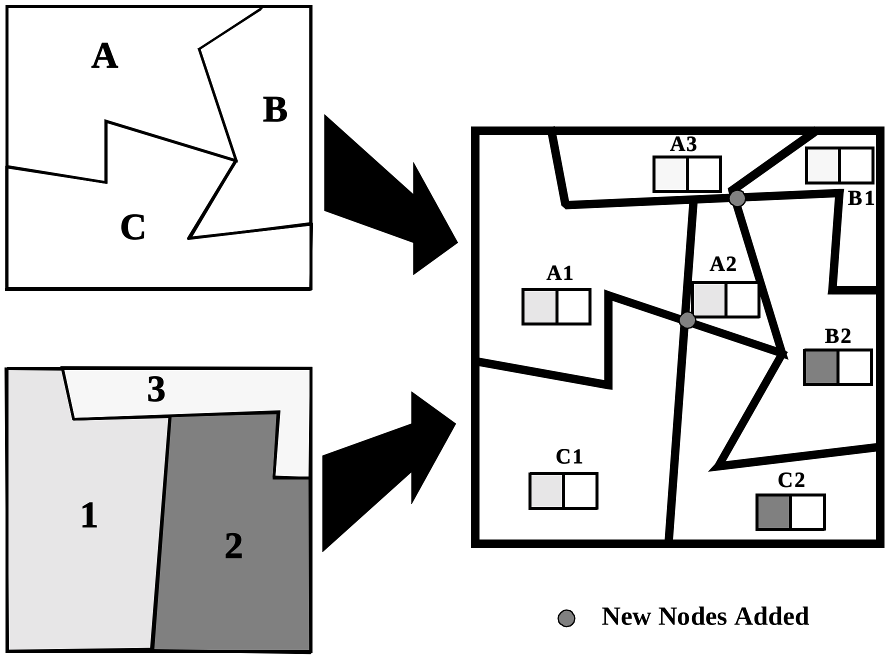

Figure 12.3 The polygon overlay problem.

an area-to-point transformation, so that the points coincide for two or more sets of regions. The second strategy is to convert one of the maps into the structure of the other. Thus a polygon entity-by-entity map can be gridded to compare with another grid. As the number of layers increases, the first of these strategies becomes more suitable, especially when inverse transformations are required.

Overlaying two maps to generate a set of most common geographic units in vector mode is not a trivial task (Goodchild, 1978). Central to the problem is the processing of line data to determine intersection points between overlapping polygons so that they can be added to both polygons in the correct place. The points then become nodes in the network of lines that constitute the most common geographic units. A summary of the contributions of computational geometry to these problems is the book by Preparata and Shamos (1985). By the late 1970s, polygon overlay was available within a number of GISs and automated mapping systems (Franklin and Wu, 1987), especially within theWhirlpool module of Odyssey (White, 1978).

The procedure for polygon overlay, illustrated in Figure 12.3, consists of three separate subproblems. First, the intersection points between lines must be found. Usually, this is done using a line segment intersection routine such as that shown in Chapter 10, recursively for all pairs of line segments in the two chains. A way to save considerable computation time is first to check the bounding rectangles of the two chains for overlap. If there is no overlap between bounding rectangles, then there is no need to test each line segment for intersection. When intersections are found, the chains must be split, and the intersection point must be labeled as a node and included as the endpoint of the new subdivided chains. Many mapping systems compute and save the map coordinates of the bounding rectangles of lines and areas automatically on data entry for this purpose.

In the next stage, the partitioned chains are reassembled into the new set of polygons that make up the most common geographic units. Finally, each polygon must be relabeled, either with a new set of sequential or other labels or with labels that record the partitioning history. Particularly difficult to process are polygons that cross and recross boundaries, and islands. A FORTRAN implementation of polygon overlay was published by Baxter (1976). Numerous improvements and refinements have been reported over the last few years, and the polygon overlay problem remains the topic of considerable work in analytical cartography.

12.2.4 Volume-to-Volume Transformations

Volumetric representation is rare in cartography because there are few truly three-dimensional means by which to symbolize cartographic objects. The usual volume-to-volume resampling transformations are changes in the grid spacing associated with grid data structures or changes in a TIN surface representation. Within a TIN, rarely is the set of points originally used to depict the surface changed, because the points are locations of real data observations and the data structure is fairly compact in the first place. When TINs are compared with data in other formats, such as grids, either the TIN is interpolated to a grid, or the data are points and are allocated to TIN triangles using a point-in-polygon test.

Grids are, however, frequently resampled to change size, to retrieve a subset, or to change the grid spacing. Invariably, unless the change is simple, such as taking every other row and column, the new grid is generated by computing the location of the new grid intersections in the coordinate space of the map and then by interpolation from neighboring grid cells. Usually, the four neighbors and a simple unweighted average are sufficient for the resampling.

12.3 VECTOR TO RASTER DATA AND BACK AGAIN

We have seen in the previous chapters that data structures for analytical and computer cartography often reflect the demands of the hardware and software they support. Just as display devices can be categorized as vector or raster, so can the cartographic data structures used by software be characterized in this way. In the past, a broad division was made between raster and vector data structures. Each structure has its advantages and disadvantages and has also had its proponents. Since the development of algorithms for efficient data structure transformation, however, the vector-versus-raster debate has become largely irrelevant.

The reasons for transformation are many, and in many automated mapping systems the conversion of raster to vector data, or vice versa, is a major consumer of computer time. The raster structure is a grid format and stores a map as individual pixels on lines similar to a television picture and a satellite image. The vector structure stores maps line by line, feature by feature. The vector format is most suitable for storing as objects and symbolizing the cartographic entities familiar to human thought (Peuquet, 1979). Most cartographic objects represent boundaries, rivers, coastlines, railroads—distinct lines or groups of lines. On the other hand, raster devices operate reliably and flexibly and have advanced data handling capability. For data storage, the manipulation of data in raster mode is quite straightforward.

The repertoire of vector algorithms for cartographic transformations was more quickly developed than for raster algorithms (Peuquet, 1979). This difference, however, rapidly diminished during the 1980s. Normally, spatial data stored in vector formats take less storage space than does their raster mode equivalent. The graphic output devices in vector form are relatively accurate and distinct, but the speed of output depends on the length of lines on a map.

The greatest need for transformation is that automatic scanners produce raster data, but vector digitizing requires human control. Therefore, to capture large data sets for cartography systematically , the conversion from a raster data structure to a vector form becomes necessary.

12.3.1 Vector-to-Raster Transformations

The conversion of vector data to raster or grid form is usually termed *rasterization*. Rasterization involves four distinct steps. First, the vector data must be read into the computer for processing. The processing can take place one polygon or line at a time or simultaneously for a whole map. Second, the appropriate scales and map transformations should be applied. Normally, the map projection transformation and any resampling transformation will have taken place before this step is reached; for example, a world map may have been transformed to a Mercator projection, clipped at 84 degrees north and 80 degrees south, and the map resampled using the Douglas-Peucker method.

We can now assume that the alignment of the grid will be to the axes of the coordinates used by the input data. The only remaining decision is how many pixels the map will be converted into, and this is determined either by the desired map resolution or the number of rows and columns. Rectangular pixels are sometimes desired, especially to meet the demands of a particular display device, algorithm, or application. This step establishes the geometry of the rasterization. The third stage depends upon technique. In many cases, the grid is partitioned as an array in the computer's memory, and as the points, lines, and polygons are rasterized, the zeros stored initially are changed to ones, or to an index number for the line or polygon.

For very large arrays a second method is used. In this method, the locations of pixels with value one, with their indices if necessary, are stored, either as a file or internally in `[row, column, index]` format. These data are then sorted by row and column, and an array is constructed by reading the sorted data, filling rows and columns either with the stored value or with a zero. An option is to fill a polygon with an index value. The final step is to store the array in the required format, such as with run-length encoding or as a bit map.

Several steps can speed the rasterization process. First, if the distance between two points in the original coordinate system is smaller than half the grid spacing, the second point can be eliminated. This reduces unnecessary data and saves processing time. A careful choice of sorting technique can also make a large difference in the processing time used.

Another step is to process the vector data, computing and storing linear equations of each two points. The purpose of the step is to compute in advance the linear equation constants for each line segment. These can be stored as real numbers, although Bresenham (1965) has shown that only integers are necessary. One problem is the special case of vertical lines, that have an infinite gradient and tend to be common on maps. In this case, the programmer can store the fact that the line is vertical.

The rasterization is now complete, yet for symbolization, the line often needs to be thickened. Simple thickening can parse the whole array and change from zero to one (not line to line) any pixel that borders a line pixel. Unfortunately, this tends to fill in the fine details of wiggly lines. Another technique, which in some display devices can be performed in hardware, is to fill in these neighboring pixels with values to be displayed at a lower light intensity than the line pixels themselves, a technique known as anti-aliasing. Anti-aliasing removes some of the effect of stair-stepping along diagonal lines in raster mode, an effect usually called the “jaggies.”

Alternatively, an algorithm that has produced thick lines will be thinned so that all raster lines are of a constant width of one pixel. The degree of fatness of the lines depends on the character of the line, the orientation of the grid, the resolution of the grid, and the algorithm used. For example, in the case of a polygon being rasterized, we could assign an edge pixel to any two or more polygons depending on the algorithm in use. If edges of polygons are treated as lines, then a significant number of pixels may be edges, rather than taking on the attributes of a polygon. If the edge pixels are to be assigned to the polygons, we can use a “dominant” criterion (the polygon owning the majority of the pixel area takes the pixel), a “center point” criterion (the polygon covering the pixel center takes the pixel), or any of several other criteria. Line thinning after the fact can be dependent on the processing direction through the array and other factors.

The algorithm `rasterize` contained on the companion disk to this book probably uses the simplest approach. The earlier function `read_strings()` is used to read the North America data set. Function `extremes()` computes the bounding rectangle for the strings, useful for computing the grid characteristics.

Function `get_gridspecs()` prompts the user for two points defining the grid extents and the *x* and *y* grid spacing. This function computes and returns the sampling distance appropriate to the grid. Sampling at any distance greater than this runs the risk of missing cells along string segments. A loop over all strings first computes the straight line equation of the line segment, testing again for vertical segments, and then applies the sampling into the grid apparent in Figure 12.4. Finally, the grid is written as an ASCII array of ones or zeros. Display of these grid files are covered later in the book.

Output from the function is shown as Figure 12.5. The North America outline, unprojected into a map projection, was gridded at three different resolutions. Notice in this figure how the level of detail decreases and the number of fat lines increases as the cell size becomes larger. Also notice the tendency for lines to be rasterized toward the cardinal directions, especially when they are only slightly diagonal, a good example being the U.S–Canada border.

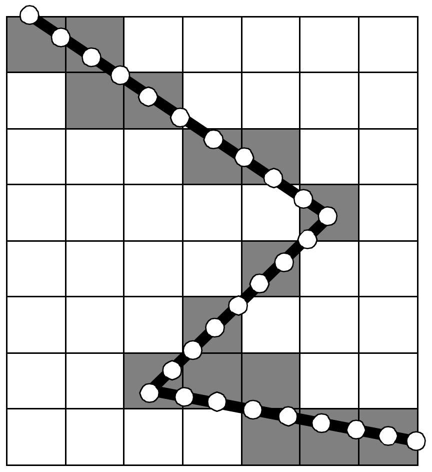

Figure 12.4 Algorithm `rasterize`. String is sampled along segments at half the grid resolution, and the resulting points are converted to integer grid indices. See the `Rasterize` directory of the companion disk.

12.3.2 Raster-to-Vector Transformations

Conversion of vector data to raster is comparatively simple compared with the inverse transformation. The demand for the raster-to-vector transformation is increasing as raster input devices such as scanners and remote sensing devices become more widespread. As a result, the raster-to-vector transformation is being built into automated mapping systems and GISs. This process is very CPU intensive and can yield topological and other errors in the resultant vector data set regardless of the quality of the input data. Analytical cartography has been slow to research issues related to this important transformation, and as a result, the published work is mostly in image processing. Peuquet (1981) noted the lack of efficient algorithms and poor computer coding in this area, a deficiency that has not been fully addressed.

Converting from raster to vector data involves three major steps. These stages, shown graphically in Figure 12.6, can be termed *skeletonization* or *line thinning*, *line extraction*, and *topological reconstruction.* Line thinning is necessary because vector lines are purely geometric, that is, they have zero width as cartographic objects yet have a width determined by the grid cell size when they appear in raster mode. The grid representation of the lines, therefore, must first be thinned so that the line consists merely of a one-cell-width sequence of connected pixels. This connectivity is usually in one of the eight major directions dictated by the eight-cell connectivity of grid cells.

The first stage, skeletonization, can be performed in one of three ways. Peuquet termed these *peeling*, *expanding*, and the *medial axis* methods. The medial axis method is the fastest, but it works on one line at a time and not on the whole grid simultaneously, and inconsistencies occur in very thick lines. Peeling and expanding methods are the inverse of each other. Peeling deals with systematically eroding the edges of lines, wheras expanding deals with expanding the space between lines.

A highly efficient peeling method is Pavlidis's asynchronous thinning algorithm (Pavlidis, 1982). This technique is not completely parallel in operation, however, and in order to maintain connectivity accepts lines with a width of more than one grid cell. Rosenfeld and Kak's thinning algorithm (Rosenfeld and Kak, 1981) is as fast as asynchronous thinning and results in a line of single-grid-cell width. This method works iteratively, at each pass deleting border grid cells that do not disconnect the local (three by three cell) neighborhood. The result of this first step is a single-cell-width line, but sometimes the thinning process introduces artifacts into the geometry of the lines.

Line extraction, the second stage, involves determining where individual lines begin and end in the thinned image, so that the lines can be rewritten as vectors connecting the endpoints in the correct sequence. Points are usually generated at the centers of the line grid cells, and long straight sequences can be eliminated to reduce the number of points in the line. Topological reconstruction is the process of generating the topological connectivity of the lines to rebuild a topological definition of the lines and polygons from the entity-by-entity objects that come from the previous two stages. As such, the third step is identical to the transformations from entity-by-entity to topological data structure discussed in the Section 12.4.

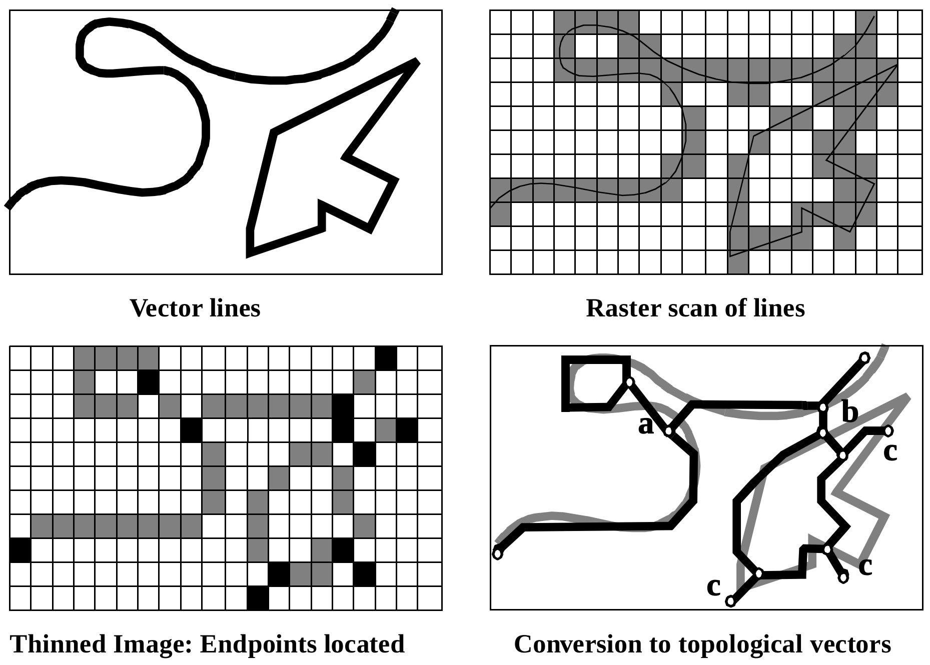
Vector lines
Raster scan of lines
Thinned Image: Endpoints located
Conversion to topological vectors
a
b
c
c
c

Figure 12.6 Stages and problems of raster-to-vector conversion.
Point a: collapsed loop. Point b: false link. Point c: collapsed spike.

Line extraction can be accomplished in one of two ways. Line following seeks a node that forms the beginning of a line and attempts to follow the line to another node. One of its disadvantages is that when a node is reached and the line terminated, no use is made other lines normally beginning at this point. The scan-line approach uses the same logic as line following, but it processes multiple lines simultaneously.

Once the lines have been extracted, they are usually written as vector lines, and any topological processing takes place in vector mode. The writing of vectors involves finding the absolute or world coordinate location of each grid cell. This can be accomplished by taking the grid cell row or column number, dividing by the number of rows or columns respectively, multiplying by the size of a grid cell in that direction, and adding the world coordinate of the lower left corner of the map. When the grid is not aligned with the coordinate system, the four values stored in the `Grid.corners[]` part of the GRID structure can be used to compute the affine transformation necessary to convert to world coordinates.

Errors associated with the conversion to a finite raster resolution and the use of the various algorithms can result in a significant number of errors, such as collapse lines, spikes, and false links (Figure 12.6). These errors depend on the resolution of the rasters and particularly on the thinning stage of conversion. Where cartographic lines are simple, such as contour lines, this may not be a significant problem. In other cases, such as along a contorted coastline or where contours run close together, the errors can be major and usually require operator intervention. In many systems that do automated line extraction, the conversion process will stop when an ambiguous topological junction is created and wait for the operator to guide the raster-to-vector converter.

When point data rather than vectors are processed, the coordinate transformation is the only one that needs to be applied. A special case is when the data to be processed are coverage polygons without boundaries. Data such as spectral classification and clustering of remote-sensing data , or data from existing raster mode GISs, are often in this format. An approach to finding the boundaries is to scan the grid from top to bottom and from side to side to reveal the boundaries

A simple algorithm to do this is illustrated in Figure 12.7. Alternatively, the grid cell can be filtered using an edge detecting filter, which emphasizes breaks in value. A disadvantage of this method is that one row and one column from each edge of the image are lost for every time the filter is applied. Clearly, when a grid cell is wholly within a polygon, the grid cell is assigned the index for that polygon.

This process is the exact inverse of the way in which polygons are created as grids from vector data. In the vector-to-raster section above, we considered only line rasterization. Points can be gridded simply as individual grid cells, but areas are different. When cells fall into two or more polygons, one way of making the assignment is to compute the area of the cell occupied by each polygon and to assign the cell to the polygon that occupies the most area.

Alternatively, the polygon boundaries can be processed as above and the indices for the polygon interior assigned by filling the bounded area within each polygon. In this case, the boundaries would remain as such and would not be considered part of the polygon. This distinction is made in the Spatial Data Transfer Standards between the ring defining a polygon and the polygon's interior area. For applications where input data are classified remotely sensed imagery, the lack of boundaries is normal, as also is a profusion of small and even single-cell clusters, which would make poor vector equivalents. These can be eliminated by filtering or by assigning small clusters to larger, neighboring clusters if certain criteria are met, such as diagonal connectivity. As remotely sensed data get better resolution, the connecting problem will diminish in significance, although it will remain for small-scale mapping, such as land-use coverage. The elimination of finer detail or "salt and pepper" will become more important with higher resolution.

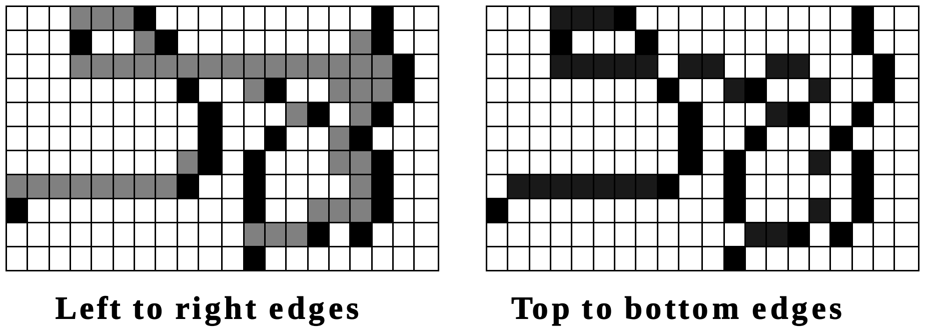

Figure 12.7 Sweep method for edge location.

12.4 DATA STRUCTURE TRANSFORMATIONS

Data structure transformations are usually necessary because of the demands of a particular mapping system and as such are pertinent to computer cartography. The need for efficient transformationsand the in-depth understanding of the cartographic implications of the transformations, however, are very much a part of analytical cartography. The multitude of data structure transformations performed during the mapping process fall into one of three types. Of these, *scaling transformations* are transformations of cartographic objects by their scale alone. Data structure transformations such as line generalization and grid resampling fall into this category. Resampling transformations, such as the rewriting of cartographic data in the same data structure at a different resolution, are the digital expression of the scale phenomenon. The study of such scale transformations has made important contributions to analytical cartography.

A second type of cartographic data structure transformation, *dimensional transformations,* involves a dimensional change. A dimension change is a change in the type of cartographic object itself rather than a data structure transformation. Such a transformation is a logical data compression. For example, a dimensional data structure change may be the selection of a point to represent an area.

Such a transformation involves considerable loss of data and information, yet gives the clarity or simplicity sometimes required during cartographic generalization. Such a transformation is invertible. We could, for example, generate Theissen polygons from the points to take the place of the original polygons, but the inverse transformation is an imperfect one and results in error. These types of transformations were considered in Chapter 11 as map data transformations,because they involve actual manipulation of the spatial properties of the cartographic objects.

A third type of data structure transformation, *structural transformations,* move cartographic objects between structures as part of the mapping process without much change of scale. For example, a gridded digital elevation model could be processed for significant points such as peaks, saddles, and pits, from which to extract and generate a TIN. The areal coverage is identical; essentially the same cartographic object, the terrain, is available for symbolization or analysis, the object's Euclidean dimension remains the same, yet the data structures are radically different.

The four major data structures we have covered can be classified into entity by entity, topological, TIN, and grid. The grid includes all raster-based structures such as quad trees, and the entity by entity covers some of the hybrid structures, such as Freeman codes. Given these four types of data structures, a matrix of transformations can be compiled that shows the 16 possible transformations (Figure 12.8). The leading diagonal of this matrix involves transformations without a change of data structure and as such these transformations must be *scaling*.

Of the remaining 12 cells in the matrix, four fall into the grid cell to entity by entity and topological structures and their inverses, which were considered above as raster-to-vector and vector-to-raster transformations. The remaining eight cells in the matrix are four transformations and their inverses. These are entity by entity to topological, entity by entity to TIN, topological to TIN, and TIN to grid.

	Entity by Entity	Topological	TIN	Grid
Entity by Entity	Scaling Dimensional	Structural	Structural	Vector to raster
Topological	Structural	Scaling Dimensional	Structural	Vector to raster
TIN	Structural	Structural	Scaling Dimensional	Structural
Grid	Raster to vector	Raster to vector	Structural	Scaling Dimensional

Figure 12.8 Possible data structure transformations.

The first of these transformations, entity by entity to topological, is the normal way by which topology is added during the geocoding process. When maps are digitized, the topology can either be entered explicitly, as in the DIME files, as separate records, or it can be extracted from the points, lines, and polygons as they are digitized. For strings, topological attributes are forward and reverse linkages and the labels of the neighboring polygons.

The linkages to other strings can be determined by storing separately the beginning and ending nodes of the strings, and then matching them against each other. Because manual digitizing results in small differences in the exact coordinate values at points, usually some tolerance is used, and matching points are given the same average location (snapping) (Figure 12.9).

Many digitizing programs maintain and check this information as each string is entered, and these programs prompt the user if an endpoint match cannot be found with the existing strings. The endpoint and topological information can be stored in a structure such as the `CHAIN`, introduced in Chapter 7.

Finding the names of the neighboring polygons is more difficult, and the usual method is either to prompt the user for the left and right polygon information when a new chain is accepted or to have the user digitize a set of label points, one per polygon. These label points can be stored separately, with the attribute data for the polygons, for example. When polygons need to be referenced, or when attributes are required in the symbolization transformation, the label points can be tested using a point-in-polygon test for inclusion within a group of chains. The relevant data structures are the `CHAIN`, the `AREA_CHAIN`, and the `NETWORK_CHAIN`.

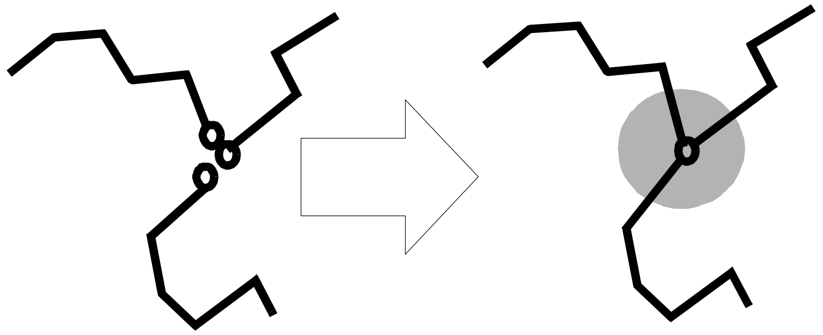

Figure 12.9 Node snapping in entity-to-entity to topological transformations.

The inverse of this transformation is the extraction of the simpler structures, such as-STRINGs, ARCs, and RINGs, from the topological structures. Because the to_node and from_node information is already available, this conversion is simply a rewriting of the information already contained within the structure. In some cases, chains would have to be written into the RINGs in opposite sequence or backwards, that is, from the end_point to the begin_point, so that the sequential nature of the RING is maintained. This is especially important if the motive for the transformation is to symbolize the polygon as a filled area under GKS or if the polygon area is to be computed.

The six remaining transformations all involve the TIN structure. They are topological to TIN, entity-by-entity to TIN, and TIN to grid and their inverses. To date, no real example of a topology-to-TIN transformation is available, although the inverse, in which the TIN is used to generate a stream, ridge line, or connected set of polygons representing visible or invisible areas, seems analytically valuable. This transformation is closer to a dimensional transformation, because the topological data structure is usually two-dimensional while the TIN is three-dimensional. The network, having no partial triangles involved, could be generated using links, with link order being downhill, for streams or ridge lines. For intervisibility problems, however, the splitting of triangles into polygons would be necessary, and simple or complex polygons would have to be formed to store the visible or invisible areas.

The transformation from entity by entity to TIN is also a dimensional transformation, because the only real example of this transformation is the transformation from point entity by entity to TIN, and vice versa. The forward transformation in this case is Delaunay triangulation, while the inverse is again simply a relisting of the entities contained in the TIN structure. Finally, the grid-to-TIN forward and reverse transformation is a true structural transformation, since neither dimensional change nor resampling takes place. This transformation can be accomplished in the forward case by selecting significant points, perhaps by filtering or special case searching, which with their elevations form the basis of the resultant TIN. The inverse transformation is accomplished by linear or other interpolation of elevations at grid cell locations from the TIN's triangles.

This three-type classification of the transformations possible between the four major cartographic data structures can be used to understand the interrelationship among the power, flexibility, efficiency, and storage size of data structures, and it helps determine when data structure transformations are appropriate within a mapping system. Analytical cartography can assist in this understanding by allowing the cartographer quantitatively and theoretically to model and predict the amount and distribution of error to be expected as a result of data structure conversions. The contention is that a cartographic entity is a complex phenomenon, and the means by which this entity is converted to a cartographic object as digital data within a cartographic data structure are both controllable and predictable. With correct understanding of these transformations, cartographers should be able to provide error models and reliability estimates along with maps that are the geographic equivalent of the statistician's normal curve and tests of significance.

12.5 THE ROLE OF ERROR

Much recent research in cartography has been centered on the problem of errors in digital cartographic databases. The focus of the problem is that because digital cartographic databases can be highly precise, many users of the maps produced from the databases also believe them to be accurate. In fact, the data stored as digital maps are often unbelievably faithful reproductions of each and every one of the human errors created in the making of papers maps.

Beard (1989) classified map errors into *source errors, use errors,* and *process errors*. Source errors are errors in the original map sources, in the digital cartographic source data, or in the data capture and geocoding processes. Use errors are caused by a lack of information about a mapping system, unexpected deviation from cartographic convention, the use of maps with a scale which is too small, a lack of timeliness, and a lack of user documentation. Beard proposed that increased attention be devoted to use error, especially because users of GIS are typically not always cartographers.

The Spatial Data Transfer Standard is designed to ensure that the source errors are revealed. The standards uses a "truth in labeling" approach, which also makes the provider of digital cartographic data responsible for providing, either internally or externally, documentation in the form of a quality report. The quality report should contain information on lineage of the data, on positional accuracy of the data, on the accuracy of the attributes associated with the data, and on the completeness of the data. SDTS assists in the task of reducing error by providing a standard set of entity and attribute terms for cartographic features that, if used, reduce ambiguities about digital cartographic data. Also, the standards establish the nomenclature and definitions for digital spatial data transfer, so that data acquired from government or other sources will be in a consistent structure, as documented in the standards.

The final type of error in Beard's classification is process error, the error resulting from the digital conversion, scale change, projection change, or the type of symbolization used. These errors are those attributable to one or more of the cartographic transformations described in this and the preceding two chapters. This type of transformational error clearly relates to the types of transformation discussed above. As we have seen, cartographic transformations can pertain to the map or the data structure, and can involve changes in resolution and changes in object dimension. Three types of process error can therefore occur.

First, errors can occur in the geometry of the map; that is, features can become mislocated in the three-dimensional geometry of the world. These mislocations are sometimes unknown, but can be the result of a controlled distortion, such as changing the map projection or statistical space fitting.

Second, errors can be due to scale change. An ideal situation would be to have a single, very high resolution database from which all others are generated using some objective generalization function (Beard, 1987). This is rarely the case, and maps will continue to be digitized from a large number of sources at different scales. Again, some errors are unknown, such as the removal of islands as they become smaller and smaller with scale, but others, such as the gridding error associated with vector-to-raster conversion, are measurable. In many cases, cartographers have suggested means for defining and measuring errors in these cartographic transformations so that they can be reduced (McMaster, 1986).

Third, errors occur due to the transformation of cartographic objects in digital form between data structures. The case can be made that none of these errors is due to unknowns, because the conversion is under the full control of the cartographer. With the encoding of topology, many consistency checks can become integral parts of GISs and computer mapping systems, ensuring that cartographic errors are more simply detected and corrected (Wagner, 1988). This approach is an integral part of the TIGER files discussed in Chapter 4.

Errors due to changes in data dimension are substantial, but are deliberate in the sense that they allow analysis or display that would otherwise be impossible. It is the straight data structure transformation errors that are most obviously manageable, and part of analytical cartography is clearly the pursuit of data structure transformations that minimize, or at least give complete accounts of, the error they introduce. The standards suggest the use of a quality map, one that maps out the error expected in the cartographic data. Only by fully understanding and modeling cartographic error can cartographers ensure that their products survive the tests of time as today's digital cartographic databases become the historical archives of the future.

12.6 REFERENCES

Baxter, R. S. (1976). *Computers and Statistical Techniques for Planners.* London: Methuen.

Beard, M. K. (1987). “How to Survive on a Single Detailed Database.” *Proceedings, AUTOCARTO 8,* Eighth International Symposium on Computer-Assisted Cartography, Baltimore, March 29–April 3, pp. 211–220.

Beard, M. K. (1989). “Use Error: The Neglected Error Component.” *Proceedings, AUTO-CARTO 9,* Ninth International Symposium on Computer-Assisted Cartography, Baltimore, April 2–7, pp. 808–817.

Bresenham, J. E. (1965). “Algorithm for Computer Control of a Digital Plotter.” *IBM Systems Journal,* vol. 4, no. 1, pp 25–30.

Buttenfield , B. (1985. “Treatment of the Cartographic Line.” *Cartographica,* vol. 22, no. 2, pp. 1–26.

Douglas, D. H., and Peucker, T. K. (1973). “Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature.” *Canadian Cartographer,* vol. 10, pp. 110–122.

Dutton, G. H. (1981). “Fractal Enhancement of Cartographic Line Detail.” *American Cartographer,* vol. 8, no. 1, pp. 23–40.

Franklin, W. R., and P. Y. F. Wu. (1987). “A PolygonOverlay System in PROLOG.” *Proceedings, AUTOCARTO 8,* Eighth International Symposium on Computer-Assisted Cartography, Baltimore, March 29–April 3, pp. 97–106.

Goodchild, M. F. (1978). “Statistical Aspects of the Polygon Overlay Problem.” *An Advanced Study Symposium on Topological Data Structures and Geographic Information Systems.* Cambridge, MA: Laboratory for Computer Graphics and Spatial Analysis, Harvard University.

McMaster, R. B. (1986). “A Statistical Analysis of Mathematical Measures for Linear Simplification.” *American Cartographer,* vol. 13, no. 2, pp. 103–116.

Muller, J. C. (1986). “Fractal Dimension and Inconsistencies in Cartographic Line Representations.” *Cartographic Journal,* vol. 23, pp. 123–130.

Pavlidis, T. (1982). “An Asynchronous Thinning Algorithm.” *Computer Graphics and Image Processing,* vol. 20, pp. 133–157.

Peuquet, D. J. (1979). “Raster Processing: An Alternative Approach to Automated Cartographic Data Handling.” *American Cartographer*, vol. 6, no. 2, pp. 129–139.

Peuquet, D. J. (1981). “An Examination of Techniques for Reformatting Digital Cartographic Data. Part 1: The Raster-to-Vector Process.” *Cartographica,* vol. 18, no. 1, pp. 34–48.

Preparata, F. P. and M. I. Shamos (1985). *Computational Geometry.* New York: Springer-Verlag.

Rosenfeld, A., and A. C. Kak (1981). *Digital Picture Processing.* 2d. ed. New York: Academic Press.

Wagner, D. F. (1988). “A Method of Evaluating Polygon Overlay Algorithms.” *Technical Papers,* ACSM-ASPRS Annual Convention, vol. 5, pp. 173–183.

White, D. (1978). “A New Method of Polygon Overlay.” *Harvard Papers on Geographic Information Systems,* vol. 6, Cambridge, MA: Harvard University.