CHAPTER 3

Map Algebra

LEARNING OBJECTIVES

On completing this chapter and combining its contents with outside readings, research, and hands-on experiences, the student should be able to do the following:

1. Identify and implement appropriate coding schemes for point, line, area, and surface data at all levels of geographic data measurement

2. Explain the difference between systematic and nonsystematic raster encoding and discuss the advantages and disadvantages of each

3. Create extended raster attribute tables for point, line, area, and surface data at all levels of geographic data measurement

4. Analyze and quantify single-theme spatial error created through one or more encoding schemes for raster data

5. Describe and illustrate the methods of dealing with encoding multiple point and line entities and their associated attributes when multiple objects occur within the geographic boundaries of a single grid cell

6. Explain the similarities and differences between map algebra and matrix algebra

7. List and define the basic operators available in map algebra and provide brief descriptions of each, including suggestions as to what they would be used for

8. List and define each of the functions of map algebra, provide brief descriptions of their purpose, and suggest, in brief, how one might employ them

9. List and define each of the basic flow control operations available in map algebra and explain why flow control operations are necessary and how they might impact the automation of modeling operations and functions

10. Explain what iteration operations are and what role they might play in the development of cartographic models
CONCEPTUALIZING ZERO-THROUGH TWO-DIMENSIONAL SPACE WITH GRID CELLS

In Chapter 2, we learned about some spatial tessellations used to quantize geographic space into discrete units or grids. We also learned some basic methods for modeling multiple thematic data within the computer. I purposely avoided a discussion of data measurement levels until now, when we combine our tessellations, raster data models, and mathematics within the central framework of cartographic modeling—map algebra (Tomlin and Berry 1979). Before we can effectively discuss map algebra, we need to review the ideas of nominal, ordinal, interval, and ratio scales of geographic data measurement as outlined in most introductory GIS textbooks, but focusing explicitly on the raster tessellation of geographic space. We will begin with zero-through two-dimensional space because most coverages inside raster GIS are not statistical surfaces. Additionally, we will discuss the conversion of analog spatial data into the basic raster tessellation so that we have a feel for exactly what it is we are modeling, at what measurement level, and with what possible positional errors we will have to contend as we build our models.

Two-dimensional space will be defined here as any nonstatistical surface-related geographic data and will include the three basic cartographic data types we saw in Chapter 2—namely, points, lines, and areas. Each of these objects represents real-world features that have been characterized by the observer and then abstracted to some degree both spatially and numerically. Such point features as electric power poles actually resemble points in reality, thus not occupying any substantial area on the ground. As such, their absolute location is often considered to be somewhere within a given grid cell, and the accuracy of the location is a direct function of the grid cell size. As we have just described it, the power pole is identified only as a single piece of nominal data in that it simply has a name associated with the location. Under such circumstances, the power pole is often recorded as a numerical value (usually a whole number) that merely indicates its existence somewhere within the grid cell. To encode this, we preselect a number to represent the grid cell and note its location using a presence/absence methodology of raster encoding (DeMers 2000a) (Figure 3.1). This method typically uses some nonzero value to indicate presence of the object and zeros to show its absence. At this point, if we are using the extended raster model, we can also indicate any additional attributes, whether numerical or categorical, as part of the linked relational database management system (RDBMS) (Table 3.1). Such attributes might include the size of the pole, its type (wood, metal, etc.), and the last time it was inspected.

Let us assume, however that we would rather encode the poles not as a single category of electrical poles but as multiple categories. For example, we might want to create a theme called power poles that explicitly categorizes them as, for example, with one, two, and four crosspieces, or each category of pole could be coded as a

![Figure 3.1 Presence/absence coding method. This method of encoding raster data is the best method applied to point data, although it is not limited to points. A grid cell is encoded 1 if the object is present somewhere within the grid cell, 0 if it is not.](image_url)
separate theme. In this way, the original theme contains a preselected category of nominal data. Then for each, we still have the ability to store, isolate, and retrieve additional attribute data as tables in the database extension for each of the specific pole types. This has some advantages by simplifying the data and allowing ease of search for, say, poles with a single crosspiece but that have not been visited for maintenance for over 6 months. If your raster GIS does not have a direct link to a database management system (DBMS), it is probably wise to produce themes with multiple categories rather than preselect each category. In this way, the grid cells themselves contain more information because of the categories, thus reducing the need for the extended DBMS.

Of course, point objects can also be coded on the basis of ordinal, interval, and ratio categories. As before, we can encode each pole either simply as a pole, with its ordinal, interval, and ratio categories stored in an extended database as attributes, or we can explicitly encode them as numbers for simple raster databases. And as before, they can be encoded using the presence/absence methodology. Some examples are shown in Table 3.2. Note the differences between the simple and extended raster models.

We have seen that there are many options as to how these point features are selected for encoding. The systematic coding strategy we used was the presence/absence method. However, we are not limited to this method for such point objects. Although the other three systematic methods—centroid of cell, percent occurrence, and dominant type (DeMers 2000a) are not particularly useful for point objects, there is one nonsystematic raster coding method that can be used. This approach, called the most important type method (Environmental Systems Research Institute Staff 1994), allows the user to selectively isolate types of objects to be included while omitting others. This can be exemplified by selecting and encoding only those power poles that are in need of inspection in a given theme. Of course, any additional thematic data can be included in tables within the extended raster model. The most important

### Table 3.1: Extended Raster Model

<table>
<thead>
<tr>
<th>Value</th>
<th>Count</th>
<th>Size</th>
<th>Type</th>
<th>Inspection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>18&quot;</td>
<td>Wood</td>
<td>1/20/97</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>22&quot;</td>
<td>Metal</td>
<td>9/19/99</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>24&quot;</td>
<td>Metal</td>
<td>9/30/99</td>
</tr>
</tbody>
</table>

Presence/absence phenomena are easier to handle with the extended raster model that allows for the inclusion of additional attributes. In this case, we are looking at power poles, each of which is of a different size, type, and inspection date.

### Table 3.2: Theme: Power Poles

<table>
<thead>
<tr>
<th>Value</th>
<th>Count</th>
<th>Size</th>
<th>Type</th>
<th>Inspection</th>
<th>Crosses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>18&quot;</td>
<td>wood</td>
<td>1/20/97</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>22&quot;</td>
<td>metal</td>
<td>9/19/99</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>24&quot;</td>
<td>metal</td>
<td>3/30/99</td>
<td>2</td>
</tr>
</tbody>
</table>

The presence/absence method of encoding can be extended with tabular data to include multiple descriptors.
Figure 3.2 One problem of presence/absence encoding. There are often situations in which two or more objects can be contained within a single grid cell, but without using an extended raster model, there is no way to note this.

type method gives the user much more control over what is important in the thematic data before modeling begins.

As we've seen, the point object has no real spatial dimension, yet the grid cell does. This creates some potential problems for situations in which two or more point objects occur within the same geographic extent of a single grid cell. For example, we may have two or three power poles that happen to occur within the extent of a grid cell (Figure 3.2). If you include a single value for each pole and you have three poles within this grid cell, you are limited to encoding only one of them with a simple raster data model. Simplifying your theme so that you can isolate poles of a particular type may eliminate this problem if each of the three poles is separable by type. Another alternative is to select a smaller grid cell size so that each pole would occur within a separate cell. Yet another approach is to encode each grid cell as either having or not having poles, then using numerical values to indicate the number of poles included for each grid cell. This approach does limit the utility of the extended raster model because you may have more than one type or category of pole at each grid cell. The extended raster model offers a similar approach to our last one in that each grid cell could be encoded as to the presence or absence of telephone poles and then the RDBMS could contain specific information about how many poles were there. As with the previous method, however, it limits the addition of attribute information. Because RDBMSs usually assume a single value for each cell (first normal form [DeMers 2000a]), you would be limited to attributes for only one pole at a time. This can be avoided by ignoring the first normal form and including separate columns for each of the three poles (Figure 3.3). Doing so will make querying the database more difficult and will limit the utility of your database. In the example from Figure 3.3, you would have to search for the value of 1 to indicate that there was a pole to begin with and then continue to search the other pole descriptors as well. This problem is well illustrated by the Mount Desert Island, Maine, database used in Exercises in GIS (DeMers 2000b).

Line features share the same levels of geographic data measurement (i.e., nominal, ordinal, interval, and ratio) as well as the same encoding options. Because line data have only one dimension, length, raster data structures generalize the spatial position of these features (Figure 3.4). In this way, a road or path with a width of 15 meters, for example, would occur somewhere within a grid cell that might be 100 meters in diameter. This again raises the need to be careful when selecting the grid cell size to encode line data.

Selection of an appropriate raster encoding scheme for linear data presents much the same set of problems as for point data. Because line data typically occupy a very
small portion of the grid cells, the use of percent occurrence, dominant type, and centroid of cell methods of raster data encoding is not appropriate. We are left with presence/absence and most important type methods. As with the point data examples, presence/absence would most often be used when the categories are kept Boolean (e.g., either we have roads or we don’t). To include more or different categories or varieties of line data, we could use the most important type method. Whichever of these methods we select, we still run the risk of having more than one linear feature appearing within the limits of a single grid cell. In fact, because roads, paths, and railroads frequently cross, the possibility is even higher than that for point data. Careful selection of line data categories allows us to avoid this problem in most situations (Figure 3.5). Of course, we still have the same options, as illustrated by point data with this same problem.

Linear data can be categorical, such as roads versus paths versus railroads, each of which is normally coded within a particular theme (Figure 3.6). The previous examples would most likely be included in a theme called transportation. In this way, each category could be included within a single theme, which would be a relatively compact way of storing them. Line data can also be ordinal (e.g., single-, double-, or multiple-lane highways), interval, or ratio (e.g., based on traffic flow or measured width). Each of these could be recorded as a separate theme, or they could be included as a single theme (e.g., roads), just as in the case of nominal categories (Figure 3.7). For extended raster data models, we also have the opportunity to store additional nominal, ordinal, interval, and ratio attribute data when necessary. And, as with point data, we could encode the roads as a single category, *roads*, and then
Figure 3.4  Generalization of lines in raster. The use of grid cells for line objects presents the problem of positional inaccuracy due to the size of the grid cell in geographic space.
Figure 3.5  Most important type method of raster encoding. By making decisions about what objects are most important prior to raster encoding, you eliminate the problem of having two or more objects intersecting at the same grid cell.

add any additional attributes, such as traffic accident reports and road condition, to our tables. In this case, if, as in the case of multiple points occurring within a single grid cell, we have several road types crossing a single grid cell, we run the risk of not being able to encode all the attribute data without violating the first normal form (Figure 3.8).

Area features have two dimensions and the same options for level of geographic data measurement for their associated attributes. The two-dimensional patterns of area features offer some interesting options for encoding not available to point and

Figure 3.6  By separating the themes out rather than keeping them combined, you can include more than one type of categorical data for a single grid cell. Note, for example, how the upper left-hand grid cell can now contain both roads and railroads.
Figure 3.7 Coding ordinal data. Note how the different ordinal categories of road can also be represented by separate themes. Alternatively, they could be included as tabular data within a single theme by using the extended raster data model.
Figure 3.8 Extended raster data model. The incorporation of line data attributes is made much easier by employing the tables within the extended raster data model. Here, we see an example of ordinantly ranked linear data with additional descriptors.
line data. Under selected circumstances, any of the four basic systematic coding schemes can be used, as well as the unsystematic most important type method. Typically, polygonal thematic data do not overlap within a single theme, but this does not eliminate the problem of multiple themes occurring within a single grid cell.

CONCEPTUALIZING THREE-DIMENSIONAL SPACE WITH GRID CELLS

Statistical surfaces are often a fundamental portion of raster databases, especially, but not exclusively where terrain modeling is concerned. By definition, statistical surfaces do not include nominal or ordinal data. Most are ratio-level data, although temperatures in Fahrenheit or Centigrade scales are a classic example of interval scale surface data. These surfaces can also include such data as chronology, attraction between objects (e.g., gravity-model data), barometric data, and precipitation. In each case, statistical surfaces require that the data either be continuous or be at least assumed to be continuous, and therefore comprising an infinite number of points. This requires that the statistical surface must be sampled to be encoded into any computer tessellation, whether it is vector or raster. For raster encoding of surface data, the procedure is somewhat different than the five typical schemes we have already visited. Instead, one records a unique z value for each grid cell. The primary decision that must be made is exactly where, within the grid cell, the value is encoded. In most cases, either the centroid of the grid cell or one of the four corners is assumed. Although this decision will have implications for modeling, the important thing is to be consistent for all your data.

In most cases, surface data do not have attributes other than the z value for each sampled or interpolated grid cell value. Older, simple raster GIS software usually recorded a single integer value for each grid cell (Figure 3.9), whereas newer simple and extended raster models often record single rational values for each grid cell. Even in the case of extended raster models, additional attribute data are not normally included in additional tables. As we saw in Chapter 2 (Figure 2.5), the only values recorded are the \( X, Y \), and \( Z \) coordinates.

Actually, statistical surfaces are often termed 2.5-dimensional rather than three-dimensional because they do not explicitly allow depth modeling. Scott (1997) has demonstrated that Map Algebra and raster data models can be extended to explicitly include volumetric data and information. Although these data models are currently somewhat experimental, it is likely they will become operational in the near future. For this text, we will abandon this extension of the statistical surface and the raster data model so we can focus on readily available data models and software. It is, however, important to acknowledge this innovation for future modeling capabilities, particularly with the advent of new object-oriented GISs.

![Contour and Raster](image)

**Figure 3.9** Simple raster representation of elevation surfaces. Note the translation of contours on the left to discrete raster values on the right.
THINKING ABOUT THE MATHEMATICS OF MAPS

As we have observed, the most flexible and elegant conceptualization of the raster GIS coverage or theme is one of a series of numerical values arranged in rows and columns, as in the MAP model. More specifically, and largely on the basis of this model, we can view each raster GIS theme as a two-dimensional array of attributes, each represented by some mathematical value (or values, in the case of the extended MAP model), whose locations on the ground are implicitly encoded on the basis of the row and column position in the array. Moreover, each grid cell location for each additional theme must, to be of use in modeling, explicitly coincide with its column–row counterpart in the other themes.

A fundamental, intuitive understanding of this construct is absolutely essential for effective raster GIS modeling. All of the operators, functions, flow control procedures, and iterative techniques necessary to create and deliver models depend on it. This is the equivalent of understanding the chessboard, its red squares versus its black squares, and the rules imposed by that particular structure. Very soon, we will move to the operations and functions of the raster GIS on the basis of its imposed structure. To continue the chess analogy, this is equivalent to understanding the movements of the individual chess pieces and the rules, capabilities, and limitations of each piece. You would not begin playing chess without understanding both the game board and the pieces. Likewise, we will learn more about the GIS equivalents. And, just as with chess, we will eventually move forward to strategy, movement combinations, offenses, and defenses so that we will become first competent, then proficient, perhaps accomplished, and, with plenty of practice, eventually expert modelers.

A COMPARISON WITH AND CONTRAST TO MATRIX ALGEBRA

Map Algebra can be envisioned as the rules and operational procedures that are employed within the MAP raster data model and the capabilities it presents us. As we have already seen, map algebra is based on the fundamental MAP data model, especially on the two-dimensional array concept for each theme. In mathematics, a two-dimensional array allows a set of mathematical procedures called matrix algebra to be applied to combine, compare, and manipulate the numbers of the matrix. Therefore, we can add matrices of numbers by taking each numerical value at each location for each matrix and adding it to its corresponding value. For example, consider the following matrix algebra equation:

\[
\begin{array}{c}
5 & 4 & 1 \\
2 & 1 & 2 \\
4 & 2 & 1
\end{array} + \begin{array}{c}
3 & 2 & 1 \\
1 & 4 & 5 \\
2 & 7 & 3
\end{array} = \begin{array}{c}
8 & 6 & 2 \\
3 & 5 & 7 \\
6 & 9 & 4
\end{array}
\]

Notice how the upper left number (5) in the first matrix is added to the upper left number (3) in the second matrix to arrive at the upper left number (8) in the output matrix. The same can be said for the remaining eight numbers in each of the matrices. Now let's make a subtle change to the matrices to produce the following:

\[
\begin{array}{c}
5 & 4 & 1 \\
2 & 1 & 2 \\
4 & 2 & 1
\end{array} + \begin{array}{c}
3 & 2 & 1 \\
1 & 4 & 5 \\
2 & 7 & 3
\end{array} = \begin{array}{c}
8 & 6 & 2 \\
3 & 5 & 7 \\
6 & 9 & 4
\end{array}
\]

Notice how the matrices now have grid cells around them. This indicates that the matrices in matrix addition are virtually identical to the grid cells used in Map Algebra and that the process of adding matrices is identical to the process of addition
within Map Algebra. It could also be visualized with a more typical display of grid cell maps (Figure 3.10).

Just as matrix addition has its exact counterpart in Map Algebra, so does matrix subtraction. For each grid cell location in the first matrix, the corresponding grid cell in another matrix of numbers is subtracted to obtain the resulting values. And the same idea is again directly translated into Map Algebra, where each grid cell in one theme is subtracted from its corresponding grid cell in another theme.

If you have studied matrix algebra, you are aware that this one-to-one locational correspondence does not apply for such functions as multiplication, division, roots, and powers. This is where matrix algebra and Map Algebra part company. In Map Algebra, the locational one-for-one translation is maintained. Thus, to multiply the following simple 4 \times 4 cell thematic maps within Map Algebra, we maintain the same rules that we applied to matrix addition and subtraction (Figure 3.11). The retention of this simple rule is essential because, unlike in mathematical matrices, the position of individual grid cells in raster themes directly corresponds to their position in geographic space. A result of this basic rule for our Map Algebra game board (our MAP-like data model) is that our grid cell values can be modified but their locations are not transposed or moved. All of the basic operators, functions, flow control, and iteration operations of Map Algebra and extensions to Map Algebra depend on this. Additionally, this knowledge is essential for those who use macro programming techniques to enhance and modify the basic data model and its user interface. We will discuss this topic later. For now, we will examine some of the basic operations available within software packages that employ some form of Map Algebra. As you go through the next sections, remember that each raster GIS approaches the use of Map Algebra differently. Try to keep the concepts rather than the commands uppermost in your mind as you learn how Map Algebra works.

**AN INTRODUCTION TO MANIPULATIONS WITH MAP ALGEBRA**

Despite its simple structure, or perhaps even because of it, Map Algebra is a very robust modeling language. Some form of it is employed in many well-known raster
GIS software packages, such as GRASS, ERDAS GIS MOD, ArcView Spatial Analyst, and Arclnsf GRID. Some will allow direct stacking of themes so that comparative Map Algebra calculations can be performed directly from theme to theme, whereas others simulate this process with the use of some form of Map Algebra calculator. The result is, for the most part, the same.

In our previous section, you might have gotten the idea that Map Algebra is simply a modified version of matrix algebra. Although this, in itself, is actually a major accomplishment in raster map manipulation, it is far from enough. Map Algebra is actually a complete modeling language, rapidly becoming the standard in the industry, that allows for program control, macro development, and iteration programming, as well as allowing for mathematical manipulation of theme grid cell values. In fact, even the analysis and manipulation of these grid cell values is not limited to mathematics but also includes a vast array of logical expressions that can be used to compare thematic values within single themes and among multiple themes. Thus, by combining elementary mathematical and logical operators into more complex functions and by using control and iteration, we can create complex models based on strategies that suit our particular data and modeling needs. Before we begin such complex modeling, we will first look at what types of operations are available to us.

**Operators**

The most basic functional characteristics of GIS packages based on the Map Algebra modeling language are the same operations with which we operate in most other modeling domains. As I have already suggested, this group of characteristics, called operators, can be divided into several groups—arithmetic, relational, bitwise, Boolean, combinatorial, logical, accumulative, and assignment. As you might guess, these include the basic functions often associated with formula translation computer languages such as FORTRAN (formula translation). Table 3.3 provides the typical sets of operators available.
### TABLE 3.3 Operator Groups

<table>
<thead>
<tr>
<th>Operator Group</th>
<th>Operator</th>
<th>Operator</th>
<th>Operator</th>
<th>Operator</th>
<th>Operator</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>mod</td>
<td>...</td>
</tr>
<tr>
<td>Relational</td>
<td>&lt;</td>
<td>&gt;</td>
<td>==</td>
<td>&gt;=</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Bitwise</td>
<td>&lt;&lt;</td>
<td>&gt;&gt;</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Boolean</td>
<td>&amp;&amp;</td>
<td></td>
<td>!</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Combinatorial</td>
<td>and</td>
<td>or</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>in</td>
<td>diff</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Accumulative</td>
<td>+=</td>
<td>*=</td>
<td>-=</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**Arithmetic Operators**  As you can see, there are many options for manipulating the grid cell values. The arithmetic functions include addition, subtraction, multiplication, division, and modulus (integers only), all required for constructing mathematical models within the raster GIS. Most of these operators will work equally well on integers as on rational numbers. The results of the operations will be based on the types of data used in the manipulations. For example, if only integers are used, the results will be integers. If floating point numbers are used in any of the operations—say, for example, floating point values multiplied by either integers or floating point values—the resultant numbers will be floating point. The only arithmetic operator that is limited by data type is the modulus (MOD) operator, which always returns integers. If MOD is applied to rational numbers, any remainder will be truncated and the result will be converted to integer. No matter which arithmetic operator you use, if there are cells without values (this assumes your GIS allows missing values or no-data values), the result will always be no data. Keep in mind that for most GISs, a zero contained within a grid cell is not necessarily the same as no data or missing data. Most often, the no-data category is contained within tabular cells in an extended raster data model. Figure 3.12 illustrates a typical use of the arithmetic operator based on multiplying two themes together.

**Relational Operators**  Relational operators are the kinds of operators you would normally expect to find within an RDBMS for use with its tables. In short, these operators evaluate a condition. If the condition is false, the output is assigned a 0, and if the condition is true, the output normally returns a 1. As with arithmetic operators, the no-data condition, when evaluated, results in no data. Conditions to be evaluated include greater than, less than, greater than or equal to, and many more. Relational operators operate on both integer and floating point numerical values and require at least two input values for comparison. Figure 3.13 demonstrates the use of the relational operator >= where it compares the numerical values of an input matrix to determine which of its grid cell values are greater than or equal to the values of the second matrix. Note how the no-data cells always return no-data cells after evaluation.

**Boolean Operators**  Boolean operators employ Boolean logic (true/false) and evaluate conditions, as we saw with relational operators. And as with relational operators, they return 1 for true and 0 for false; operate on grids, scalars, numerals, or combinations; and require at least two input values. No-data values are evaluated as no-data. Figure 3.14 shows a typical relational operator using the & (and) function, also known as the intersection. The figure demonstrates how grid cells that have val-
Figure 3.12 A $4 \times 4$ matrix example of map multiplication shows how two input matrices are multiplied to obtain the final result. This is functionally identical to the example in Figure 3.11 but shows that each set of values represents a map as a matrix of values.

Figure 3.13 Relational operator. The grid cell values of the first input matrix are compared to those of the second. When the values of the first are larger than those of the second, a value of 1 is recorded in the output matrix. A value of 0 is recorded when this is not the case.
Figure 3.14  **Boolean operators.** The two input matrix values are compared to evaluate presence or absence of values in both sets of grid cells. When nonzero values exist in both, a truth value of 1 is recorded. If one or more corresponding grid cells contains a 0, a value of 0 is returned. Finally, when one or more values are missing (i.e., no data are present), the software returns no data.

Values in both themes are output as 1's whereas those that lack values in at least one of the two themes are assigned a value of 0. Note also the no-data output cells.

**Bitwise Operators.** Bitwise operators compute on a binary representation of a single (input) set of matrix values and work only on integer values. If rational numbers are used as the input, they will be truncated first before they are evaluated, which means that the output values are always going to be integers. As with Boolean and relational operators, the results of using no-data values will always result in no-data output. Figure 3.15 demonstrate the use of the bitwise operator $<<1$, meaning all nonzero values will be converted to their binary equivalents. As you might imagine,

<table>
<thead>
<tr>
<th>Input Matrix</th>
<th>Output Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1</td>
<td>2 0 2 2</td>
</tr>
<tr>
<td>2 4 1</td>
<td>4 8 2</td>
</tr>
<tr>
<td>1 2 4 2</td>
<td>2 4 8 4</td>
</tr>
<tr>
<td>2 1 4 2</td>
<td>4 2 8 4</td>
</tr>
</tbody>
</table>

= No Data

Figure 3.15  **Bitwise operators.** Bitwise comparison of grid cells to a value of 1. Values of 1 are converted to their bitwise equivalent (2), values of 2 are converted to 4, and so on.
numbers in the input matrix that are equal to 1 will be converted to 2 in the output, 2 will be converted to 4, 4 to 8, and so on.

**Combinatorial Operators** Combinatorial operators share much in common with Boolean operators, except that they assign specific values to the results of evaluating two or more themes or matrices. This is a generalization of the Boolean operator, where a true condition is evaluated as a value of 1 and a false condition is evaluated as 0. In the case of the combinatorial operator, if both (or all) input values are evaluated as true (nonzero), the cell locations on the output grid are assigned some numerical value that preserves the uniqueness of the combination. So, for example, a 1 compared with a 2 would be assigned a different output value than a 1 compared with a 3. These numbers are asymmetrical also, in that a 1 on the first matrix compared with a 3 on the second will not return the same output value as a 3 on the first matrix compared with a 1 on the second matrix. Some older software (e.g., OSU-MAP-for-the-PC) allowed the user to specify the output values as the comparisons were made. To ensure that the unique results from analysis of asymmetrical input and output is preserved, in some software, such as ArcGrid, the assigned values are set on the basis of the order in which they are evaluated. Thus, for example, if the comparison of a 1 in matrix 1 with a 4 in matrix 2 is encountered before any comparison of a 4 in matrix 1 with a 1 in matrix 2, the 1 versus 4 is assigned a lower value than is a 4 versus 1. Table 3.4 provides an example of how this might be achieved. Notice that when similarly ordered value sets (e.g., 1 compared with 4) in one pair of grid cells is encountered again, the same value will be assigned. In this way, all identically ordered grid cell combinations will receive the same output value. Figure 3.16 shows a simple example of the application of combinatorial operators.

**Logical Operators** In addition to the Boolean operators we have already seen, there are some additional operators employing set-based logic. Three basic logical operators are generally included in this set—difference (DIFF), contained in (IN), and OVER. Each of these operators typically compares values based on pairs of matrices. DIFF compares two input matrices to determine whether the values in each matrix are the same or different. Although this is not universal, most software retains the first input matrix value if it is different from the matrix to which it is compared. If the

<table>
<thead>
<tr>
<th>Matrix 1</th>
<th>Matrix 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

With combinatorial operators, the geographic information system modeler can decide what values are applied to each combination of grid cells. Each paired comparison is asymmetrical in that a 1 compared with a 2 is not the same as a 2 compared with a 1.
two matrix grid cell values are the same, the software returns a 0. In other words, in
the output, all nonzero numbers indicate some change, perhaps a change from one
time to another.

The IN operator, like the DIFF operator, accepts and compares two inputs, but they
need not be grid maps. In most cases, the first input is an expression (typically a list
or grid) and the second input is a set of numbers. The idea is to preselect a set of
numbers against which you wish to compare the values in your grid matrix. If, for
example you want to isolate several (let us say five) land uses coded with five indi-
vidual integer values, and you further wish to zero out all others (sometimes called
a mask in remote sensing), this technique is very useful. The output retains all val-
ues from the first input that are also contained in the second input (the set). Those
that are not found in the second input are set to zero on output.

The OVER operator, which also accepts two inputs, searches for zeros. All nonzero
values from the first input matrix are returned as output. If a zero is discovered, the
software will return the second input value as output. This operation is similar to the
IN operation, except that both inputs are matrices. Figure 3.17 is an example of the
OVER operation.

Accumulative Operators The accumulative operators are designed for latter
movement across a raster map, especially for cumulative surface analysis through
scanning types of operations. A single grid is used as input and assigned an accu-
mulative result to a scalar. If, for example, you were to use a + operator, the GIS
starts at a corner (upper-left cell in a matrix is used by GRID), then moves to the first
cell to the right and adds the value of that cell to the value of the first cell. Then the
scan proceeds to the next cell and adds its value to the sum of the values of the first
two cells. The process then continues until you run out of grid cells. As with other
operators we have seen, no-data cells are ignored; in this case, they are not used at
all for accumulation calculations. Figure 3.18 illustrates the + accumulative opera-
tor case.
### Logical Operators

The **OVER** logical operator returns the value from the first input matrix unless a 0 is encountered in the first matrix, in which case the second grid cell value is returned.

<table>
<thead>
<tr>
<th>First Input Matrix</th>
<th>Output Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4 0 8</td>
<td>1 4 9 8</td>
</tr>
<tr>
<td>5 6 1 3</td>
<td>5 6 1 3</td>
</tr>
<tr>
<td>6 1 0 2</td>
<td>6 1 3 2</td>
</tr>
<tr>
<td>0 2 0 7</td>
<td>4 2 1 7</td>
</tr>
</tbody>
</table>

The **OVER** operator is defined as:

\[
\begin{array}{cccc}
1 & 4 & 0 & 8 \\
5 & 6 & 1 & 3 \\
6 & 1 & 0 & 2 \\
0 & 2 & 0 & 7 \\
\end{array} \Rightarrow \begin{array}{cccc}
5 & 1 & 9 & 1 \\
2 & 5 & 0 & 7 \\
1 & 0 & 3 & 1 \\
4 & 7 & 1 & 8 \\
\end{array}
\]

### Assignment Operators

The final group of operators is the assignment operators. These store the results of expressions in an output (normally a grid cell matrix). This operator is actually as simple as assigning all cells of an input grid to a single value. More complex mathematical expressions and arithmetic operators can also be included. So, for example, an output matrix could be created by multiplying one theme by another, or an output grid matrix could be created by multiplying a grid matrix by a single value (e.g., an input grid matrix multiplied by 5). This latter case is demonstrated by Figure 3.19. Additionally, tabular values from an extended raster model could also be used to manipulate the raster values in a separate grid.

<table>
<thead>
<tr>
<th>Input Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1</td>
</tr>
<tr>
<td>2 4 1</td>
</tr>
<tr>
<td>1 2 4 2</td>
</tr>
<tr>
<td>2 1 4 2</td>
</tr>
</tbody>
</table>

The **+** operator is defined as:

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
2 & 4 & 1 \\
1 & 2 & 4 & 2 \\
2 & 1 & 4 & 2 \\
\end{array} \Rightarrow \begin{array}{cccc}
1 & 0 & 1 & 1 \\
2 & 4 & 1 \\
1 & 2 & 4 & 2 \\
2 & 1 & 4 & 2 \\
\end{array} \Rightarrow VALUE = 28
\]
Functions

Functions are higher-order GIS operations built up of the more basic operators we have just examined and designed to provide a vehicle for model implementation. Functions are grouped as local, focal, block, zonal, global, and special types. In the next chapter, we will look more closely at the functions; however, we will provide some basic concepts and definitions here as important building blocks of the Map Algebra modeling language.

Local functions, also called by-cell functions, are designed to operate on a cell-by-cell basis. In other words, each grid cell at a given location in a first matrix is operated on either by an expression or by a grid cell at a corresponding location in another matrix.

Focal functions, or by-neighborhood functions, characterize or reclassify a selected cell on the basis of characteristics of a predefined neighborhood of grid cells.

Block functions, or by-area functions, are similar to focal functions in that they do evaluate groups of grid cells to perform its reassignments. In this case, however, the results are assigned to whole blocks of cells, and the groups of cells do not overlap.

Zonal, or by-zone, functions use zones identified from another coverage to evaluate and reclassify a target cell. The zones are actually geographic areas, whether contiguous, fragmented, or perforated, and are usually defined by internal attribute homogeneity.

Global functions, as the name implies and unlike the previous types of functions, tend to operate on the entire grid matrix all at once. These are the types of functions most often associated with Euclidean geometric analysis, distance and shortest path analysis, and visibility and viewshed analysis.

Beyond these are an even more complex set of special functions, often based on selected integration of simpler functions. These specialized functions are primarily used for such functions as complex geometric analysis, hydrological modeling and characterization, and surface analysis and characterization.

Not all GIS software contains all of these types of functions, but most professional raster GIS software does contain macro language capabilities for implementing these capabilities. As you might already suspect, the existence of a MAP-like data model is useful, if not essential, for optimal model implementation. Before such implementation can take place, however, it is necessary to be able to issue commands that control the flow of operations. This flow control functionality is explained next.
Flow control is an integral component of Map Algebra. It provides for basic functions such as starting and stopping, and for a set of conventions that can be combined to create a command line framework within which the user can interface with the GIS software. Although an early variant of this framework was first developed by Tomlin and Berry (1979) for the original Map Analysis Package, it has been modified to a lesser or greater degree for all of the variants of this original data model. As such, the exact nature and structure of the flow control framework will vary from package to package. We will adopt the same format employed by Tomlin (1990).

Whether your GIS software uses a graphical user interface (GUI) or is strictly command-line driven, the process is essentially the same because the commands and structure of the language are embedded in the GUI. The basic format for flow control is composed of two distinct, yet linked elements—statements and programs—that work with the operations and functions we have already seen. These two elements are linked in a hybrid language that resembles elements and structure of both algebra and English. Some non-English versions of this flow control language also exist.

**Statements** A statement is a verbal representation of the operations. It provides a declarative command structure that links operators, functions, and programming commands in a logical progression. Much like declarative computer languages, the order of operations is vital to proper functioning of the model. In fact, some GIS packages use a set of flow control procedures that resemble such computer languages as FORTRAN or versions of C or BASIC. For ease of use, I will describe the typical form of the language that more closely resembles a declarative English sentence. Because of its flexible, natural language-like structure, this flow control methodology can also be extended to create algorithms in higher-level GIS macro languages.

The statements used in the flow control methodology include an ordered sequence of letters, numbers, symbols (representing operators), and blank spaces. As with a declarative natural language sentence, the sequence forms a declarative statement that indicates the subjects under consideration, modifiers to the subjects, and objects on which the subjects will act. Consider the following statement:

\[
\text{TotalCostMap} = \text{LocalSum of FirstCost and SecondCost and ThirdCost}
\]

This statement begins with the subject. (For most modern GISs, this would be the name of output maps.) In this case, the subject is TotalCostMap. This is the thematic map or other output that will result when the operations on the right of the equals sign are executed.

Statements also have the ability to accept modifiers that correspond to prepositions, adjectives, adverbs, nouns, conjunctions, and even punctuation that add significance or change the meaning of the sentence. The equal sign in the statement above is a modifier because it acts as a verb providing action on the subject. Other modifiers include the preposition of and the conjunction and. These words provide meaning, enhance meaning, and link other terms in the statement just as they would in a normal sentence.

The objects LocalSum, FirstCost, SecondCost, and ThirdCost provide items and actions on those items that, together, create the output (the subject). In this case, the LocalSum term is a function (a local function, in fact), and it operates on each cost grid cell for each of three separate matrices by adding them together. For most Map Algebra-based raster software, the objects may be map names (as in the case above), nouns, adverbs, or numbers. Special codes are also possible that represent special values such as the null set (−0), the highest value (++) the lowest value (−−). The code
... can be used to indicate a missing portion of a numerical series, such as 1 2 ... 8 9 10, which indicates that the sequence values 3 4 5 6 7 are part of the arithmetic series.

In some cases, the actual values of portions of a Map Algebra statement are either not known or will be evaluated, and therefore created, during evaluation. These portions are called variables and are analogous to variables in a traditional programming language. In the traditional implementation of Map Algebra, these variables are noted by all-capital letters. Thus, a statement could look like this:

\[
\text{NewMap} = \text{LocalSum of FIRSTMAP and SECONDMAP and THIRDMAP and FOURTHMAP}
\]

Where the objects FIRSTMAP, SECONDMAP, THIRDMAP, and FOURTHMAP suggest the nature of the values that will, on evaluation, replace the generic variables. This generalized statement allows for flexible algorithm development and implementation no matter what the actual values will be.

To add more flexibility to the programming environment, Map Algebra also allows the inclusion of optional portions of a statement. This might be required if, as in the previous statement, we know that at least two of the maps will be needed in the LocalSum operation whereas the remaining maps are optional and need be evaluated only if they are present. The modified statement often uses brackets to define which of the portions are optional. So the previous statement would be rewritten to look like the following:

\[
\text{NewMap} = \text{LocalSum of FIRSTMAP and SECONDMAP \{and THIRDMAP and FOURTHMAP\}}
\]

The assumption from the statement above, however, is that there are four potential maps to be evaluated and that only the first two are required. What if you are not aware of exactly how many maps will be included in the LocalSum operation? Fortunately, Map Algebra allows for this with still more flexibility by adding the term etc., which indicates that the statement portions enclosed within the brackets can be continued. Rather than explicitly including the variables THIRDMAP and FOURTHMAP in brackets, we use a more generic variable such as NEXTMAP. The statement for NewMap based on the LocalSum can now be rewritten thusly:

\[
\text{NewMap} = \text{LocalSum of FIRSTMAP and SECONDMAP \{and NEXTMAP\} etc.}
\]

This shows that any number of maps beyond the first two can be included in the calculations in addition to the mandatory first two.

As you have observed from the statements above, we have been using the term LocalSum. From our preliminary discussion of functions, you should recognize that this is a member of the group of functions we called local or by-cell. Any function, no matter which group it belongs to, can be incorporated into our statements to create procedures that will evaluate our grid maps. In the next chapter, we will look more closely at the functional options available in Map Algebra-based GIS packages. Before that, however, we need to briefly examine programs and iteration.

Programs The notational representation of a procedure in Map Algebra is called a program. Although this sounds like the same thing as a statement, it is not. Rather, a program is an ordered sequence of statements, each statement placed on a separate line and collectively designed to perform a wide array of interconnected activities. You should note that I used the term ordered sequence. This indicates that, just as in any other programming language, the order in which the statements appear will most often indicate the order in which the processing takes place. The program is the notational specification of a GIS model and indicates which maps will be included, what operators will be applied to each specified grid cell or cells, what intermediate
maps will be produced, and how they in turn will be manipulated. You should also note that I stated that the order of statements will most often indicate the processing order. This would assume that the program is strictly linear and that each step directly stems from the one previously implemented before it. This would limit our programs to very simple tasks. We need to add one more feature to our Map Algebra language to provide us with more flexibility for modeling in complex situations.

**Iteration**

Fortunately, Map Algebra is a robust language allowing the GIS modeler or programmer to vary the order in which statements occur. For example, there are steps that one might wish to skip under specified circumstances. Thus, if the evaluation of a given statement results in values that do not achieve a specific threshold value for our model (say, a threshold value for total erosion on a soil map in an evaluation of housing sites), we might be able to eliminate procedures that would otherwise be used to add financial resources to make development economically viable. We might also have a need to insert some operations under given situations. In fact, we could rewrite our Map Algebra program of housing site development in such a way that we would normally assume that the need for additional financial input would not be encountered, and if the erosion threshold is exceeded, we could then ask the program to include the necessary statements and procedures to implement the addition of financial resources. Essentially, this is the reverse of the first situation. These two situations are akin to an “if-then-else” type of programming statement available in nearly all modern programming languages.

There will also be common situations in which some procedures and statements will need to be repeated to achieve the final result. This is often the case when statistical or numerical procedures require multiple steps for their final evaluation. In the case of map algebra, a classic example would be a model in which many time steps are to be processed. Each successive step must be performed and its intermediate map output stored and then retrieved for the next time step. The computer language analog would be a “do loop” structure that requires that operations continue until some predefined stopping point or until the data set is exhausted.

Finally, you should think of Map Algebra as a complete high-level spatial modeling language that includes basic elements (operators), more complex elements (functions), and a formalized structure (statements), together with all the necessary programming features that allow complex models to be developed and implemented. To become proficient at GIS modeling in raster, it is essential that you become familiar with the structure, operating rules, and components of the individual version, implementation, or modification of Map Algebra that your particular software employs. We will look at some of the more powerful functions available within the Map Algebra language in the next chapter. Be sure to compare these to those available with your software before you begin modeling. A thorough knowledge of the modeling capabilities of your software will improve efficiency, inspire more ideas, and suggest new, as yet unimplemented capabilities that will take you beyond the GIS button bar into the exciting and potentially lucrative realm of applications development.

**Chapter Review**

Point, line, area, and statistical surface attribute data can be conceptualized and represented as nominal, ordinal, interval, or ratio scales of geographic data measure-
ment. Each of these measurement scales provides both opportunities and restrictions as to how they are stored either within simple raster models or extended raster models. The dimensionality, the associated measurement levels, and the potential for locational overlap among objects all contribute to determining which of the five raster input methods (four systematic and one unsystematic) would be the most appropriate. Statistical data generally do not have additional attribute values associated with them and are also the most common data represented as floating point (rational) values rather than integers.

The raster tessellation amounts to a matrix not unlike what one might encounter with matrix algebra. Within this construct, most modern raster GIS packages have adopted a modeling language as some variant of Map Algebra. Map Algebra is a formalized, simplified version of matrix algebra that maintains the locational fidelity of each grid cell within the matrix. Beyond simple mathematical procedures, Map Algebra also includes a wide array of relational, logical, combinatorial, accumulative, and assignment procedures. Collectively, these are called operators, which can be combined with higher-level procedures called functions, within a natural language-like structure called statements, to allow program control to build programs used for GIS model development and deployment.

Map Algebra also allows for flow control through ordered sequences of Map Algebra statements called programs. The sequencing of program statements imposes a framework on which maps are selected and operated on in which order to achieve the desired model outcomes. Additional programming flexibility is incorporated through the use of statements that allow some procedures to be skipped, included, or iterated whenever necessary. This flexibility gives Map Algebra the same power and flexibility most often associated with typical computer programming languages.

Discussion Topics

1. What impact does dimensionality of objects have on the selection of appropriate raster encoding scheme?

2. What is nonsystematic raster encoding, and what types of criteria might you employ in selecting and implementing it?

3. Using a hypothetical or a real set of map themes, discuss which themes would appropriately use the extended raster data model and provide some concrete examples.

4. How does the raster encoding methodology selected impact the spatial accuracy of point, line, and area entities?

5. Discuss the impact of point, line, and area objects that occur within the area occupied by a single grid cell. Describe some situations in which this is likely to occur and explain some solutions to the problem.

6. How do the mathematics of Map Algebra and matrix algebra differ? Why are the mathematics of Map Algebra different? Couldn't we simply have added some of the Map Algebra structure to the mathematics of matrix algebra?

7. List and provide a simple, one-sentence description of each of the basic types of functions.

8. Describe some different types of operators and functions and provide examples of how they operate within the statement structure of Map Algebra.

9. Provide examples of statements that illustrate the different aspects of flow control. In your statements, include variables, objects, modifiers, and other statement
parts. When these are completed, label the parts of the statements, much as you would diagram a sentence for an English class.

Learning Activities

1. In this chapter, we learned five separate ways of encoding raster data. Create a coding system in which each code indicates one of these. For example, PA could stand for presence/absence, DT for dominant type, etc. Now create a table that shows the dimensions of the geographic entities on the vertical axis and the encoding scheme methods on the horizontal axis. In each cell, place an X where the coding scheme could be used for each dimension of data type.

2. Provide five examples of how the most important type methodology of grid cell encoding could be employed for raster data. Be specific about how you made your decisions—exactly what basis you are using to decide what the most important type is.

3. Create, or copy real-world examples from available GIS databases, of extended raster database management tables for data of the following types:
   a. Land use example (polygons)
   b. Linear infrastructure example (power lines, street networks, highways, etc.)
   c. Point examples (wildlife, stores, wells, etc.)

4. Pick up a copy of a linear algebra (matrix algebra) text and illustrate examples of matrix multiplication, division, square root, and square. Use real numbers to solve these problems. Now, create illustrations of Map Algebra multiplication, division, square root, and square. Again, use real numbers (identical to those for the matrix algebra examples) and work through the solutions. Describe the results to illustrate the differences between the two.

5. On the basis of the introductory material you have been given in this chapter, create simple examples of Map Algebra statements that include operators and functions to derive the following output maps. The titles of the output maps are meant to be descriptive of the methodology you need to employ. Remember, the output map names are only descriptive. Your results may vary depending on how you interpret what the output map means.
   a. BiggestMap =
   b. SmallestMap =
   c. AverageMap =
   d. DifferenceMap =
   e. TimeChangeMap =

6. Create a simple, fictitious Map Algebra program (an algorithm, because you are not using real data yet) that includes at least three of the statements you just created in question 5 above and any others you wish to use. Describe what the program is doing and what its output is meant to represent. For an extra challenge, try adding at least one flow control statement that allows iteration or condition and also includes the use of a variable.