US Army Corps
of Engineers
Construction Engineering
Research Laboratories

USACERL Technical Report EC-94/13
April 1994

r.mapcalc: An Algebra for GIS and Image Processing

by
Michael Shapiro and James Westervelt

As geographic information systems are enhanced with more robust query capabilities, they evolve from basic spatial data storage, retrieval, and display systems to more powerful spatial modeling systems. Languages that empower a user to design and create spatial models will advance this development even further.

This study describes a map algebra language that can form the foundation for overlay and neighborhood analyses in both raster geographic information systems and image-processing systems. The language combines maps and images in expressions with arithmetic and logical operators and mathematical functions to produce new maps and images. This algebra has been implemented in the Geographical Resources analysis Support System r.mapcalc module. The language syntax is first described and then examples relevant to geographic information systems and image-processing applications are presented.

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED

DO NOT RETURN IT TO THE ORIGINATOR

USER EVALUATION OF REPORT

REFERENCE: USACERL Technical Report EC-94/13, *r.mapcalc: An Algebra for GIS and Image Processing*

Please take a few minutes to answer the questions below, tear out this sheet, and return it to USACERL. As user of this report, your customer comments will provide USACERL with information essential for improving future reports.

1. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which report will be used.)

_____

_____

_____

2. How, specifically, is the report being used? (Information source, design data or procedure, management procedure, source of ideas, etc.)

_____

_____

3. Has the information in this report led to any quantitative savings as far as manhours/contract dollars saved, operating costs avoided, efficiencies achieved, etc.? If so, please elaborate.

_____

_____

4. What is your evaluation of this report in the following areas?

a. Presentation: _____

b. Completeness: _____

c. Easy to Understand: _____

d. Easy to Implement: _____

e. Adequate Reference Material: _____

f. Relates to Area of Interest: _____

g. Did the report meet your expectations? _____

h. Does the report raise unanswered questions? _____

i. General Comments. (Indicate what you think should be changed to make this report and future reports of this type more responsive to your needs, more usable, improve readability, etc.)

5. If you would like to be contacted by the personnel who prepared this report to raise specific questions or discuss the topic, please fill in the following information.

Name: _____

Telephone Number: _____

Organization Address: _____

6. Please mail the completed form to:

Department of the Army
CONSTRUCTION ENGINEERING RESEARCH LABORATORIES
ATTN: CECER-IMT
P.O. Box 9005
Champaign, IL 61826-9005

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1994	Final

4. TITLE AND SUBTITLE
r.mapcalc: An Algebra for GIS and Image Processing

5. FUNDING NUMBERS
4A162720
A896
NN-TS2

6. AUTHOR(S)
Michael Shapiro and James Westervelt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Construction Engineering Research Laboratories (USACERL)
P.O. Box 9005
Champaign, IL 61826-9005

8. PERFORMING ORGANIZATION REPORT NUMBER
EC-94/13

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Office of the Assistant Chief of Staff for Installation Management (ACS(IM))
ATTN: DAIM-ED-N
600 Army, Pentagon
Washington, DC 20310-0600

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Copies are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

As geographic information systems are enhanced with more robust query capabilities, they evolve from basic spatial data storage, retrieval, and display systems to more powerful spatial modeling systems. Languages that empower a user to design and create spatial models will advance this development even further.

This study describes a map algebra language that can form the foundation for overlay and neighborhood analyses in both raster geographic information systems and image-processing systems. The language combines maps and images in expressions with arithmetic and logical operators and mathematical functions to produce new maps and images. This algebra has been implemented in the Geographical Resources analysis Support System r.mapcalc module. The language syntax is first described and then examples relevant to geographic information systems and image-processing applications are presented.

14. SUBJECT TERMS
geographic information systems
spatial modeling system
image processing
geographic resources analysis support system

15. NUMBER OF PAGES
26

16. PRICE CODE

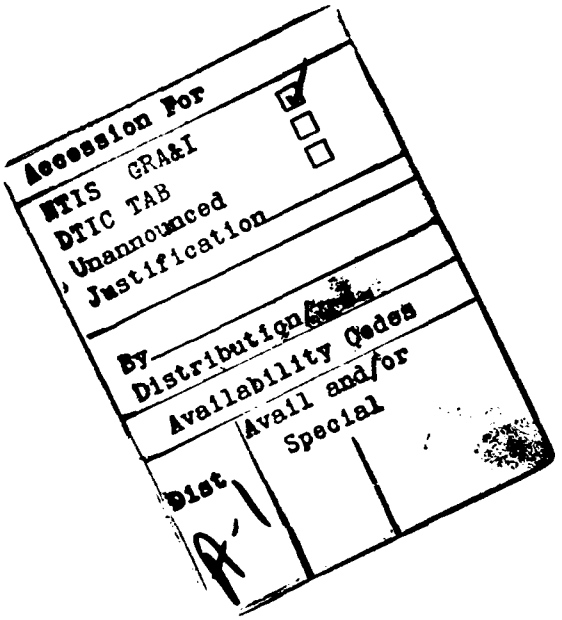
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Foreword

This research was conducted for the U.S. Army Office of the Assistant Chief of Staff for Installation Management (ACS(IM)) under Project 4A162720A896, "Base Facility Environmental Quality;" Work Unit NN-TS2, "Imagery Data for Training Area Management." The technical monitor was Dr. Victor Diersing, DAIM-ED-N.

The work was performed by the Environmental Compliance Division (EC), Environmental sustainment Laboratory (EL), U.S. Army Construction Engineering Research Laboratories (USACERL). The USACERL principal investigator was Michael Shapiro. Dr. John Bandy is Chief, CECER-ED and William Goran is Chief, CECER-EL.

LTC David J. Rehbein is Commander, USACERL, and Dr. L.R. Shaffer is Director.



Contents

1 INTRODUCTION

Background

Because geographic information systems (GISs) have been enhanced with more robust query capabilities, they have evolved from basic spatial data storage, retrieval, and display systems to more powerful spatial modeling systems. Languages that empower an end-user to design and create spatial models will advance this development even further. Tomlin (1990) described a spatial modeling language for raster data with three major classes of map operations: functions of individual cells (e.g., map recoding and Boolean overlay); functions of cells within neighborhoods (e.g., filters and slope-aspect generation); and functions of cells within regions or zones (e.g., area calculations). This classification could be applied to image-processing operations as well because they are inherently raster-based and employ analysis methods very similar to some GIS operations (Burrough 1986).

A GIS map algebra was needed that would accommodate Tomlin's map classifications, allow image-processing operations and could be implemented in the Geographical Resources Analysis Support System (GRASS). GRASS is a public domain, image-processing and geographic information system originally developed by researchers in the Environmental Division of the U.S. Army Construction Engineering Research Laboratories (USACERL) in Champaign, IL. The GRASS system is used to input, analyze, and output geographic data by users in both military and nonmilitary, public and private agencies, based in North America, Europe, and other parts of the world. It is a component of the Integrated Training Area Management program implemented by USACERL to assist in the management of Army training lands.

Objective

The objective of this research was to develop a map algebra language flexible enough to accommodate operations in Tomlin's first two classes of map operations and that would allow developers and end-users to construct a core of GIS analysis and image-processing operations.

Approach

A map algebra language syntax was first defined and developed and then tested and illustrated with examples relevant to GISs and image-processing applications. The sample geographic data used in the examples came from the Spearfish, SD database–a database distributed with GRASS software.

Mode of Technology Transfer

The map algebra has been implemented in the *r.mapcalc* module of the Geographic Resources Analysis Support System. GRASS is transferred to the field through the following mechanisms: training programs, hands-on experience, a user documentation center, newsletters, institutional structures at the Army and Interagency levels, communication networks, and other forums. GRASS source codes can be obtained via Internet computer software from a file transfer protocol server at ftp moon.cecer.army.mil. Current information is available from the GRASS Information

Center at USACERL, which can be contacted by phone at 217-373-7220, by fax at 217-373-7222, or by e-mail at gic@zorro.cecer.army.mil.

2 LANGUAGE SYNTAX

The *r.mapcalc* algebra employs mathematical operators and functions in expressions involving maps or images that are stored in raster grid-cell format and are two-dimensional matrices of integer values.

The syntax for the algebra is *result=expression*, where *expression* is built using *maps* and *images*, mathematical *operators*, *functions*, and temporary *variables*. The *result* map is produced by evaluating the expression for each cell in the matrix.

For example, the expression:

$$sum = map1 + map2$$

would produce a map *sum* where each cell is the sum of the values of the corresponding cells in *map1* and *map2*.

Maps and Images

Maps and images are database files stored in raster format, i.e., two-dimensional matrices of integer values. The values stored in maps may represent categorical (e.g., soil types) or continuous (e.g., elevation) data, whereas the values stored in images usually represent continuous data (e.g., satellite sensor data). Although the information represented by the values within a map or an image varies, the data format is the same. Therefore, this distinction between map and image will be dropped, and the term *map* will be used for either.

Map naming rules are flexible; however, a map name must not contain any special characters (operators, parentheses, etc.), and it must be distinguishable from numbers. Common names include *elevation*, *soils*, *vegetation*, *roads*, *band.1*, etc., although the format *12oct89* is also acceptable.

Map names may be followed by a *neighborhood* modifier which specifies a relative offset from the current cell being evaluated. The format is *map[r,c]*, where *r* is the row offset and *c* is the column offset. For example, *map[1,2]* refers to the cell one row below and two columns to the right of the current cell, *map[-2,-1]* refers to the cell two rows above and one column to the left of the current cell, and *map[0,1]* refers to the cell one column to the right of the current cell. This syntax permits the development of neighborhood-type filters within a single map or across multiple maps.

At present, GRASS map files may contain only integer values. However, GRASS does permit floating point values to be associated with each integer value in a map.[1] Maps may be prefixed by the @ modifier which translates the integer values in a map to their associated floating point values.

[1] These values are stored in a separate attribute file known as the "category label file."

Operators

The operators for the map algebra are:

operator	meaning	type	precedence
*	multiplication	arithmetic	4
/	division	arithmetic	4
%	modulus	arithmetic	4
+	addition	arithmetic	3
−	subtraction	arithmetic	3
==	equal	comparison	2
!=	not equal	comparison	2
<	less than	comparison	2
<=	less than or equal	comparison	2
>	greater than	comparison	2
>=	greater than or equal	comparison	2
&&	and	logical	1
||	or	logical	1

They are applied from left to right, with those of higher precedence applied before those of lower precedence. Parentheses may be used to control the order of evaluation. The arithmetic operations have their usual meanings, except that division by zero equals zero. If both operands are integer, the result is integer, otherwise the result is floating point. The comparisons evaluate to 1 if the comparison is true, otherwise they evaluate to 0. The || operation evaluates to 1 if either operand is nonzero (true), and to 0 otherwise. The && operation evaluates to 1 only if both operands are nonzero (true), and to 0 otherwise.

Functions

Functions perform specific operations on a list of expressions and result in a single operand. The *r.mapcalc* algebra includes the following functions:

abs(x)	$\|x\|$
exp(x)	e^x
exp(x,y)	x^y
log(x)	$\ln x$
log(x,y)	$\log_y x$
sqrt(x)	$\sqrt{x}$
atan(x)	$\tan^{-1} x$ $[-90°, 90°]$
atan(x,y)	$\tan^{-1} y/x$ $[0°, 360°]$
cos(x)	$\cos x°$
sin(x)	$\sin x°$
tan(x)	$\tan x°$
min(x,y,...)	minimum of $\{x, y, ...\}$
max(x,y,...)	maximum of $\{x, y, ...\}$
float(x)	convert x to floating point
round(x)	round x to nearest integer
int(x)	convert x to largest integer less than or equal to x
if(x)	1, if x != 0; 0 otherwise
if(x,y)	y, if x != 0; 0 otherwise
if(x,y,z)	y, if x != 0; z otherwise
if(x,p,z,n)	p, if $x > 0$; z, if $x == 0$; n otherwise
eval(a,b,...,x)	x (but all arguments are evaluated)

Temporary Variables

Expressions can become complicated and values computed in one part of an expression may need to be computed again in a later part. Such a value can be assigned a temporary *variable* to be used in place of that value in the later part of the expression. For example, *result=(map+2)*(map+2)* may be expressed, using temporary variable x, as *result=(x=map+2)*x*. Additional examples are given on pages 10 and 13.

3 GIS Examples

The *r.mapcalc* algebra supports a variety of GIS operations, some of which are described in the following sections; these, however, do not exhaust the flexibility and power of the language. In the examples, the following maps are used:

elevation 30-meter digital elevation, range 1066-1840 meters

rushmore Camp Rushmore (a fictitious military installation)

0	outside the installation
1	inside the installation

slope slope in degrees

vegcover vegetation cover

1	irrigated agriculture
2	rangeland
3	coniferous forest
4	deciduous forest
5	mixed forest
6	disturbed

Map Recoding

One standard GIS operation is the recoding of values in a map. For example, a map of forest cover versus nonforest cover can be created from the *vegcover* map by recoding forest categories to 1, and nonforest categories to 2 as follows:

$$simple.cover = if(vegcover == 3 \,||\, vegcove == 4 \,||\, vegcover == 5,\ 1) + \ \backslash$$
$$if(vegcover == 1 \,||\, vegcover == 2 \,||\, vegcover == 6,\ 2)$$

One adds the results of both *if* functions because each *if* selects a distinct subset of the input data. Note the use of \ to indicate that the expression continues on multiple lines.

To make a map of elevations between 1200 and 1375 meters (with elevations outside this range recoded to 0), use:

$$upland = if(elevation >= 1200 \,\&\&\, elevation <= 1375, elevation)$$

Selections

An operation related to map recoding is the selection or identification of cells meeting some specified criteria in one or more maps. Selection differs from recoding in two respects: (1) it may involve more than one map; and (2) the resulting map often has only the values 0 and 1, 1 indicating cells that meet the criteria and 0 indicating those that do not. The logical operators (&& and ||) together with the comparison operators (==, !=, <, <=, >, and >=) provide this selection capability.

For example, to create the map *upland*, with elevations between 1200 and 1375 meters coded as 1, and elevations outside this range coded as 0, use:

$$upland = elevation >= 1200 \,\&\&\, elevation <= 1375$$

To create the map *forest*, indicating where forests are found, use:

$$forest = vegcover == 3 \;||\; vegcover == 4 \;||\; vegcover == 5$$

Then, to create the combined map *upland.forest*, indicating where upland forests are found use:

$$upland.forest = forest \;\&\&\; upland$$

Or, more directly, make the map *upland.forest* without first creating *upland* and *forest* by using:

$$\begin{aligned} upland.forest \;=\; & (elevation >= 1200 \;\&\&\; elevation <= 1375) \;\&\&\; \backslash \\ & (vegcover == 3 \;||\; vegcover == 4 \;||\; vegcover == 5) \end{aligned}$$

Region Growing

A region-growing operation adds a one-cell border around an area of a map and can be implemented with a simple algorithm. Each zero value in a map is replaced by a nonzero value from the cell to the left, to the right, above, or below.

For example, to add a one-cell border to the Camp Rushmore map, *rushmore*, one can create *rushmore.grow*:

$$\begin{aligned} rushmore.grow \;=\; & if(rushmore, rushmore, \backslash \\ & \quad if(rushmore[0,-1], rushmore[0,-1], \backslash \\ & \quad\quad if(rushmore[0,1], rushmore[0,1], \backslash \\ & \quad\quad\quad if(rushmore[-1,0], rushmore[-1,0], \backslash \\ & \quad\quad\quad\quad if(\text{rushmore[1,0]}, rushmore[1,0])))) \end{aligned}$$

The border itself, *rushmore.border*, can now be extracted by subtracting the original map from the new map:

$$rushmore.border = rushmore.grow - rushmore$$

Note that a count of just the border cells can be used to approximate the perimeter of the installation.[2]

Slope and Aspect

The neighborhood syntax of the *r.mapcalc* algebra enables a user to calculate a slope gradient (the maximum rate of change in altitude) and its aspect (the compass direction of the slope) from elevation values. The basic formulas (Dozier and Strahler 1983) are:

$$\begin{aligned} tan(slope) &= \sqrt{(\delta z/\delta x)^2 + (\delta z/\delta y)^2} \\ tan(aspect) &= \frac{\delta z/\delta x}{\delta z/\delta y} \end{aligned}$$

[2]Cell counting, a regional operation, is not supported by *r.mapcalc*. Cell counts must be determined using another tool (e.g., the GRASS *r.stats* command).

where $\delta z/\delta x$ and $\delta z/\delta y$ are the partial derivatives in the east-west and north-south directions, respectively. Numerical methods can estimate these derivatives (Skidmore 1989). A method given by Horn (1981) is:

$$[\delta z/\delta x]_{y,x} = (z_{y-1,x-1} + 2z_{y,x-1} + z_{y+1,x-1} - z_{y-1,x+1} - 2z_{y,x+1} - z_{y+1,x+1})/8\Delta x$$
$$[\delta z/\delta y]_{y,x} = (z_{y-1,x-1} + 2z_{y-1,x} + z_{y-1,x+1} - z_{y+1,x-1} - 2z_{y+1,x} - z_{y+1,x+1})/8\Delta y$$

where $z_{y,x}$ is the elevation value at row y column x, Δx is the east-west (i.e., column) grid spacing, and Δy is the north-south (i.e., row) grid spacing. This can be expressed more clearly with a matrix of coefficients:

$$\delta z/\delta x = \frac{\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}}{8\Delta X} \quad \delta z/\delta y = \frac{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}}{8\Delta Y}$$

The *elevation* map, with its 30-by-30-meter horizontal grid spacing, can be processed with *r.mapcalc* as follows:

```
slope = eval( x = (elevation[-1,-1] + 2 * elevation[0,-1] + elevation[1,-1] \
                   -elevation[-1,1] - 2 * elevation[0,1] - elevation[1,1]    \
                   )/(8.0 * 30.0) ,                                          \
              y = (elevation[-1,-1] + 2 * elevation[-1,0] + elevation[-1,1] \
                   -elevation[1,-1] - 2 * elevation[1,0] - elevation[1,1]    \
                   )/(8.0 * 30.0) ,                                          \
              atan(sqrt(x * x + y * y))                                      \
            )

aspect = eval( x = (elevation[-1,-1] + 2 * elevation[0,-1] + elevation[1,-1] \
                    -elevation[-1,1] - 2 * elevation[0,1] - elevation[1,1]   \
                    )/(8.0 * 30.0) ,                                         \
               y = (elevation[-1,-1] + 2 * elevation[-1,0] + elevation[-1,1] \
                    -elevation[1,-1] - 2 * elevation[1,0] - elevation[1,1]   \
                    )/(8.0 * 30.0) ,                                         \
               a = round(atan(x,y)) ,                                        \
               if(x||y, if(a,a,360))                                         \
             )
```

Note the use of the temporary variables a, x, and y to capture intermediate results for use in a latter part of the expression.

Slope is calculated using the single-argument version of *atan*(), which produces angles in the range of 0 to 90 degrees. For terrain with low relief, however, where degree values may not give enough information, one could multiply the result by 10 to get 10ths of a degree, or omit the *atan*() function to get *tan(slope)* instead.

Aspect is calculated using the two-argument version of atan(), which produces angles in the range of 0 to 360 degrees. However, because aspect is undefined if both x and y are zero (i.e., flat terrain), additional care is required. The code:

```
a = round(atan(x,y))
if(x||y, if(a,a,360))
```

produces values from 1 to 360 for cells that have nonzero slope, and a zero value for flat terrain.

Frank (1988) used an alternative technique to compute $\delta z/\delta x$ and $\delta z/\delta y$:

$$\delta z/\delta x = \frac{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 8 & 0 & -1 & -8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}{12\Delta X} \quad \delta z/\delta y = \frac{\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -8 & 0 & 0 \end{bmatrix}}{12\Delta Y}$$

This is expressed in *r.mapcalc* as:

```
slope = eval( x = (elevation[0,-2] + 8 * elevation[0,-1] \
                   -elevation[0,2] - 8 * elevation[0,1]   \
                   )/(12.0 * 30.0) ,                      \
              y = (elevation[-2,0] + 8 * elevation[-1,0]  \
                   -elevation[2,0] - 8 * elevation[1,0]   \
                   )/(12.0 * 30.0) ,                      \
              atan(sqrt(x * x + y * y))                   \
            )

aspect = eval( x = (elevation[0,-2] + 8 * elevation[0,-1] \
                   -elevation[0,2] - 8 * elevation[0,1]   \
                   )/(12.0 * 30.0) ,                      \
               y = (elevation[-2,0] + 8 * elevation[-1,0] \
                   -elevation[2,0] - 8 * elevation[1,0]   \
                   )/(12.0 * 30.0) ,                      \
               a = round(atan(x,y)) ,                     \
               if(x||y, if(a,a,360))                      \
             )
```

Hydrologic Simulation

r.mapcalc can construct a simple hydrologic model that iteratively "flows" water across a landscape. A constant amount of water is first deposited in each cell in the landscape. Then, at each time-step, a portion of the water in each cell is drained into its eight surrounding neighbors. The basic logic is:

```
for each cell
    for each neighbor
        let height difference = (cell elevation + water height)
                minus (neighbor elevation + water height)
        if the height difference is positive
        then
            if the cell elevation is greater than
                the neighbor's elevation + water height
            then
```

```
                drain out a portion of the water
            otherwise
                drain out a portion of the height difference
        otherwise
            if the neighbor's elevation is greater than
                the cell's elevation + water height
            then
                drain in a portion of the neighbor's water
            otherwise
                drain in a portion of the height difference
    end
end
```

The outer loop is for each cell in the landscape and is done automatically by *r.mapcalc*. The inner loop applies to the eight neighboring cells and must be explicitly coded by the user.

Before running this model, the elevation map must be converted to the same units as the water map and must be filtered to smooth the elevation. In this example, let us assume that the water will be in units of 0.1 inches. To convert elevation from meters to 0.1-inch units:

$$elev = elevation * 393.7$$

The map *elev* can then be smoothed with the following filter:

$$\frac{\begin{bmatrix} 1 & 2 & 1 \\ 2 & 8 & 2 \\ 1 & 2 & 1 \end{bmatrix}}{20}$$

as follows:

$$\begin{aligned} elev \;=\; (\;& elev[-1,-1] + 2 * elev[-1,0] + elev[-1,1] \;\backslash \\ + \;& 2 * elev[0,-1] + 8 * elev + 2 * elev[0,1] \;\backslash \\ + \;& elev[1,-1] + 2 * elev[1,0] + elev[1,1] \;\backslash \\ ) \;& /20 \end{aligned}$$

(This filter produces a map with smoother contours. It also tends to create dams at terrain choke points, which will form little lakes during the simulation.)

The model itself is coded as follows:

```
water = water + eval(x = elev + water,                                    \
        if  (x > (y = elev[-1,0] + water[-1,0]),                          \
            -.15 * if  (elev > y,            water,          x - y),      \
             .15 * if  (elev[-1,0] > x,      water[-1,0],    y - x))+     \
        if  (x > (y = elev[1,0] + water[1,0]),                            \
            -.15 * if  (elev > y,            water,          x - y),      \
             .15 * if  (elev[1,0] > x,       water[1,0],     y - x))+     \
        if  (x > (y = elev[0,-1] + water[0,-1]),                          \
            -.15 * if  (elev > y,            water,          x - y),      \
             .15 * if  (elev[0,-1] > x,      water[0,-1],    y - x))+     \
        if  (x > (y = elev[0,1] + water[0,1]),                            \
            -.15 * if  (elev > y,            water,          x - y),      \
             .15 * if  (elev[0,1] > x,       water[0,1],     y - x))+     \
        if  (x > (y = elev[-1,1] + water[-1,1]),                          \
            -.10 * if  (elev > y,            water,          x - y),      \
             .10 * if  (elev[-1,1] > x,      water[-1,1],    y - x))+     \
        if  (x > (y = elev[1,1] + water[1,1]),                            \
            -.10 * if  (elev > y,            water,          x - y),      \
             .10 * if  (elev[1,1] > x,       water[1,1],     y - x))+     \
        if  (x > (y = elev[1,-1] + water[1,-1]),                          \
            -.10 * if  (elev > y,            water,          x - y),      \
             .10 * if  (elev[1,-1] > x,      water[1,-1],    y - x))+     \
        if  (x > (y = elev[-1,-1] + water[-1,-1]),                        \
            -.10 * if  (elev > y,            water,          x - y),      \
             .10 * if  (elev[-1,-1] > x,     water[-1,-1],   y - x)))
```

The resulting map *water* represents the water depth after a single iteration (time-step). Note that the eight sections in this model (beginning with $if(x > \ldots)$) are similar, each handling a neighboring cell. The first four drain 15 percent of the water to or from the cells above, below, and to either side; the last four drain 10 percent of the water to or from the cells at the diagonal positions.

This model has been applied to a section of the Spearfish database with realistic-looking results. (The data set entitled *spearfish* is a sample database included with the GRASS release and covers the Spearfish, South Dakota vicinity.) Water drains away from the uplands forming temporary streams and ponds. For those wishing to repeat the experiment with GRASS, enter the model code in a file called *water.mapcalc*, then create a controlling shell script:

```
#!/bin/sh
r.mapcalc water=120 # start with 12 inches of water in each cell
d.rast water        # display the water map
i=1
while [ $i != 100 ]
do
    n=1
    while [ $n != 10 ]
```

```
    do
        r.mapcalc < water.mapcalc # run the simulation
        d.rast water              # display the water map
        n='expr $n + 1'
    done
    g.copy rast=water,water.$i    # save a snapshot
    i='expr $i + 1'
done
```

This script will run 1000 iterations. The depth starts at 120 (12 inches, or 30.48 cm, of water) across the entire map. After every tenth iteration, the resultant map is copied to a map with a unique name[3] for rapid display later. One can run the shell script and watch the water "flow" off the slopes of South Dakota's Black Hills.

This model could be improved by considering additional factors such as: (1) *evaporation and transpiration*—in each time-step (iteration), some water will evaporate, which could be modeled either as a constant value or as a simple function of groundcover type; (2) *flow impedance*—the amount of water that can leave a cell in a given time step could be modified using groundcover information; and (3) *ground saturation*—the rate at which soils absorb water during rainfall can be estimated based on soil type and length of exposure to water.

4 IMAGERY EXAMPLES

The *r.mapcalc* algebra also supports many image-processing operations. Examples of these follow; again, they are not exhaustive, but are intended to illustrate the power of the algebra.

Spectral Ratios

Ratio transformation of spectral data is one technique used in the analysis of remotely sensed data (Lillesand and Kiefer 1987). Between-band ratios, which eliminate multiplicative effects, are easily created with *r.mapcalc*:

$$ratio = band.1/band.2$$

Ratios of between-band differences, which eliminate additive effects, are also straightforward:

$$ratio = (band.1 - band.2)/(band.3 - band.2)$$

A normalized vegetation index (*nvi*) for AVHRR data is computed as:

$$nvi = (avhrr.2 - avhrr.1)/(avhrr.2 + avhrr.1)$$

These results could be produced with a very simple algebra that uses only arithmetic operators. The transformed vegetative index (*tvi*) for the Landsat Thematic Mapper illustrates the use of functions in *r.mapcalc*:

$$tvi = 100 * sqrt((tm.4 - tm.3)/(tm.4 + tm.3) + 0.5)$$

[3]These maps will be named *water.1, water.2, ... water.100.*

Note that 0.5 is added to help ensure that *sqrt*() is not applied to a negative value. This is guaranteed by using the *max*() function:

$$tvi = 100 * sqrt(max(0,\ (tm.4 - tm.3)/(tm.4 + tm.3) + 0.5))$$

If the results are to be in the range 0 to 255, this ratio will produce such results:

$$ratio = 255.0/90.0 * atan(float(tm.1)/float(tm.2))$$

The *float*() function ensures that the division is floating point, not integer, division. Because the output map can hold only integer values, *r.mapcalc* will round the results to integers before they are stored in the map.

Spatial Filters

A common image-processing operation is local-neighborhood filtering. A small window is moved over an image, and the center pixel is replaced by a combination of all the pixels in the window.

Low-frequency filters (also called low-pass filters) deemphasize high spatial frequencies and involve computing a weighted average of all pixels in the window. A simple average, which smoothes the image, is based on the filter:

$$\frac{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}}{9}$$

This is expressed with *r.mapcalc* as:

```
ave3x3 = (  image[-1,-1] + image[-1,0] + image[-1,1]  \
         +  image[0,-1] + image[0,0] + image[0,1]     \
         +  image[1,-1] + image[1,0] + image[1,1]     \
         ) / 9
```

Use high-frequency filters (high-pass filters) to emphasize high spatial frequencies and for edge enhancement. The technique is the same as for low-frequency filters, except that some of the weights are negative. One such filter is:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

This is expressed with *r.mapcalc* as:

```
high3x3 = (  -image[-1,-1] - image[-1,0] - image[-1,1]  \
             -image[0,-1] + 9 * image[0,0] - image[0,1]  \
             -image[1,-1] - image[1,0] - image[1,1]      \
          )
```

The Sobel filter is a nonlinear edge-enhancement filter:

$$\sqrt{\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}^2 + \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}^2}$$

It is similar to the method to determine slope values from elevation values and is expressed in *r.mapcalc* as:

$$\begin{aligned} sobel3x3 = eval(\quad & x = \quad image[-1,1] + 2 * image[0,1] + image[1,1] \quad \backslash \\ & \quad\quad -image[-1,-1] - 2 * image[0,-1] - image[1,-1] \,, \quad \backslash \\ & y = \quad image[-1,-1] + 2 * image[-1,0] + image[-1,1] \quad \backslash \\ & \quad\quad -image[1,-1] - 2 * image[1,0] - image[1,1] \,, \quad \backslash \\ & sqrt(x * x + y * y) \quad \backslash \\ ) & \end{aligned}$$

Another type of filter is an adaptive one, which replaces the center pixel only if some criteria are met. Eliason and McEwen (1990) describe two whereby the center pixel is replaced by the average value in the neighborhood if the difference between them exceeds a threshold. One filter sets the threshold to a multiple of the statistical variance in the neighborhood. The other sets the threshold to a multiple of the statistical variance in the entire image. Both filters can be coded with *r.mapcalc*; the variance within the entire image is a regional calculation and must first be computed by another command[4] and then inserted into the *r.mapcalc* code. The first filter is illustrated.

Let P be the value of the center pixel, s the sum of all values in the neighborhood, n the number of pixels in the neighborhood, and ss the sum of the squares of the values in the neighborhood. The average is given by:

$$\mu = \frac{s}{n}$$

and the variance is given by:

$$\sigma^2 = \frac{ss}{n} - \mu^2$$

The center pixel is replaced if:

$$(P - \mu)^2 > C\sigma^2$$

where C is usually between 1.0 and 4.0. The *r.mapcalc* code for a 3x3 window with C set to 2.25 is:

[4]The GRASS *r.stats* command could be used to compute the variance across the entire image.

$$
\begin{aligned}
filter = eval(\ & s = image[-1,-1] + image[-1,0] + image[-1,1] + \ \backslash \\
& \quad image[0,-1] + image[0,0] + image[0,1] + \ \backslash \\
& \quad image[1,-1] + image[1,0] + image[1,1]\ , \ \backslash \\
& ss = image[-1,-1] * image[-1,-1] + \ \backslash \\
& \quad image[-1,0] * image[-1,0] + \ \backslash \\
& \quad image[-1,1] * image[-1,1] + \ \backslash \\
& \quad image[0,-1] * image[0,-1] + \ \backslash \\
& \quad image[0,0] * image[0,0] + \ \backslash \\
& \quad image[0,1] * image[0,1] + \ \backslash \\
& \quad image[1,-1] * image[1,-1] + \ \backslash \\
& \quad image[1,0] * image[1,0] + \ \backslash \\
& \quad image[1,1] * image[1,1]\ , \ \backslash \\
& ave = s/9.0\ , \ \backslash \\
& var = ss/9.0 - ave * ave\ , \ \backslash \\
& x = image - ave\ , \ \backslash \\
& if(x * x > 2.25 * var, ave, image) \ \backslash \\
)
\end{aligned}
$$

Radiometric Calibration

Radiometric calibration converts digital values recorded by a remote sensing system to radiance values at the top of the atmosphere, possibly followed by a conversion to surface reflectance values. Radiance values can be computed using the following formula (Chavez 1989):

$$rad_i(x,y) = (dn_i(x,y) - offset_i)/gain_i$$

where *rad* is the radiance for band *i* at pixel *x,y*; *dn* is the value recorded by the sensors; and *offset* and *gain* are the sensor offset and gain for band *i*. This formula is easily handled by *r.mapcalc*. For example, using the values for the offset and gain as reported by Chavez for October 3, 1988 Thematic Mapper data, radiance images are constructed as follows:

$$
\begin{aligned}
rad.1 &= (tm.1 - 2.4899)/16.5993 \\
rad.2 &= (tm.2 - 2.3871)/8.5104 \\
rad.3 &= (tm.3 - 1.4815)/12.4074 \\
rad.4 &= (tm.4 - 1.8418)/12.2790 \\
rad.5 &= (tm.5 - 3.4240)/92.5292 \\
rad.7 &= (tm.7 - 2.6323)/175.4878
\end{aligned}
$$

The radiance formula for the SPOT sensor appears in a slightly modified form (USA SPOT Image Corporation 1989):

$$rad_i(x,y) = dn_i(x,y)/A_i$$

where A_i are the absolute calibration coefficients. The May 27, 1989 SPOT multispectral image in the Spearfish database can be converted to radiance, using *r.mapcalc*, as follows:

$$
\begin{aligned}
rad.1 &= spot.ms.1/1.05586 \\
rad.2 &= spot.ms.2/1.12140 \\
rad.3 &= spot.ms.3/0.97244
\end{aligned}
$$

Surface reflectance calculations are more involved because the formulas must incorporate atmospheric effects. There are various methods for modeling these effects (Richards 1986, Price 1987, Chavez 1989); a formula given by Chavez is:

$$R_i(x,y) = \frac{\pi dist^2[rad_i(x,y) - haze_i]}{E_i slope(x,y) \cdot sun \cdot mhaze_i}$$

where R is the surface reflectance image, *rad* is the radiance image, *slope* is a slope map, *dist* is the Earth-Sun distance, *haze* and *mhaze* are the additive and multiplicative atmospheric haze factors, E is exoatmospheric spectral irradiance, and *sun* is the sun elevation at the time the image was captured. Combining the nonmap factors into single constants, the formula can be written in the form:

$$R_i(x,y) = \frac{A_i[rad_i(x,y) - B_i]}{slope(x,y)}$$

which, after substituting the appropriate values for the constants A and B, is expressed with *r.mapcalc* as:

$$\begin{aligned} R.1 &= A_1 * (rad.1 - B_1)/slope \\ R.2 &= A_2 * (rad.2 - B_2)/slope \\ R.3 &= A_3 * (rad.3 - B_3)/slope \\ &\vdots \end{aligned}$$

Principal Components

Principal components analysis involves transforming a set of correlated variables into a new, uncorrelated set. The variables are the band files from a multispectral sensor. The transformation is a linear combination of the band data:

$$component_i = \sum_{j=1}^{N} w_{ij} * band_j \quad i = 1, \ldots, N$$

where N is the number of band files and w_{ij} are the eigenvectors for the between-band covariance matrix. Although neither the covariance computation nor the determination of the eigenvectors can be accomplished using the map algebra, the components can be computed. For example, suppose the three multispectral bands from a SPOT image had the following covariance matrix:

$$\begin{bmatrix} 462.88 & 480.41 & 281.76 \\ 480.41 & 513.02 & 278.92 \\ 281.76 & 278.91 & 336.33 \end{bmatrix}$$

A set of eigenvectors for this matrix (in decreasing order of spectral variance) is:

$vector_1$	21.24	22.15	14.77
$vector_2$	2.91	4.46	−10.87
$vector_3$	1.82	−1.62	−0.18

To compute the second component with *r.mapcalc*:

$$pc.2 = 2.91 * spot.ms.1 + 4.46 * spot.ms.2 - 10.87 * spot.ms.3$$

Merging Panchromatic Data With Multispectral Data

The French SPOT satellite produces a 10-meter panchromatic image plus three spectral bands with 20-meter resolution. Landsat Thematic Mapper produces six spectral bands with 30-meter resolution (as well as a 120-meter resolution thermal band). A sharpened color image can be produced by merging the high-resolution panchromatic image with the lower resolution spectral bands. With the assumption that the spectral data have been registered and resampled to the 10-meter panchromatic data, a number of techniques have been used to perform the merger, most of which can be done with the map algebra.

Welch and Ehlers (1987) described two direct methods:

$$\begin{aligned} M_i' &= a_i(w_1 M_i + w_2 P) + b_i \\ M_i' &= a_i(M_i P)^{\frac{1}{2}} + b_i \end{aligned}$$

where M_i is spectral band i, P is the panchromatic band, w_1 and w_2 are weights, and a_i and b_i are chosen to make the results fall within the range 0 to 255. Both these methods can be implemented using the *r.mapcalc* algebra.

A simple average, based on the first method, is coded as follows:

$$\begin{aligned} merge.1 &= (spot.ms.1 + spot.pan)/2 \\ merge.2 &= (spot.ms.2 + spot.pan)/2 \\ merge.3 &= (spot.ms.3 + spot.pan)/2 \end{aligned}$$

Or a combination of both methods can be used:

$$\begin{aligned} merge.1 &= sqrt(spot.pan * spot.ms.1) \\ merge.2 &= sqrt(spot.pan * spot.ms.2) \\ merge.3 &= (spot.pan + 3 * spot.ms.3)/4 \end{aligned}$$

A third method transforms three selected bands from red, green, and blue color space to intensity, hue, and saturation color space. The intensity is replaced by the panchromatic image, and the new intensity, hue, and saturation combination is transformed back into red, green, and blue.

Intensity, Hue, Saturation

Conversion from red, green, and blue colors to intensity, hue, and saturation involves formulas that can be represented using the *r.mapcalc* algebra. The following is based on the hue, saturation, and value (HSV) algorithm found in Foley and Van Dam (1984). Assuming that R, G, and B represent the red, green, and blue components of an image, the value (V), saturation (S),

and hue (H) can be formulated as follows:

$$
\begin{array}{rcl}
V & = & max(R,G,B) \\
S & = & 255.0*(V-min(R,G,B))/V \\
H & = & if(S,eval(\quad \backslash \\
 & & m = float(min(R,G,B)), \quad \backslash \\
 & & r = (V-R)/(V-m), \quad \backslash \\
 & & g = (V-G)/(V-m), \quad \backslash \\
 & & b = (V-B)/(V-m), \quad \backslash \\
 & & h = if(R==V,b-g)+ \quad \backslash \\
 & & \quad if(G==V,2+r-b)+ \quad \backslash \\
 & & \quad if(B==V,4+g-r), \quad \backslash \\
 & & if(h<=0,h+6,h)*60))
\end{array}
$$

Although saturation is normally defined in the range from 0 to 1, it is multiplied here by 255 to retain more information after conversion to integer. Hue will be in the range of 0 to 360, with 0 representing undefined hues (i.e., white, black, or gray).

The inverse transformation is:

$$
\begin{array}{rcl}
R & = & eval(\quad \backslash \\
 & & s = S/255.0, \quad \backslash \\
 & & h = if(H>=360,H-360,H)/60.0, \quad \backslash \\
 & & i = int(h), \quad \backslash \\
 & & f = h-i, \quad \backslash \\
 & & p = V*(1-s), \quad \backslash \\
 & & q = V*(1-s*f), \quad \backslash \\
 & & t = V*(1-s*(1-f)), \quad \backslash \\
 & & if(i==0,V)+if(i==1,q)+if(i==2,p)+ \quad \backslash \\
 & & if(i==3,p)+if(i==4,t)+if(i==5,V))
\end{array}
$$

$$
\begin{array}{rcl}
G & = & eval(\quad \backslash \\
 & & s = S/255.0, \quad \backslash \\
 & & h = if(H>=360,H-360,H)/60.0, \quad \backslash \\
 & & i = int(h), \quad \backslash \\
 & & f = h-i, \quad \backslash \\
 & & p = V*(1-s), \quad \backslash \\
 & & q = V*(1-s*f), \quad \backslash \\
 & & t = V*(1-s*(1-f)), \quad \backslash \\
 & & if(i==0,t)+if(i==1,V)+if(i==2,V)+ \quad \backslash \\
 & & if(i==3,q)+if(i==4,p)+if(i==5,p))
\end{array}
$$

$$
\begin{aligned}
B \;=\; & eval(& & \backslash \\
& s \;=\; & S/255.0\,, & \backslash \\
& h \;=\; & if(H >= 360, H - 360, H)/60.0\,, & \backslash \\
& i \;=\; & int(h)\,, & \backslash \\
& f \;=\; & h - i\,, & \backslash \\
& p \;=\; & V * (1 - s)\,, & \backslash \\
& q \;=\; & V * (1 - s * f)\,, & \backslash \\
& t \;=\; & V * (1 - s * (1 - f))\,, & \backslash \\
& if(i == 0, p) + if(i == 1, p) + if(i == 2, t) + & & \backslash \\
& if(i == 3, V) + if(i == 4, V) + if(i == 5, q))
\end{aligned}
$$

5 CONCLUSION

Providing a map algebra for manipulation of raster map data results in a flexible geographic information system. Given appropriate raster data layers, the type and number of potential data manipulations are virtually limitless. For many applications, users are freed from software limits and are bound only by their ability to employ the algebra. *r.mapcalc* supports three levels of usage: (1) as a resource for users who need to perform specific algebraic functions that are not provided by other GRASS programs; (2) as a foundational tool for advanced GIS users to develop a limitless set of image and map analysis functions; and (3) as a resource for programmers and developers to use in building new functions.

r.mapcalc is perhaps most important at the intermediate level, as a tool for advanced GIS users. A common implementation model for a GIS in larger organizations is one or two sophisticated users supporting numerous casual users. In this context, *r.mapcalc* becomes a language that opens an array of possibilities for spatial analysis and provides a means to develop macros to support routine operations used by less sophisticated users.

6 REFERENCES

Burrough, P. A., *Principles of Geographical Information Systems for Land Resources Assessment*, Monograph on Soils and Resources Survey No. 12 (Oxford University Press, 1986).

Chavez, Pat S., Jr., "Radiometric Calibration of Landsat Thematic Mapper Multispectral Images," *Photogrammetric Engineering and Remote Sensing*, Vol 55, No. 9 (1989), pp 1285-1294.

Dozier, J., and A. H. Strahler, "Ground Investigations in Support of Remote Sensing," in *Manual of Remote Sensing*, Vol 1, edited by R. N. Colwell (American Society of Photogrammetry, 1983).

Eliason, Eric M., and Alfred S. McEwen, "Adaptive Box Filters for Removal of Random Noise from Digital Images," *Photogrammetric Engineering and Remote Sensing*, Vol 56, No. 4 (1990), pp 453-458.

Foley, James D., and Andries Van Dam, *Fundamentals of Interactive Computer Graphics* (Addison-Wesley, 1984).

Frank, Thomas D., "Mapping Dominant Vegetation Communities in the Colorado Rocky Mountain Front Range with Landsat Thematic Mapper and Digital Terrain Data," *Photogrammetric Engineering and Remote Sensing*, Vol 54, No. 12 (1988), pp 1727-1734.

Horn, B. K. P., "Hill Shading and the Reflectance Map," *Proceedings of the I.E.E.E.*, No. 69 (1981).

Lillesand, Thomas M., and Ralph W. Kiefer, *Remote Sensing and Image Interpretation*, 2d ed. (John Wiley & Sons, 1987).

Price, John C., "Calibration of Satellite Radiometers and the Comparison of Vegetation Indices," *Remote Sensing of Environment*, No. 21 (1987), pp 15-27.

Richards, John A., *Remote Sensing Digital Image Analysis* (Springer-Verlag, 1986).

Skidmore, Andrew K., "A Comparison of Techniques for Calculating Gradient and Aspect from a Gridded Elevation Model," *International Journal of Geographical Information Systems*, Vol 3, No. 4 (1989), pp 323-334.

SPOT User's Handbook, Vol 2 (USA SPOT Image Corporation, 1989), pp 2-49.

Tomlin, C. Dana, *Geographic Information Systems and Cartographic Modeling* (Prentice-Hall, 1990).

Welch, R., and M. Ehlers, "Merging Multiresolution SPOT HRV and Landsat TM Data," *Photogrammetric Engineering and Remote Sensing*, Vol 53, No. 3 (1987), pp 301-305.

USACERL Distribution

Chief of Engineers
ATTN: CEHEC-IM-LH (2)
ATTN: CEHEC-IM-LP (2)
ATTN: CERD-L
ATTN: DAEN-ZCI-P (2)
ATTN: CECW-PF
ATTN: CECW-EP-S
ATTN: CECW-RE
ATTN: CEIM-P
ATTN: CECW-OR
ATTN: DAIM-ED-N

US Army Engineer District
ATTN: Chief, Regulatory Functions
Buffalo 14207
Norfolk 23510
Huntington 25701
Wilmington 28402
Charleston 29402
St. Paul 55101
Chicago 60606
Little Rock 72203
Galveston 77550
Albuquerque 87103
Los Angeles 90053
San Francisco 94105
Sacramento 95814
Portland 97208
Seattle 99362
New England 02254
ATTN: CENED-OD-R
New York 10278
ATTN: CENAN
Philadelphia 19106
ATTN: CENAP
Baltimore 21203
ATTN: CENAB-OP-R
Savannah 31402
ATTN: CESAS-OP-FP
Jacksonville 32232
ATTN: CESAJ-R-RD
Mobile 36628
ATTN: CESAM-OP-S
Nashville 37202
ATTN: CEORN
Memphis 38103
ATTN: CELMM-CO-R
Vicksburg 39180
ATTN: CELMV
Louisville 40201
ATTN: CEORL
Detroit 48226
ATTN: CENCE-CO
Rock Island 61204
ATTN: CENCR
St. Louis 63101
ATTN: CELMS-OD-F
Kansas City 64106
ATTN: Chief, Permits Section
Omaha 68102
ATTN: CEMRO-OP
New Orleans 71060
ATTN: Chief, Permits Section
Tulsa 74121
ATTN: CEWST
Fort Worth 76102
ATTN: CESWF-OD-O

US Army Engr Divisions
North Central 60606
ATTN: CENCD-CO
Southwest 75242
ATTN: CESWD-CO-R

US Military Academy 10996
ATTN: Dept of Geo & Envr Engr (2)
ATTN: Natural Resources Branch

Aberdeen Proving Ground, MD 21005
ATTN: ISC-TECOM
ATTN: HSHB-CI

US EPA Research Lab 97333

Yakima Firing Center 98901

USATHAMA 21010
ATTN: CETHA-RM-I
ATTN: AMXTH-RM
ATTN: CETHA-IR-S

CECPW 22060
ATTN: CECPW-FN
ATTN: CECPW-SP
ATTN: CECPW-FM-A

Bureau of Land Management
WASH DC 20240
Fort Collins 80526
Denver 80225

US Dept of Commerce 20233

Army National Guard 20310
ATTN: NGB-AREC

US Army Concepts Analysis Agency 20814

FBI Academy 22135

USA Foreign Science Tech Ctr 22901

Naval Oceanographic Office 39522

Redstone Arsenal 35898
ATTN: AMSMI-RA-EH-MP-PC

CEWES
ATTN: CEWES-IM-DA 39181
ATTN: CADD Center 39180

Wright-Patterson AFB 45433
ATTN: NSBIT AL/OEBN

Michigan Dept of Military Affairs 48913

Twin Cities Army Ammo Plant (2) 55112

Camp Ripley 56345
ATTN: Ofc of Archeology & Engr (2)

5th Inf. Fort Polk 71459
ATTN: Envr & Nat Rsrcs Mgmt Div (2)

US Army Materiel Cmd 61299
ATTN: AMXEN-U (2)

US Army Europe
HQ USAREUR 09403
ATTN: AEAEN-FE-E (2)
V Corps 09079
ATTN: AETV-EHF-R

Texas Army Nat'l Guard 78763

Lone Star Army Ammo Plant 75505

Red River Army Depot
ATTN SDSRR-GB 75507

Dugway Proving Ground 84122
ATTN: DPG-EN-E (2)

Yuma Proving Ground 85365
ATTN: ATEYP-ES-E

White Sands Missile Range
ATTN: STEWS-ES-E 88002

Envr Response & Info Ctr
ATTN: ENVR-EP 20310

Nat'l Geophysical Data Ctr
ATTN: Code E-GCI 80303

Hohenfels Training Area 09173
ATTN: AETTH-DEH
ATTN: AETTH-DEH-ENV-APO

US Army Forts
Fort Belvoir, VA 22060
ATTN: CEETL-CL-GC (2)
ATTN: CETEC-CA-D
ATTN: AMSEL-RD-NV-VMD-TST
ATTN: Envr & Nat Res Div
Fort Monroe, VA 23651
ATTN: ATBO-GE
ATTN: ATEN-FN
Fort Drum 13603
ATTN: AFZS-EH-E
Fort Jackson 29207
ATTN: ATCJ-EHN
Fort Gillem 30050
ATTN: FCEN-CED-E
Fort Gordon 30905
ATTN: ATZH-DIE (2)
Fort Stewart 31314
ATTN: AFZP-DEN-W
Fort Benning 31905
ATTN: Nat. Resource Mgmt Div (2)
Fort McClellan 36205
ATTN: ATZN-FEE
Fort Rucker 36362
ATTN: ATZQ-EH
Fort Knox 40121
ATTN: ATZK-EHE
Fort Campbell 42223
ATTN: AFZH-DEH
Fort Benjamin Harrison 46216
ATTN: ATZI-ISP (2)
Fort McCoy 54656
ATTN: AFZR-DEN
Fort Riley 66442
ATTN: AFZN-DE-N (2)
Fort Chaffee 72905
ATTN: ATZR-ZFE (2)
Fort Sill 73503
ATTN: Fish & Wildlife Br (2)
Fort Leonard Wood 65473
ATTN: ATZT-DEH-EE
Fort Dix 08640
ATTN: ATZD-EHN
Fort Eustis 23604
ATTN: Ranges & Targets Dir
Fort Worth 76115
ATTN: Cartographic Ctr (2)
Fort Hood 76544
ATTN: AFZF-DE-ENV
Fort Bliss 79916
ATTN: ATZC-DEH-E
Fort Carson 80913
ATTN: AFZC-ECM-NR
Fort Huachuca 85613
ATTN: ATZS-EHB
Fort Irwin 92310
ATTN: AFZJ-EH
Fort Lewis 98433
ATTN: AFZH-DEQ
ATTN: ATZH-EHQ
Fort Richardson 99505
ATTN: DEH
Fort Bragg 28307
ATTN: DEH

National Weather Service 20910

US Geological Survey 22092

Pine Bluff Arsenal 17602
ATTN: SMCPB-EMB

US Army Cold Regions Research & Engr Lab
ATTN: CECRL-IS 03755

NASA/SSC/STL 39529

Defense Technical Info Center
ATTN: DTIC-FAB (2)

145
+20
2/93

This publication was reproduced on recycled paper.

☆ U.S. GOVERNMENT PRINTING OFFICE: 1994 — 3510-S/00042