8

Map Data Structures

8.1 WHY MAP DATA STRUCTURES ARE DIFFERENT

A distinction can be made between map and attribute data structures. Map data structures are different because they store information about location, scale, dimension, and other geographic properties. Attribute data structures may be linked to the data structures for cartographic objects, but they contain attribute information about the objects, or links between the objects only. In this chapter, we consider only the data structures dealing explicitly with locative information. In Chapter 9, we focus on attribute data structures. Map data structures are particularly cartographic and have been developed by cartographers and others to deal with computer and analytical needs and for use in GISs. Attribute data structures can also be called data models and are used to organize text or numerical data as well as cartographic data. Also essential are how these two types of data structures are related to each other, and how data can be transformed between structures. This theme is the topic of Chapter 12.

In the early days of computer cartography, the data sets that went along with cartographic software contained the minimum amount of information necessary to produce the desired map and as such used data structures that were nonanalytic, device specific, and as a result short-lived. These data structures, which have become known as entity-by-entity, or "cartographic spaghetti, " structures remain in circulation and for many purposes still serve the needs for which they were intended. Early on, however, attention became focused on the potential power of topologically encoded cartographic data. A group at Harvard University, at the time producing the ODYSSEY package, sponsored a conference on topological data structures that became a milestone in the acceptance of the need for topological encoding (Dutton, 1979). These topological structures have become the data structures for a large number of data sets such as DLG, GIRAS, and TIGER, and they form the core of a majority of the systems for the display and analysis of cartographic data.

In 1975, Tom Poiker (formerly Peucker) and Nick Chrisman published a paper entitled "Cartographic Data Structures", that reported a way in which the information required for the topological encoding of cartographic data could be converted into a data structure that was particularly geographic. Their structure, part of a program called POLYVRT, became the model for many cartographic systems and was highly influential in the evolution of data structures (Peucker and Chrisman, 1975). Since then numerous new, efficient, and thought-provoking data structures have been devised for cartographic data. The grid structure, the quad tree, and tessellations, in addition to the entity-by-entity, and the topological structures, have found acceptance and use within computer mapping systems.

A *data structure* can be defined as a logical organization of information to preserve its integrity and facilitate its use. By *information* here we mean the information content inherent in cartographic data, especially that necessary to produce a map. The information relates to the fundamental properties of geographic objects, that is, information about size, location, shape, distribution and so forth. We have to *preserve the integrity* of this information. In other words, we have to store it with correct precision, in a coherent and consistent manner, and in such a way that we can retrieve the right information at the right time. To *facilitate the use* of the information we need to have some way of organizing our data physically on a storage medium in a method of representation that allows us to symbolize cartographic objects on maps with relative ease and within a reasonable length of time. Much work in computer cartography consists of writing data into cartographic data structures and transforming the data between structures. Different structures suit different kinds of mapping and different sets of demands.

The general characteristics of cartographic data structures are largely input determined. The data structures inherent in the geocoded data are usually determined by whatever input device was used to capture the data. Whether it was a digitizing tablet or an orbiting satellite, the structure that the data collection instrument imposed on the data is almost always the form in which we get the data. As the digital cartographic data standards gain acceptance, most existing data will be in more logical formats, but for the present, the input-determined nature of the data is a fact.

In many cartographic applications, the data need only be a simple representation of the map in order to produce a graphic. Integrating data sets, such as plotting the various separations that make up a multicolor topographic map, simply involves overlaying one data layer onto another. The cartographer need only decide upon an order of precedence. In additive color systems, the last color plotted at one location takes priority, and no mixing takes place, although there are exceptions. In these types of data sets we have no explicit relations between the cartographic objects. The only geographic property represented is location, and the map interpreter is left the task of searching for geographic relationships.

Thus we have two types of data structure in analytical and computer cartography. First, we have data structures to store strictly locational data about the map, with perhaps any necessary topological or other data required to produce the map. In addition, we have attribute data structures, which encode the attributes associated with cartographic objects, plus the indexing necessary to support query and retrieval functions on the objects.

A map data structure is the minimum required for a computer mapping system, plus the actual data with its representational characteristics, and its origins in geocoding.

We can now respond to the issue raised at the beginning of this section. Why are map data structures different? Map data structures are required for computer mapping, and they are different from attribute data structures in that their purpose is to support computer cartography and not necessarily analytical cartography. A map plus an attribute data structure is the minimum requirement for the additional analytical functions we may use in analytical cartography, in the more sophisticated displays of advanced computer mapping systems, or in GISs. Also, many of these systems owe their power and capabilities to how the map and attribute data structures are related to each other, both logically and physically. Such maps—with both map and attribute, or graphic and non-map, data fully integrated—are often called "smart maps."

8.2 VECTOR AND RASTER TECHNOLOGIES AND DATA STRUCTURES

We have already seen a hardware division between raster and vector systems. Just as there are certain input controls on data structures, there are certain output constraints on data structures. Both input and display technologies have reflected generally a movement from vector-based to raster-based over time. Vector input devices such as digitizing tablets and output devices such as pen plotters are increasingly being replaced with raster technology, such as scanners for input and raster displays for output. This has influenced thinking about data structures considerably.

Both raster and vector data structures have been proposed as the answer for structuring geographic data. To use either one to the exclusion of the other, however, means accepting the inherent disadvantages of one of the systems. Many recent computer mapping systems and GISs have solved this problem by supporting both structures and allowing the user to transform between structures as appropriate. Peuquet (1979) showed that most algorithms using a vector data structure have an equivalent raster-based algorithm, in many cases more computationally efficient. Logically, therefore, the user should structure data according to the relative merits of each data structure for a particular type of cartographic entity or a particular mapping circumstance. Burrough (1986) has provided an extensive list of the advantages and disadvantages of each type of data structure. The advantages and disadvantages are worth considering, because they reflect the correct choice of structure for dealing with particular types of data and particular geographic properties.

8.2.1 Advantages of Vector Data Structures

Burrough considered vectors a good representation of data mapped as lines and polygons, because vectors can follow the lines very closely. This means that maps produced from vector-based mapping systems are capable of greater resolution and accuracy. This, however, assumes that the lines that have been captured from the map as vectors are thin, and line thickness at the map scale is not an issue. The vector structure is also compact in that it can represent very complex lines with a minimal amount of information. We have seen

that in vector mode we can reduce redundancy because if we have a fairly long line segment we only need to represent it with two x, y pairs, the two endpoints. Many other systems embed redundancy in representing straight lines.

As far as topological data structures are concerned, vectors are preferred, because topology can be completely described within vector-based systems as network linkages. Vector structures support interactive retrieval, so that updating and generalization of map data and data attributes are possible. For example, if we have a vector representation of a line at 1:25,000, it is very easy to resample the points along that line to give a generalization of that line so that we can plot the map at another scale, such as 1:250,000. Also, in editing the map we can remove an entire chain and reenter it if we have made an error.

8.2.2 Disadvantages of Vector Data Structures

Vector data structures are complex and are less intuitively understood by the users of cartographic data. The biggest disadvantage, however, is that the combination by overlay of two or more vector-based maps is a very computationally intensive task. Map overlay, or computation of the most common geographic units, means processing networks to find where they intersect. Just finding the intersections is difficult because we have to test every line segment of every chain to find an intersection unless we use some kind of preprocessing based on the ranges in x and y of the chains. We next have to reorder the polygons by the new intersections.

Overlay is a classic geographic problem, and many geographic phenomena need to be redistributed between sets of irregular areas. Not the least important, for example, is the redrawing of congressional and voting districts in the United States after each decennial census.

Display and plotting of cartographic data in vector mode can be expensive, particularly higher-quality color and cross-hatching. A vector-mode device especially does not do area fills very well, because plot devices try to fill areas by dragging the pen backward and forward, a time-consuming process. Vector mode display technology is also expensive, particularly for the more sophisticated software and hardware.

8.2.3 Advantages of Raster Data Structures

The principal advantage of raster data structures is their simplicity. The grid is an integral part of cartography and has long been used to structure geographic information. More recently, cartographic data have been acquired in grid formats directly, for example, in satellite and scanned air photo data. This aspect is becoming increasingly important as remote sensing becomes a source of new cartographic data.

Using the grid structure, analytical operations are easier, such as computation of variograms, and some autocorrelation statistics, interpolation, and filtering. Resolution on raster devices is improving steadily, and the cost of the graphic memory boards that support display in raster mode is falling, whereas vector technology has not changed significantly in cost or capability since the mid-1980s.

8.2.4 Disadvantages of Raster Data Structures

The principal drawback to raster data structures is the required volume of data. Grids embed much redundancy, so have a much larger data volume. If we have extremely variable data, such as topography, then grids use storage inefficiently, because compression methods do not save significant amounts of space. Also, the use of large cells to reduce data volumes means that some cartographic entities can be lost, simply slipping through the sampling net. Raster maps are considerably less visually effective than maps drawn with lines. Text on low-resolution raster devices is often illegible. Also, curved and angled text is difficult to produce. In raster mode, network linkages are difficult to establish. Cartographic entities such as stream networks, or boundaries between polygons, or linkages between town centers, are not well supported as cartographic objects. It is difficult, for example, to establish links within a stream network in raster mode. Map projection transformations are time-consuming using raster data structures, unless special algorithms or hardware are used. A disadvantage often ignored is orientation. If we use a grid, we imply an orientation to a grid, which automatically means that the sampling strategy has a particular directional bias. We normally orient our grids north/south and so bias the data against other directions.

As we have seen, neither of the data structures holds all solutions for all data. Grids are usually used for mapping extensive geographic information at small scales, while vectors are used for detailed information at large scales. Systems that support both structures allow data to be structured to take advantage of the strengths of each structure type.

8.3 ENTITY-BY-ENTITY DATA STRUCTURES

Cartographic entities are usually classified by dimension into point features, line features, and area features. The simplest means of digitally representing cartographic entities as objects is to use the feature itself as the lowest common denominator. Using entity-by-entity data structures, we are concerned with discrete sets of connected numbers that represent an object in its entirety, not as the combination of features of lesser dimension.

The simplest entity-by-entity structure is the point. In the digital cartographic data standards, the cartographic objects listed as “graphics only” are closest to entity-by-entity structures in type. Thus the point, especially the entity point, is the simplest form for dimension 0; the string, the arc, and the ring, for dimension 1; and the polygon and grid cell for dimension 2.

Chapter 6 included program segments for reading and storing data in these structures, and Chapter 11 contains the necessary code for generating the map from the data in these structures. The grid cell structure is covered in detail later in this chapter.

An important difference between entity-by-entity and topological structures is shown by the ring structure. As far as a map is concerned, a lake can be shown on a map using a RING data storage structure. The first and last points match perfectly in x and y is simply to close the polygon visually. If we wish to compute the lake area, or the length of the boundary, or even shade the lake with color, the ring structure is adequate. If we wish to find the polygons that border the lake, or determine which rivers flow into the lake, however, it would require a large amount of systematic searching to find endpoint matches.

Similarly, for a point, we are able simply to plot a symbol at the point, but we do not know which other features are related to the point and how.

8.3.1 Point Objects

Attributes associated with cartographic objects stored in entity-by-entity data structures rarely go beyond those contained in the C language structures in Chapter 6. Point features need only a feature code, because the feature code can be used as a key into attribute database of independent records for each feature. An entity-by-entity list of cities could contain their locations stored as a set of points, and if populations were required to produce a proportional circle map, for example, the cities could be stored as label points and the label itself could be used as an index into a file of attributes. The files in Function 8.1 could contain all the data necessary to produce a proportional symbol map.

Function 8.1

```
File : City_Populations
Format : ASCII population est. in 1986
Source : Goode's World Atlas, 17th Ed.
Structure : Flat File

Quito                          918884
Rabat                          367620
Rangoon                        2276000
Rawalpindi                     452000
Recife                         1204738
Riga                           875000

File : City_Points
Format : ASCII Lat/Long in DDD.MM format
Structure : Label Point

Quito             -0.17        -78.32
Rabat             33.59        -6.47
Rangoon           16.46        96.09
Rawalpindi        33.40        73.10
Recife            -8.09        -34.59
Riga              57.56        23.05
```

8.3.2 Line Objects

In an entity-by-entity structure, a line is usually represented by a string, that is, an ordered set of points, which connected together in sequence trace out the line. In vector mode the string is unconstrained, although usually lines are terminated and restarted when crossing takes place. Similarly, there is no obstacle to double specification of a single line, a common problem resulting from the manual digitizing process. A line can carry an attribute, such as a traffic volume, a color, a thickness, or a feature type such as highway or railroad. Attributes are not really assignable to strings directly, but a separate file of attributes by string number can be maintained. Alternatively, separate files for each feature type can be maintained, similar to separating line colors by separation in a manual color production process.

A common alternative to keeping a file containing each string listed point by point is to use a point dictionary (Figure 8.1). In this file arrangement, separate files are kept for points and for lines. The point file is simply a listing of every point used on the map, containing an identifier, often in a sequential order, and an easting and northing. The second file, the line file, contains line numbers, perhaps the line attributes, and the starting and ending point numbers of sequential groups of points in the points file that make up the line. The point numbers act as pointers into the point dictionary. An alternative system has a list in the lines file of every point identifier in the line. This system, although much more cumbersome to implement and use, has the advantage that the point data file can be sorted at will to facilitate searching or processing. Point dictionary systems make editing and data entry relatively difficult, but have found their uses in mapping systems.

Several grid-type structures exist for entity-by-entity data structures for lines. If a grid is used, lines are typically labeled with attribute numbers that are assigned to grid cells

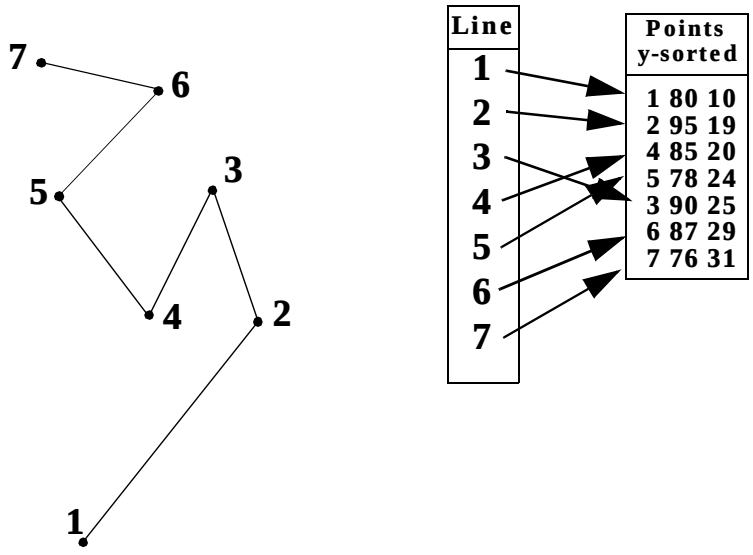


Figure 8.1 Line defined as entity-by-entity reference to point dictionary.

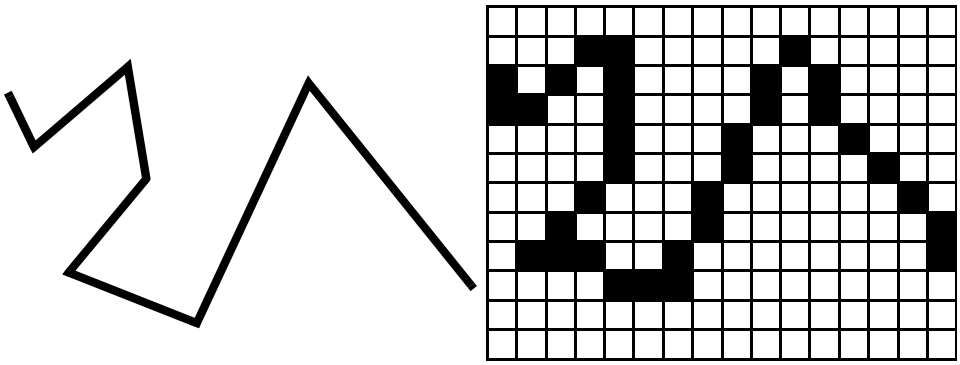


Figure 8.2 Representation of lines in entity-by-entity structures.

as data. Thus to give an entity-by-entity definition of a river for use in a grid mapping system, we could assign a value for the river to every pixel into which the river passes. Connectivity can be limited to the four adjacent cells or enlarged to permit diagonal or eight-cell connectivity. It is normal to thin the lines to one pixel width for processing and display (Figure 8.2). If lines are stored as attributes or index numbers in the grid, problems result at line intersections. These are sometimes stored separately; otherwise, the attribute or color plotted at an intersection is the last one referenced.

The Freeman code is an entity-by-entity line data structure that falls between vector and grid. The Freeman code is a representation of a line as a sequence of numerical codes, each representing the direction of a step moved along the line (Figure 8.3). Freeman codes that use one octal digit per code are eight-pointed, while one hexadecimal digit (nibble) per code allows 16 values. A problem of converting grid data to Freeman codes is that diagonals have length of $\sqrt{2}$ over 2, wheras the primary directions have length of one over two in grid units. Both strategies can be used. Anstaett and Moellering (1985) gave a simple algorithm for computing areas of polygons bounded by sets of Freeman codes. Freeman codes can also be run-length encoded, meaning that files containing long straight lines, with repeating identical Freeman codes, can be compressed considerably.

8.3.3 Area Objects

Because we are dealing with entity-by-entity structures, polygons of two-dimensions can be represented cartographically by one-dimensional objects unless we wish to fill the polygon. Appropriate structures in the C language, as given in Chapter 6, are the two forms of the RING, in its form as a set of strings or as a set of arcs. As a two-dimensional object, the POLYGON structure is appropriate. Strictly speaking, the complex polygon with holes cannot be stored as an entity-by-entity object. The hole can be part of the boundary, with a “bridge” between the two lines, or we can use alternative means to include the hole.

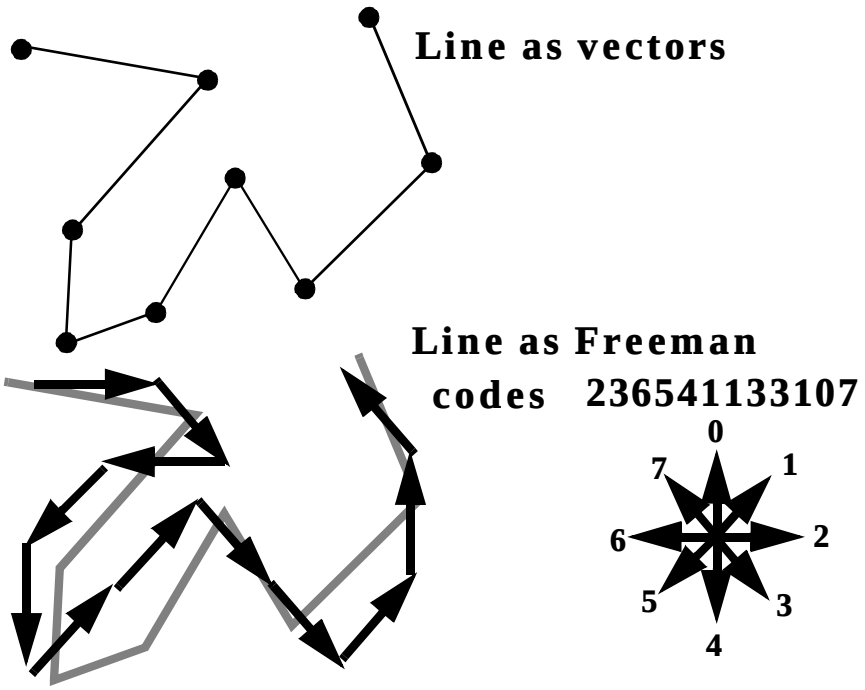


Figure 8.3 Freeman codes as a line data structure.

Just as we used the point dictionary for a line, we can also use the point dictionary for an entity-by-entity area (Figure 8.4). In this case the second file is the area file, which contains the polygon identifier and a list of the points that make up the boundary. There may be a good reason for using this structure, because a sorting of the point dictionary makes it simple to determine shading instructions for filling the polygon. A polygon list, in addition, is the simplest means for computing the area of a polygon and for drawing the polygon as a single entity against a plain background on a map.

Using the grid structure, polygon representation is simple. Membership is usually assigned to the grid cell with the highest area content for any polygon, and all cells within a polygon are assigned one value, either an index or an attribute (Figure 8.5)

Alternatively, the class of the center of the grid cell can be assigned. Holes can simply be assigned last, so that their attributes fall on top of any prior values. In this way, areas can be computed by cell counting, a rapid though less accurate technique. Similarly, polygons can be made up of strings of Freeman codes, although in this case care must be taken to assure closure.

Closure is when the vector sum of the individual Freeman code vectors is zero, but again no requirement to prevent crossing is included. Crossed-over polygons (Figure 8.6) are sometimes called weird or splintered polygons. These are often errors in digitized maps that we seek to detect and eliminate. Area calculations and other operations are not appropriate for these polygons.

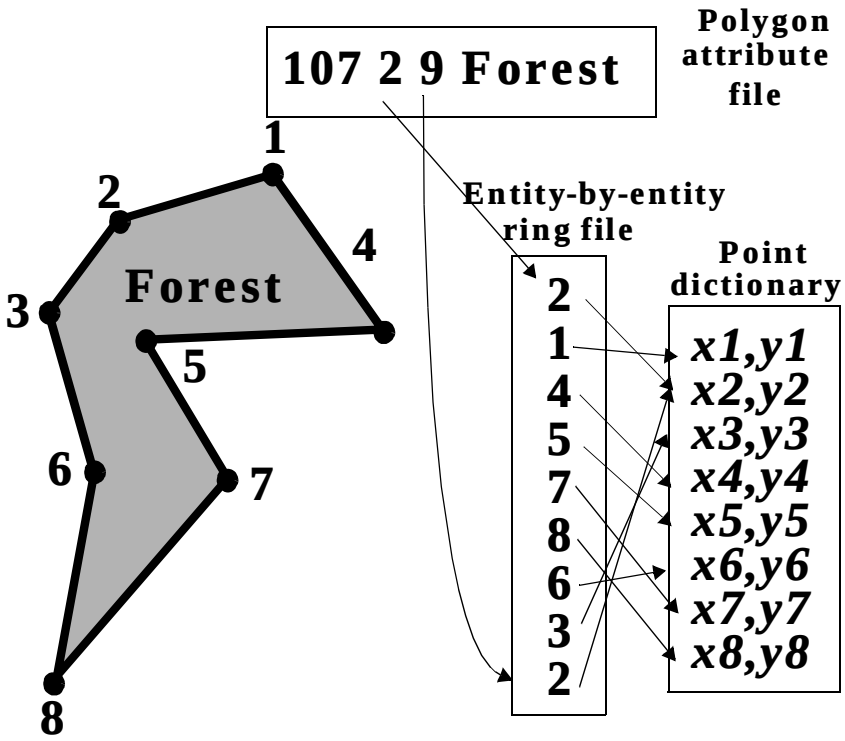


Figure 8.4 Point dictionaries for a polygon.

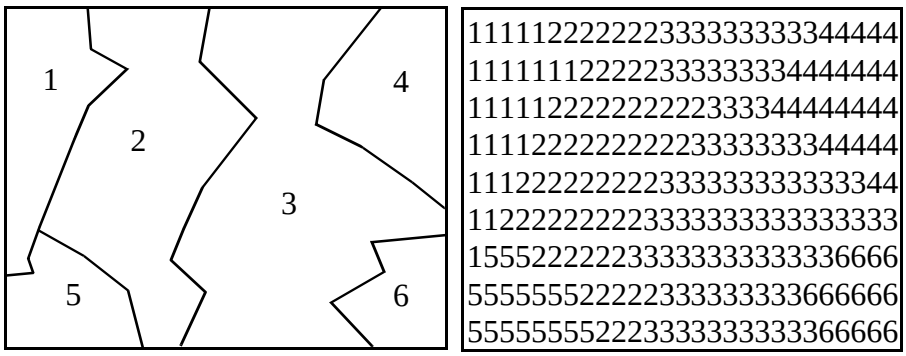


Figure 8.5 Polygon structures for a grid.

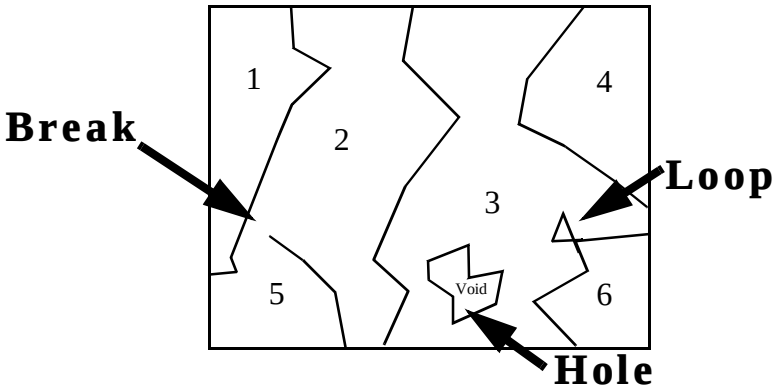


Figure 8.6 Weird polygons.

8.4 TOPOLOGICAL DATA STRUCTURES

Entity-by-entity data structures, although useful for the symbolization of cartographic objects, are often unsuitable for more complex analysis because they store no information about topology. Furthermore, we have no guarantee with entity-by-entity structures that impossible geometric configurations such as weird polygons and slivers do not exist. Topological data structures store the additional characteristics of connectivity and adjacency. As such, they allow immediate checking, perhaps during data entry, for errors in geocoding and data storage.

The basic objects are the node, the link, and the chain. To these are added direction, as in the link, and higher structure, as in the network chain. Nodes are points with topological significance and as such must be labeled. The link and the network chain are similar in that they support linkages to adjacent objects and nodes. Connectivity to nodes is essential to give a graphic frame of reference and to link lines in the network together. The direction and the links to other primitive objects can be summarized as properties of a winged segment (Figure 8.7).

Linkages are sometimes stored as forward linkages and reverse linkages. This is because chains can begin only at one end, and to piece together objects it is very often required to traverse a chain in either direction. Forward linkages are the segment identifiers of connected segments that are pointed to when a chain is traversed in the direction in which its x and y values are stored.

We can have forward linkages to other links, directed links, or chains. Usually, there is a finite limit to the number of chains that can meet at a node. This is not a problem for most maps; for example, the 48 contiguous states have only one node where four lines meet (at the Four Corners), but sometimes at the poles, for example, many lines meet together at once.

To traverse an entire network we only need know immediate right-hand and left-hand turns. Eventually, if we keep following around the network, we come back to the start point along some other chain. If we have made an error and one of the chains has not been

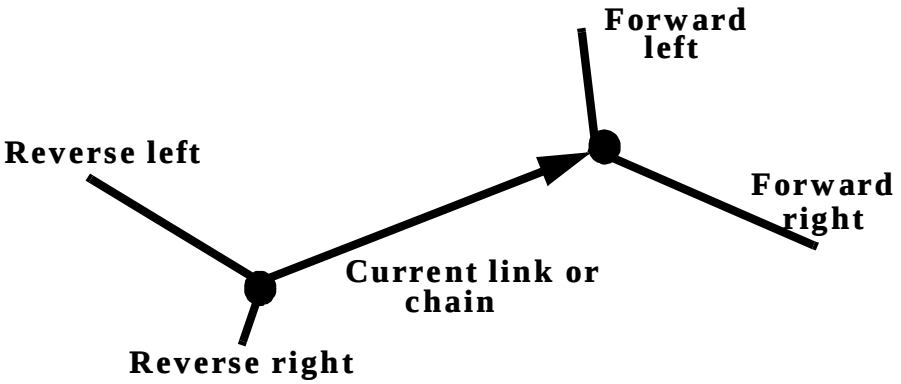


Figure 8.7 A winged segment.

snapped to a node, or if not all intersections are nodes, the network tracking algorithm in trying to close the polygon will go off along the wrong chains, going around through the network trying to close a polygon.

Topological data structures also store polygon information, which makes the data structure more analytically powerful. Right-hand and a left-hand polygons are assigned according to the direction of the chain or directed link. This allows us to presort the chains to select all chains that say that they are neighbors of a particular polygon, for example. These chains can then be connected either by their chain linkage information or by matching their end points, and they can be converted into a polygon list as an entity-by-entity structure or simply plotted in order.

8.5 TESSELLATIONS AND THE TIN

Tessellations are connected networks that partition space into a set of sub-areas. They consist of the network itself and the spaces left within the areas. Special cases of tessellations are topological structures, in which the space is partitioned into regions of geographic interest such as states and counties and grids, in which the tessellation is a network of connected squares and the grid cells they divide the space into. The remainder of tessellations include a number of systems used infrequently in analytical and computer cartography, such as equilateral triangles and hexagons, which are geometric and regular in nature, and one irregular system, which has gained widespread use, the *triangulated irregular network* (TIN).

Mark (1975) has suggested that TINs are more accurate and use less space than any other data structure for topography, and McCullagh and Ross (1980) demonstrated that TINs can be generated from point data faster than the alternative data structure for terrain, the grid. TIN data structures can describe more complex surfaces than a grid, including vertical drops and irregular boundaries. Because single points can be easily added, deleted, or moved, this structure has gained widespread acceptance in CADD systems, surveying, engineering, and terrain analysis.

The TIN structure was proposed by Peucker et al. (1976) and takes into account an important property of geographic data collection,that is, that map data collection often tabulates data at points and that the points themselves have an inherent significance as far as the information content is concerned. For example, a surveyor measuring land surface elevations seeks "high information content" points on the landscape, such as mountain peaks, the bottoms of valleys and depressions, and saddle points and break points in slopes, and he or she measures the elevation at these points. The intervening elevations are then often interpolated from the significant points.

The simplest form of interpolation is to assume that between triplets of points the land surface forms a plane. This is the foundation of the TIN structure. Triplets of points forming irregular triangles are connected to form a network. Rules are followed in partitioning the points into the nodes of the triangles. The space is then divided into a set of triangles, their edges, and their nodes. Data can be associated with any of these elements. Two approaches are in common use to arrive at the "best" triangulation. In the first method, an initial triangulation is assigned and is then refined until the triangles meet the Delaunay criterion. Alternatively, some methods seek to compute the best triangulation in one step. McKenna (1987) reviewed these approaches.

The first problem in TIN creation is how to allocate the triangles between points. This problem is usually solved using the Delaunay triangulation (Figure 8.8). This is an iterative solution that begins by searching for the closest two nodes and then assigns additional nodes to the network if the triangles they create satisfy a criterion, such as selecting the next triangle that is closest to a regular equilateral triangle. Many different methods for performing this space partitioning exist and are constantly being improved and made more efficient (McKenna, 1987)

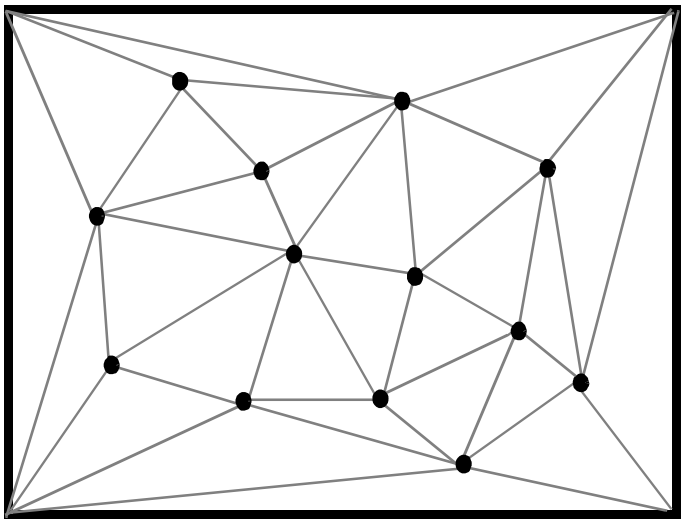


Figure 8. 8 Delaunay triangulation to yield a TIN.

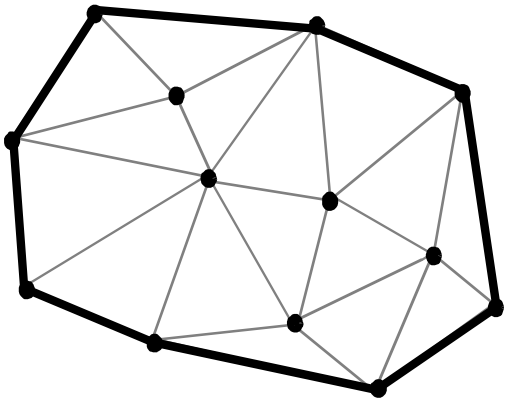


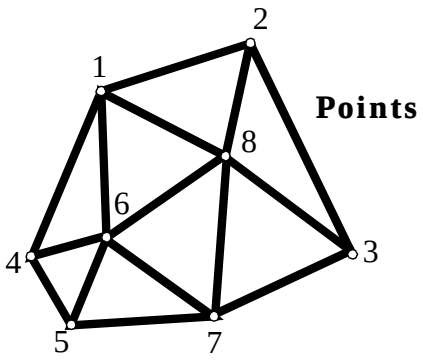
Figure 8.9 A convex hull.

The Delaunay triangulation yields a convex hull (Figure 8.9). This is because if we make an attachment that makes a concavity in the edge of the network, the network can be extended until the concavity is removed. So in other words, there are no concavities in the hull's edges. The convex hull presents another problem in constructing a TIN, because no data can be stored in the structure for the region between the edge of the convex hull and the square or rectangular edges of the map. A common solution is to include the map corners, and perhaps points along the map edges, in the Delaunay triangulation or to go beyond the map edges with additional points outside the area of interest.

Two data structures have been used to physically represent TINs within the computer's memory. The first uses the triangle as the basic cartographic object, while the other uses the vertices of the triangles. Most TIN structures use the triangles as the object, and store links to neighboring triangles (for example, see Gold et al., 1977). In this case, data are contained in two files (Figure 8.10).

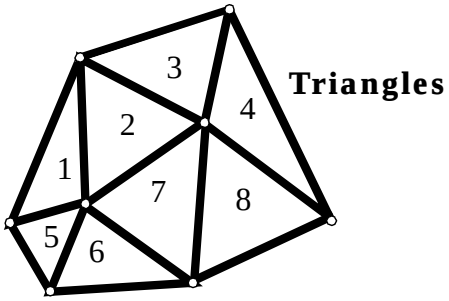
The first file contains points, stored as `[x,y,z]` triplets with their eastings, northings, and elevations. The second file contains one record per triangle and contains three attributes that are pointers into the point file, plus three additional pointers to adjacent triangles. Function 8.1 gives an example of data stored in this data structure.

The original TIN structure used in Poiker and Chrisman's paper (Peucker and Chrisman, 1975), used a vertex-based system. In this data structure, the point record contains `[x,y,z]` as before, but also a pointer to a "connected points" file. The connected points file is sequential and contains, at each location pointed to in the points file, a list of nodes that are connected to the point in question. The list is terminated with a zero to allow an arbitrary number of nodes (Figure 8.11). This data structure uses about half the storage of the triangle object data structure. Each of these two structures has advantages and disadvantages depending on the particular application. In both cases the second file is constructed from the points file during the Delaunay triangulation phase of the TIN construction.



Point file

1 x y z
2 x y z
3 x y z
4 x y z
5 x y z
6 x y z
7 x y z
8 x y z



Triangle file

1	**1**	**6**	**4**	2	5	X	
2	**1**	**8**	**6**	1	3	7	
3	**1**	**2**	**8**	X	2	4	
4	**2**	**3**	**8**	3	8	X	
5	**4**	**6**	**5**	1	6	X	
6	**6**	**7**	**5**	5	7	X	
7	**6**	**8**	**7**	2	6	8	
8	**8**	**3**	**7**	4	X	7	

Figure 8.10 The TIN data structure based on triangles.

Function 8.1

```
/* C language structure to hold a TIN */
#define MAXTRIANGLES 20 /* Max # triangles in TIN */
typedef struct {
      POINT Point;
      int elevation;
} TRIPLET;
typedef struct {
      int Vertex[3];
      int Neighbor[3];
} TIN;
TIN Tin[MAXTRIANGLES];
/* the tin array holds the triangle file */
TRIPLET Triplet[MAXPOINTS];
/* the triplet array holds the points file */
```

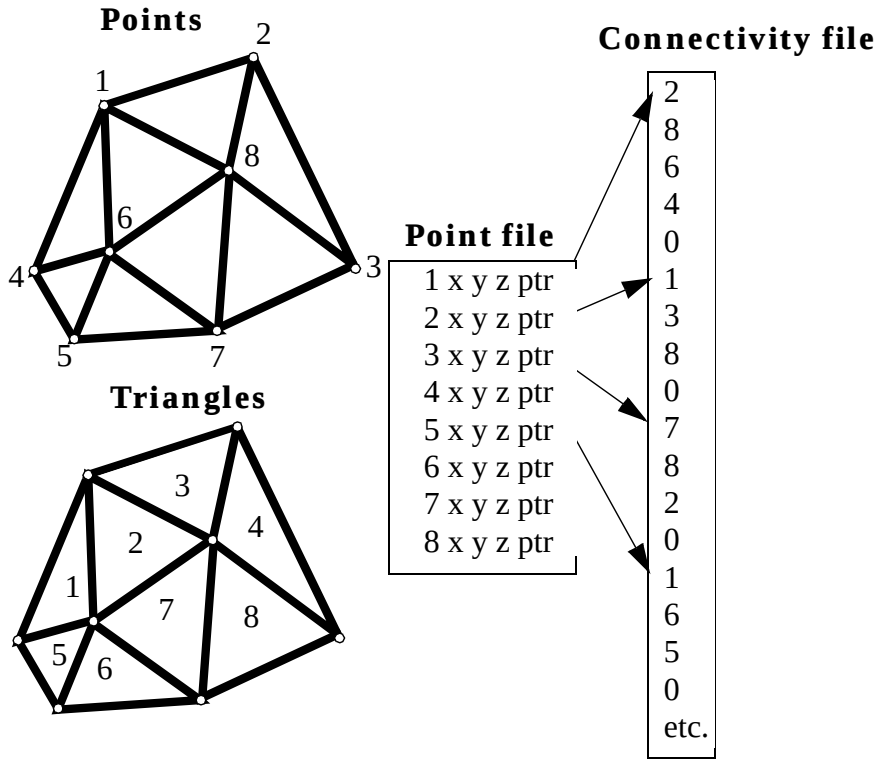


Figure 8.11 The TIN data structure based on vertices.

A major advantage of the TIN structure is that many of the problems associated with automated contouring, hidden-line processing, and surface shading for cartographic symbolization are simpler to tackle in this structure. The structure is particularly suitable for workstations that support polygon manipulation and shading in three dimensions in hardware, allowing interactive production of perspective views and realistic perspectives. The structure is also advantageous for modeling, especially of hill slopes and streams, which flow along the edges of the triangles across the terrain. The TIN data structure is also very suitable for the intervisibility problem, which seeks to determine in three dimensions solutions to which facets in the TIN are visible from points and lines, perhaps at different altitudes.

8.6 QUAD TREE DATA STRUCTURES

The quad tree is a data structure that has received increasing use in recent years. At first, quad trees were used exclusively in image processing for binary images and for partitioning lines. More recently, a number of advances in algorithms have made quad trees a viable data structure for cartographic data. Quad tree equivalent algorithms now exist for area computation, centroid calculation, image comparison, connected component labeling, neighbor detection, distance transformations, regionalization, smoothing, and edge

enhancement (Tobler and Chen, 1986). Mark and Lauzon (1985), among others, pointed out the feasibility and advantages of the data structure for map data, and Samet (1984) provided the definitive survey.

Quad trees are areal indexing systems and assume that the basic data element is the area, as represented by the extent of a grid cell. Quad trees, used as a cartographic data structure, also assume that all areas to be stored are grid cells with a side that is some power of 2 of the smallest resolution element. The quad tree data structure allows very rapid area searches and relatively fast display. The gain in efficiency is the result of the exploitation of the fact that areas on maps are typically some combination of large homogeneous areas and smaller heterogeneous areas. Quad trees are tessellation data structures, in that the space on the map is partitioned by a covering network of spaces, but for the quad tree, the partitioning is into nested squares. Division of squares is into quadrants, giving the "quad" part of the name.

At the highest level, the entire map can be thought of as one square. Rectangular or irregular maps can be subdivided into a set of square regions to become starting points in the quad tree. Each square is then divided into four quadrants, the NE, NW, SE, and SW, if and only if the square contains detail at a level greater than the current area of the square. The path to each quadrant then becomes a branch, from the root of the quad tree to its leaves, which are the highest level of detail.

This system is familiar to users of the U.S. Public Land Survey System. In this system, square regions (and rectangular, and indeed irregular shapes due to errors) are given a reference within a township and range system. For example, a small land tract may have the following reference

```
Northeast Quarter of the Southwest Quarter of the Southeast
      Quarter of Section 21, Township 24N, Range 2E.
```

When the land holding is large, it can be referenced simply; for example, we could record that all of Section 21, Township 24N, Range 2E is farmland. When the landholdings are small and various, we need the full list to specify the plot.

To index a very small area within a quad tree structure, we could use a list such as NE, SW, NE, NW, SE, and so forth.

More efficient, however, is to store the Morton number. The Morton number is a unique identifier assigned to quad tree divisions as they are constructed. Figure 8.12 shows how Morton numbers are assigned. In their simplest form, quad trees consist of strings of Morton numbers, each of which contains the index or pointer to a data value associated with the highest-level quadrant. When large squares are sufficient, the Morton sequence is short, and vice versa.

Quad trees are most efficient when the basic cartographic entity is the polygon. The cells in the quad tree then become the cartographic objects, and the data structures contain the coordinates of the region in question, the Morton sequences for each unique cell division in the quad tree, and the attributes for these cells. More than this, the efficiency of the quad tree is at its highest when a mix of large homogeneous areas and smaller polygons exists, because the storage gain is mostly from two-dimensional run-length encoding of the large regions.

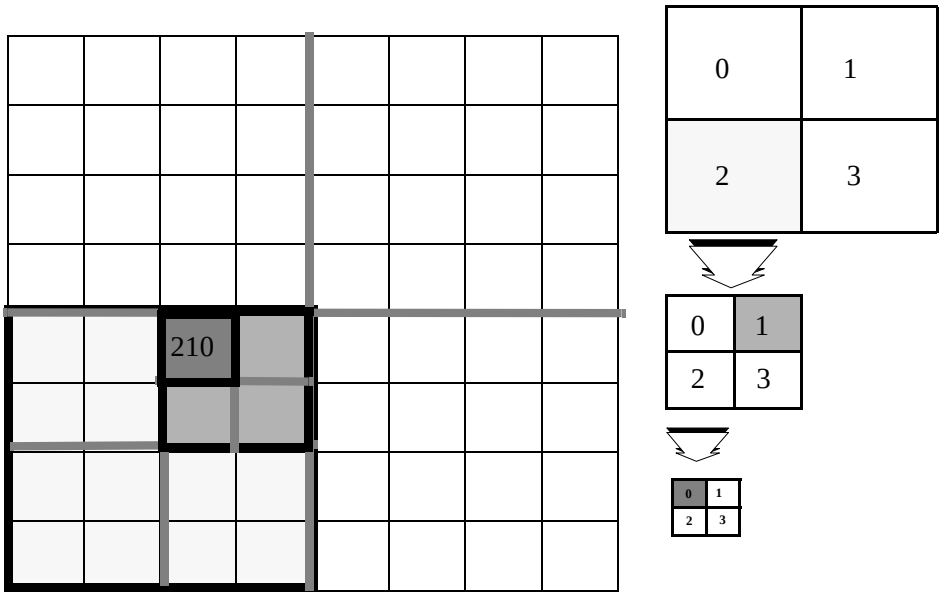


Figure 8.12 Selection of Morton code 210.

Land-use maps are good examples, when this is the case, as also are political units such as counties. Land-use maps generally have a background category that occupies most of the space, such as forest or agriculture. The map then has patches on top of this background, such as roads, commercial districts, lakes, and wetlands. In these cases the quad tree may be the most effective structure for several types of analysis to be performed on the data. Given that many applications in remote sensing generate regionalized or "segmented" grid maps, the strong link among the quad tree, remote sensing, and the grid data structure is self-evident.

8.7 MAPS AS MATRICES

The logical organization of raster data is easily converted into a physical representation in storage. This is because almost all computer programming languages support the array as a data structure directly. Thus in C, the array is given as `z[i][j]`, in Pascal as `z[i,j]`, and in FORTRAN as `Z(I,J)`. As such, the mathematical term for this organization is *matrix*. In simple terms, we can consider a map to be a matrix. Each element in the matrix falls as a grid cell on the ground and can be set to coincide with the spacing and orientation of the coordinate system and map projection in use. For example, we could divide the world up into grid cells based on 10-degree increments of latitude and longitude and store data for the cells in a matrix. Given the corners of the matrix, and the

```
grid.z[i][j]
where (0 <= i < nrows) and (0 <= j < ncols)
```

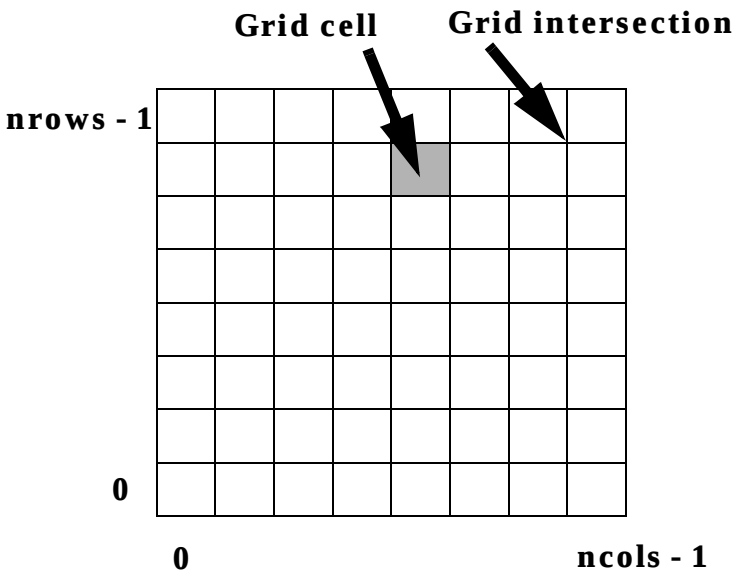


Figure 8.13 Generic structure for a grid.

spacing, we can implicitly refer to each matrix element just by referring to its row and column numbers, rather than the explicit way we give eastings and northings for points (Figure 8.13).

When we read a matrix into the computer's memory, we read it row by row and column by column, one element at a time. Having the grid stored in this way has the distinct advantage of simplicity. The disadvantage is that elements that are spatially next to each other are not necessarily close together in the file. Fortunately, when the data are within the array in a program, the immediate neighbors can be found easily.

Most automated mapping systems that use pixel-based display or use remotely sensed data in any form use the grid as their basic data structure. Usually, the files actually stored contain the registration information, such as the coordinates of the corners, the grid spacing, and so forth, plus the image itself, often in binary or compressed form to save storage. Files can be very large using the grid structure. For example, many display devices support images of 512 rows by 512 columns. A single data set, storing one byte per record, takes up 262 kilobytes without compression.

Both the Spatial Data Transfer Standard and GKS define a basic grid as a primitive element. In SDTS, the elements are the grid cell and the pixel, and the composite object of an image. Under GKS, the elements are the cell array and the pixel. As in the data standards, the GKS cell array grid cell need not be a single pixel. Using these elements in

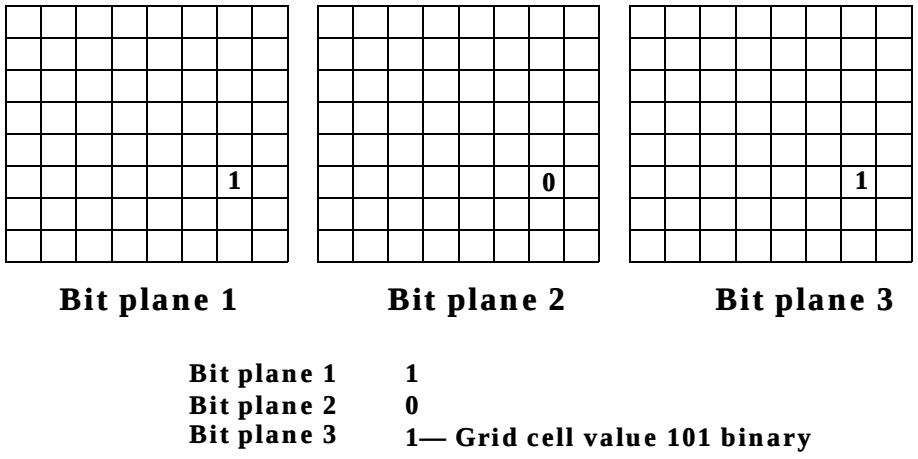


Figure 8.14 Bit plane storage of a grid map.

GKS is termed *raster graphics*, defined as a display image composed of an array of pixels arranged in rows and columns. Because this terminology is part of the standard, the issue of raster or vector representation is trivial and either type can be displayed on any device

A common format for the provision of raw grid data is the DEM format of the U. S. Geological Survey. A program to read the DEM format, as the files are available on tape or over the Internet, is provided on the companion disk as `read_dem`, and a sample data set is provided. The program `draw_image` reads the grid files output by this program, converts the grid to an image, and uses GKS to plot the maps. It is covered in detail in Chapter 12. Examples of data stored in a grid format have been given in Chapter 7, and include the DEM. Issues associated with grid data are registration problems, map projections, boundary effects, and edge matching, especially when the map edges do not neatly fit along the edges of the grid.The problem of piecing multiple grid files together is called mosaicking.

An alternative way of storing information in an array is to use *bit planes* (Figure 8.14). Bit planes are much more associated with storage and display than with manipulation, and are particularly useful for large areas with similar values, although they are universally applicable. In some cases, bit-mapped terminals can be used directly using this structure. Very high resolution monochrome terminals are often bit-mapped.

Bit planes can be used in two ways. Bit planes can represent colors on a display as a map of the pixels. One bit plane can be made to correspond to each color. For the six major colors (red, green, blue, magenta, cyan, and yellow), we need only three bit planes. For a single cell, if all three planes are 0, the screen shows black. If all three planes are 1, the screen shows white. Red, green, and blue correspond to the three planes directly, with a 1 in the correct plane determining the color. Magenta has a 1 in red and blue bit planes, cyan has a 1 in the green and blue bit planes, and yellow has a 1 in green and red.

With three bit planes, we can store eight colors, and because the bit arrays can be

placed directly into the video display channels, a color screen can be drawn quickly. Four and more bit planes become complex to use, although storing data bit by bit in bit planes allows extremely fast overlay of images and permits binary operations between planes, such as AND, OR, and NOR. Each bit plane, in addition, can be run-length encoded to drastically reduce the amount of memory required (Figure 8.15).In run-length encoding, we store pairs of bytes, the first representing the value stored in the array, the second giving the number of times that the value should be repeated. Major space savings are possible using run-length encoding when the map is spatially homogeneous.

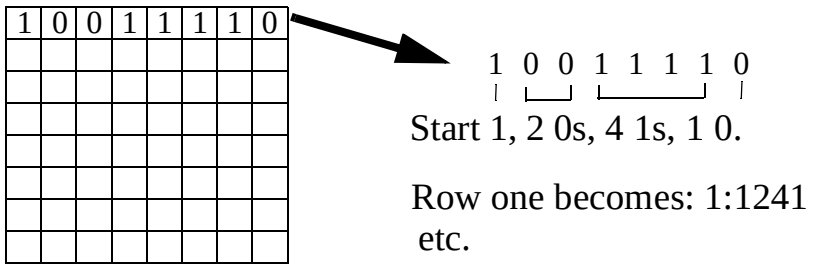


Figure 8.15 Run-length encoding for a bit map.

8.8 MAP AND ATTRIBUTE DATA STRUCTURES

In this chapter, we surveyed the major data structures in use for the storage of cartographic objects within their mapping context. Analytical and computer cartography, however, requires far more than simply the map data to produce an effective map, to use the data for new cartographic applications, and to support the demands of GISs. In many ways, map data structures are the determining factor for computer cartography. Many of the issues of geocoding, data conversion, error control, analytical power, and the choice of map display are really determined by the map data structure used. Understanding the strengths and weaknesses of these structures is critical to the understanding of analytical cartography.

In Chapter 9 we will consider how attribute data can be similarly structured and examine the need for effective links between the map and the attribute data. It is on the strength of the map to attribute link that many mapping systems and GISs will stand or fall as new and innovative uses for cartographic data find their way into the everyday life of noncartographers. We should remember, therefore, that a good cartographic data structure must both preserve the data integrity and also make the geographic information it contains available to make maps that can be used and understood by all.

8.9 REFERENCES

Anstaett, M. R., and H. Moellering (1985). "Area Calculations Using Pick's Theorem on Freeman-encoded Polygons in Cartographic Systems." *Proceedings, AUTO-CARTO 7,* Seventh International Symposium on Computer-Assisted Cartography, Washington, DC, March 11–14, pp. 11–21.

Burrough, P. A. (1986). *Principles of Geographical Information Systems for Land Resources Assessment*. Monographs on Soil and Resources Survey Number 12, Oxford Science Publications. Oxford: Clarendon Press.

Dutton, G., ed. (1979). *Harvard Papers on Geographic Information Systems.* First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems. Reading, MA: Addison-Wesley.

Gold, C. M., T. D. Charters, and J. Ramsden (1977). "Automated Contour Mapping Using Triangular Element Data Structures and an Interpolant Over Each Irregular Triangular Domain." *Computer Graphics,* vol. 11, pp. 170–175.

Mark D. M. (1975). "Computer Analysis of Topography: A Comparison of Terrain Storage Methods." *Geografiska Annaler,* vol. 57, pp. 179–188.

Mark, D. M., and J. P. Lauzon (1985). "Approaches for Quadtree-based Geographic Information Systems at Continental or Global Scales." *Proceedings, AUTO-CARTO 7, S*eventh International Symposium on Computer-Assisted Cartography, Washington, DC, March 11–14, pp. 355–364.

McCullagh, M. J., and C. G. Ross (1980). "Delaunay Triangulation of a Random Data Set for Isarithmic Mapping." *Cartographic Journal,* vol. 17, no. 2, pp. 93–99.

McKenna, D. G. (1987). "The Inward Spiral Method: An Improved TIN Generation Technique and Data Structure for Land Planning Applications." *Proceedings, AUTOCARTO 8,* Eighth International Symposium on Computer-Assisted Cartography, Baltimore, MD, March 29–April 3, pp. 670–679.

Peucker, T. K., and N. Chrisman (1975). "Cartographic Data Structures. " *American Cartographer,* vol. 2, no. 1, pp. 55–69.

Peucker, T. K., R. J. Fowler, J. J. Little, and D. M. Mark (1976). *Digital Representation of Three-dimensional Surfaces by Triangulated Irregular Networks (TIN).* Technical Report No. 10, U.S. Office of Naval Research, Geography Programs.

Peuquet, D. J. (1979). "Raster Processing: An Alternative Approach to Automated Cartographic Data Handling." *American Cartographer*, vol. 6, pp. 129–139.

Samet, H. (1984). "The Quadtree and Related Hierarchical Data Structures." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no. 3, pp. 298–303.

Tobler, W. R., and Z. T. Chen (1986). "A Quadtree for Global Information Storage." *Geographical Analysis*, vol. 18, no. 4, pp. 360–371.