GRASS 4.2 Reference Manual

Section VI - Raster Commands

S.F. Clamons and B.W. Byars

GRASS Research Group
Baylor University
Waco, Texas

Table of Contents

r.answers

NAME

r.answers - Menu-driven interface from GRASS to ANSWERS

SYNOPSIS

r.answers

DESCRIPTION

r.answers integrates ANSWERS with GRASS. ANSWERS (Areal Non-point Source Watershed Environmental Response Simulation) is an event oriented, distributed parameter model that was developed to simulate the behavior of watersheds having agriculture as their primary land use. Its primary applications are watershed planning for erosion and sediment control on complex watersheds, and water quality analysis associated with sediment associated chemicals.

r.answers provides a menu of steps to complete the input required to run an ANSWERS simulation. Each simulation is treated as "project" by *r.answers*. The inputs collected for the steps completed are recorded under a project name, so that they may be copied or recalled for further completion or modification. The first menu one encounters when running *r.answers* includes functions to create a new project, work on existing projects, copy an existing project, and remove existing projects. The main menu (shown below) lists steps to be completed to prepare ANSWERS input, to run ANSWERS, plus other miscellaneous functions.

```
ANSWERS on GRASS Project Manager Main Menu
Project Name: [sample]

Status Option Description
_____________________________
0    Quit
1    Set mask, region, and resolution
2    Catalogue soils parameters
3    Catalogue land use and surface parameters
4    Identify elevation-based input layers
5    Prepare rain gauge data
6    Identify outlet cell
7    Specify areas with subsurface drainage
8    Catalogue channel parameters
9    Define channel slopes
10    Specify BMP's in watershed
11    Prepare ANSWERS input and run simulation
12    Miscellaneous Command Menu

Option: 0__
```

Steps 1-11 record and display their status to the left of the step number. If a step has not been run, no status is displayed (as seen above). If the step has been successfully completed, the status will be listed as "done". In some cases, a change in one step will cause the need to run another step again, in which case the status will read "rerun". If a step has a status of "done" or "rerun", if it is run again it will attempt to offer previous inputs as defaults.

Interface Operation Notes

Throughout *r.answers* two primary types of interface/input are used:

1) Text input that can be completed by hitting the RETURN key. In most cases, if no text was entered, the given ques- tion or operation is canceled. Often times text input will consist of the name of a new or existing map or project name, in which case entering the word "list" will provide a list of currently used names.

2) Text or menu options that can be completed by hitting the ESC (escape) key. This type of interface is used for menus or for entering tables of parameters. All menus have a default answer of Exit (0), so that by simply hitting ESC one may leave the program's menus. The following keystroke guide is helpful to know when using the parameter entry worksheets that use thisinterface:

[RETURN] moves the cursor to next prompt field
[CTRL-K] moves the cursor to previous prompt field
[CTRL-H] moves the cursor backward non-destructively within the field
[CTRL-L] moves the cursor forward non-destructively within the field
[CTRL-A] writes a copy of the screen to a file named "visual_ask" in your home directory
[CTRL-C] where indicated (on bottom line of screen) can be used to cancel operation

Description of Main Menu Steps
The following section describes each option of the main menu. All steps are verbose to provide as much immediate information as is practical, however it is necessary that the user also be familiar with the operation of ANSWERS. (Obtain a copy of the ANSWERS User's Manual (1991) by David Beasley and Larry Huggins. Available from Bernard Engel, Agricultural Engineering, Purdue University, W. Lafayette, Indiana, 47907).

Steps 1 through 10 collect inputs (either maps from the currently available mapsets or other text/numerical inputs) in order to create or extract the necessary portions of ANSWERS inputs for that step. After steps 1 through 10 are done, step 11 can be run to assemble an ANSWERS input file. ANSWERS can then be run using the inputs, and the output from the simulation is captured and processed, as described under step 11.

Step 1 Set mask, region, and resolution
Map input: Watershed mask
Other inputs: Project resolution, project region (optional)
Description: All raster values in the input mask map greater than zero will be used to create reclass rules to set the project MASK to the watershed area. Each time the project is called, the MASK will be automatically set. Project resolution is input in meters and it used to set the size of the watershed elements to be used in the simulation. The part of this step attempts to find the minimal region needed to con- tain the watershed mask at the given resolution. A region will be calculated to allow at least a one-cell border around the watershed area. This region is then presented in an input screen (much like that of r.region) for editing or approval. After the project mask, region and resolution are set, the information is recorded and will be reset automatically each time the project is called. This step will create a new raster map in the user's current mapset entitled project name].ELEMENT. This map will act as a reference to ANSWERS' methods of refering to raster cells. Raster values of the map will indicate element number, with the category description giving row and column numbers. If any of the inputs in this step are subsequently reset, all other steps that may have been completed will be marked with a status of "rerun", since changing mask, resolution or region will require that inputs will have to be resampled.

Step 2 Catalogue soils parameters
Map input: Soils
Other input: Soils parameters, tile drainage coefficient, groundwater release fraction.
Description: This step prompts for the name of a soil map, then reads the map and lists all soil categories found in the watershed mask. For each soil found in the watershed, ANSWERS requires values for the parameters listed below. The Project Manager facilitates preparation parameters by input into a table.

```
Soil Parameters for ANSWERS (see ANSWERS Users Manual for more details)
1   total porosity (percent pore space volume of soil)
2   field capacity (percent saturation)
3   steady state infiltration rate (mm/hour)
4   difference between steady state and maximum infiltration rate (mm/hour)
5   exponent in infiltration equation
6   infiltration control zone depth (mm)
7   antecedent soil moisture (percent saturation)
8   USLE 'K'
```

After the soil parameters are input, a creen will prompt for groundwater release fraction and tile drainage coeffi- cient, which will apply to the entire watershed. The tile drainage coefficient indicates the design coefficient (mm/day) of tile drains in those areas designated as having tile drainage. The groundwater release fraction is measure of the contribution of lateral groundwater movement or interflow to total runoff.

After this step is completed, it will provide an option to save the entered parameters to a file or printer for reference.

ANSWERS soils inputs will then be extracted and stored. This step may be rerun to change any of the information. Previously entered information will be recalled and may be modified.

Step 3 Catalogue land use and surface parameters
Map input: Land cover/use.
Other input: Land cover parameters
Description: For each category in the land use map found in the watershed, ANSWERS requires values for the parameters listed below. The Project Manager facilitates preparation parameters by input into a table.

```
Land Cover Parameters for ANSWERS (see ANSWERS Users Manual for details)

1    short (8 characters) description of land use and management
(program will attempt to use map category description, if any)
2    mm of potential rainfall interception by land cover
3    percentage of surface covered by specified land use
4    oughness coefficient of the surface (a shape factor)
5    m of maximum roughness height of the surface profile
6    Manning's n (a measure of flow retardance of the surface)
7    relative erosiveness (function of time and USLE 'C' and 'P')
```

After this step is completed, it will provide an option to save the entered parameters to a file or printer for reference. ANSWERS cover inputs will then be extracted and stored. This step may be rerun to change any of the information. Previously entered information will be recalled and may be modified.

Step 4 Identify elevation-based input layers
Map input: Slope and aspect.
Description: This step prompts for the names of previously prepared maps of slope and aspect for the project watershed. It is important to note that the required format of slope and aspect maps vary from that created by the *r.slope.aspect* program. Programs have been developed to process an elevation surface map and create ANSWERS slope and aspect map. The elevation map should be true elevations in meters. The elevation map can be "filtered" to remove "pits" and other potential problems to ANSWERS with the *r.fill.dir* program. The *r.direct* program can be used to prepare an ANSWERS aspect map from the elevation layer created by *r.fill.dir*. The *r.slope program* can be used to prepare an ANSWERS slope map from the elevation layer created by *r.fill.dir*. ANSWERS requires slope values which are percent multiplied by 10 (so a slope map value of 35 indicated a slope of 3.5%). The aspect map is a critical input to ANSWERS, since it defines the routing of runoff through the watershed, and should be carefully examined, since the *r.direct* program is unable to create flawless output. The *d.rast.arrow* and the *d.rast.edit* programs have been developed to assist this manual inspection and editing process. When editing the flow direction map, pay careful attention to 1) cells on the watershed border, which all must flow into the watershed. 2) cells that will be declared as "channels" must flow directly from one to another (therefore it is suggested that channels should be identified in conjunction with this step). 3) flow from two cells must not point directly to each other (-][-) or otherwise form circuitous routes. In the final flow map, one should be able to start at any cell in the watershed and follow the flow directions from cell to cell until arriving at the outlet cell.

Step 5 Prepare rain gauge data
Map input: Rain gauge areas (for multiple gauges)
Other input: Rain gauge data
Description: This step is designed to organize data used to describe the precipitation event to be simulated. ANSWERS permits up to four rain gauges to be used, each of which will require a table of rainfall data (time in minutes and rainfall intensity in millimeters per hour). Data from at least one rain gauge are required. If more than one gauge is used, you will need to prepare a raster map of the watershed area to indicate which cells are to represented by a given gauge's data.

To facilitate the modeling of a number of storms this step will prompt for a rainfall event name. The data tables entered will be stored in the ANSWERS database under the event name.

Rain gauge data for ANSWERS consists two columns of numbers. The first is Time (in minutes) and the second is Rainfall Intensity (in mm/hour). Decimal values will be rounded to the closest whole number. To input rain gauge data to the Project Manager, a file must first be prepared with rain gauge data. If multiple gauges are to be used, one input file is still used, data for each gauge are separated by should occur sequentially by gauge; so that data for gauge 1 is first in the file, data for gauge two is the second group of data, and so on.

Example rain gauge data input files:

```
one gauge   |two gauges
-------------------------------------------------------
00    |    00
103   |   111 data for gauge 1
20    |   10   |   257
35    |   22   |   -1 ——— delimiter
559   |    00
674   |   156 data for gauge 2
1000  |   104
-------------------------------------------------------
```

This step will prompt to determine if multiple rain gauges are to be used. If so, it will prompt for the name of a map that represents areas to be assigned to the given gauges. The number of categories and their value should match the number of rain gauges. Next the program prompts for the name of the rain gauge data file. The program reads the file and displays what it found to the screen for approval. Having this, it will create the appropriate ANSWERS input files.

Step 6 Identify outlet cell
Map input: none
Other input: row and column number of watershed outlet element
Description: ANSWERS needs to know the row and column number of the element at the watershed outlet. To facilitate your finding this information, the raster map [project name].ELEMENT has been created. The category values of this map are the sequentially numbered cells of the watershed. The category descriptions are the cell's row and column number. Using a tool such as *d.what.rast*, the row and column number of the outlet cell can be queried from the displayed element map.

Step 7 Specify areas with subsurface drainage
Map input: Areas with subsurface drainage (optional)
Description: This step offers a menu which allows the delineation of 1) all the watershed with subsurface drainage, 2) none of the watershed with subsurface drainage, or 3) areas with subsurface drainage specified with a raster map (all elements with a value greater than zero will be input to ANSWERS as having subsurface drainage. Note: the drainage coefficient for areas with subsurface is set with the other soils parameters in step 2. If "all" or "none" of the watershed is simulated as having subsurface drainage, no input map is required; otherwise a raster map is used to specify areas with subsurface drainage.

Step 8 Catalogue channel parameters
Map input: Channels
Other input: Channel width and roughness coefficient for each category of channel
Description: Watershed cells with a well-defined channel should be defined to ANSWERS. ANSWERS assumes the channel is rectangular in cross-section and is sufficiently deep to handle runoff.

To prepare channel data for use with ANSWERS, the following is needed: a raster map layer of the channels in the watershed and a description of width (meters) and roughness (Manning's "n") for each channel category found in the layer.

It is suggested that the aspect map from Step 4 b created in conjunction with the map, since ANSWERS will abort operation if a one channel element does not flow directly into another adjacent channel element.

Step 9 Define channel slopes
Map input: Channel slope
Description: An optional input to ANSWERS is the slope of channels. If a channel slope input is not given, ANSWERS assumes the slope for the channel is the same as the overland slope for the element.

If desired, a raster map may be used to define channel slope values. To do so, a raster map should be prepared with category values for channel slopes in tenths of a percent (i.e. a category value of 31 would indicate a channel slope of 3.1 percent).

Note: Even though channel slopes are an optional input to ANSWERS, this step must be run if only to say no map will be used.

Step 10 Specify BMP's in watershed
Map input: Tile Outlet Terrace, Sedimentation Pond, Grassed Waterway, and/or Field Borders.
Other input: Grassed waterway or field border width (meters)
Description: This step provides a menu to prepare any or none of the four structural Best Management Practices (BMPs) that ANSWERS recognizes. Many BMPs can be described to ANSWERS by changing variables describing the surface condition of the soil. Practices which are tillage-oriented, for example, are described in the soils and land use sections. Gully structures such as a drop spillway may be simulated by reducing channel slope. On the other hand, BMPs which are structural in nature require a change in land use (row crop to grass for waterways, for example). ANSWERS recognizes four optional BMP structures. Even though the use of BMP structures is optional, this step still must be run to ver- ify this. NOTE: Since ANSWERS will recognize one BMP for a given watershed element, the most effective BMP should be used. The following is a brief discussion of the BMPs:

1. ANSWERS Tile Outlet Terrace Assumptions:

- Trap efficiency of 90%
- Only lowermost terraces are described

Also, if a terrace exists only in a portion of an element, the assumption is made that all incoming flow is influenced by the BMP. Thus, elements which have only a small portion of the practice within their boundaries should not be given credit for the practice.

2. ANSWERS Sedimentation Pond Assumptions:

- Trap efficeincy of 95%
- Only ponds in upland areas should be defined. In stream structures are treated differently.

Also, if a pond exists only in a portion of an element, the assumption is made that all incoming flow is influenced by the BMP. Thus, elements which have only a small portion of the practice within their boundaries should not be given credit for the practice.

3. ANSWERS Grassed Waterway Assumptions:

- The vegetated area with in the affected element is no longer subject to any sediment detachment.
- The model deliberately prohibits deposition within the vegetation of a grass waterway, since any waterway that effectively traps sediment would soon fill and become ineffective.

For each category found in the layer, you will be prompted for width of the waterway

4. ANSWERS Field Border Assumptions

- The vegetated area with in the affected element is no longer subject to any sediment detachment. For each category found in the layer, you will be prompted for width of the field border.

Step 11 Prepare ANSWERS input and run simulation
Description: Steps 1-10 must have a status of "done" before this step can be run. (Even steps for optional inputs must be run before an ANSWERS input file can be completed). Each of the prior steps will have prepared their part of the ANSWERS input. The first function of this step is to compile all the parts together. Once the input file is complete, the simulation can be run. (NOTE: *r.answers* will call the ANSWERS program, which must be compiled as a part of the r.*answers* installation. The source code for ANSWERS should be part of the software distributed with *r.answers*.) The error messages that ANSWERS may send to "standard output" are captured to a file by *r.answers* and displayed. If none, a message to that effect will be printed to the screen (although this doesn't mean that the simulation ran entirely error-free). The primary output of the simulation is captured to another file, then processed to separate it into component parts of 1) text - the verbose reiteration of the inputs and summary of watershed characteristics. This is useful for checking to insure that inputs were read in by ANSWERS properly; 2) outlet hydrograph data of rainfall, runoff and sediment yield and concentration. If these data are in order, they will be processed into a format readable by the *d.linegraph* program for display; 3) individual element net sedimentation showing sediment loss or deposition, if any, for each raster element in the watershed. Also, sediment deposition in channel elements. This step will prompt for names to use for new watershed maps it will create by extracting these data from the output. If the simulation event did not create sediment loss or

deposition, or channel deposition for the scenario, the given map will not be created. To find out how to access the output files, see the description of step 12, below.

Step 12 Miscellaneous Command Menu
This step calls a menu that allows access to the project files in the project database and to a function that prepares a summary of the project's current status.

The project database is where *r.answers* stores all the inputs, output, and other non-map data associated with the project. See the "FILES" section (below) for more information. There are two sections to the project data, since rainfall data are kept in a separate directory. When using this step to access database files, the program will list both the project data and the rain data files, and ask which section you wish to access. Next you will be prompted for the name of the file to access. This request will be turned over to the "file handler program" which facilitates sending a file to the screen, copying to another file, or printing.

The project status function available under step 12 creates a helpful summary of the project, and then passes control to the "file handler program" for display, copying to a file, or printing.

Index of ANSWERS on GRASS database
Each project will create and use the following files in $LOCATION/answers/[project name]/data. For the most part, there isn't much to see, unless something is not working right. If that is the case, The first thing to check would be files listed here under the Output section or Input file. Furthermore, attempting to fix a problem by editing any of these files could prove to have unpredictable results. Once a problem is identified (with the input maps or parameters, most likely) fix the input maps if need be, run *r.answers* again to make any changes, such a using a different map or correcting parameters. Remember that if a map is changed the menu step that uses it must be run again to resample the inputs. Run step 11 again to create a new input file and re-run ANSWERS.

General project data
reclass reclass rules to create project MASK regionproject region coordinates

ANSWERS Input file
answers_input file created to use as input to ANSWERS

ANSWERS Output
When ANSWERS is run, output from stdout is sent to answers_output and anything that may go to stderr is cap- tured in answers_error. After that the output is cut into sections. (if something unpredictable happened when ANSWERS ran, then the output and the files extracted from it may be garbled; reading answers_output and answers_error may provide clues).

answers_output complete output from running answers
answers_error errors captured when answers is run
out_chnl channel deposition data
out_sediment element sediment deposition/loss data
out_text verbose input reiteration
out_hydro outlet hydrograph data

The outlet hydrograph data is broken into 5 files to use as input to *d.linegraph*

hydro_time time increments of simulation (minutes)
hydro_rain rainfall (mm/h)
hydro_runoff runoff at outlet (mm/h)
hydro_sed1 cumulative sediment at outlet (kg)
hydro_sed2 sediment concentration in runoff (mg/l)

ANSWERS Element data
Element data files are extracted from input maps. Each line is data for a watershed cell element. When answers input is created, these files are used to create the element data section.

in_row_col watershed row and column number
in_soil soil type
in_cover land use
in_elev slope and aspect
in_chnl channel element data
in_rain rain gauge number
in_tile subsurface drainage flag

ANSWERS Predata
The following files are used to form the "predata" section of the answers input file.

chnl_predata description of channel types
cover_predata description of cover parameters
soil_predata description of soil parameters
rain_predata rain gauge data

Parameter data
These files are used by the project manager to "remember" parameters used to create the respective predata files, allowing the parameters to be read back by the program for editing.

chnl_data channel parameters
cover_data cover parameters
soil_data soil parameters

SEE ALSO
d.INTRO, d.rast.edit, d.rast.num, d.what.rast, r.slope, r.fill.direct, r.direct,

AUTHOR
Chris Rewerts, Agricultural Engineering, Purdue University

r.average

NAME

r.average - Finds the average of values in a cover map within areas assigned the same category value in a user-specified base map.
(GRASS Raster Program)

SYNOPSIS

r.average
r.average help
r.average [-c] base=name cover=name output=name

DESCRIPTION

r.average calculates the average value of data contained in a cover raster map layer for areas assigned the same category value in the user-specified base raster map layer. These averaged values are stored in the category labels file associated with a new output map layer.

The values to be averaged are taken from a user-specified cover map. The category values for the cover map will be averaged, unless the *-c* flag is set. If the *-c* flag is set, the values that appear in the category labels file for the cover map will be averaged instead (see example below).

The output map is actually a reclass of the base map (see r.reclass), and will have exactly the same category values as the base map. The averaged values computed by *r.average* are stored in the output map's category labels file.

If the user simply types *r.average* on the command line, the user is prompted for the flag setting and parameter values through the standard parser interface (see parser manual entry). Alternately, the user can supply all needed flag settings and parameter values on the command line.

Flag:
-c Take the average of the values found in the category labels for the cover map, rather than the average of the cover map's category values.

Parameters:
base=name An existing raster map layer in the user's current mapset search path. For each group of cells assigned the same category value in the base map, the values assigned these cells in the cover map will beaveraged.

cover=name An existing raster map layer containing the values (in the form of cell category values or cell category labels) to be averaged within each category of the base map.

output=name The name of a new map layer to contain program output (a reclass of the base map). Averaged values will be stored in the output map's category labels file under the user's $LOCATION/cats directory.

EXAMPLE

Assume that farms is a map with 7 farms (i.e., 7 categories), and that soils.Kfactor is a map of soil K factor values with the following category file:

```
cat   cat
value   label
0no soil data
1.10
2.15
3.17
4.20
5.24
6.28
7.32
8.37
9.43
```

Then

r.average -c base=farms cover=soils.Kfactor output=K.by.farm

will compute the average soil K factor for each farm, and store the result in the output map K.by.farm, which will be a reclass of farms with category labels as follows (example only):

```
catcat
value   label
1.1023
2.1532
3.172
4.3872
5.003
6.28
7.2345
```

NOTES
The *-c* option requires that the category label for each category in the cover map be a valid number, integer, or decimal. To be exact, if the first item in the label is numeric, then that value is used. Otherwise, zero is used. The following table covers all possible cases:

```
category label   value used by -c
_____________________________________
.12               .12
.80 KF            .8
no data            0
```

(This flag is very similar to the @ operator in *r.mapcalc*, and the user is encouraged to read the manual entry for *r.mapcalc* to see how it works there.)

The user should use the results of *r.average* with care. Since this utility assigns a value to each cell which is based on global information (i.e., information at spatial locations other than just the location of the cell itself), the resultant map layer is only valid if the geographic region and mask settings are the same as they were at the time that the result map was created.

Results are affected by the current region settings and mask.

SEE ALSO
g.region, r.cats, r.clump, r.describe, r.mapcalc, r.mask, r.mfilter, r.mode, r.neighbors, r.reclass, r.stats and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering ResearchLaboratory

r.basins.fill

NAME
r.basins.fill - Generates a raster map layer showing watershed subbasins.
(GRASS Raster Program)

SYNOPSIS
r.basins.fill
r.basins.fill help
r.basins.fill number=value c_map=name t_map=name result=name

DESCRIPTION
r.basins.fill generates a raster map layer depicting subbasins, based on input raster map layers for the coded stream network (where each channel segment has been "coded" with a unique category value) and for the ridges within a given watershed. The raster map layer depicting ridges should include the ridge which defines the perimeter of the watershed. The coded stream network can be generated as part of the *r.watershed* program, but the map layer of ridges will need to be created by hand, either through digitizing done in v.digit, or through the on-screen digitizing option accessible within *d.display* or *d.digit*.

The resulting output raster map layer will code the subbasins with category values matching those of the channel segments passing through them. A user-supplied number of passes through the data is made in an attempt to fill in these subbasins. If the resulting map layer from this program appears to have holes within a subbasin, the program should be rerun with a higher number of passes.

The user can run *r.basins.fill* either interactively or non- interactively. If the user simply types *r.basins.fill* without other arguments on the command line, the program will prompt the user for the needed parameters using the standard GRASS parser interface (see manual entry for parser).

If the user wishes to run the program non-interactively, the following parameter values must be specified on the command line:

Parameters:

number=value The number of passes to be made through the dataset.

c_map=name The coded stream network file name.

t_map=name The thinned ridge network file name.

result=name The resultant watershed partition file name.

NOTES
The current geographic region setting is ignored. Instead, the geographic region for the entire input stream's map layer is used.

SEE ALSO
d.digit, d.display, r.watershed, v.digit, and parser
See Appendix A of the GRASS Tutorial *r.watershed* for further details on the combined use of *r.basins.fill* and *r.watershed*

AUTHORS
Dale White, Dept. of Geography, The Pennsylvania StateUniversity
Larry Band, Dept. of Geography, University of Toronto, Canada

r.binfer

NAME

r.binfer - Bayesian expert system development program.
(GRASS Raster Program)

SYNOPSIS

r.binfer
r.binfer help
r.binfer [-v] input=name [output=name]

DESCRIPTION

r.binfer is an expert system shell containing an inference engine based on Bayesian statistics (reasoning from past experience). It is designed to assist human experts in a field develop computerized expert systems for land use planning and management. These expert systems are designed to aid non-experts make decisions about land use.

In Bayesian expert system programs like *r.binfer*, the system bases the probable impacts of a future land use action on the conditional probabilities about the impact of similar past actions.

OPTIONS

Flags:

-v Run verbosely, displaying messages on debugging output to standard output. Includes a listing of the symbol table used by *r.binfer*.

Parameters:

input=name Name of an existing file containing analysis instructions.

output=name Name to be assigned to the file to contain program output.
Default: *binfer.out*

Using appropriate *r.binfer* syntax, the human expert structures an input knowledge/control script with an appropriate combination of map layer category values (GRASS raster map layers that contain data on soil texture, slope, density, etc.) and attributes relevant to decision-making (e.g., rainfall, temperature, season, subjective judgement, etc.). Options exist for specifying a user interface and a data base containing prior and conditional probabilities necessary to infer the value of a goal attribute. The expert also specifies the format for display of end results (raster map layers) in the input script. New raster map layers — one for each possible inferred attribute value — are created that contain the probability of the inferred attribute value occurring in each grid cell.

Alternately, a single new map layer called *r.binfer* (or whatever output name is specified by the user) is also output. This map shows, for each grid cell, the inferred attribute value that has the highest probability of occurring in each grid cell, given the values of the input raster map layer and contextual attributes.

r.binfer scripts are typed into a file by the user using a system editor like *vi*, and then input to *r.binfer* as the input file named on the command line. For a complete description of the input syntax, see the document GRASS Tutorial: *r.binfer*. For example *r.binfer* scripts see the EXAMPLES section below. The results are used to generate the new raster map layers in the user's current mapset.

As stated above, *r.binfer* scripts contain descriptions of two types of input attributes. The map layer type attributes are actual GRASS raster map layers, with the values defined to be ranges of the categories within that raster map layer. For example, if the user chooses slope as one of the layer attributes, the possible values for the slope attribute might be the following:

flat (slopes between 0 and 5 degrees)
low (slopes between 6 and 10 degrees)
medium (slopes between 11 and 30 degrees)
steep(slopes greater than 31 degrees)

The contextual attributes are those that do not represent raster map layers, but rather, information that reflects criteria relevant to the specific decision being contemplated. For example, if the user chooses "rainfall amount" as one of the contextual attributes, possible values assigned to the "rainfall amount" attribute might be the following:

low (rainfall amounts less than an inch)
medium (rainfall amounts between one and three inches)
high (rainfall amounts grater than three inches)

The inferred attribute values are specified along with a prior probability and a table of conditional probabilities that indicate the probability of that inferred attribute value occurring given that an input attribute value has occurred.

r.binfer will determine the value of contextual attributes by prompting the user for input. It will then open each of the raster map layers corresponding to each map layer attribute. *r.binfer* then determines the values for all map layer attributes in each grid cell. Using the conditional probability tables, the prior probabilities, and Bayes' theorem, *r.binfer* calculates the output probabilities for each inferred value and writes its probability of occurrence as a percentage. It also determines which value is most likely to occur in that cell and writes that to the output file name.

EXAMPLES

The two sample scripts shown below illustrate only the use of *r.binfer* to: (1) estimate the probability that an avalanche will occur, and (2) infer the probability of finding pine mountain beetles, at each cell across a landscape, given the input map layer attributes shown below. The author makes no claims as to the correctness of using these criteria to infer either event.

Some Notes on Script Construction.

1. No Data (or what to do with category zero).

If category zero is excluded from the ranges of any layer attribute value, it is treated as "no data" and the resulting probability and combined maps will reflect this.

Otherwise, category zero is treated just like any other cell value.

2. Category ranges for layer attributes.

The category ranges are specified using r.reclass rules. For example, a value list for slope might look like this:

(flat [0 1 thru 3], gentle [4 thru 8], moderate [9 thru 15], other [16 thru 89]).

3. Question Attachments.

Question attachments can be supplied for and context attribute or attribute value. If names are chosen cleverly, the default menu should be sufficient.

4. Determinant List.

At this time the determinant list serves no real purpose.

Planned extensions to *r.binfer* will make use of this list, so just don't use it for now.

5. Probabilities.
The conditional probability table is very important, try to be sure of its accuracy.

```
        #
        # Filename:  avalanche.binfer
        #
        # This is a r.binfer script that infers the probability of an
        # avalanche occurring, given the values of the input attributes.
        #
        # NOTE:  Execute r.binfer as follows:
        #   r.binfer avalanche.binfer [output=name]
        #   If the user does not specify an output file name,
        #   the combined map will be named binfer.
        #
        # Script file output keywords:
        #
        #CombinedMap (Colortable) - assigns the combined map the given colortable.
        #NoCombinedMap - only generates probability maps
        # (one for each inferred attribute value).
        #NoProbabilityMaps - only generates combined map.
        #(Colortable) can be any of the following keywords:
        #Aspect - aspect colors,
        #Grey,Gray - grey scale,
        #Histo - histogram stretched grey scale,
        #Rainbow - rainbow colors,
        #Ramp - color ramp (default),
        #Random - random colors,
        #RYG - red yellow green,
        #Wave - color wave.
        #
        #
        # Start layer attribute section.
        #
        layer:
        #
        # Layer attribute #1 is aspect
        #
        aspect:
        #
        #   all southern exposures = 1.
        #   all eastern exposures  = 2.
        #   all western exposures  = 3.
        #   all northern exposures = 4.
        #   all others = 0.
        #
         (south[16 thru 22],east[22 23 1 thru 4],west[11 thru 15], north[5 thru 10]).
        #
        # Layer attribute #2 is slope
        #
        slope:
        #
        # low - 0 to 9 degrees
        # moderate - 10 - 19 degrees
        # steep - 20 - 29 degrees
        # severe - 30 - 88 degrees
        #
         (low[1 thru 10],moderate[11 thru 19],steep[20 thru 30],severe[31 thru 89]).
        %
        # End of layer section

        #
        # Start context section
        #
        context:
        #
        # Contextual attribute #1 is temperature
        # NOTE: A menu will be constructed using the attribute name and
        #  the names of the attribute values.
        #  The user will be prompted to enter his choice.
#
```

```
temperature:
 (freezing,cold,warm,hot).
#
# Contextual attribute #2 is snowfall_amt
# NOTE: A menu will be constructed using the question attachments
#  supplied here.
#  The user will be prompted to enter his choice.
#
snowfall_amt:
 (a {question “Less than one foot.”},
  b {question “Between a foot and four feet.”},
  c {question “More than four feet.”})
  {question “How much snow has accumulated ?”}.
%
# End of context section.

#
# Start inferred section
#
inferred:
#
# Inferred attribute is avalanche.
#
avalanche:
#
# Inferred attribute value “high”.
# A colortable of Ramp will be assigned (default).
# NOTE: Prior probability, and conditional probabilities are given in
#  this section.
#
 (high <0.20>
 [0.10,0.50,0.20,0.20;
  0.05,0.15,0.20,0.60;
  0.80,0.15,0.00,0.05;
  0.05,0.35,0.60;] ,
#
# Inferred attribute value “moderate”.
# A colortable of Grey will be assigned.
#
 moderate Grey <0.30>
 [0.15,0.35,0.25,0.25;
  0.10,0.20,0.20,0.50;
  0.75,0.20,0.00,0.05;
  0.05,0.35,0.60;] ,
#
# Inferred attribute value “low”.
# A colortable of Rainbow will be assigned.
#
 low Rainbow <0.50>
 [0.25,0.25,0.25,0.25;
  0.25,0.25,0.25,0.25;
  0.50,0.30,0.10,0.10;
  0.10,0.40,0.50;] ).
%
# End of inferred section.
# End of avalanche.binfer script.



#
# Filename: bugs.binfer
#
# This is a r.binfer script that infers the probability of finding
# pine mountain beetles, given the input layer attributes below.
# NOTE:  Execute r.binfer as follows:
#binfer bugs.binfer [output=name]
#   if the user does not specify an output name, the combined map
#   will be named binfer.
#
# Script file output keywords:
#
```

```
#CombinedMap (Colortable) - assigns the combined map the given colortable.
#NoCombinedMap - only generates probability maps
# (one for each inferred attribute value).
#NoProbabilityMaps - only generates combined map.
#(Colortable) can be any of the following keywords:
#Aspect - aspect colors,
#Grey,Gray - grey scale,
#Histo - histogram stretched grey scale,
#Rainbow - rainbow colors,
#Ramp - color ramp (default),
#Random - random colors,
#RYG - red yellow green,
#Wave - color wave.
#
# Choose the combined map colortable to be a color wave
CombinedMap Wave
#
# Start layer attribute section.
#
layer:
#
# Layer attribute #1 is slope
#
slope:
#
#
(low[1 thru 10],moderate[11 12 13 14 thru 20],steep[21 thru 30],severe[31 thru 89]).
#
# Layer attribute #2 is aspect
#
aspect:
#
#
 (south[16 thru 22],east[22 23 1 thru 4],west[11 thru 15],
    north[5 thru 10]).
#
# Layer attribute #3 is vegcover
#
vegcover:
(other[1 thru 2],coniferous[3],deciduous[4],mixed[5],disturbed[6]).
#
# Layer attribute #4 is (forest) density
#
density:
(nonforest[1],sparse[2],moderate[3],dense[4]).
%
# End of layer section.

#
# Start inferred section
#
inferred:
#
# Inferred attribute is bugs
#
bugs:
#
# Inferred attribute value “bugs”.
# A colortable of Ramp will be assigned (default).
# NOTE: Prior probability, and conditional probabilities are given in
#  this section.
       #
(bugs <0.20>
 [0.124,0.416,0.371,0.090;# conditionals corresponding to slope,
  0.180,0.292,0.292,0.239;# myaspect,
  0.011,0.798,0.022,0.169,0.0; # vegcover,
  0.202,0.326,0.213,0.258;],   # and density (one per value).

#
# Inferred attribute value “nobugs”.
# A colortable of Rainbow will be assigned.
```

```
#
 nobugs Rainbow <0.80>
   [0.404,0.416,0.157,0.011;
    0.225,0.281,0.281,0.225;
    0.281,0.427,0.135,0.056,0.0;
    0.584,0.112,0.202,0.112;]).
%
# End of inferred section.
# End of bugs.binfer script.
```

SEE ALSO
GRASS Tutorial *r.binfer*

AUTHOR
Kurt Buehler, Purdue University

r.buffer

NAME

r.buffer - Creates a raster map layer showing buffer zones surrounding cells that contain non-zero category values. (GRASS Raster Program)

SYNOPSIS

r.buffer
r.buffer help
r.buffer [-q] input=name output=name distances=value[,value,...] [units=name]

DESCRIPTION

r.buffer creates a new raster map layer showing buffer (aka, "distance" or "proximity") zones around all cells that contain non-zero category values in an existing raster map layer. The distances of buffer zones from cells with non- zero category values are user-chosen. Suppose, for example, that you want to place buffer zones around roads. This program could create the raster map layer shown below on the right based on road information contained in the raster map layer shown on the left.

```
00000000000000000000000  11112222222222222333333
11100000000000000000000  00011111111112222222222
00011111111110000000000  11100000000011112222222
00000000100001110000000  22111111011110001111111
00000000100000001111111  22222221012211110000000
00000000100000000000000  32222221012222211111111
00000000100000000000000  33333221012222222222222
00000000100000000000000  33333221012233222222222
00000000100000000000000  33333221012233333333333
Category 0: No roads  Category 0: Road location
Category 1: Roads Category 1: Buffer Zone 1 around roads
Category 2: Buffer Zone 2 around roads
Category 3: Buffer Zone 3 around roads
```

INTERACTIVE PROGRAM USE

The user can run the program interactively by simply typing *r.buffer* without program arguments on the command line. The program then prompts the user for parameter values.

(1) You are requested to identify the existing raster map layer from which distance-from calculations shall be based, and a name (of your choice) for the new raster map layer which will contain the results.

(2) Then, identify the units of measurement in which buffer (distance) zones are to be calculated, and the distance of each buffer zone from each non-zero cell in the input map. The user has the option of identifying up to 60 continuous zones. The zones are identified by specifying the upper limit of each desired zone (*r.buffer* assumes that 0 is the starting point). ("Continuous" is used in the sense that each category zone's lower value is the previous zone's upper value. The first buffer zone always has distance 0 as its lower bound.) Distances can be entered in one of four units: meters, kilometers, feet, and miles.

(3) Last, calculate distances from cells containing user- specified category values, using the "fromcell" method. [The "fromcell" method goes to each cell that contains a category value from which distances are to be calculated, and draws the requested distance rings around them. This method works very fast when there are few cells containing the category values of interest, but works slowly when there are numerous cells containing the category values of interest spread throughout the area.]

The *r.buffer* program now runs the process in "background" and returns keyboard control to the user. These processes can occasionally take up to an hour or more to finish; however, because they run in the background, you are free to do other things with the computer in the interim.

NON-INTERACTIVE PROGRAM USE

The user can run *r.buffer* specifying all parameter values on the command line, using the form:

r.buffer [-q] input=name output=name distances=value[,value,...] [units=name]

Flags:

-q Run quietly

Parameters:

input=name The name of an existing raster map layer whose non-zero category value cells are to be surrounded by buffer zones in the output map.

output=name The name assigned to the new raster map layer containing program output. The output map will contain buffer zones at the user-specified distances from non-zero category value cell in the input map.

distances=value[,value,...] The distance of each buffer zone from cells having non-zero category values in the input map.

units=name The unit of measurement in which distance zone values are to be calculated. Possible choices for name are: meters, kilometers, feet, and miles. The default units used, if unspecified by the user, are meters.

EXAMPLE
In the example below, the buffer zones would be (in the default units of meters): 0-10, 11-20, 21-30, 31-40 and 41-50.

Format:

r.buffer input=name output=name distances=value[,value,...] [units=name]

Example:

r.buffer input=map.in output=map.out distances=10,20,30,40,50 units=meters

NOTES
r.buffer measures distances from center of cell to center of cell using Euclidean distance measure for planimetric databases (like UTM) and using ellipsoidal geodesic distance measure for latitude/longitude databases.

r.buffer calculates distance zones from all cells having non-zero category values in the input map. If the user wishes to calculate distances from only selected input map layer category values, the user should run (for example) r.reclass prior to r.buffer, to reclass all categories from which distance zones are not desired to be calculated into category zero.

SEE ALSO
r.region, r.mapcalc,r.reclass

AUTHORS
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
James Westervelt, U.S. Army Construction Engineering Research Laboratory

r.cats

NAME
r.cats - Prints category values and labels associated with user-specified raster map layers.
(GRASS Raster Program)

SYNOPSIS
r.cats
r.cats help
r.cats map=name [cats=range[,range,...]] [fs=character|space|tab]

DESCRIPTION
r.cats prints the category values and labels for the raster map layer specified by map=name to standard output. The user can specify all needed parameters on the command line, and run the program non-interactively. If the user does not specify any categories (e.g., using the optional *cats=range[,range,...]* argument), then all the category values and labels for the named raster map layer that occur in the map are printed. The entire map is read, using *r.describe*, to determine which categories occur in the map. If a listing of categories is specified, then the labels for those categories only are printed. The cats may be specified as single category values, or as ranges of values. The user may also (optionally) specify that a field separator other than a space or tab be used to separate the category value from its corresponding category label in the output, by using the *fs=character|space|tab* option (see example below). If no field separator is specified by the user, a tab is used to separate these fields in the output, by default. The output is sent to standard output in the form of one category per line, with the category value first on the line, then an ASCII TAB character (or whatever single character or space is specified using the *fs* parameter), then the label for the category.

If the user simply types *r.cats* without arguments on the command line the program prompts the user for parameter values using the standard GRASS parser interface described in the manual entry for parser.

EXAMPLES

r.cats map=soils

prints the values and labels associated with all of the categories in the soils raster map layer;

r.cats map=soils cats=10,12,15-20

prints only the category values and labels for soils map layer categories 10, 12, and 15 through 20; and

r.cats map=soils cats=10,20 fs= :

prints the values and labels for soils map layer categories 10 and 20, but uses ":" (instead of a tab) as the character separating the category values from the category values in the output.

Example output:

```
10:Dumps, mine, Cc
20:Kyle clay, KaA
```

NOTES
Any ASCII TAB characters which may be in the label are replaced by spaces.
The output from *r.cats* can be redirected into a file, or piped into another program.

SEE ALSO
UNIX Manual entries for awk and sort
r.coin, r.describe, r.rast.what, r.support and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.clump

NAME
r.clump - Recategorizes data in a raster map layer by grouping cells that form physically discrete areas into unique categories.
(GRASS Raster Program)

SYNOPSIS
r.clump
r.clump help
r.clump [-q] input=name output=name [title="string"]

DESCRIPTION
r.clump finds all areas of contiguous cell category values in the input raster map layer name. It assigns a unique category value to each such area ("clump") in the resulting output raster map layer name. If the user does not provide input and output map layer names on the command line, the program will prompt the user for these names, using the standard parser interface (see manual entry for parser).

Category distinctions in the input raster map layer are preserved. This means that if distinct category values are adjacent, they will NOT be clumped together. (The user can run *r.reclass* prior to *r.clump* to recategorize cells and reassign cell category values.)

Flag:

-q Run quietly, without printing messages on program progress to standard output.

Parameters:

input=name Name of an existing raster map layer being used for input.

output=name Name of new raster map layer to contain program output.

title="string" Optional title for output raster map layer, in quotes. If the user fails to assign a title for the output map layer, none will be assigned it.

ALGORITHM
r.clump moves a 2x2 matrix over the input raster map layer. The lower right-hand corner of the matrix is grouped with the cells above it, or to the left of it. (Diagonal cells are not considered.)

NOTES
r.clump works properly with raster map layers that contain only "fat" areas (more than a single cell in width). Linear elements (lines that are a single cell wide) may or may not be clumped together depending on the direction of the line — horizontal and vertical lines of cells are considered to be contiguous, but diagonal lines of cells are not considered to be contiguous and are broken up into separate clumps.

A random color table and other support files are generated for the output raster map layer.

SEE ALSO
r.average, r.buffer, r.combine, r.grow, r.infer, r.mapcalc, r.mfilter, r.neighbors, r.poly, r.reclass, r.support, r.weight, parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering ResearchLaboratory

r.coin

NAME

r.coin - Tabulates the mutual occurrence (coincidence) of categories for two raster map layers.
(GRASS Raster Program)

SYNOPSIS

r.coin
r.coin help
r.coin [-qw] map1=name map2=name units=name

DESCRIPTION

r.coin tabulates the mutual occurrence of two raster maplayers' categories with respect to one another. This analysis program respects the current geographic region and mask settings.

The user can run the program non-interactively by specifying all needed flags settings and parameter values on the command line, in the form:

r.coin [-qw] map1=name map2=name units=name

Flags:

-q Run quietly, and supress the printing of program status messages to standard output.

-w Print a wide report, in 132 columns
Default=80 columns

Parameters:

map1=name Name of first raster map layer.

map2=name Name of second raster map layer.

units=name Units of measure in which to output report results.
Options: c, p, x, y, a, h, k, m

Alternately, the user can run *r.coin* interactively by simply typing *r.coin* without command line arguments; in this case, the user will be prompted for the names of the two raster map layers which will be the subjects of the coincidence analysis. *r.coin* then tabulates the coincidence of category values among the two map layers and prepares the basic table from which the report is to be created. This tabulation is followed by an indication of how long the coincidence table will be. If the table is extremely long, the user may decide that viewing it is not so important after all, and may cancel the request at this point. Assuming the user continues, r.coin then allows the user to choose one of eight units of measure in which the report results can be given. These units are:

c cells
p percent cover of region
x percent of <map name> category (column)
y percent of <map name> category (row)
a acres
h hectares
k square kilometers
m square miles

Note that three of these options give results as percentage values: "p" is based on the grand total number of cells; "x" is based on only column totals; and "y" is based on only row totals. Only one unit of measure can be selected per report output. Type in just one of the letters designating a unit of measure followed by a <RETURN>. The report will be printed to the screen

for review. After reviewing the report on the screen, the user is given several options. The report may be saved to a file and/or sent to a printer. If printed, it may be printed with either 80 or 132 columns. Finally, the user is given the option to rerun the coincidence tabulation using a different unit of measurement.

Below is a sample of tabular output produced by *r.coin*. Here, map output is stated in units of square miles. The report tabulates the coincidence of the Spearfish sample database's owner and road raster map layers' categories. The owner categories in this case refer to whether the land is in private hands (category 1) or is owned by the U.S. Forest Service (category 2). The roads map layer categories refer to various types of roads (with the exception of category value "0", which indicates "no data"; i.e., map locations at which no roads exist). *r.coin* does not report category labels. The user should run *r.report* or *r.cats* to obtain this information.

The body of the report is arranged in panels. The map layer with the most categories is arranged along the vertical axis of the table; the other, along the horizontal axis. Each panel has a maximum of 5 categories (9 if printed) across the top. In addition, the last two columns reflect a cross total of each column for each row. All of the categories of the map layer arranged along the vertical axis are included in each panel. There is a total at the bottom of each column representing the sum of all the rows in that column. A second total represents the sum of all the non-zero category rows. A cross total (Table Row Total) of all columns for each row appears in a separate panel.

Note how the following information may be obtained from the sample report.

In the Spearfish data base, in area not owned by the Forest Service, there are 50.63 square miles of land not used for roads. Roads make up 9.27 square miles of land in this area. Of the total 102.70 square miles in Spearfish, 42.80 square miles is owned by the Forest Service. In total, there are 14.58 square miles of roads. There are more category 2 roads outside Forest Service land (2.92 mi. sq.) than there are inside Forest land boundaries (0.72 mi. sq.).

Following is a sample report.

```
+----------------------------------------------------------------------------+
|COINCIDENCE TABULATION REPORT                                               |
|----------------------------------------------------------------------------|
|Location: spearfish    Mapset: PERMANENT   Date: Wed Jun 1  13:36:08        |
|                                                                            |
| Layer 1: owner— Ownership                                                  |
| Layer 2: roads— Roads                                                      |
| Mask:    none                                                              |
|                                                                            |
| Units:   square miles                                                      |
|----------------------------------------------------------------------------|
| Window:              North: 4928000.00                                     |
|          West: 590000.00           East: 609000.00                         |
|                      South: 4914000.00                                     |
+----------------------------------------------------------------------------+
```

Panel #1 of 1

```
+-------------------------------------------+
|       |    owner      |   Panel Row Total    |
| cat#  |   1   |   2   | w cat 0  | w/o cat 0 |
|-------------------------------------------|
| r0    |50.63  | 37.49 |  88.12   |   88.12   |
| o1    | 1.53  | 0.68  |   2.21   |   2.21    |
| a2    | 2.92  | 0.72  |   3.64   |   3.64    |
| d3    | 3.97  | 2.57  |   6.54   |   6.54    |
| s4    | 0.65  | 1.36  |   2.00   |   2.00    |
|  5    | 0.19  | 0.00  |   0.19   |   0.19    |
|-------------------------------------------|
|Total  |       |       |          |           |
|with 0 |59.90  | 42.80 | 102.70   |  102.70   |
|-------------------------------------------|
|w/o 0  | 9.27  | 5.32  |14.58     |   14.58   |
+-------------------------------------------+
```

	Table Row Total	
cat#	w cat 0	w/o cat 0
r0	88.12	88.12
o1	2.21	2.21
a2	3.64	3.64
d3	6.54	6.54
s4	2.00	2.00
5	0.19	0.19
Total with 0	102.70	102.70
w/o 0	14.58	14.58

NOTES

It is not a good idea to run *r.coin* on a map layer which has a monstrous number of categories (e.g., unreclassed elevation). Because *r.coin* reports information for each and every category, it is better to reclassify those categories (using reclass) into a more manageable number prior to running r.coin on the reclassed raster map layer.

r.coin calculates the coincidence of two raster map layers. Although *r.coin* allows the user to rerun the report using different units, it is not possible to simply rerun the report with different map layers. In order to choose new map layers, it is necessary to rerun *r.coin.*

SEE ALSO

r.region, r.cats, r.describe, r.mask, r.reclass, r.report, r.stats

AUTHORS

Michael O'Shea, U.S. Army Construction Engineering Research Laboratory
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.colors

NAME

r.colors - Creates/Modifies the color table associated with a raster map layer.
(GRASS Raster Program)

SYNOPSIS

r.colors
r.colors help
r.colors [-wq] map=name color=type

DESCRIPTION

r.colors allows the user to create and/or modify the color table for a raster map layer. The map layer (specified on the command line by *map=name*) must exist in the user's current mapset search path. The color table specified by *color=type* must be one of the following:

color type description

```
aspect   (aspect oriented grey colors)
grey     (grey scale)
grey.eq  (histogram-equalized grey scale)
gyr      (green through yellow to red colors)
rainbow  (rainbow color table)
ramp     (color ramp)
random   (random color table)
ryg      (red through yellow to green colors)
wave     (color wave)
rules    (create new color table based on user-specified rules)
```

If the user specifies the *-w* flag, the current color table file for the input map will not be overwritten. This means that the color table is created only if the map does not already have a color table. If this option is not specified, the color table will be created if one does not exist, or modified if it does.

If the user sets the *-q* flag, *r.colors* will run quietly, Without printing numerous messages on its progress to standard output.

Color table types aspect, grey, grey.eq (histogram-equalized grey scale), gyr (green-yellow-red), rainbow, ramp, ryg (red-yellow-green), random, and wave are pre-defined color tables that *r.colors* knows how to create without any further input.

The rules color table type will cause *r.colors* to read color table specifications from standard input (stdin) and will build the color table accordingly. Using color table type rules, there are three ways to build a color table: by color list, by category values, and by "percent" values.

Building a customized color table by color list is the simplest of the three rules methods: just list the colors you wish to appear in the color table in the order that you wish them to appear. Use the standard GRASS color names: white, black, red, green, blue, yellow, magenta, cyan, aqua, grey, gray, orange, brown, purple, violet, and indigo.

For example, to create a color table for the raster map layer elevation that assigns greens to low map category values, browns to the next larger map category values, and yellows to the still larger map category values, one would type:

r.colors map=elevation color=rules
green
brown
yellow
end

To build a color table by category values' indices, the user should determine the range of category values in the raster map layer with which the color table will be used. Specific category values will then be associated with specific colors. Note that a color does not have to be assigned for every valid category value because *r.colors* will interpolate a color ramp to fill in

where color specification rules have been left out. The format of such a specification is as follows:

category_value color_name
category_value color_name
....
....
category_value color_name
end

Each category value must be valid for the raster map layer, category values must be in ascending order and only use standard GRASS color names (see above).

Colors can also be specified by color numbers each in the range 0-255. The format of a category value color table specification using color numbers instead of color names is as follows:

category_value red_number green_number blue_number
category_value red_number green_number blue_number
.....
.....
category_value red_number green_number blue_number
end

Specifying a color table by "percent" values allows one to treat a color table as if it were numbered from 0 to 100. The format of a "percent" value color table specification is the same as for a category value color specification, except that the category values are replaced by "percent" values, each from 0-100, in ascending order. The format is as follows:

percent_value% color_name
percent_value% color_name
....
....
percent_value% color_name
end

Using "percent" value color table specification rules, colors can also be specified by color numbers each in the range 0-255. The format of a percent value color table specification using color numbers instead of color names is as follows:

percent_value% red_number green_number blue_number
percent_value% red_number green_number blue_number
.....
.....
percent_value% red_number green_number blue_number
end

Note that you can also mix these three methods of color table specification; for example:

0 black
10% yellow
78 blue
magenta
purple
brown
100% 0 255 230
end

EXAMPLES
(1) The below example shows how you can specify colors for a three category map, assigning red to category 1, green to category 2, and blue to category 3. Start by using a text editor, like *vi*, to create the following rules specification file. Save it with the name rules.file.

1 red
2 green
3 blue
end

The color table can then by assigned to map threecats by typing the following command at the GRASS> prompt:

cat rules.file | r.colors map=threecats color=rules

(2) To create a natural looking LUT for true map layer elevation, use the following rules specification file. It will assign light green shades to the lower elevations (first 20% of the LUT), and then darker greens (next 15%, and next 20%) and light browns (next 20%) for middle elevations, and darker browns (next 15%) for higher elevations, and finally yellow for the highest peaks (last 10% of LUT).

0%	*0*	*230*	*0*
20%	*0*	*160*	*0*
35%	*50*	*130*	*0*
55%	*120*	*100*	*30*
75%	*120*	*130*	*40*
90%	*170*	*160*	*50*
100%	*255*	*255*	*100*

SEE ALSO
d.colormode, d.colors, d.colortable, d.display, d.legend, p.colors, r.support

AUTHORS
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
David Johnson, DBA Systems, Inc. supplied the idea to create this program

r.combine

NAME

r.combine - Allows category values from several raster map layers to be combined.
(GRASS Raster Program)

SYNOPSIS

r.combine
r.combine < inputfile

DESCRIPTION

r.combine accepts commands that are similar to those used for boolean combinations (AND, OR, NOT) in order to overlay user-selected groups of categories from different raster map layers. After the *r.combine* program is started, the users are asked if they want the graphic output to go to a color graphics monitor. If a color graphics monitor is not used, the graphic output is displayed on the terminal screen. This display is, of course, quite rough. It consists of numerals representing the various categories that result from the r.combine analysis. Following this question, the user will see a [1]:. This is the first prompt, and indicates that *r.combine* is ready to receive input from the user.

The following commands perform operations in *r.combine*:

Command [Alias]	Followed by	Such as
NAME [name]	name for raster map output	sandstone
GROUP [group] [grp]	category values and a raster map	1-40 (elevation.255)
AND [and] [&][&&]	expression describing a raster map and categories	(grp 4 (soils)) (grp 2 (owner))
OR [or] [|][| |]|	expression describing a raster map and categories	(grp 4 (soils)) (grp 2 (owner))
NOT [not] [~]	expression describing a raster map and categories	(grp 2 3 (roads))
OVER [over] [overlay]	existing raster map and color	sandstone yellow
COVER [cover]	existing raster map	sandstone

r.combine uses the same colors for all the operating commands. This is the *r.combine* color table:

0 black	4 blue	8 grey	12 blue/grey
1 red	5 purple	9 red/grey	13 purple/grey
2 yellow	6 green	10 yellow/grey	14 green/grey
3 orange	7 white	11 orange/grey	15 dark grey

The user may enter commands either line-by-line from within *r.combine*, or by typing the commands into a file which is then read into *r.combine* using the UNIX redirection symbol < . The command format is the same for the two methods. The line-by-line method, however, does not allow as much flexibility as does use of an input file. If a line containing a syntax error is entered on the *r.combine* command line, it is cleared; the line must then be reentered in its entirety. Input files containing mistakes, however, can easily be modified (rather than recreated). An input file is especially advantageous when a more complex series of statements is input to r.combine.

r.combine uses two types of commands: those which perform operations, and those which have some other function. *r.combine* can probably best be learned by following examples, so pay special attention to those included below with the operating command descriptions. Notice two things in particular:

1) All parentheses must be closed. A raster map layer name must often be enclosed within parentheses; each time one of the above commands is used, it and its appropriate companions must also be enclosed within parentheses.

2) Certain spaces are important. Generally, *r.combine* requires at least one space before an opening parenthesis (except when it is the first character in an expression). *r.combine* ignores extra spaces and tab characters.

OPERATING COMMANDS

Below is a summary of the syntax of the operating commands, a description of each command, and examples using the Spearfish sample data base.

NAME
(NAME new_map_name (Expression))
Allows graphic output to be saved in the raster map layer new_map_name, so that it is available for additional analysis or for future viewing. The results of performing the expression in parentheses is then placed into the named output raster map layer (here, new_map_name). Note that this means that r.combine may be used to create new raster map layers from existing ones. r.combine automatically creates a color table for the new raster map layers; however, the user should run the GRASS program r.support to fill in category assignments and history information if the new raster map layer is to be saved for future use in the mapset.

example:
(NAME sandstone (GROUP 4 (geology)))
The above command will result in the creation of a new raster map layer named sandstone, noting the locations of cells with geology category value 4. You must then run the GRASS program r.support in order to label the categories present in the new raster map layer. Resultant categories:

0 - black: other than sandstone
1 - red: sandstone

GROUP
(GROUP category_values (existing raster map layer))
Selects out categories of the desired values from the existing raster map layer which is indicated in parentheses directly after the category grouping. It also works to select out just one category from the map layer. Any of the following are legal category groupings:

2
1-18
1 2 5-7.

example:
(GROUP 1-40 (elevation.255))
Depicts only the area with elevation 1187 meters or less (i.e., elevation map layer category values 1 through 40 only). Resultant categories:

0 - black: elevation > 1187 m
1 - red : elevation <= 1187 m

example:
(NAME low.hi (GROUP 1-40 238-255 (elevation.255)))
Depicts only those areas with elevations of either 1187 meters or less, or in excess of 1787 meters (elevation categories 1-40, and 238-255). The graphic output is saved in the new raster map layer called low.hi. Resultant categories:

0 - black : elevation > 1187 m and < 1787 m
1 - red : elevation <= 1187 m and >= 1787 m

AND
(AND (Expression A) (Expression B))
Combines two map layers and creates a new one, when BOTH of the category values associated with the same given cell location in the two combined map layers are non-zero, a category value of 1 is assigned to that cell in the new map layer. If, however, either map layer assigns a category value of zero to the same given cell location, the category value associated with this cell's location in the resultant map layer also becomes zero.

For example,

raster map 1 2 2 0
2 1 0
0 0 0 1 0 0 results
AND—> 1 1 0
raster map 2 1 0 1 0 0 0
1 1 0
1 1 0

example:

(AND (GROUP 4 7-9 (geology)) (GROUP 2 (owner)))
Depicts the occurrences of categories 4, 7, 8, and 9 from the map layer geology whenever they occur on U.S. Forest Service property. Results are displayed to the terminal screen. Resultant categories:

0 - black : no data occurred in one or the other of the raster map layers
1 - red : the AND condition is met

Note that if neither map layer contained any areas of "no data", the resultant raster map layer would include only 1's.

Example:
(NAME sand (AND (GROUP 4 7-9 (geology)) (GROUP 2 (owner))))
Same as above, except the results are saved in the map layer sand.

OR
(OR (Expression A) (Expression B))
Combines two map layers and creates a new one; when EITHER of the category values associated with the same given cell location in the two combined map layers is non-zero, a category value of 1 is assigned to that cell in the new map layer. If, however, both map layers assign a category value of zero to the same given cell location, the category value of this cell in the resultant map layer also becomes zero. Only two map layers may be combined at one time. For example:

raster map 1 2 2 0
2 1 0
0 0 01 1 1 results
OR —> 1 1 0
raster map 2 1 0 11 1 0
1 1 0
1 1 0

Example:
(OR (GROUP 4 7-9 (geology)) (GROUP 2 (owner)))
Depicts all occurrences of categories 4, 7, 8, and 9 from the map layer geology as well as showing all the land which is U.S. Forest Service property. Results are displayed to the terminal screen. Resultant categories:

```
0 - black: this area has neither the values of 4, 7, 8, or 9 nor is it on U.S.Forest Service property
1 - red: this area meets one or the other of the conditions noted above
```

Note that no distinction is made between those places where conditions are met in both map layers and where they are met in only one. See the *r.combine* command OVER if it is necessary to make that distinction.

NOT(NOT (Expression))
Negates Expression in order to define a new map layer which contains the opposite of what is defined by Expression. The new raster map layer will contain category values of either 0 or 1. 0 values would indicate that the NOT conditions were not met. Cell values of 1 would indicate that the NOT conditions were met. In order to specify the map layer in which to save the output from NOT, use the *r.combine* command NAME.

Example:
(NAME rds (NOT (GROUP 0 (roads))))
Areas containing category zero in the existing map layer roads indicate those locations within the data base where roads do not exist. Negating that expression leaves us with all other areas - i.e., those locations at which roads do exist. Here, the graphic output is saved in the raster map layer named rds. Resultant categories:

```
0 - black: no roads
1 - red  : roads
```

The same results could have been obtained with: (NAME rds (GROUP 1-5 (roads))). NOT is most useful in those cases where it is simpler to define something on the basis of what it is not than on the basis of what it is.

OVER
(OVER color (Expression)) or (OVER existing_rastermap color (Expression))
Performs a transparent overlay operation. This means that when a map layer which depicts some feature in blue is overlain with one which depicts a feature in yellow, the resulting raster map layer will show areas of overlap in green; areas in the two raster map layer that do not overlap other areas maintain their original colors (i.e., yellow or blue).

OVER may be run with or without an existing map layer name. If the user does not specify an existing raster map layer name, OVER applies the color specified to the expression in parentheses and displays the results. If an existing raster map layer name is specified, OVER applies the color to the expression (just as before) and then overlays the results on top of the existing raster map layer. In order to make sense of the colors which result, use only those existing map layers created using OVER.

OVER allows the user to specify just four colors:

color	value
red	1
yellow	2
blue	4
grey	8

These four colors are then combined to form other colors. The number of progressive overlays allowed is limited to four (one for each of the basic colors above). The actual number of colors on the resultant raster map layer, however, varies depending on the distribution of the features and on the interaction of the features from the different map layers which are overlain. When two or more of these colors are overlain, new colors are created. The numerical values associated with the colors above are significant, in that the values of any additional colors created reflect the sum of two or more of the four above. These overlain color values appear on the resultant overlay as cell (category) values. The user should know what these values represent in order to know what category information is to be associated with the new map layer (entered using the GRASS *r.support* command), and to know the significance of this and subsequent analyses involving the new map layer.

Any of these colors and category values may result from OVER. Note that this is the same as the *r.combine* color table listed above.

0 black	4 blue	8 grey	12 blue/grey
1 red	5 purple	9 red/grey	13 purple/grey
2 yellow	6 green	10 yellow/grey	14 green/grey
3 orange	7 white	11 orange/grey	15 dark grey

The syntax for OVER makes no provision for a new raster map layer name. It is necessary to use the r.combine operator NAME to specify a new raster map layer name in which to save the graphic output generated by OVER. If the user runs OVER without specifying an output raster map layer name, output is displayed to the terminal. However, this output is available for future use only if it is saved using the NAME command.

example:
(NAME park.or.priv (OVER red (GROUP 1 (owner))))
The new raster map layer park.or.priv displays private land (i.e., category 1 of the raster map layer owner) in red, and displays U.S. Forest Service land (i.e., "no data" areas within the owner map layer) as black. Resultant categories:

```
0 - black:  park
1 - red :  private land
```

example:
(NAME roads.or.not (OVER park.or.priv yellow (GROUP 0 (roads))))
Category 0 in the map layer roads is overlain in yellow on top of the park.or.priv map layer created above. The output is placed in a new map layer named roads.or.not. Resultant categories in roads.or.not are:

```
0 - black  :  park; road
1 - red    :  private; road
2 - yellow :  park; no road
3 - orange :  private; no road
```

example:
(NAME low.elev (OVER park.or.priv blue (GROUP 1-19 (elevation.255))))
The elevation categories of 1123 meters or less from the map layer elevation.255 are assigned the color blue and then overlain on park.or.priv (generated in the previous example). Resultant categories in the new map layer low.elev are:

```
0 - black  :  park; > 1123 m
1 - red    :  private; > 1123m
4 - blue   :  park; <= 1123 m
5 - purple :  private; <= 1123m
```

Note how category 5 is the sum of red (1) + blue (4) (i.e., the intersection of areas containing low elevations and private lands with roads).

COVER
(COVER existing_map (Expression))
Performs an opaque overlay operation. This means that where the top map layer contains "holes" (cell category values of 0), the bottom map layer will show through. Where the top map layer contains information on a feature, it will cover (substitute its category value for) whatever is below it. The top map layer is that which is defined by Expression. The bottom map layer is existing_map; this map layer must already exist.

The user does not specify colors with COVER. COVER uses the default color table that is listed above with OVER. Colors are assigned starting with the lower map layer. The category values are assigned the color from the table that corresponds with that value. For example, 1 would be red; 2, yellow; 3, orange, etc. Moving to the upper map layer COVER starts wherever it left off after the lower one. If the highest value of the lower map layer was 5, then all non-zero (i.e., places where a feature exists) cells of the upper map layer would be assigned the value of 6 (green). Note that if, in this case, the upper map layer did not have any cells of value zero, then the entire resulting new map layer would be green. The upper map layer would have been assigned the value 6 and would have completely covered that which was below it.

This is what happens:

Expression 1 1 1 0
(top raster map) 1 1 0 0
0 0 0 0 6 6 6 0 result
—> 6 6 2 0
oldmap2 5 0 0 5 5 2 2
(bottom raster map) 0 5 2 0
5 5 2 2

As many map layers may be overlain as is desired. However, there is a practical limit on the number of map layers that can be used while still generating sensible output. That number depends on the features involved in each map layer, and how

many cells within the upper (overlying) map layers contain category values of zero (holes through which underlying data can be seen).

COVER has no provision for saving graphic output. Use the *r.combine* command NAME to save output in a raster map layer.

Example:
(NAME lo.elev (COVER owner (GROUP 1-19 (elevation.255))))
The categories that indicate elevation of 1123 meters or less are placed on top of the existing map layer owner. Output is saved in lo.elev. Resultant categories:

```
1 - red    : private ownership; elev > 1123 m
2 - yellow : park property; elev > 1123 m
3 - orange : park or private; elev <= 1123 m
```

Example:
(NAME sand.lo (COVER lo.elev (GROUP 4 (geology))))
Category 4 of geology (sandstone) is placed on top of lo.elev, the raster map layer created in the previous example. The output is saved in sand.lo. Resultant categories:

```
1 - red    : private ownership; elev > 1123 m; no sandstone
2 - yellow : park property; elev > 1123 m ; no sandstone
3 - orange : park or private; elev <= 1123 m; no sandstone
4 - blue   : park or private; any elev; sandstone
```

ADDITIONAL COMMANDS
r.combine also contains a number of commands which are not used for operations, but serve a variety of other functions. Additional commands:

Command	Alias	Followed By
QUIT	quit q exit bye	
CATS	categories cats	existing raster map
EXP	exp expr	number of an expression
!		shell command e.g. vi comb.1
<		existing input file
WINDOW	window	existing raster map layer
HISTORY	history hist	
HELP	help	combine command for which help is needed
ERASE	erase	

QUIT
Allows the user to exit from *r.combine* while remaining within the GRASS session.

CATS raster map
Gives user an on-line listing of categories and labels for the map layer specified.
For example:

[1]:CATS owner

EXPEXP expression number
During an *r.combine* session, each completed expression and command is assigned a number. This number may be used to reference the expression to which it is assigned; this means that the user can substitute the number of the expression for the expression itself.
For example:
[4]:(GROUP 5 (geology))
[5]:(NAME limestone (EXP 4))

Use the UNIX history mechanism (explained below) to determine the specific numbers associated with particular expressions in your current *r.combine* session.

!!shell command
Allows user to temporarily suspend *r.combine* and go run another command, as in the two examples below:

!vi input
!g.list type=rast

Unless otherwise specified by the user, when a file is created using a system editor (like *vi*) from within *r.combine*, this file will be placed in the user's mapset under the COMBINE directory. After the command is completed, control returns to *r.combine*.

<< input filename
Takes input from the specified filename containing *r.combine* commands. The user, of course, must previously have entered the commands into this named input file. If no pathname is given, the input file is assumed to be in the user's mapset under the COMBINE directory. For example, the user would perform the following steps to redirect input from the file comb.in into the r.combine program (while within *r.combine*):

First, the user would create the file:

!vi comb.in

Second, the user would direct *r.combine* to take its input from the file:

< comb.in

WINDOWWINDOW raster_map
Gives on-line geographic region (window) information about the raster map layer specified.

HISTORY
Provides a listing of all previously completed expressions used within the current *r.combine* session, and the numbers associated with the execution of these commands.

HELP command
An on-line help facility for r.combine commands only. Type in the name of the *r.combine* command for which help is needed, to see the entry for that command.

ERASE
Will cause the color graphics monitor to clear.

NOTES
In all of the above examples, only a single line of input was provided to *r.combine*. However, since *r.combine* conveniently ignores extra spaces and tabs, it is possible to type input to *r.combine* in the manner outlined below. Users may find this to more clearly exhibit the relationships involved and parentheses needed. This can be typed as shown below either directly at the *r.combine* command line, or redirected into *r.combine* from an already existing file.

example:

```
(NAME good.place
(AND
(OR
(GROUP 1 2 5 (geology))
(GROUP 1-5 (elevation.255))
)
(NOT
(GROUP 1-4 (landuse))
)
)
)
```

Such involved input to *r.combine* might conveniently be typed into an input file, and then input to *r.combine* using the UNIX redirection mechanism < .

SEE ALSO
GRASS Tutorial:, *r.combine, r.infer, r.mapcalc, r.weight*

AUTHORS
L. Van Warren, U.S. Army Construction Engineering Research Laboratory
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
James Westervelt, U.S. Army Construction Engineering Research Laboratory

r.compress

NAME
r.compress - Compresses and decompresses raster files.
(GRASS Raster Program)

SYNOPSIS
r.compress
r.compress help
r.compress [-u] map=name[,name,...]

DESCRIPTION
The GRASS program *r.compress* can be used to compress and decompress raster map layers.

During compression, this program reformats raster files using a run-length-encoding (RLE) algorithm. Raster map layers which contain very little information (such as boundary, geology, soils and land use maps) can be greatly reduced in size. Some raster map layers are shrunk to roughly 1% of their original sizes. Raster map layers containing complex images such as elevation and photo or satellite images may increase slightly in size. GRASS uses a new compressed format, and all new raster files are now automatically stored in compressed form (see FORMATS below). GRASS programs can read both compressed and regular (uncompressed) file formats. This allows the use of whichever raster data format consumes less space.

As an example, the Spearfish data base raster map layer owner was originally a size of 26600 bytes. After it was compressed, the raster file became only 1249 bytes (25351 bytes smaller).

Raster files may be decompressed to return them to their original format, using the *-u* option of *r.compress*. If *r.compress* is asked to compress a raster file which is already compressed (or to decompress an already decompressed file), it simply informs the user of this and asks the user if he wishes to perform the reverse operation.

PROGRAM OPTIONS
r.compress can be run either non-interactively or interactively. In non-interactive use, the user must specify the name(s) of the raster map layer(s) to be compressed (or decompressed) on the command line, using the form map=name[,name,...] (where each name is the name of a raster map layer to be compressed or decompressed). To decompress a map, the user must include the -u option on the command line. If the -u option is not included on the command line, r.compress will attempt to compress the named map layer(s).

If the user simply types *r.compress* without specifying any map layer name(s) on the command line, r.compress will prompt the user for the names of the map layers to be compressed/decompressed, and ask whether these maps are to be compressed or decompressed. This program interface is the standard GRASS parser interface described in the manual entry for parser.

Flags:

-u If set, *r.compress* converts a compressed map to its uncompressed format. If not set, *r.compress* will attempt to compress the named map layer(s).

Parameters:

map=name[name,....] The name(s) of raster map layer(s) to be compressed or decompressed.

FORMATS
Conceptually, a raster data file consists of rows of cells, with each row containing the same number of cells. A cell consists of one or more bytes. The number of bytes per cell depends on the category values stored in the cell. Category values in the range 0-255 require 1 byte per cell, while category values in the range 256-65535 require 2 bytes, and category values in the range above 65535 require 3 (or more) bytes per cell.

The decompressed raster file format matches the conceptual format. For example, a raster file with 1 byte cells that is 100

rows with 200 cells per row, consists of 20,000 bytes. Running the UNIX command ls -l on this file will show a size of 20,000. If the cells were 2 byte cells, the file would require 40,000 bytes. The map layer category values start with the upper left corner cell followed by the other cells along the northern boundary. The byte following the last byte of that first row is the first cell of the second row of category values (moving from left to right). There are no end-of-row markers or other syncing codes in the raster file. A cell header file (cellhd) is used to define how this string of bytes is broken up into rows of category values.

The compressed format is not so simple, but is quite elegant in its design. It not only requires less disk space to store the raster data, but often can result in faster execution of graphic and analysis programs since there is less disk I/O. There are two compressed formats: the pre- version 3.0 format (which GRASS programs can read but no longer produce), and the version 3.0 format (which is automatically used when new raster map layers are created).

PRE-3.0 FORMAT:

First 3 bytes (chars) - These are a special code that identifies the raster data as compressed.

Address array (long) - array (size of the number of rows + 1) of addresses pointing to the internal start of each row. Because each row may be a different size, this array is necessary to provide a mapping of the data.

Row by row, beginning at the northern edge of the data, a series of byte groups describes the data. The number of bytes in each group is the number of bytes per cell plus one. The first byte of each group gives a count (up to 255) of the number of cells that contain the category values given by the remaining bytes of the group.

POST-3.0 FORMAT:

The 3 byte code is not used. Instead, a field in the cell header is used to indicate compressed format.

The address array is the same.

The RLE format is the same as the pre-3.0 RLE, except that each row of data is preceded by a single byte containing the number of bytes per cell for the row, and if run-length- encoding the row would not require less space than non-run- length-encoding, then the row is not encoded.

These improvements give better compression than the pre-3.0 format in 99% of the raster data layers. The kinds of raster data layers which get bigger are those in which each row would be larger if compressed (e.g., imagery band files). But even in this case the raster data layer would only be larger by the size of the address array and the single byte preceding each row.

SEE ALSO
r.support, and parser

AUTHORS
James Westervelt, U.S. Army Construction Engineering Research Laboratory
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.contour

NAME
r.contour - Produces a GRASS binary vector map of specified contours from GRASS raster map layer.

SYNOPSIS
r.contour
r.contour help
r.contour input=name output=name [levels=value,value,...,value] [minlevel=value] [maxlevel=value] [step=value]

DESCRIPTION
r.contour produces a contour map of user-specified levels from a raster map layer. This program works two ways:

1. Contours are produced from a user-specified list of levels.

2. Contours are produced at some regular increment from user-specified minimum level to maximum level. If no minimum or maximum level is specified, minimum or maximum data value will be used.

Parameters:

input=name Name of input raster map layer.

output=name Name of the binary vector file created.

levels=value,value,...,value Comma separated list of desired levels.

minlevel=value Beginning (lowest) value to be used when stepping through contours. Default is minimum data value.

maxlevel=value Ending (highest) value to be used when stepping through contours. Default is maximum data value.

step=value Increment between contour levels.

r.contour may be run interactively or non-interactively. To run the program non-interactively, the user must specify the input and output file names, either a list of levels or a step value and, optionally, minimum and maximum levels:

r.contour input=name output=name[levels=value,value,...,value][minlevel=value][maxlevel=value][step=value]

To run the program interactively, the user may simply type *r.contour* at the command line and will be prompted for parameter values.

NOTES
r.contour will either step through incremental contours or produce contours from a list of levels, not both. If both a list of levels and a step are specified, the list will be produced and the step will be ignored.

Currently, 0 is treated as a valid data value by *r.contour*.

If a contour level exactly matches a category value in the raster file, the contour line may backtrack on itself, causing illegal arcs to be produced in the output GRASS vector file.

AUTHOR
Terry Baker, U.S. Army Construction Engineering Research Laboratory

NAME

r.cost - Outputs a raster map layer showing the cumulative cost of moving between different geographic locations on an input raster map layer whose cell category values represent cost.
(GRASS Raster Program)

SYNOPSIS

r.cost
r.cost help
r.cost [-vk] input=name output=name [coordinate=x,y[,x,y,...]] [stop_coordinate=x,y[,x,y,...]]

DESCRIPTION

r.cost determines the cumulative cost of moving to each cell on a cost surface (the input raster map layer) from other user-specified cell(s) whose locations are specified by their geographic coordinate(s). Each cell in the original cost surface map will contain a category value which represents the cost of traversing that cell. r.cost will produce an output raster map layer in which each cell contains the lowest total cost of traversing the space between each cell and the user-specified points. (Diagonal costs are multiplied by a factor that depends on the dimensions of the cell.) This program uses the current geographic region settings.

OPTIONS

r.cost can be run either non-interactively or interactively. The program will be run non-interactively if the user specifies the names of raster map layers and any desired options on the command line, using the form:

r.cost [-vk] input=name output=name [coordinate=x,y[,x,y,...]] [stop_coordinate=x,y[,x,y,...]]

where the input name is the name of a raster map layer representing the cost surface map, the output name is the name of a raster map layer of cumulative cost, and each x,y coordinate pair gives the geographic location of a point from which the transportation cost should be figured.

Alternately, the user can simply type *r.cost* on the command line, without program arguments. In this case, the user will be prompted for parameter values using the standard GRASS parser interface described in the manual entry for parser.

r.cost can be run with two different methods of identifying the starting point(s). One or more points (geographic coordinate pairs) can be provided on the command line. In lieu of command line coordinates, the output map (e.g., output) is presumed to contain starting points. All non- zero cells are considered to be starting points. Beware: doing this will overwrite output with the results of the calculations. If output does exist and points are also given on the command line, the output is ignored and the coordinates given on the command line are used instead.

Flags:

-v Processing is tracked verbosely. This program can run for a very long time.

-k The Knight's move is used which improves the accuracy of the output. In the diagram below, the center location (O) represents a grid cell from which cumulative distances are calculated. Those neighbors marked with an X are always considered for cumulative cost updates. With the *-k* option, the neighbors marked with a K are also considered.

```
. . . . . . . . . . . . . . .
.   .   . K .   . K .   .   .
. . . . . . . . . . . . . . .
.   . K . X . X . X . K .   .
. . . . . . . . . . . . . . .
.   .   . X . O . X .   .   .
. . . . . . . . . . . . . . .
.   . K . X . X . X . K .   .
. . . . . . . . . . . . . . .
.   .   . K .   . K .   .   .
. . . . . . . . . . . . . . .
```

Parameters:

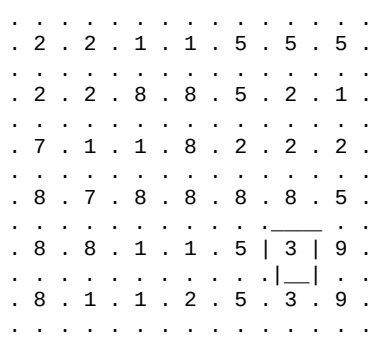
input=name Name of input raster map layer whose category values represent surface cost.

output=name Name of raster map layer to contain output. Also can be used as the map layer of the input starting points. If so used, the input starting point map will be overwritten by the output.

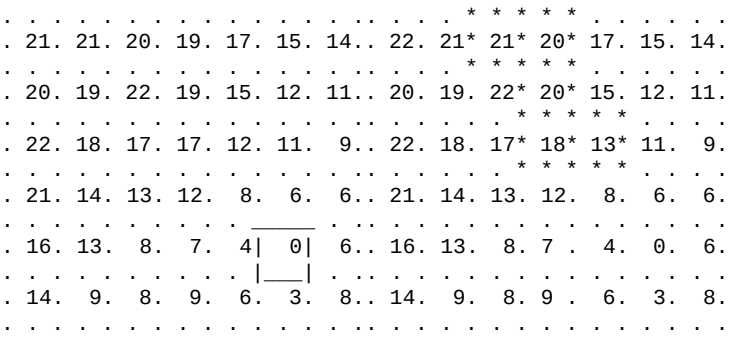
coordinate=x,y[,x,y,x,y, ...] Each x,y coordinate pair gives the easting and northing (respectively) geographic coordinates of a starting point from which to figure cumulative transportation costs for each cell. As many points as desired can be entered by the user.

stop_coordinate=x,y[,x,y,x,y, ...] Each x,y coordinate pair gives the easting and northing (respectively) geographic coordinates of a stopping point. During execution, once the cumulative cost to all stopping points has been determined, processing stops. As many points as desired can be entered by the user.

EXAMPLE
Consider the following example:
Input:



Output (using -*k*): Output (not using -*k*):



The user-provided ending location in the above example is the boxed 3 in the left-hand map. The costs in the output map represent the total cost of moving from each box (“cell”) to one or more (here, only one) starting location(s). Cells surrounded by asterisks are those that are different between operations using and not using the Knight’s move (*-k*) option. This output map can be viewed, for example, as an elevation model in which the starting location(s) is/are the lowest point(s). Outputs from *r.cost* can be used as inputs to *r.drain*, in order to trace the least-cost path given in this model between any given cell and the *r.cost* starting location(s). The two programs, when used together, generate least-cost paths or corridors between any two map locations (cells).

NOTES
If you submit the starting point map on the command line by specifying:

output=start_pt_map

the starting point map will be overwritten by the calculated output. It is wise to copy or rename (e.g., using *g.copy* or *g.rename*) the map of starting points to another name before submitting it to *r.cost*; otherwise, its contents will be overwritten.

Sometimes, when the differences among cell category values in the *r.cost* cumulative cost surface output are small, this cumulative cost output cannot accurately be used as input to *r.drain* (*r.drain* will output bad results). This problem can be circumvented by making the differences between cell category values in the cumulative cost output bigger. It is recommended that, if the output from *r.cost* is to be used as input to *r.drain*, the user multiply the input cost surface map to *r.cost* by the value of the map's cell resolution, before running *r.cost*. This can be done using *r.mapcalc* or other programs. The map resolution can be found using *g.region*.

SEE ALSO
g.copy, g.region, g.rename, r.drain, r.in.ascii, r.mapcalc, r.out.ascii, and parser

AUTHOR
Antony Awaida, Intelligent Engineering Systems Laboratory, M.I.T.
James Westervelt, US Army Construction Engineering Research Lab

r.covar

NAME
r.covar - Outputs a covariance/correlation matrix for user specified raster map layer(s).
(GRASS Raster Program)

SYNOPSIS
r.covar
r.covar help
r.covar [-mrq] map=name[,name,...]

DESCRIPTION
r.covar outputs a covariance/correlation matrix for user- specified raster map layer(s). The output can be printed, or (if run non-interactively) saved by redirecting output into a file.

The output is an N x N symmetric covariance (correlation) matrix, where N is the number of raster map layers specified on the command line. For example,

r.covar map=layer.1,layer.2,layer.3

would produce a 3x3 matrix (values are example only):

```
462.876649   480.411218   281.758307
480.411218   513.015646   278.914813
281.758307   278.914813   336.326645
```

OPTIONS
The program will be run non-interactively, if the user specifies the names of raster map layers and any desired options on the command line, using the form

r.covar [-mrq] map=name[,name,...]

where each name specifies the name of a raster map layer to be used in calculating the correlations, and the (optional) flags *-m*, *-r*, and *-q* have meanings given below. If these flags are not specified on the command line, their answers default to "no".

Flags:

-m Include zero values in the correlation calculations, due to the mask.

-r Print out the correlation matrix.

-q Run quietly (without comments on program progress).

Parameters:

map=name[,name,...] Existing raster map layer(s) to be included in the covariance/correlation matrix calculations.

Alternately, the user can simply type *r.covar* on the command line, without program arguments. In this case, the user will be prompted for flag settings and parameter values using the standard GRASS parser interface described in the manual entry for parser.

PRINCIPLE COMPONENTS
This module can be used as the first step of a principle components transformation. The covariance matrix would be input into a system which determines eigen values and eigen vectors. An NxN covariance matrix would result in N real eigen values and N eigen vectors (each composed of N real numbers). In the above example, the eigen values and corresponding eigen vectors for the covariance matrix are:

```
component      eigen value                    eigen vector
 1             1159.745202    <   0.691002    0.720528    0.480511   >
 2             5.970541       <   0.711939   -0.635820   -0.070394   >
 3             146.503197     <   0.226584    0.347470   -0.846873   >
```

The component corresponding to each vector can be produced using r.*mapcalc* as follows:

*r.mapcalc 'pc.1 = 0.691002*layer.1 + 0.720528*layer.2 + 0.480511*layer.3'*
*r.mapcalc 'pc.2 = 0.711939*layer.1 - 0.635820*layer.2 - 0.070394*layer.3'*
*r.mapcalc 'pc.3 = 0.226584*layer.1 + 0.347470*layer.2 - 0.846873*layer.3'*

Note that based on the relative sizes of the eigen values, pc.1 will contain about 88% of the variance in the data set, pc.2 will contain about 1% of the variance in the data set, and pc.3 will contain about 11% of the variance in the data set.

Also, note that the range of values produced in pc.1, pc.2, and pc.3 will not (in general) be the same as those for layer.1, layer.2, and layer.3. It may be necessary to rescale pc.1, pc.2 and pc.3 to the desired range (e.g. 0- 255). This can be done with *r.rescale*.

NOTES
If your system has a FORTRAN compiler, then the program *m.eigensystem* in src.contrib can be compiled and used to generate the eigen values and vectors.

SEE ALSO
i.pca, m.eigensystem, r.mapcalc, r.rescale, and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.cross

NAME
r.cross - Creates a cross product of the category values from multiple raster map layers.
(GRASS Raster Program)

SYNOPSIS
r.cross
r.cross help
r.cross [-qz] input=name,name[,name,...] output=name

DESCRIPTION
r.cross creates an output raster map layer representing all unique combinations of category values in the raster input layers (*input=name,name,name, ...*). At least two, but not more than ten, input map layers must be specified. The user must also specify a name to be assigned to the output raster map layer created by *r.cross*.

The program will be run non-interactively if the user specifies the names of between 2-10 raster map layers be used as input, and the name of a raster map layer to hold program output, using the form:

r.cross [-qz] input=name,name[,name,...] output=name

where each input name specifies the name of a raster map layer to be used in calculating the cross product, the output name specifies the name of a raster map layer to hold program output, and the options *-q* and *-z* respectively specify that the program is to run quietly and exclude zero data values.

Alternately, the user can simply type *r.cross* on the command line, without program arguments. In this case, the user will be prompted for needed input and output map names and flag settings using the standard GRASS parser interface described in the manual entry for parser.

Flags:

-q Run quietly. Supresses output of program percent complete messages. If this flag is not used, these messages are printed out.

-z Do not cross zero data values. This means that if a zero category value occurs in any input data layer, the combination is assigned to category zero in the resulting map layer, even if other data layers contain non-zero data. In the example given above, use of the *-z* option would cause 3 categories to be generated instead of 5.

If the *-z* flag is not specified, then map layer combinations in which not all category values are zero will be assigned a unique category value in the resulting map layer.

Parameters:

input=name,name[,name,...] The names of between two and ten existing raster map layers to be used as input. Category values in the new output map layer will be the cross-product of the category values from these existing input map layers.

output=name The name assigned to the new raster map layer created by *r.cross*, containing program output.

EXAMPLE
For example, suppose that, using two raster map layers, the following combinations occur:

map1	map2
0	1
0	2
1	1
1	2
2	4

r.cross would produce a new raster map layer with 5 categories:

```
map1   map2   output
_____________________
 0      1       1
 0      2       2
 1      1       3
 1      2       4
 2      4       5
```

Note: The actual category value assigned to a particular combination in the result map layer is dependent on the order in which the combinations occur in the input map layer data and can be considered essentially random. The example given here is illustrative only.

SUPPORT FILES

The category file created for the output raster map layer describes the combinations of input map layer category values which generated each category. In the above example, the category labels would be:

```
category       category
value          label
_____________________________
   1           layer1(0) layer2(1)
   2           layer1(0) layer2(2)
   3           layer1(1) layer2(1)
   4           layer1(1) layer2(2)
   5           layer1(2) layer2(4)
```

A random color table is also generated for the output map layer.

NOTES

When run non-interactively, *r.cross* will not protect existing files in the user's mapset. If the user specifies an output file name that already exists in his mapset, the existing file will be overwritten by the new *r.cross* output.

SEE ALSO

r.corr, r.covar, r.stats,and parser

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.describe

NAME
r.describe - Prints terse list of category values found in a raster map layer.
(GRASS Raster Program)

SYNOPSIS
r.describe
r.describe help
r.describe [-1rqd] map=name

DESCRIPTION
r.describe prints a terse listing of category values found in a user-specified raster map layer.

The program will be run non-interactively, if the user specifies the name of a raster map layer and any desired flags on the command line, using the form

r.describe [-1rqd] map=name

where the map name is the name of a raster map layer whose categories are to be described, and the (optional) flags *-1*, *-r*, *-q*, and *-d* have the meanings described below.

Alternately, the user can simply type *r.describe* on the command line, without program arguments. In this case, the user will be prompted for needed flag settings and the parameter value using the standard GRASS parser interface described in the manual entry for parser.

PROGRAM USE
The user can select one of the following two output reports from *r.describe*:

(1) RANGE. A range of category values found in the raster map layer will be printed. The range is divided into three groups: negative, positive, and zero. If negative values occur, the minimum and maximum negative values will be printed. If positive values occur, the minimum and maximum positive values will be printed. If zero occurs, this will be indicated.

(2) FULL LIST. A list of all category values that were found in the raster map layer will be printed.

The following sample output from *r.describe:*

```
0 2-4 10-13
```

means that category data values 0, 2 through 4, and 10 through 13 occurred in the named map layer. The user must choose to read the map layer in one of two ways:

(1) DIRECTLY. The current geographic region and mask are ignored and the full raster map layer is read. This method is useful if the user intends to reclassify or rescale the data, since these functions (*r.reclass* and *r.rescale*) also ignore the current geographic region and mask.

(2) REGIONED and MASKED. The map layer is read within the current geographic region, masked by the current mask.

NON-INTERACTIVE PROGRAM USE
r.describe examines a user-chosen raster map layer. If run non-interactively, the layer name must be supplied on the command line.

A compact list of category values that were found in the data layer will be printed.

Following is a sample output:

```
0 2-4 10-13
```

-1 Print the output one value per line, instead of the default short form. In the above example, the *-1* option would output:

```
0
2
3
4
10
11
12
13
```

-r Only print the range of the data. The highest and lowest positive values, and the highest and lowest negative values, are output. In the above example, the *-r* option would output:

```
0 2 13
```

If the *-1* option is also specified, the output appears with one category value per line.

-q Quiet. The *-q* option will tell *r.describe* to be silent while reading the raster file. If not specified, program percentage-completed messages are printed.

-d Use the current geographic region settings. Normally, *r.describe* will read the data layer directly, ignoring both the current region settings and mask. The *-d* option tells *r.describe* to read the map layer in the current region masked by the current mask (if any).

NOTES
The range report will generally run faster than the full list.

SEE ALSO
g.region, r.mask, r.reclass, r.rescale, and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.digit

NAME
r.digit - Interactive tool used to draw and save vector features on a graphics monitor using a pointing device (mouse)

SYNOPSIS
r.digit

DESCRIPTION
The GRASS tool *r.digit* provides the user with a way to draw lines, areas, and circles on a monitor screen, and to save these features in a cell file. Lines, areas, and circles are to be drawn using a pointing device (mouse). A mouse button menu indicates the consequences of pressing each mouse button. The user is requested to enter the category number associated with the line, area, or circle subsequently drawn by the user. Lines, areas, and circles are defined by the series of points marked by the user inside the map window. *r.digit* will close areas when the user has not. By drawing a series of such features, the user can repair maps, identify areas of interest, or simply draw graphics for advertisement. When drawing is completed, a raster map based on the user's instructions is generated. It is available for use as a mask, in analyses, and for display.

Digitizing is done in a "polygon" method. Each area is circumscribed completely. Two or more areas and/or lines might define a single part of a map. Each part of the map, however, is assigned only the LAST area or line which covered it.

THE PROCESS:
Step 1: Choose to define an area or line, quit, or finish. If you quit, the session exits with nothing created. If you choose to finish (done), you will be prompted for a new map name; the new map is then created.

Step 2: If you choose to make an area or line you must identify the category number for that area or line.

Step 3: Using the mouse trace the line or circumscribe the area; or, finish (go to Step 1).

SEE ALSO
v.digit - Highly, interactive tool for digitizing, editing, and labeling vector data
d.display - Tool for displaying and producing maps
d.mapgraph - Draws simple graphics on a map
r.in.poly - Tool for importing "polygon" data to raster format

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.drain

NAME
r.drain - Traces a flow through an elevation model on a raster map layer.
(GRASS Raster Program)

SYNOPSIS
r.drain
r.drain help
r.drain input=name output=name [coordinate=x,y[,x,y,...]]

DESCRIPTION
r.drain traces a flow through a least-cost path in an elevation model. The elevation surface (a raster map layer input) might be the cumulative cost map generated by the *r.cost* program. The output result (also a raster map layer) will show one or more least-cost paths between each user- provided location(s) and the low spot (low category values) in the input model.

The program will be run non-interactively, if the user specifies the names of raster map layers and any desired options on the command line, using the form

r.drain input=name output=name [coordinate=x,y[,x,y,...]]

where the input name is the name of a raster map layer to be used in calculating drainage, the output name is the name of the raster map layer to contain output, and each x,y coordinate pair is the geographic location of a point from which drainage is to be calculated.

Alternately, the user can simply type *r.drain* on the command line, without program arguments. In this case, the user will be prompted for needed parameter values using the standard GRASS parser interface described in the manual entry for parser.

Parameters:

input=name Name of raster map layer containing cell cost information.

output=name Name of raster map layer to contain program output.

coordinate=x,y[,x,y,...] Each x,y pair is the easting and northing (respectively) of a starting point from which a least-cost corridor will be developed. As many points as desired can be input. (But, see BUGS below.)

EXAMPLE
Consider the following example:

```
Input:                                Output:
   ELEVATION SURFACE                    LEAST COST PATH
 . . .---  . . . . . . . . . .        . . . . . . . . . . . . . .
 . 20| 19| 17. 16. 17. 16. 16.        .   . 1 . 1 . 1 .   .   .   .
 . . |___| . . . . . . . . . .        . . . . . . . . . . . . . .
 . 18. 18. 24. 18. 15. 12. 11.        .   .   .   .   . 1 .   .   .
 . . . . . . . . . . . . . . .        . . . . . . . . . . . . . .
 . 22. 16. 16. 18. 10. 10. 10.        .   .   .   .   . 1 .   .   .
 . . . . . . . . . . . . . . .        . . . . . . . . . . . . . .
 . 17. 15. 15. 15. 10. 8 . 8 .        .   .   .   .   .   . 1 .   .
 . . . . . . . . . . . . . . .        . . . . . . . . . . . . . .
 . 24. 16. 8 . 7 . 8 . 0 .12 .        .   .   .   .   .   . 1 .   .
 . . . . . . . . . . . . . . .        . . . . . . . . . . . . . .
 . 17. 9 . 8 . 7 . 8 . 6 .12 .        .   .   .   .   .   .   .   .
 . . . . . . . . . . . . . . .        . . . . . . . . . . . . . .
```

The user-provided starting location in the above example is the boxed 19 in the left-hand map. The path in the output shows the least-cost corridor for moving from the starting box to the lowest (smallest) possible point. This is the path a raindrop would take in this landscape.

BUGS
Currently, *r.drain* will not actually provide output for more than one pair of input coordinates stated on the command line.

r.drain also currently finds only the lowest point (the cell having the smallest category value) in the input file that can be reached through directly adjacent cells that are less than or equal in value to the cell reached immediately prior to it; therefore, it will not necessarily reach the lowest point in the input file. It currently finds pits in the data, rather than the lowest point present.

Only one least-cost path is currently printed to the output file for the user.

Sometimes, when the differences among cell category values in the *r.cost* cumulative cost surface output are small, this cumulative cost output cannot accurately be used as input to *r.drain* (*r.drain* will output bad results). This problem can be circumvented by making the differences between cell category values in the cumulative cost output bigger. It is recommended that, if the output from *r.cost* is to be used as input to *r.drain*, the user multiply the input cost surface map to *r.cost* by the value of the map resolution, before running *r.cost*. This can be done using *r.mapcalc* or other programs. The map resolution can be found using *g.region*.

SEE ALSO
g.region, r.cost, r.mapcalc, and parser

AUTHOR
Kewan Q. Khawaja, Intelligent Engineering Systems Laboratory, M.I.T.

r.grow

NAME
r.grow - Generates an output raster map layer with contiguous areas grown by one cell (pixel).
(GRASS Raster Program)

SYNOPSIS
r.grow
r.grow help
r.grow [-bq] input=name output=name

DESCRIPTION
r.grow adds one cell around the perimeters of all areas in a user-specified raster map layer and stores the output in a new raster map layer.

An area consists of any contiguous clump of cells with non-zero category values. No distinction is made between differing category values within an area. Rather, a border is grown around the outside of each entire contiguous set of non-zero cells.

The output raster map layer will not go outside the boundaries set in the current geographic region. Thus, if a contiguous area in the input raster map layer extends to the geographic edge of the current map layer, no new border cells can be added to that side of the area.

Growth around a rectangular area in the input raster map layer will occur straight out from each edge, but not diagonally from the corners of the rectangle. Thus, the "grown" border area will contain lines along the edge of the original rectangle, but the corners of the border will not be squared off. Instead, the lines of the border which go along each side of the original rectangle will touch only at the corners of the cells at the end of each line.

OPTIONS
The user can run *r.grow* either interactively or non- interactively. The program is run interactively if the user types *r.grow* without specifying flag settings and parameter values on the command line. In this case, the user will be prompted for input.

Alternately, the user can run *r.grow* non-interactively, by specifying the names of an input and output map layer, and including any desired flags, on the command line.

Flags:

-b Output a binary raster map layer having only zero-one category values, regardless of the category values in the input map layer. In this case, all cells with a non-zero category value in the input map layer are assigned to category 1 in the output map layer. If the *-b* flag is not used, these cells will retain their original non-zero category values. In either case, all cells whose category value is changed from 0 during the growing process are assigned a category value of 1 in the output map.

-q Run quietly, supressing printing of information about program progress to standard output.

Parameters:

input=name Name of an existing raster map layer in the user's current mapset search path containing areas to be "grown".

output=name Name of the new raster map layer to contain program output. This map will be binary if the user sets the *-b* flag. Otherwise, input map cells having non-zero category values will retain their original values. In either case, all cells whose values changed during growth will be assigned category value 1 in the output map.

NOTES
The *r.grow* command can be used to represent the boundary of one or more areas. In this case, the zero-one (binary) output option should NOT be used. Then the input map layer can be subtracted from the output map layer using the *r.mapcalc* command. All original non-zero category values will be subtracted out, leaving the boundary areas only. This resulting zero-one boundary depiction can be displayed over other related raster map layers using the overlay option of *d.rast*.

If the resolution of the current geographic region does not agree with the resolution of the input raster map layer, unintended resampling of the original raster map layer may occur. The user should be sure that the current geographic region is set properly.

SEE ALSO
d.rast, g.region, r.mapcalc, and r.poly

AUTHOR
Marjorie Larson, U.S. Army Construction Engineering Research Laboratory

r.in.ascii

NAME
r.in.ascii - Convert an ASCII raster text file into a (binary) raster map layer.
(GRASS Raster Data Import Program)

SYNOPSIS
r.in.ascii
r.in.ascii help
r.in.ascii input=name output=name [title="phrase"] [mult=multiplier]

DESCRIPTION
r.in.ascii allows a user to create a (binary) GRASS raster map layer from an ASCII raster input file with (optional) title.

Parameters:

input=name Name of an existing ASCII raster file to be imported.

output=name Name to be assigned to resultant binary raster map layer.

title="phrase" Title to be assigned to resultant raster map layer.

mult=multiplier Multiply all raster cell values by multiplier. *multiplier* is a floating point value, and has a default value of 1.0.

The input file has a header section which describes the location and size of the data, followed by the data itself. The header has 6 lines:

```
north:   xxxxxx.xx
south:   xxxxxx.xx
east:    xxxxxx.xx
west:    xxxxxx.xx
rows:    r
cols:    c
```

The north, south, east, and west field values entered are the coordinates of the edges of the geographic region. The rows and cols field values entered describe the dimensions of the matrix of data to follow. The data which follows is r rows of c integers.

EXAMPLE
The following is a sample input file to *r.in.ascii:*

```
 _____________________________________
| north:          4299000.00         |
| south:          4247000.00         |
| east:           528000.00          |
| west:           500000.00          |
| rows:                  10          |
| cols:                  15          |
|                                    |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15|
|____________________________________|
```

NOTES
The geographic coordinates north, south, east, and west describe the outer edges of the geographic region. They run along the edges of the cells at the edge of the geographic region and NOT through the center of the cells at the edges.

The data (which follows the header section) must contain r x c values, but it is not necessary that all the data for a row be on one line. A row may be split over many lines.

r.in.ascii handles floating point cell values, but truncates them into integer values. The mult option allows the number of significant figures of a floating point cell to be increased before truncation to an integer. Multiples of ten are the most functional multipliers.

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.in.erdas

NAME
r.in.erdas - Creates raster files from ERDAS files. It creates one raster file for each band, and creates color support if an ERDAS trailer file is specified.
(SCS GRASS Raster Program)

SYNOPSIS
r.in.erdas
r.in.erdas help
r.in.erdas erdas=name [trl=name] prefix=name

DESCRIPTION
This command prompts the user twice:

First the user is asked if he wishes to select a subset of the bands available in the ERDAS file for output. If unspecified by the user, all bands are used, by default.

Second, the user is asked if he wishes to select a subregion of the image available in the ERDAS file. If unspecified by the user, the complete image is used, by default.

Note:
GRASS raster files will be named prefix.band Remember that it is necessary to run: *r.support*: to create the histogram or change the color table and *i.group*: to associate the individual raster files as an image group.

OPTIONS
Parameters:

erdas=name Input ERDAS file name.

trl=name Input ERDAS trailer file name.

prefix=name Prefix of the GRASS raster files to be created.

AUTHOR
M.L. Holko, USDA, SCS, NHQ-CGIS

r.in.ll

NAME
r.in.ll - Converts raster data referenced using latitude and longitude coordinates to a UTM-referenced map layer in GRASS raster format.
(GRASS Raster Data Import Program)

SYNOPSIS
r.in.ll
r.in.ll help
r.in.ll[-s]input=name output=name bpc=value corner=corner,lat,lon dimension=rows,cols res=latres,lonres spheroid=name

DESCRIPTION
This program converts raster data referenced using latitude and longitude coordinates to a UTM-referenced map layer in GRASS raster format. *r.in.ll* is primarily used as the final program in converting DTED and DEM digital elevation data to GRASS raster format, but is not limited to this use. *r.in.ll* uses the user's current geographic region settings. Only data that falls within the current geographic region will appear in the final raster map layer.

r.in.ll requires the user to enter the following information:

OPTIONS
Flags:

-s Signed data (high bit means negative value).

Parameters:

input=name Name of an existing input raster map layer.

output=name Name to be assigned to the output raster map layer.

bpc=value Number of bytes per cell.

corner=corner,lat,lon One corner latitude and longitude of the input.
Format: {nw|ne|sw|se},dd:mm:ss{N|S},ddd:mm:ss{E|W}

The latitude and longitude are specified as dd.mm.ssH where dd is degrees, mm is minutes, ss is seconds, and H is the hemisphere (N or S for latitudes, E or W for longitudes).

For example, to specify the southwest corner:

corner=sw,46N,120W

Note: the latitude and longitude specified are for the center of the corner cell.

dimension=rows,cols Number of rows and columns in the input file.

res=latres,lonres Resolution of the input (in arc seconds).

spheroid=name Name of spheroid to be used for coordinate conversion.
Options:airy, australian, bessel, clark66, everest, grs80, hayford, international, krasovsky, wgs66, wgs72, wgs84

EXAMPLE
The command line:

r.in.ll input=rot.out output=import.out dimension=358,301 bpc=2 res=3,3 corner=sw,37:13N,103:45W spheroid=wgs72

reads data from the file rot.out, converts the data, and stores them in the file import.out. The data to be converted are made up of 358 rows and 301 columns, and have a resolution of 3x3 arc seconds.

NOTES

In the conversion of DTED and DEM elevation data to raster map layer format, *r.in.ll* follows execution of the data rotation program *m.rot90*. Because the user can glean information on the number of rows and columns, the resolutions of the latitude and longitude, and the number of bytes per column from the header file produced by the tape extraction programs *m.dted.extract* and *m.dmaUSGSread*, the user should recall that *m.rot90* has rotated the files produced by the tape extraction programs 90 degrees; this means that the user should INTERCHANGE the numbers of rows and columns present in the header file for input to *r.in.ll*. The number of rows shown in the tape extract header file now become the number of columns in the *m.rot90* output file; the number of columns shown in the tape extract header file are now the number of rows present in the *m.rot90* output file.

The user should also note that the raster map layer imported into GRASS will be based on the current geographic region settings. The boundaries of this geographic region should therefore be checked before importing the raster map layer. Data outside of the geographic region will not be imported and missing data will be assigned the category value “no data”.

SEE ALSO

m.dmaUSGSread, m.dted.examine, m.dted.extract, m.rot90

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.in.miads

NAME
r.in.miads - Converts a MIADS output ASCII text file into an GRASS raster import (*r.in.ascii*) formatted file.
(SCS GRASS Raster Program)

SYNOPSIS
r.in.miads
r.in.miads help
r.in.miads input=name output=name strip=value line=value cell=value Northing=value Easting=value size=value

DESCRIPTION
r.in.miads allows a user to create a r.in.ascii formatted ASCII file from a MIADS output printer format ASCII file. The program will actively read the MIADS data file, selectively remove and process each strip, creating a individual *r.in.ascii* formatted file for each strip, and finally create one category file for all strips. The program also produces a report file summarizing all pertinent information for each strip.

The resulting *r.in.ascii* files for each strip are then used in conjunction with the GRASS commands *r.in.ascii* and *r.patch* to create a complete raster file.

OPTIONS
Parameters:

input=name MIADS input file name.

output=name GRASS raster data output file name.

strip=value MIADSs strip number of reference cell.

line=value MIADSs line number of reference cell.

cell=value MIADS cell number of reference cell.

Northing=value UTM Easting at the cell reference.

Easting=value UTM Northing at the cell reference.

size=value Cell size (length one side) in meters.

SCS has developed scripts *run.miads*, *getstrip*, and *strip.99s*. These scripts make the MIADS to GRASS conversion easier.

run.miads - SCS macro to perform the complete conversion of a MIADS printer format data set to a GRASS raster file.

getstrip - Reads each MIADS strip file and converts it to a independent GRASS data file. Support may be run on any one of these strip files; however, there is no category information available. Each strip may be viewed at this time with d.rast or d.display.

strip.99s - Special pre-*r.in.miads* macro that removes the 99's from the MIADS data file. This effectively removes border information, replacing it with "no data" values.

SEE ALSO
d.display, d.rast, r.in.ascii, r.patch, r.support

AUTHOR
r.in.miads - R.L.Glenn, USDA, SCS, NHQ-CGIS
run.miads, strip.99s - Harold Kane, USDA, SCS, Oklahoma State Office

r.in.poly

NAME
r.in.poly - Create raster maps from ascii polygon/line data file
(GRASS Raster Program)

SYNOPSIS
r.in.poly
r.in.poly help
r.in.poly input=name output=name [title="phrase"] [rows=value]

DESCRIPTION
r.in.poly allows the creation of GRASS ASCII files containing polygon and linear features.

OPTIONS
Parameters

input=name Unix input file, in ascii format, containing the polygon and linear features. The format of this file is described in the section INPUT FORMAT below.

output=name Raster output file

title="phrase" Title for resultant raster map (optional)

rows Number of rows to hold in memory (default: 512). This parameter allows users with less memory (or more) on their system to control how much memory *r.in.poly* uses.

INPUT FORMAT
The input format for the input file consists of sections describing either polygonal areas or linear features.
The basic format is:

```
A  <for polygonal areas>
    easting northing
    .
=   cat# label
L  <for linear features>
    easting northing
    .
=   cat# label
```

The A signals the beginning of a polygon. It must appear in the first column. The L signal the beginning of a linear feature. It also must appear in the first column. The coordinates of the vertices of the polygon, or the coordinates defining the linear feature follow and must have a space in the first column and at least one space between the easting and the northing. To give meaning to the features, the = indicates that the feature currently being processed has category value cat# (which must be an integer) and a label (which may be more than one word, or which may be omitted).

SEE ALSO
r.digit - interactive on-screen polygon/line digitizing for raster maps
r.colors - creates color tables for raster maps

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.in.sunrast

NAME
r.in.sunrast - Converts a SUN raster file to a GRASS raster file.
(GRASS Raster Data Import Program)

SYNOPSIS
r.in.sunrast

DESCRIPTION
This program converts a SUN raster file that has been created by SUN's "screendump" utility to a GRASS raster file. Output is placed in the /cell directory under the user's current GRASS mapset.

The program prompts the user to enter the name of the SUN raster file to be converted and the name to be assigned to the GRASS raster file to contain the resultant image.

It is recommended that this program be used in an x,y database (as opposed to, for example, a UTM data base), since the cell header is created with nonsense coordinates (i.e., coordinates designed only to specify the number of rows and columns in the image). Of course, the user can adjust the cell header after import using *r.support*.

The user must, of course, first create the SUN raster file to be converted, either by running the SUN "screendump" utility (to capture a displayed image) or by some other means (e.g., from a scanning system that produces SUN raster file format).

NOTE
If you are using the screendump utility on a SUN workstation to create the sun rasterfile, do not use the *-e* option. This option creates a sun rasterfile format that *r.in.sunrast* does not understand.

SEE ALSO
SUN screendump utility
r.support and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.infer

NAME

r.infer - Outputs a raster map layer whose category values represent the application of user-specified criteria (rules statements) to other raster map layers' category values. (GRASS Raster Program)

SYNOPSIS

r.infer
r.infer help
r.infer [-vt] rulesfile=name

DESCRIPTION

r.infer is an inference engine which applies a set of user- specified rules to named raster map layers. A new raster map layer named infer is created as output, whose category values reflect the ability of each cell in the named input layers to satisfy the named conditions. *r.infer* commands (conditions and consequences) are typed into a file by the user using a system editor like *vi*, and then input to *r.infer* as the rulesfile named on the command line. The results are used to generate a new raster map layer named infer in the user's current mapset. This program performs analyses similar to *r.combine*, but uses a (possibly) more pleasing syntax and approach. The program will be run non-interactively if the user specifies the name of a rules file and any desired flags on the command line, using the form:

r.infer [-vt] rulesfile=name

where name is the name of an ASCII file containing valid input rules to *r.infer*, and the (optional) flags *-v* and *-t* have the meanings described in the OPTIONS section, below. Alternately, the user can simply type *r.infer* on the command line, without program arguments. In this case, the user will be prompted for the needed parameter value and flag settings using the standard GRASS *parser* interface described in the manual entry for *parser*.

Flags:

-t Allows the user to run *r.infer* in test mode. The user is questioned about the truth of each condition named in the file. r.infer then outputs the value that would be placed in the new layer infer for a cell meeting conditions specified by the user. When no sets of conditions stated in the input file are satisfied (based upon the user's answers), cell values of zero are output. Test mode is used to test the accuracy of the user's logic. Users are encouraged to run r.infer in test mode prior to actually creating map layers.

-v Makes *r.infer* run verbosely, giving information about each cell as it is analyzed according to the statement conditions.

Parameter:

rulesfile=name Allows the user to input rules to *r.infer* from an ASCII file, rather than from standard input. This rulesfile must exist in the user's current working directory or be given by its full pathname. File rules statements take the same form as those given on the command line. Examples of valid rules statements are given in the sections below.

COMMANDS AND STATEMENTS

The following commands are available in *r.infer*:

Command	Aliases	Followed By	Such As
IFMAP	ANDIFMAP ANDMAP	cellmap cat#	geology 2
IFNOTMAP	ANDNOTMAP	cellmap cat#	geology 2
THENMAPHYP		cat# [statement]	3 nice vacation spot
THEN		statement condition	No sandstone
IF	AND ANDIF	predefined statement condition	No sandstone

These five commands may be used to formulate statements with functions ranging from a simple reclassification to a more complex expert system type application. Statements are composed of one or more conditions followed by one or more hypotheses and/or conclusions. The use of aliases is provided to allow for the use of a command which has an English meaning consistent with the logic at that point.

Following is a description of each of the five commands. The map layers used in the examples are in the Spearfish sample data base.

IFMAP
Map condition.
Map conditions are questions to each cell about the presence of specified map layer category values. *r.infer* questions each cell in the named map layer (here, geology) about its contents (i.e., category value). Cells which satisfy the named condition(s) stated by IFMAP (i.e., here, those cells which contain geology map layer category values 4 or 5) will be assigned the subsequently-stated map conclusion or hypothesis (category), in the new map layer infer. Cells which fail to satisfy named map condition(s) will continue to move down through the user's rulesfile (searching for conditions it is able to satisfy) if any additional conclusions/hypotheses are stated in the file, or will be assigned category zero in the new map layer infer (if no additional conclusions/hypotheses are possible in this rulesfile).

example: *IFMAP geology 4 5*

IFNOTMAP
Map condition.
Like IFMAP, but instead questions each cell about the absence of specified map layer categories. Cells which meet the IFNOTMAP conditions (i.e., below, those cells which do NOT include owner map layer category value 2) will be assigned the named conclusionothesis, in the new map layer infer.

example: *IFNOTMAP owner 2*

THENMAPHYP
Map conclusion.
Assigns each cell a specified category value in the new map layer infer based on the cell's ability or failure to meet conditions named above this THENMAPHYP statement in the rulesfile. The user should note that although the user can specify a uniquely-named rulesfile, *r.infer* always directs its output to a file named infer in the current mapset (overwriting whatever is currently in this file). Therefore, if the user wishes to save this file for future use, this file should be renamed before the user next runs *r.infer* (e.g., using the GRASS command g.rename).

It is important to realize that r.infer runs through the conditions stated in the named rulesfile one cell at a time, moving from the top of the raster input file to the bottom of the raster input file. As soon as the cell currently being examined by r.infer satisfies a set of conditions, it is assigned a category value in the new map layer infer. *r.infer* does NOT check to see if that same cell satisfies other conditions named further down in the input file, too. Instead, it moves on to the next cell, and begins anew with the conditions named at the top of the input file. Essentially, this means that conclusions made higher-up in the input file have precedence over conditions named further down in the input file.

example: *IFMAP density 1 THENMAPHYP 1 no trees*

In the above example, all cells having a category value of 1 (non-forest) in the map layer density, are assigned a category value of 1 in the resultant map layer infer. The trailing text "no trees" is entered into the category support file for category 1 in the new map layer infer.

THEN
Statement hypothesis.
At the conclusion of one or several condition statements, instead of making a map conclusion as with THENMAPHYP, the conditions are used to create a hypothesis. This may then be referenced in later statements using the IF command. The trailing text at the end of the THEN statement is used as the means with which to reference the hypothesis. An example follows the description of IF below.

IF
Statement condition.
States a condition based on an hypothesis that was created by a previous THEN statement. IF may be used only after a THEN has set up the group of statements that are to be referenced later.

example:

IFMAP elevation.255 170-255 ANDIFMAP density 3 4 THEN high elevation with trees!
IF high elevation with trees ANDIFMAP owner 2 THENMAPHYP 1 this is the place

The above example queries each cell for the presence of both elevations greater than 1580 meters (i.e., for elevation.255 category values 170-255) and a medium to high density of trees (i.e., density category values 3 4). All areas (i.e., cells) that satisfy these criteria are assigned to the hypothesis "high elevation with trees." The "!" simply tells *r.infer* to ignore whatever appears on that line (a comment statement), and is used here for readability.

The IF statement then references cells having "high elevation with trees" (i.e., those cells that satisfied both of the above conditions named by the IFMAP and ANDIFMAP statements). If a cell both has "high elevations with trees" and owner map layer category 2 (areas owned by the Forest Service), it is assigned by the THENMAPHYP statement to category 1 in the new map layer infer. The trailing text "this is the place" is automatically entered into the category support file for the new map infer. Cells failing to meet all of the conditions stated in this input file will be assigned category 0 in the new map layer infer.

SEE ALSO
GRASS Tutorial: *r.infer*
g.rename, r.combine,< r.mapcalcr.weight, and parser

AUTHOR
James Westervelt, U.S. Army Construction Engineering Research Laboratory
George W. Hageman, SOFTMAN Enterprises P.O. Box 11234 Boulder, Colorado 80301
Daniel S. Cox, In Touch 796 West Peachtree St. NE Atlanta, GA 30308

r.info

NAME
r.info - Outputs basic information about a user-specified raster map layer.
(GRASS Raster Program)

SYNOPSIS
r.info
r.info help
r.info map=name

DESCRIPTION
r.info reports some basic information about a user-specified raster map layer. This map layer must exist in the user's current mapset search path. Information about the map's boundaries, resolution, projection, data type, category number, data base location and mapset, and history are put into a table and written to standard output. The types of information listed can also be found in the /cats, /cellhd, and /hist directories under the mapset in which the named map is stored.

The program will be run non-interactively if the user specifies the name of a raster map layer on the command line, using the form *r.info map=name* where name is the name of a raster map layer on which the user seeks information. The user can save the tabular output to a file by using the UNIX redirection mechanism (<); for example, the user might save a report on the soils map layer in a file called soil.rpt by typing:

r.info map=soils > soil.rpt

Alternately, the user can simply type *r.info* on the command line, without program arguments. In this case, the user will be prompted for the name of a raster map layer using the standard GRASS parser interface described in the manual entry for parser. The user is asked whether he wishes to print the report and/or save it in a file. If saved, the report is stored in a user-named file in the user's home directory.

Below is the report produced by *r.info* for the raster map geology in the Spearfish sample data base.

```
|-----------------------------------------------------------|
| Layer:    geology   Date: Mon May  4 10:00:14 1987 :      |
| Location: spearfish Login of Creator: grass               |
| Mapset:   PERMANENT                                       |
| Title:    Geology                                         |
|-----------------------------------------------------------|
|   Type of Map:  rasterNumber of Categories: 9             |
|   Rows:    140                                            |
|   Columns: 190                                            |
|   Total Cells:  26600                                     |
|   Projection: UTM (zone 13)                               |
|       N: 4928000.00    S: 4914000.00   Res: 100.00       |
|       E:  609000.00    W:  590000.00   Res: 100.00       |
|                                                           |
|   Data Source:                                            |
|   Raster file produced by EROS Data Center                |
|                                                           |
|   Data Description:                                       |
|   Shows the geology for the map area                      |
|                                                           |
|   Comments:                                               |
|-----------------------------------------------------------|
```

SEE ALSO
g.mapsets, r.coin, r.describe, r.report, r.stats, r.support, r.what, and parser

AUTHOR
Michael O'Shea, U.S. Army Construction Engineering Research Laboratory

r.line

NAME
r.line - Creates a new binary GRASS vector (*v.digit*) file by extracting linear features from a thinned raster file. (GRASS Raster Program)

SYNOPSIS
r.line
r.line help
r.line input=name output=name [type=name]

DESCRIPTION
r.line scans the named raster map layer (*input=name*) and extracts thinned linear features into the named vector file (*output=name*).

OPTIONS
The user can run this program either non-interactively or interactively. The program will be run non-interactively if the user specifies program arguments on the command line, using the form:

r.line input=name output=name [type=name]

If the user specifies input raster and output vector map names on the command line, any other parameter values left unspecified on the command line will be set to their default values (see below). Alternately, the user can simply type *r.line* on the command line, without program arguments. In this case, the user will be prompted for parameter values using the standard GRASS *parser* interface described in the manual entry for parser.

Parameters:
input=name Name of existing raster file to be used as input.

output=name Name of new vector file to be output.

type=name Line type of the extracted vectors; either line or area. Specifying line will type extracted vectors as linear edges. Specifying area will type extracted vectors as area edges.
Options: line or area. Default: type=line

r.line assumes that the input map has been thinned using *r.thin*.

NOTES
r.line extracts vectors (aka, "arcs") from a raster file. These arcs may represent linear features (like roads or streams), or may represent area edge features (like political boundaries, or soil mapping units). The attribute type option allows the user to establish the use of either linear or area edge attributes for all of the extracted vectors. *r.poly* may be used to extract vectors that represent area features (like soil mapping units, elevation ranges, etc.) from a raster file.

The user must run *v.support* on the resultant vector (*v.digit*) files to build the dig_plus information.

r.thin and *r.line* may create excessive nodes at every junction, and may create small spurs or "dangling lines" during the thinning and vectorization process. These excessive nodes and spurs may be removed using *v.trim*.

BUGS
The input raster file MUST be thinned by *r.thin*; if not, *r.line* may crash.

SEE ALSO
r.poly, r.thin, v.digit, v.support, v.trim, and parser

AUTHOR
Mike Baba, DBA Systems, Inc. 10560 Arrowhead Drive Fairfax, Virginia 22030

r.los

NAME
r.los - Line-of-sight raster analysis program.
(GRASS Raster Program)

SYNOPSIS
r.los
r.los help
r.los input=name output=name coordinate=x,y [patt_map=name] [obs_elev=value] [max_dist=value]

DESCRIPTION
r.los generates a raster map output in which the cells that are visible from a user-specified observer location are marked with integer values that represent the vertical angle (in degrees) required to see those cells.

The program can be run either non-interactively or interactively. To run *r.los* non-interactively, the user must specify at least an input file name, output file name, and the geographic coordinates of the user's viewing location on the command line; any remaining parameters whose values are unspecified on the command line will be set to their default values (see below). Non-interactive usage format is:

r.los input=name output=name coordinate=x,y [patt_map=name] [obs_elev=value] [max_dist=value]

Alternately, the user can type simply *r.los* on the command line; in this case, the program will prompt the user for parameter values using the standard GRASS interface described in the manual entry for parser.

Parameters:

input=name Name of a raster map layer containing elevation data, used as program input.

output=name Name assigned to the file in which the raster program output will be stored.

coordinate=x,y Geographic coordinates (i.e., easting and northing values) identifying the desired location of the viewing point.

patt_map=name Name of a binary (1/0) raster map layer in which cells within the areas of interest are assigned the category value '1', and all other cells are assigned the category value '0'. If this parameter is omitted, the analysis will be performed for theole area within a certain distance of the viewing point inside the geographic region boundaries.
Default: assign all cells within the max_dist and the user's current geographic region boundaries a value of 1.

obs_elev=value Height of the observer (in meters) above the viewing point's elevation.
Default: 1.75 (meters)

max_dist=value Maximum distance (in meters) from the viewing point inside of which the line of sight analysis will be performed. The cells outside this distance range are assigned the category value '0'.
Options: 0-99999 (stated in map units) Default: 100

NOTES
For accurate results, the program must be run with the resolution of the geographic region set equal to the resolution of the data (see g.region). It is advisable to use a 'pattern layer' which identifies the areas of interest in which the line of sight analysis is required. Such a measure will reduce the time taken by the program to run.

SEE ALSO
g.region, r.pat.place, and parser

AUTHOR
Kewan Q. Khawaja, Intelligent Engineering Systems Laboratory, M.I.T.

r.mapcalc

NAME
r.mapcalc - Raster map layer data calculator.
(GRASS Raster Program)

SYNOPSIS
r.mapcalc
r.mapcalc [result=expression]

DESCRIPTION
r.mapcalc performs arithmetic on raster map layers. New raster map layers can be created which are arithmetic expressions involving existing raster map layers, integer or floating point constants, and functions.

PROGRAM USE
If used without command line arguments, *r.mapcalc* will read its input, one line at a time, from standard input (which is the keyboard, unless redirected from a file or across a pipe). Otherwise, the expression on the command line is evaluated. *r.mapcalc* expects its input to have the form:

result=expression

where result is the name of a raster map layer to contain the result of the calculation and expression is any legal arithmetic expression involving existing raster map layers, integer or floating point constants, and functions known to the calculator. Parentheses are allowed in the expression and may be nested to any depth. result will be created in the user's current mapset.

The formula entered to *r.mapcalc* by the user is recorded both in the result map title (which appears in the category file for result) and in the history file for result.

Some characters have special meaning to the command shell. If the user is entering input to *r.mapcalc* on the command line, expressions should be enclosed within single quotes. See NOTES, below.

OPERATORS AND ORDER OF PRECEDENCE
The following operators are supported:

Operator	Meaning	Type	Precedence
%	modulus (remainder upon division)	Arithmetic	4
/	division	Arithmetic	4
*	multiplication	Arithmetic	4
+	addition	Arithmetic	3
-	subtraction	Arithmetic	3
==	equal	Logical	2
!=	not equal	Logical	2
>	greater than	Logical	2
>=	greater than or equal	Logical	2
<	less than	Logical	2
<=	less than or equal	Logical	2
&&	and	Logical	1
||	or	Logical	1

The operators are applied from left to right, with those of higher precedence applied before those with lower precedence. Division by 0 and modulus by 0 are acceptable and give a 0 result. The logical operators give a 1 result if the comparison is true, 0 otherwise.

RASTER MAP LAYER NAMES
Anything in the expression which is not a number, operator, or function name is taken to be a raster map layer name. Examples:

elevation x3 3d.his

Most GRASS raster map layers meet this naming convention. However, if a raster map layer has a name which conflicts with the above rule, it should be quoted. For example, the expression

x = a-b

would be interpreted as: x equals a minus b, whereas

x = "a-b"

would be interpreted as: x equals the raster map layer named a-b.

Also

x = 3107

would create x filled with the number 3107, while

x = "3107"

would copy the raster map layer 3107 to the raster map layer x. Quotes are not required unless the raster map layer names look like numbers or contain operators, OR unless the program is run non-interactively. Examples given here assume the program is run interactively. See NOTES, below.

r.mapcalc will look for the raster map layers according to the user's current mapset search path. It is possible to override the search path and specify the mapset from which to select the raster map layer. This is done by specifying the raster map layer name in the form:

name@mapset

For example, the following is a legal expression:

result = x@PERMANENT / y@SOILS

The mapset specified does not have to be in the mapset search path. (This method of overriding the mapset search path is common to all GRASS commands, not just *r.mapcalc*.)

THE NEIGHBORHOOD MODIFIER

Maps and images are data base files stored in raster format, i.e., two-dimensional matrices of integer values. In *r.mapcalc*, maps may be followed by a neighborhood modifier that 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.

RASTER MAP LAYER VALUES FROM THE CATEGORY FILE

Sometimes it is desirable to use a value associated with a category's contents instead of the category value itself. If a raster map layer name is preceded by the @ operator, then the labels in the category file for the raster map layer are used in the expression instead of the category value.

For example, suppose that the raster map layer soil.ph (representing soil pH values) has a category file with labels as follows:

```
cat     label
_____________
0       no data
1       1.4
2       2.4
3       3.5
4       5.8
5       7.2
6       8.8
7       9.4
```

Then the expression:

*result = @soils.ph * 10*

would produce a result with category values 0, 14, 24, 35, 58, 72, 88 and 94.

Note that this operator may only be applied to raster map layers and produces a floating point value in the expression. Also the category label must start with a valid number. Missing labels, or labels that do not start with a number will (silently) produce a 0 value for that category.

GREY SCALE EQUIVALENTS AND COLOR SEPARATES

It is often helpful to manipulate the colors assigned to map categories. This is particularly useful when the spectral properties of cells have meaning (as with imagery data), or when the map category values represent real quantities (as when category values reflect true elevation values). Map color manipulation can also aid visual recognition, and map printing.

The # operator can be used to either convert map category values to their grey scale equivalents or to extract the red, green, or blue components of a raster map layer into separate raster map layers.

result = #map

converts each category value in map to a value in the range 0-255 which represents the grey scale level implied by the color for the category. If the map has a grey scale color table, then the grey level is what #map evaluates to. Otherwise, it is computed as:

*.18 * red + .81 * green + .01 * blue*

The # operator has three other forms: r#map, g#map, b#map. These extract the red, green, or blue components in the named raster map, respectively. The GRASS shell script blend.sh extracts each of these components from two raster map layers, and combines them by a user-specified percentage. These forms allow color separates to be made. For example, to extract the red component from map and store it in the new 0-255 map layer red, the user could type:

red = r#map

To assign this map grey colors type:

r.colors map=red color=rules
black
white

To assign this map red colors type:

r.colors map=red color=rules
black
red

FUNCTIONS

The functions currently supported are listed in the table below. The type of the result is indicated in the last column. F means that the functions always results in a floating point value, I means that the function gives an integer result, and * indicates that the result is float if any of the arguments to the function are floating point values and integer if all arguments are integer.

function	description	type
abs(x)return	absolute value of x	*
atan(x)	inverse tangent of x (result is in degrees)	F
cos(x)	cosine of x (x is in degrees)	F
eval([x,y,...,]z)	evaluate values of listed expr, pass results to z	*
exp(x)	exponential function of x	F

```
exp(x,y)              x to the power y                                  F
float(x)              convert x to floating point                       F
if                    decision options:                                 *
   if(x)                 1 if x not zero, 0 otherwise
   if(x,a)               a if x not zero, 0 otherwise
   if(x,a,b)             a if x not zero, b otherwise
   if(x,a,b,c)           a if x > 0, b if x is zero, c if x < 0
int(x)                convert x to integer [ truncates ]                I
log(x)                natural log of x                                  F
log(x,b)              log of x base b                                   F
max(x,y[,z...])       largest value of those listed                     *
median(x,y[,z...])    median value of those listed                      *
min(x,y[,z...])       smallest value of those listed                    *
rand(x,y)             random value between x and y                      *
round(x)              round x to nearest integer                        I
sin(x)                sine of x (x is in degrees)                       F
sqrt(x)               square root of x                                  F
tan(x)                tangent of x (x is in degrees)                    F
```

FLOATING POINT VALUES IN THE EXPRESSION

Floating point numbers are allowed in the expression. A floating point number is a number which contains a decimal point:

2.3 12. .81

Floating point values in the expression are handled in a special way. With arithmetic and logical operators, if either operand is float, the other is converted to float and the result of the operation is float. This means, in particular that division of integers results in a (truncated) integer, while division of floats results in an accurate floating point value. With functions of type * (see table above), the result is float if any argument is float, integer otherwise.

However, GRASS raster map layers can only store integer values. If the final value of the expression is a floating point value, this value is rounded to the nearest integer before storing it in the result raster map layer.

Note that raster map layers in the expression are considered to be integers.

EXAMPLES

To compute the average of two raster map layers a and b:

ave = (a + b)/2

To form a weighted average:

*ave = (5*a + 3*b)/8.0*

To produce a binary representation of the raster map layer a so that category 0 remains 0 and all other categories become 1:

mask = a/a

This could also be accomplished by:

mask = if(a)

To mask raster map layer b by raster map layer a:

result = if(a,b)

REGION/MASK

The user must be aware of the current geographic region and current mask settings when using r.mapcalc. All raster map layers are read into the current geographic region masked by the current mask. If it is desired to modify an existing raster map layer without involving other raster map layers, the geographic region should be set to agree with the cell header for the raster map layer. For example, suppose it is determined that the elevation raster map layer must have each category value

increased by 10 meters. The following expression is legal and will do the job:

new_elevation = elevation + 10

Since a category value of 0 is used in GRASS for locations which do not exist in the raster map layer, the new raster map layer will contain the category value 10 in the locations that did not exist in the original elevation. Therefore, in this example, it is essential that the boundaries of the geographic region be set to agree with the cell header.

However, if there is a current mask, then the resultant raster map layer is masked when it is written; i.e., 0 category values in the mask force zero values in the output.

NOTES

Extra care must be taken if the expression is given on the command line. Some characters have special meaning to the UNIX shell. These include, among others:

** ( ) > & |*

It is advisable to put single quotes around the expression; e.g.:

*result = 'elevation * 2'*

Without the quotes, the *, which has special meaning to the UNIX shell, would be altered and *r.mapcalc* would see something other than the *.

If the input comes directly from the keyboard and the result raster map layer exists, the user will be asked if it can be overwritten. Otherwise, the result raster map layer will automatically be overwritten if it exists.

Quoting result is not allowed. However, it is never necessary to quote result since it is always taken to be a raster map layer name.

For formulas that the user enters from standard input (rather than from the command line), a line continuation feature now exists. If the user adds \ to the end of an input line, *r.mapcalc* assumes that the formula being entered by the user continues on to the next input line. There is no limit to the possible number of input lines or to the length of a formula.

If the *r.mapcalc* formula entered by the user is very long, the map title will contain only some of it, but most (if not all) of the formula will be placed into the history file for the result map.

When the user enters input to *r.mapcalc* non-interactively on the command line, the program will not warn the user not to overwrite existing map layers. Users should therefore take care to assign program outputs raster file names that do not yet exist in their current mapsets.

BUGS

Continuation lines must end with a and have NO trailing white space (blanks or tabs). If the user does leave white space at the end of continuation lines, the error messages produced by *r.mapcalc* will be meaningless and the equation will not work as the user intended.

Error messages produced by *r.mapcalc* are almost useless. In future, *r.mapcalc* should make some attempt to point the user to the offending section of the equation, e.g.:

*x = a * b ++ c*

ERROR: somewhere in line 1: ... b ++ c ...

Currently, there is no comment mechanism in *r.mapcalc*. Perhaps adding a capability that would cause the entire line to be ignored when the user inserted a # at the start of a line as if it were not present, would do the trick. The function should require the user to type "end" or "exit" instead of simply a blank line. This would make separation of multiple scripts separable by white space.

SEE ALSO
r.mapcalc: an Angebra for GIS and Image Processing,
"GRASS Tutorial: *r.mapcalc*,"
blend.sh, g.region, r.colors, r.combine, r.infer, r.mask, r.weight

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.mask.points

NAME

r.mask.points -Examines and filters lists of points constituting lines to determine if they fall within current region and mask and optionally an additional raster map.
(GRASS Raster Program)

SYNOPSIS

r.mask.points
r.mask.points help
r.mask.points [-r] [mask=name] [input=name] [fs=name]

DESCRIPTION

r.mask.points filters a list of points based on the current region and current mask. The point list consists of lines which have the following format

easting northing [text]
.
.
easting northing [text]

The eastings and northings define points in the coordinate space. Each line is examined to determine if the point falls within the current region, current mask (if any), and optionally an additional raster map that acts as a secondary mask. If the point falls outside the current region or falls in a grid cell that has value zero (either in the current mask, or the specified mask file), then the entire line is suppressed. otherwise it is printed exactly as it is input. There may be arbitrary text following the coordinate pairs and this text is output as well.

OPTIONS

Flags

-r Coordinates are reversed: north east

Normal input has the east first and the north second. This option allows the order of the coordinates to be north first and east next.

Parameters:

mask Raster map used to mask points. This parameter is optional. If not specified, then the points are mask by the default mask (if there is one). If it is specified, then the points are mask by this layer as well as the default mask.

input Unix input containing point list. If not specified it is assumed that the user will either redirect the input from a file:

r.mask.points < file

or pipe the results from some other process (e.g., a DBMS query) into *r.mask.points*

some_process | r.mask.points

fs Input field separator character. If the coordinates are not separated by white space, but by some other character, this option specifies that character. For example, if a colon is used between the east and north, then r.mask.point can be told this by:

r.mask.points fs=:

NOTES
Lines that make it through the filtering are output intact. This means that if the coordinates are reversed they will remain reversed on output. If there is a field separator, it will also be output.

SEE ALSO
r.mask, s.out.ascii, s.in.ascii, d.points

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.mask

NAME
r.mask - Establishes or removes the current working mask.
(GRASS Raster Program)

SYNOPSIS
r.mask

DESCRIPTION
The *r.mask* program allows the user to block out certain areas of a map from analysis, by "hiding" them from sight of other GRASS programs. This is done by establishing a mask. While a mask exists, most GRASS programs will operate only on data falling inside the masked area, and ignore any data falling outside of the mask.

Because the mask is actually only a reclass file called "MASK" that is created when the user identifies a mask using *r.mask*, it can be copied, renamed, removed, and used in analyses, just like other GRASS raster map layers. The user should be aware that a mask remains in place until a user renames it to something other than "MASK", or removes it using *r.mask* or *g.remove*.

r.mask provides the following options:

```
  1  Remove the current mask
  2  Identify a new mask
RETURN  Exit from program
```

The user establishes a new mask by choosing option (2). The user will be asked to name an existing raster map layer from among those available in the current mapset search path. Once done, the user is shown a listing of this map's categories, and is asked to assign a value of "1" or "0" to each map category. Areas assigned category value "1" will become part of the mask's surface, while areas assigned category value "0" will become "no data" areas in the MASK file.

If a category is not assigned category value "1" it will automatically be assigned the category value "0" in the resulting MASK file. Any cells falling in category "0" will fall outside the newly formed mask, and their presence will be ignored by GRASS programs run later on, as long as the MASK file remains in place.

NOTES
The above method for specifying a "mask" may seem counterintuitive. Areas inside the mask are not hidden; areas outside the mask will be ignored until the MASK file is removed. This program actually creates a raster map layer (reclass type) called MASK, which can be manipulated (renamed, removed, copied, etc.) like any other raster map layer. Somewhat similar program functions to those performed by *r.mask* can be done using *r.mapcalc*, *g.region*, and other programs.

This program can only be run interactively.

Note that some programs, like *r.stats*, have options that allow the user to see the effects of the current mask without removing the current mask. See, for example, use of the *-m* option for *r.stats*.

SEE ALSO
g.copy, g.region, g.remove, g.rename, r.combine, r.infer, r.mapcalc, r.reclass, r.stats, r.weight

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.median

NAME
r.median - Finds the median of values in a cover map within areas assigned the same category value in a user-specified base map.
(GRASS Raster Program)

SYNOPSIS
r.median
r.median help
r.median base=name cover=name output=name

DESCRIPTION
r.median calculates the median category of data contained in a cover raster map layer for areas assigned the same category value in the user-specified base raster map layer. These median values are stored in the new output map layer. The output map is actually a reclass of the base map.

If the user simply types *r.median* on the command line, the user is prompted for the parameter values through the standard parser interface (see parser manual entry).

Alternately, the user can supply all needed parameter values on the command line.

Parameters:
base=name An existing raster map layer in the user's current mapset search path. For each group of cells assigned the same category value in the base map, the median of the values assigned these cells in the cover map will be computed.

cover=name An existing raster map layer containing the values to be used to compute the median within each category of the base map.

output=name The name of a new map layer to contain program output (a reclass of the base map). The median values will be stored in the output map.

NOTES
The user should use the results of *r.median* with care. Since this utility assigns a value to each cell which is based on global information (i.e., information at spatial locations other than just the location of the cell itself), the resultant map layer is only valid if the geographic region and mask settings are the same as they were at the time that the result map was created.

Results are affected by the current region settings and mask.

SEE ALSO
g.region, r.average, r.cats, r.clump, r.describe, r.mapcalc, r.mask, r.mfilter, r.mode, r.neighbors, r.reclass, r.stats, and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.mfilter

NAME
r.mfilter - Raster file matrix filter.
(GRASS Raster Program)

SYNOPSIS
r.mfilter
r.mfilter help
r.mfilter [-qpz] input=name output=name filter=name [repeat=value] [title="phrase"]

DESCRIPTION
r.mfilter filters the raster input to produce the raster output according to the matrix filter designed by the user (see FILTERS below). The filter is applied repeat times (default value is 1). The output raster map layer can be given a title if desired. (This title should be put in quotes if it contains more than one word.)

OPTIONS
The program can be run either non-interactively or interactively. To run *r.mfilter* non-interactively, the user should specify desired flag settings and parameter values on the command line, using the form:

r.mfilter [-qpz] input=name output=name filter=name [repeat=value] [title="phrase"]

If the user specifies input, output, and filter file names on the command line, other parameters whose values are unspecified on the command line will be set to their default values (see below).

Alternately, the user can simply type *r.mfilter* on the command line, without program arguments. In this case, the user will be prompted for flag settings and parameter values using the standard GRASS parser interface described in the manual entry for parser.

Flags:

-q *r.mfilter* will normally print messages to indicate what is is doing as it proceeds. If the user specifies the -q flag, the program will run quietly.

-z The filter is applied only to zero category values in the input raster map layer. The non-zero category values are not changed. Note that if there is more than one filter step, this rule is applied to the intermediate raster map layer — only zero category values which result from the first filter will be changed. In most cases this will NOT be the desired result. Hence -z should be used only with single step filters.

Parameters:

input=name The name of an existing raster file containing data values to be filtered.

output=name The name of the new raster file to contain filtered program output.

filter=name The name of an existing, user-created UNIX ASCII file whose contents is a matrix defining the way in which the input file will be filtered. The format of this file is described below, under FILTERS.

repeat=value The number of times the filter is to be applied to the input data.
Options: integer values Default: 1

title="phrase" A title to be assigned to the filtered raster output map. If the title exceeds one word, it should be quoted.
Default: (none)

FILTERS
The filter file is a normal UNIX ASCII file designed by the user. It has the following format:

```
TITLE      title
MATRIX     n
 .
n lines of n integers
 .
DIVISOR    d
TYPE       S/P
```

TITLEA one-line title for the filter. If a title was not specified on the command line, it can be specified here. This title would be used to construct a title for the resulting raster map layer. It should be a one-line description of the filter.

MATRIX The matrix (n x n) follows on the next n lines. n must be an odd integer greater than or equal to 3. The matrix itself consists of n rows of n integers. The integers must be separated from each other by at least 1 blank.

DIVISOR The filter divisor is d. If not specified, the default is 1. If the divisor is zero (0), then the divisor is dependent on the category values in the neighborhood (see HOW THE FILTER WORKS below).

TYPE The filter type. S means sequential, while P mean parallel. If not specified, the default is S.

Sequential filtering happens in place. As the filter is applied to the raster map layer, the category values that were changed in neighboring cells affect the resulting category value of the current cell being filtered.

Parallel filtering happens in such a way that the original raster map layer category values are used to produce the new category value.

More than one filter may be specified in the filter file. The additional filter(s) are described just like the first. For example, the following describes two filters:

```
EXAMPLE FILTER FILE
TITLE     3x3 average, non-zero data only, followed by 5x5 average
MATRIX    3
1 1 1
1 1 1
1 1 1
DIVISOR   0
TYPE      P

MATRIX    5
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
DIVISOR   25
TYPE      P
```

HOW THE FILTER WORKS
The filter process produces a new category value for each cell in the input raster map layer by multiplying the category values of the cells in the n x n neighborhood around the center cell by the corresponding matrix value and adding them together. If a divisor is specified, the sum is divided by this divisor, rounding to the nearest integer. (If a zero divisor was specified, then the divisor is computed for each cell as the sum of the MATRIX values where the corresponding input cell is non-zero.)

If more than one filter step is specified, either because the repeat value was greater than one or because the filter file contained more than one matrix, these steps are performed sequentially. This means that first one filter is applied to the entire input raster map layer to produce an intermediate result; then the next filter is applied to the intermediate result to produce another intermediate result; and so on, until the final filter is applied. Then the output cell is written.

NOTES
If the resolution of the geographic region does not agree with the resolution of the raster map layer, unintended resampling of the original data may occur. The user should be sure that the geographic region is set properly.

SEE ALSO
g.region, r.clump, r.neighbors, and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Laboratory

r.mode

NAME

r.mode - Finds the mode of values in a cover map within areas assigned the same category value in a user-specified base map. (GRASS Raster Program)

SYNOPSIS

r.mode
r.mode help
r.mode base=name cover=name output=name

DESCRIPTION

r.mode calculates the most frequently occurring value (i.e., mode) of data contained in a cover raster map layer for areas assigned the same category value in the user-specified base raster map layer. These modes are stored in the new output map layer. The output map is actually a reclass of the base map.

If the user simply types *r.mode* on the command line, the user is prompted for the parameter values through the standard parser interface (see parser manual entry).

Alternately, the user can supply all needed parameter values on the command line.

Parameters:

base=name An existing raster map layer in the user's current mapset search path. For each group of cells assigned the same category value in the base map, the mode of the values assigned these cells in the cover map will be computed.

cover=name An existing raster map layer containing the values to be used to compute the mode within each category of the base map.

output=name The name of a new map layer to contain program output (a reclass of the base map). The mode values will be stored in the output map.

NOTES

The user should use the results of *r.mode* with care. Since this utility assigns a value to each cell which is based on global information (i.e., information at spatial locations other than just the location of the cell itself), the resultant map layer is only valid if the geographic region and mask settings are the same as they were at the time that the result map was created.

Results are affected by the current region settings and mask.

SEE ALSO

g.region, r.average, r.cats, r.clump, r.describe, r.mapcalc, r.mask, r.median, r.mfilter, r.neighbors, r.reclass, r.stats, and parser

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.neighbors

NAME

r.neighbors - Makes each cell category value a function of the category values assigned to the cells around it, and stores new cell values in an output raster map layer.
(GRASS Raster Program)

SYNOPSIS

r.neighbors
r.neighbors help
r.neighbors [-aq] input=name output=name method=name size=value [title="phrase"]

DESCRIPTION

r.neighbors looks at each cell in a raster input file, and examines the category values assigned to the cells in some user-defined "neighborhood" around it. It outputs a new raster map layer in which each cell is assigned a category value that is some (user-specified) function of the values in that cell's neighborhood. For example, each cell in the output layer might be assigned a category value equal to the average of the category values appearing in its 3 x 3 cell "neighborhood" in the input layer.

The program will be run non-interactively if the user specifies program arguments (see OPTIONS) on the command line. Alternately, the user can simply type *r.neighbors* on the command line, without program arguments. In this case, the user will be prompted for flag settings and parameter values.

OPTIONS

The user must specify the names of the raster map layers to be used for input and output, the method used to analyze neighborhood category values (i.e., the neighborhood function or operation to be performed), and the size of the neighborhood. Optionally, the user can also specify the title to be assigned to the raster map layer output, elect to not align the resolution of the output with that of the input (the *-a* option), and elect to run *r.neighbors* quietly (the *-q* option). These options are described further below.

Neighborhood Operation Methods: The neighborhood operators determine what new category value a center cell in a neighborhood will have after examining category values inside its neighboring cells. Each cell in a raster map layer becomes the center cell of a neighborhood as the neighborhood window moves from cell to cell throughout the map layer. *r.neighbors* can perform the following operations:

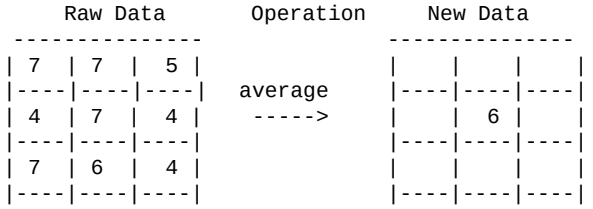
average The average category value within the neighborhood. In the following example, the result would be: (7*4 + 6 + 5 + 4*3)/9 = 5.66 The result is rounded to the nearest integer (in this case 6).

median The category value found half-way through a list of the neighborhood's category values, when these are ranged in numerical order.

mode The most frequently occurring category value in the neighborhood.

minimum The minimum category value within the neighborhood.

maximum The maximum category value within the neighborhood.



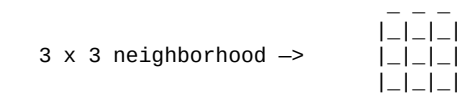
stddev The statistical standard deviation of category values within the neighborhood (rounded to the nearest integer).

variance The statistical variance of category values within the neighborhood (rounded to the nearest integer).

diversity The number of different category values within the neighborhood. In the above example, the diversity is 4.

interspersion The percentage of cells containing categories which differ from the category assigned to the center cell in the neighborhood, plus 1. In the above example, the interspersion is: 5/8 * 100 + 1 = 63.5 The result is rounded to the nearest integer (in this case 64).

Neighborhood Size: The neighborhood size specifies which cells surrounding any given cell fall into the neighborhood for that cell. The size must be an odd integer. Options are: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, and 25. For example,



-a If specified, *r.neighbors* will not align the output raster map layer with that of the input raster map layer. The r.neighbors program works in the current geographic region. It is recommended, but not required, that the resolution of the geographic region be the same as that of the raster map layer. By default, if unspecified, *r.neighbors* will align these geographic region settings.

-q If specified, *r.neighbors* will run relatively quietly (i.e., without printing to standard output notes on the program's progress). If unspecified, the program will print messages to standard output by default.

NOTES

The *r.neighbors* program works in the current geographic region with the current mask, if any. It is recommended, but not required, that the resolution of the geographic region be the same as that of the raster map layer. By default, *r.neighbors* will align these geographic region settings. However, the user can elect to keep original input and output resolutions which are not aligned by specifying this (e.g., using the *-a* option).

r.neighbors copies the GRASS color files associated with the input raster map layer for those output map layers that are based on the neighborhood average, median, mode, minimum, and maximum. Because standard deviation, variance, diversity, and interspersion are indices, rather than direct correspondents to input category values, no color files are copied for these map layers. (The user should note that although the color file is copied for averaged neighborhood function output, whether or not the color file makes sense for the output will be dependent on the input data values.)

SEE ALSO

g.region, r.clump, r.mapcalc, r.mask, r.mfilter, r.support

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.out.ascii

NAME
r.out.ascii - Converts a raster map layer into an ASCII text file.
(GRASS Raster Data Export Program)

SYNOPSIS
r.out.ascii
r.out.ascii help
r.out.ascii [-h] map=name [digits=value]

DESCRIPTION
r.out.ascii converts a user-specified raster map layer (map=name) into an ASCII text file suitable for export to other computer systems. The *digits=value* option (where value is a number of the user's choice) can be used to request that numbers in the ouput be equally-spaced (i.e., columnar output). Each category value in the ASCII map layer will then take up value number of spaces. However, to use this, the user should know the maximum number of digits that will occur in the output file, and add one to this number (to leave a space between each column). The user can find the maximum number of digits occurring in the output file by running *r.out.ascii* without the *digits=value* option.

The GRASS program *r.in.ascii* can be used to perform the reverse function, converting an ASCII file in suitable format to GRASS raster file format.

Flag:

-h Suppress printing of header information.

Parameters:

map=name Name of an existing raster map layer.

digits=value The minimum number of digits (per cell) to be printed.

r.out.ascii can be run either non-interactively or interactively. The program will be run non-interactively if the user specifies the name of a raster map layer and (optionally) a value for digits, using the form

r.out.ascii map=name [digits=value]

where name is the name of a raster map layer to be converted to ASCII format, and value is the minimum number of digits (per cell) to be printed to output. The user can also the *-h* option to suppress the output of file header information.

Alternately, the user can simply type *r.out.ascii* on the command line, without program arguments. In this case, the user will be prompted for parameter values using the standard GRASS parser interface described in the manual entry for parser.

NOTES
The output from r.out.ascii may be placed into a file by using the UNIX redirection mechanism; e.g.:

r.out.ascii map=soils > out.file

The output file out.file can then be printed or copied onto a magnetic tape or floppy disk for export purposes.

SEE ALSO
r.in.ascii and parser

AUTHOR
Michael Shapiro, U.S. Construction Engineering ResearchLaboratory

r.patch

NAME

r.patch - Creates a composite raster map layer by using known category values from one (or more) map layer(s) to fill in areas of "no data" in another map layer.
(GRASS Raster Program)

SYNOPSIS

r.patch
r.patch help
r.patch [-q] input=name[,name,...] output=name

DESCRIPTION

The GRASS program *r.patch* allows the user to assign known data values from other raster map layers to the "no data" areas (those assigned category value 0) in another raster map layer. This program is useful for making a composite raster map layer from two or more adjacent map layers, for filling in "holes" in a raster map layer's data (e.g., in digital elevation data), or for updating an older map layer with more recent data.

The program will be run non-interactively if the user specifies program arguments on the command line, using the form

r.patch [-q] input=name[,name,...] output=name

where each input name is the name of a raster map layer to be patched, the output name is the name assigned to the new composite raster map layer containing the patched result, and the (optional) *-q* flag directs *r.patch* to run quietly.

The first name listed in the string *input=name,name,name, ...* is the name of the base map whose zero data values will be attempted to be filled by non-zero data values in the second through tenth input name maps listed. The second through tenth input name maps will be used to supply remaining missing (zero) data values for the first input map name, based on the order in which they are listed in the string *input=name,name,name, ...*

Alternately, the user can simply type *r.patch* on the command line, without program arguments. In this case, the user will be prompted for the flag setting and parameter values using the standard GRASS parser interface described in the manual entry for parser.

Flag:

-q Directs that *r.patch* run quietly, suppressing output messages on program progress to standard output.

Parameters:

input=name,name,... The name(s) of between one and ten existing raster map layers to be patched together. The first of the ten maps listed will be used as a base map, and the second through tenth maps listed will be used to supply missing (zero) category values for the first map.

output=name The name of the new raster map to contain the resultant patched output.

EXAMPLE

Below, the raster map layer on the far left is patched with the middle (patching) raster map layer, to produce the composite raster map layer on the right.

```
1 1 1 0 2 2 0 0    0 0 1 1 0 0 0 0    1 1 1 1 2 2 0 0
1 1 0 2 2 2 0 0    0 0 1 1 0 0 0 0    1 1 1 2 2 2 0 0
3 3 3 3 2 2 0 0    0 0 0 0 0 0 0 0    3 3 3 3 2 2 0 0
3 3 3 3 0 0 0 0    4 4 4 4 4 4 4 4    3 3 3 3 4 4 4 4
3 3 3 0 0 0 0 0    4 4 4 4 4 4 4 4    3 3 3 4 4 4 4 4
0 0 0 0 0 0 0 0    4 4 4 4 4 4 4 4    4 4 4 4 4 4 4 4
```

Switching the patched and the patching raster map layers produces the following results:

```
0 0 1 1 0 0 0 0    1 1 1 0 2 2 0 0    1 1 1 1 2 2 0 0
0 0 1 1 0 0 0 0    1 1 0 2 2 2 0 0    1 1 1 1 2 2 0 0
0 0 0 0 0 0 0 0    3 3 3 3 2 2 0 0    3 3 3 3 2 2 0 0
4 4 4 4 4 4 4 4    3 3 3 3 0 0 0 0    4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4    3 3 3 0 0 0 0 0    4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4    0 0 0 0 0 0 0 0    4 4 4 4 4 4 4 4
```

NOTES
Frequently, this program is used to patch together adjacent map layers which have been digitized separately. The programs *v.mkquads* and *v.mkgrid* can be used to make adjacent maps align neatly.

The user should check the current geographic region settings before running *r.patch*, to ensure that the region boundaries encompass all of the data desired to be included in the composite map.

Use of *r.patch* is generally followed by use of the GRASS programs *g.remove* and *g.rename*; *g.remove* is used to remove the original (un-patched) raster map layers, while *g.rename* is used to then assign to the newly-created composite (patched) raster map layer the name of the original raster map layer.

r.patch creates support files for the patched, composite output map.

SEE ALSO
g.region, g.remove, g.rename, r.mapcalc, r.support, v.mkgrid, v.mkquads, and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.poly

NAME
r.poly - Extracts area edges from a raster map layer and converts data to GRASS vector format.
(GRASS Raster Program)

SYNOPSIS
r.poly
r.poly help
r.poly [-l] input=name output=name

DESCRIPTION
r.poly scans the named input raster map layer, extracts area edge features from it, converts data to GRASS vector format, and smoothes vectors. *r.poly* first traces the perimeter of each unique area in the raster map layer and creates vector data to represent it. The cell category values for the raster map layer will be used to create attribute information for the resultant vector area edge data.

A true vector tracing of the area edges might appear blocky, since the vectors outline the edges of raster data that are stored in rectangular cells. To produce a better-looking vector map, r.poly smoothes the corners of the vector data as they are being extracted. At each change in direction (i.e., each corner), the two midpoints of the corner cell (half the cell's height and width) are taken, and the line segment connecting them is used to outline this corner in the resultant vector file. (The cell's cornermost node is ignored.) Because vectors are smoothed by this program, the resulting vector map will not be "true" to the raster map from which it was created. The user should check the resolution of the geographic region (and the original data) to estimate the possible error introduced by smoothing.

OPTIONS
The user can run this program either non-interactively or interactively. The program will be run non-interactively if the user specifies program arguments and flag settings on the command line using the form:

r.poly [-l] input=name output=name

Alternately, the user can simply type r.poly on the command line without program arguments. In this case, the user will be prompted for parameter values and flag settings using the standard GRASS parser interface described in the manual entry for parser.

Flag:
-l Smooth corners.

Parameters:
input=name Use the existing raster map name as input.

output=name Set the new vector output file name to name.

NOTES
r.poly extracts only area edges from the named raster input file. If the raster file contains other data (i.e., line edges, or point data) the output may be wrong. The user must run *v.support* on the resultant file to build the needed topology information stored in the dig_plus file.

SEE ALSO
v.support, and parser

AUTHOR
Original version of *r.poly*: Jean Ezell, U.S. Army Construction Engineering Research Laboratory and Andrew Heekin, U.S. Army Construction Engineering Research Laboratory

Modified program for smoothed lines: David Satnik, Central Washington University, WA

r.profile

NAME
r.profile - Outputs the raster map layer values lying on user-defined line(s).
(GRASS Raster Program)

SYNOPSIS
r.profile
r.profile help
r.profile map=name [result=type] [width=value] line=east,north,east,north[,east,north,east,north,...]

DESCRIPTION
This program outputs, in ASCII, the values assigned to those cells in a raster map layer that lie along one or more lines ("profiles"). The lines are described by their starting and ending coordinates. The profiles may be single-cell wide lines, or multiple-cell wide lines. The output, for each profile, may be the category values assigned to each of the cells, or a single aggregate value (e.g., average or median value).

OPTIONS
Parameters:
map=name Raster map to be queried.

result=type Type of result to be output.
Options: raw, median, average Default: raw

Raw results output each of the category values assigned to all cells along the profile. Median and average output a single value per profile: average outputs the average category value of all cells under the profile; median outputs the median cell category value.

line=east,north,east,north[,east,north,east,north,...] The geographic coordinates of the starting and ending points that define each profile line, given as easting and northing coordinate pairs. The user must state the starting and ending coordinates of at least one line, and may optionally include starting and ending coordinates of additional lines.

width=value Profile width, in cells (odd number).
Default: 1

Wider profiles can be specified by setting the width to 3, 5, 7, etc. The profiles are then formed as rectangles 3, 5, 7, etc., cells wide.

OUTPUT FORMAT
The output from this command is printed to the standard output in ASCII. The format of the output varies slightly depending on the type of result. The first number printed is the number of cells associated with the profile. For raw output, this number is followed by the individual cell values. For average and median output, this number is followed by a single value (i.e., the average or the median value). These examples are for the elevation.dem raster map layer in the spearfish sample data set distributed with GRASS:

Single-cell profile:

r.profile map=elevation.dem line=593655,4917280,593726,491735

```
4 1540 1551 1557 1550
```

3-cell wide profile:

r.profile map=elevation.dem line=593655,4917280,593726,4917351 width=3

```
22 1556 1538 1525 1570 1555 1540 1528 1578 1565 1551 1536 1523 1569 1557 1546 1533 1559 1550 1542
1552 1543 1548
```

3-cell wide profile average:

r.profile map=elevation.dem line=593655,4917280,593726,4917351 width=3 result=average

```
22 1548.363636
```

3-cell wide profile median:

r.profile map=elevation.dem line=593655,4917280,593726,4917351 width=3 result=median

```
22 1549.000000
```

SEE ALSO
d.profile, r.transect

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.random

NAME
r.random - Creates a raster map layer and site list file containing randomly located sites.
(GRASS Raster Program)

SYNOPSIS
r.random
r.random help
r.random [-qz] input=name nsites=number[%] [raster_output=name] [sites_output=name]

DESCRIPTION
The program *r.random* allows the user to create a raster map layer and a site list file containing geographic coordinates of points whose locations have been randomly determined. The program locates these randomly generated sites within the current geographic region and mask (if any), on non-zero category value data areas within a user-specified raster map layer. If the user sets the -z flag, sites will be randomly generated across all cells (even those assigned category value 0).

The raster_output raster map layer is created in the user's current mapset. The category values and corresponding category names already associated with the random site locations in the input map layer are assigned to these sites in the raster_output map layer. The site_lists file created by r.random contains a listing of the sites' geographic coordinates; these coordinates are the center points of the randomly selected cells.

OPTIONS
The user may specify the quantity of random locations to be generated either as a positive integer (e.g., 10), or as a percentage of the raster map layer's total area (e.g., 10%, or 3.05%). If unspecified, the number of sites is set to '0' by default. If stated as a percentage of the raster map's total size, the number of random locations generated will be set equal to the number of cells contained within the stated percentage of the raster map layer. Options are 0-100; percentages less than one percent may be stated as decimals. The default percentage value used, if unspecified by the user, is '0'. (Note that choosing 1% of a raster map's cells frequently produces an abundance of random locations.)

r.random can be run interactively or non-interactively. The user may provide program arguments on the command line, specifying an input map layer name (*input=name*), output raster map layer name (*raster_output=name*), output site list file name *(sites_output=name*), and (optionally) give the number of sites to be randomly generated as a total number of sites (*nsites=number*) or as a percentage of the map's size (*nsites=number%*). The user can also direct that *r.random* run quietly (using the *-q*) option, and/or direct *r.random* to also generate random site locations against cells containing category zero (using the *-z* option).

Alternately, the user can simply type *r.random* on the command line without program arguments. In this case, the user will be prompted for needed inputs and option choices using the standard GRASS user interface described in the manual entry for parser.

Flags:

-q Run quietly. *r.random* will normally print output messages to standard output as it runs. The *-q* option will suppress the printing of these messages.

-z Include areas assigned a category value of zero within the pool of areas within which *r.random* will randomly generate site locations. If the *-z* option is specified, sites that fall in areas assigned a category value of zero in the input map layer will be assigned to a newly-created category in the output raster map layer. If the *-z* flag is not set, cells having category value 0 in the output layer will represent the areas at which randomly- located sites were not placed.

Parameters:

input=name An existing raster map layer in the user's current mapset search path. *r.random* will randomly generate sites on a user- specified portion of the cells in this input raster map.

nsites=number or *nsites=number%* Allows the user to specify the quantity of sites to be randomly generated as either a positive integer, or as a percentage value of the number of cells in the input map layer. If stated as a positive integer, number is the number of sites (i.e., number of cells) to appear in the raster_output layer and/or sites_output file. Options: Non-percentage values should be given as positive integer values less than or equal to the number of cells in the input map layer. Percentage values given should be within the range 0.00 - 100.00 (decimal values are allowed).

raster_output=name The new raster map layer to hold program output. This map will contain the sites randomly generated by *r.random*. If the *-z* flag is not set, all sites will be assigned whatever category values were assigned these cell locations in the input raster map layer. If the *-z* flag is set, all sites except those falling on cells assigned category value 0 in the input value will be assigned the category values assigned these cells in the input layer; sites falling on cells assigned category value 0 in the input layer will be assigned to a newly created category in the raster_output layer.

sites_output=name The new GRASS site_lists file to hold program output. If no sites_output file name is given on the command line, no site_lists file will be created by *r.random*. (See *raster_output* parameter description, above.)

Note. Although the user need not request that *r.random* output both a raster map layer (raster_output) and a site list file (sites_output), the user must specify that at least one of these outputs be produced.

NOTES
To create random site locations within some, but not all, non-zero categories of the input raster map layer, the user must first create a reclassified raster map layer of the original raster map layer (e.g., using the GRASS program *r.reclass*) that contains only the desired categories, and then use the reclassed raster map layer as input to *r.random*.

SEE ALSO
g.region, r.mask, r.reclass, and parser

AUTHOR
Dr. James Hinthorne, GIS Laboratory, Central Washington University

r.reclass.scs

NAME
r.reclass.scs - Create a new raster map layer based on an existing raster map.
(SCS GRASS Raster Program)

SYNOPSIS
r.reclass.scs
r.reclass.scs help

DESCRIPTION
r.reclass.scs is an interface to the GRASS r.reclass program. The program will reclassify the category values in a raster map layer based on reclass instructions entered by the user. The user can enter map reclassification rules to *r.reclass.scs* either from standard input or from a file. The program then issues r.reclass commands to produce the new reclassified raster map layer.

Input to *r.reclass.scs* consists of a list of category names or category values that will be grouped into the same category in the output (reclassified) map. Only one category name should appear on each line of input. Input can be entered either interactively, or from a file.

A file containing these reclass rules can be created using a text editor, word-processor, DBMS, etc. It is no more than a list of category names which will have the same category value after the reclassification.

SEE ALSO
r.reclass, r.resample, r.rescale

AUTHOR
R.L. Glenn, USDA, SCS, NHQ-CGIS

r.reclass

NAME
r.reclass - Creates a new map layer whose category values mare based upon the user's reclassification of categories in an existing raster map layer.
(GRASS Raster Program)

SYNOPSIS
r.reclass
r.reclass help
r.reclass input=name output=name [title=name]

DESCRIPTION
r.reclass creates an output map layer based on an input raster map layer. The output map layer will be a reclassification of the input map layer based on reclass rules input to r.reclass, and can be treated in much the same way that raster files are treated. A title for the output map layer may be (optionally) specified by the user.

The reclass rules are read from standard input (i.e., from the keyboard, redirected from a file, or piped through another program).

The program will be run non-interactively if the user specifies the name of the raster map layer to be reclassified, the name of an output layer to hold reclass rules, and (optionally) the name of a title for the output map:

r.reclass input=name output=name [title=name]

After the user types in the above information on the command line, the program will (silently) prompt the user for reclass rules to be applied to the input map layer categories. The form of these rules is described in further detail in the sections on non-interactive program use reclass rules and examples, below.

Alternately, the user can simply type r.reclass on the command line, without program arguments. In this case, the user will be prompted for all needed inputs.

Before using *r.reclass* one must know the following:

1 The new categories desired; and, which old categories fit into which new categories.

2 The names of the new categories.

INTERACTIVE PROGRAM USE: EXAMPLE
Suppose we want to reclassify the raster map layer roads, consisting of five categories, into the three new categories: paved roads, unpaved roads, and railroad tracks. The user is asked whether the reclass table is to be established with each category value initially set to 0, or with each category value initially set to its own value. A screen like that shown below then appears, listing the categories of the roads raster map layer to be reclassified and prompting the user for the new category values to be assigned them.

```
ENTER NEW CATEGORY NUMBERS FOR THESE CATEGORIES

   OLD CATEGORY NAME          OLD NUM   NEW NUM
   no data                    0         0__
   Hard Surface, 2 lanes      1         0__
   Loose Surface, 1 lane      2         0__
   Improved Dirt              3         0__
   Unimproved Dirt Trail      4         0__
   Railroad, single track     5         0__

   AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
  (OR <Ctrl-C> TO CANCEL)
```

In the following screen the new category values have been entered beside the appropriate old category names. Cells assigned category values 2, 3, and 4 in the old raster map layer are now assigned the new category value 2 in the reclassed map; cell data formerly assigned to category value 5 in the old raster map map are now assigned the new category value 3 in the reclassed map.

```
ENTER NEW CATEGORY NUMBERS FOR THESE CATEGORIES

   OLD CATEGORY NAME          OLD NUM   NEW NUM
   no data                    0         0__
   Hard Surface, 2 lanes      1         1__
   Loose Surface, 1 lane      2         2__
   Improved Dirt              3         2__
   Unimproved Dirt Trail      4         2__
   Railroad, single track     5         3__

   AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
  (OR <Ctrl-C> TO CANCEL)
```

Hitting the escape key <ESC> will bring up the following screen, which prompts the user to enter a new title and category label for the newly reclassed categories.

```
 ENTER NEW CATEGORY NAMES FOR THESE CATEGORIES

 TITLE:  Roads Reclassified
       CAT             NEW CATEGORY NAME
       NUM
       0               no data
       1               Paved Roads
       2               Unpaved Roads
       3               Railroad, single track

 AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)
```

Based upon the information supplied by the user in the above sample screens, the new output map, supporting category, color, history, and header files are created.

NON-INTERACTIVE PROGRAM USE: RECLASS RULES

In non-interactive program use, the names of an input map, output map, and output map title are given on the command line. However, the reclass rules are still read from standard input (i.e., from the keyboard, redirected from a file, or piped through another program).

Once the user has specified an input raster map layer, output map layer name, and (optionally) output map layer title by typing:

r.reclass input=name output=name [title=name]

Each line of input must have the following format:

input_categories=output_category [label]

where the input lines specify the category values in the input raster map layer to be reclassified to the new output_category category value. Specification of a label to be associated with the new output map layer category is optional. If specified, it is recorded as the category label for the new category value. The equal sign (=) is required. The input_category(ies) may consist of single category values or a range of such values in the format “low thru high.” The word “thru” must be present.

A line containing only the word end terminates the input.

NON-INTERACTIVE PROGRAM USE: EXAMPLES
The following examples may help clarify the reclass rules.

1 This example reclassifies categories 1, 3 and 5 in the input raster map layer to category 1 with category label “poor quality” in the output map layer, and reclassifies input raster map layer categories 2, 4, and 6 to category 2 with the label “good quality” in the output map layer.

1 3 5 = 1 poor quality
2 4 6 = 2 good quality

2 This example reclassifies input raster map layer categories 1 thru 10 to output map layer category 1, input map layer categories 11 thru 20 to output map layer category 2, and input map layer categories 21 thru 30 to output map layer category 3, all without labels.

1 thru 10 = 1
11 thru 20 = 2
21 thru 30 = 3

3 Subsequent rules override previous rules. Therefore, the below example reclassifies input raster map layer categories 1 thru 19 and 51 thru 100 to category 1 in the output map layer, input raster map layer categories 20 thru 24 and 26 thru 50 to the output map layer category 2, and input raster map layer category 25 to the output category 3.

1 thru 100 = 1 poor quality
20 thru 50 = 2 medium quality
25 = 3 good quality

4 The previous example could also have been entered as:

1 thru 19 51 thru 100 = 1 poor quality
20 thru 24 26 thru 50 = 2 medium quality
25 = 3 good quality

or as:

1 thru 19 = 1 poor quality
51 thru 100 = 1
20 thru 24 = 2
26 thru 50 = 2 medium quality
25 = 3 good quality

The final example was given to show how the labels are handled. If a new category value appears in more than one rule (as is the case with new category values 1 and 2), the last label which was specified becomes the label for that category. In this case the labels are assigned exactly as in the two previous examples.

NOTES
In fact, the *r.reclass* program does not generate any new raster map layers (in the interests of disk space conservation). Instead, a reclass table is stored which will be used to reclassify the original raster map layer each time the new (reclassed) map name is requested. As far as the user (and programmer) is concerned, that raster map has been created. Also note that although the user can generate a *r.reclass* map which is based on another r.reclass map, the new *r.reclass* map map will be stored in GRASS as a reclass of the original raster map on which the first reclassed map was based. Therefore, while GRASS allows the user to provide *r.reclass* map layer information which is based on an already reclassified map (for the user’s convenience), no *r.reclass* map layer (i.e., reclass table) will ever be stored as a r.reclass of a *r.reclass*.

To convert a reclass map to a regular raster map layer, set your geographic region settings to match the settings in the header for the reclass map (an ASCII file found under the cellhd directory, or viewable by running r.support) and then run *r.resample*.

r.mapcalc can also be used to convert a reclass map to a regular raster map layer:

r.mapcalc raster_map=reclass_map

where raster_map is the name to be given to the new raster map, and reclass_map is an existing reclass map.

BEWARE
Because *r.reclass* generates a table referencing some original raster map layer rather than creating a reclassed raster map layer, a *r.reclass* map layer will no longer be accessible if the original raster map layer upon which it was based is later removed.

An *r.reclass* map is not a true raster map layer. Rather, it is a table of reclassification values which reference the input raster map layer. Therefore, users who wish to retain reclassified map layers must also save the original input raster map layers from which they were generated.

Category values which are not explicitly reclassified to a new value by the user will be reclassified to 0.

SEE ALSO
r.resample, r.rescale

AUTHORS
James Westervelt, U.S. Army Construction Engineering Research Laboratory
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.report

NAME
r.report - Reports statistics for raster map layers.
(GRASS Raster Program)

SYNOPSIS
r.report
r.report help
r.report [-hmfqe] map=name[,name,...] [units=name[,name,...]] [pl=value] [pw=value] [output=name]

DESCRIPTION
r.report allows the user to set up a series of report parameters to be applied to a raster map layer, and creates a report. If invoked with command line arguments, the report will print out to the screen only. However, output may be redirected to a file or another program using the UNIX redirection mechanism. If invoked without command line arguments, the user is given the option of printing out each report and/or saving output to a file.

The program will be run non-interactively, if the user specifies the names of raster map layers and any desired options on the command line, using the form

r.report [-hmfqe] map=name[,name,...] [units=name[,name,...]] [pl=value] [pw=value]

where each map name is the name of a raster map layer on which to report, each unit name is a unit of measure in which results are to be reported, the *pl* value gives the page length, the *pw* value gives the page width, and the (optional) flags *-h, -e, -m, -f*, and *-q* have the meanings stated below.

Flags:
-h Suppress the printout of page headers.

-m Report on zero values, because a mask is being used.

-f Use formfeeds between pages when printing report output.

-q Run quietly, without printing program messages to standard output.

-e Use scientific format for the numbers that are too long to fit in the tab table field if their decimal form is used.

-z Report only non-zero data values. Zero data will not be reported. However, for multiple map layers this means that if zero values occur in every map layer, they will not be reported; if non-zero category values occur in any map layer (along with zeros in others), the non- zero values along with the zero values will be reported.

Parameters:
map=name,name,... Names of raster map(s) on which to report.

units=name Units of measure in which results are to be reported. These units are based on the number of cells in the user's area of interest (i.e., cells within the current geographic region definition, and the current mask [if any]). These are established with the programs g.region and r.mask, respectively.
Options: Possible units of measurement are:
mi (cover measured in square miles)
me (cover measured in square meters)
k (cover measured in square kilometers)
a (cover measured in acres)
h (cover measured in hectares)
c (the number of cells in the area of interest)
p (the percent cover, excluding no data areas)

pl=value Page length, in lines, in which report will be output.
Default: 0 (lines)

pw=value Page width, in characters, in which report will be output.
Default: 79 (characters)

output=name The name of a file to store the report in. If not specified, the report is printed on the terminal screen.

Alternately, the user can simply type *r.report* on the command line, without program arguments. In this case, the user will be prompted for program flag settings and parameter values. The report itself consists of two parts, a header section and the main body of the report. The header section of the report identifies the raster map layer(s) (by map layer name and title), location, mapset, report date, and the region of interest. The area of interest is described in two parts: the user's current geographic region is presented, and the mask is presented (if any is used). The main body of the report consists of from one to three tables which present the statistics for each category and the totals for each unit column. Note that, unlike *r.stats*, *r.report* allows the user to select the specific units of measure in which statistics will be reported.

Following is the result of a r.report run on the raster map layer geology (located in the Spearfish, SD sample data base), with the units expressed in square miles and acres. Here, r.report output is directed into the file report.file.

EXAMPLE:
r.report map=geology units=miles,acres > report.file

```
 _____________________________________________________________________
| RASTER MAP CATEGORY REPORT                                          |
| LOCATION: spearfish   Fri Sep 2 09:20:09                            |
|_____________________________________________________________________|
|          north:   4928000.00   east:   609000.00                    |
| REGION:  south:   4914000.00   west:   590000.00                     |
|          res:    100.00   res:  100.00                              |
|_____________________________________________________________________|
| MASK:none                                                           |
|_____________________________________________________________________|
| MAP: geology in PERMANENT                                           |
|_____________________________________________________________________|
|    Category Information                        |    Acres  |   Square |
|    #      description                          |           |   Miles  |
|________________________________________________|___________|__________|
|    0   |  no data                              |     415.13|     0.65 |
|    1   |  metamorphic                          |    2597.02|     0.46 |
|    2   |  transition                           |      32.12|     0.05 |
|    3   |  igneous                              |    8117.24|    12.68 |
|    4   |  sandstone                            |   16691.60|    26.08 |
|    5   |  limestone                            |   13681.93|    21.38 |
|    6   |  shale                                |   10304.07|    16.10 |
|    7   |  sandy shale                          |    2517.95|     3.93 |
|    8   |  claysand                             |    3229.60|     5.05 |
|    9   |  sand                                  |   8141.95|    12.72 |
|________________________________________________|___________|__________|
|  TOTAL                                         |  65728.60|   102.70 |
|________________________________________________|___________|__________|
```

NOTES
If the user runs r.report interactively and saves the report output in a file, this file will be placed into the user's current working directory. If the user runs *r.report* non-interactively, report output can be saved by redirecting it to a file or a printer using the UNIX redirection mechanism.

SEE ALSO
g.region, r.coin, r.describe, r.info, r.mask, r.stats

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.resample

NAME
r.resample - GRASS raster map layer data resampling capability.
(GRASS Raster Program)

SYNOPSIS
r.resample
r.resample help
r.resample [-q] input=name output=name

DESCRIPTION
r.resample resamples the data values in a user-specified raster input map layer name (bounded by the current geographic region and masked by the current mask), and produces a new raster output map layer name containing the results of the resampling. The category values in the new raster output map layer will be the same as those in the original, except that the resolution and extent of the new raster output map layer will match those of the current geographic region settings (see g.region).

The program will be run non-interactively if the user specifies program arguments on the command line, using the form

r.resample [-q] input=name output=name

where the input name is the name of the raster map layer whose data are to be resampled, the output name is the name of the raster map layer to store program output, and the -q option, if present, directs that *r.resample* run quietly (supressing the printing of program messages to standard output).

Alternately, the user can simply type *r.resample* on the command line, without program arguments. In this case, the user will be prompted for needed inputs and option choices using the standard GRASS parser interface described in the manual entry for parser.

NOTES
The method by which resampling is conducted is “nearest neighbor” (see *r.neighbors*). The resulting raster map layer will have the same resolution as the resolution of the current geographic region (set using *g.region*).

The resulting raster map layer may be identical to the original raster map layer. The *r.resample* program will copy the color table and history file associated with the original raster map layer for the resulting raster map layer and will create a modified category file which contains description of only those categories which appear in resampled file.

When the user resamples a GRASS reclass file, a true raster file is created by *r.resample*.

SEE ALSO
g.region, r.mapcalc, r.mask, r.mfilter, r.neighbors, r.rescale, and parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.rescale.inf

NAME
r.rescale.inf - Generate a raster map layer in which the categories represent values in a database column which have been divided into equal interval units.
(GRASS-RDBMS Raster Interface Program)

SYNOPSIS
r.rescale.inf
r.rescale.inf help
r.rescale.inf tab=name key=name col=name cats=name input=name output=name [join=tab,tabkey,pkey]

DESCRIPTION
r.rescale.inf creates a reclassed raster map layer by dividing the values in a numeric column in the currently selected database into equal interval units. The number of resulting categories is determined by the user via the command line parameter *[cats=]*. *r.rescale.inf* evaluates the range of values for the database column and subsets these values into equal interval groups of records returned by the query. For example, if the database column contains values which range from 1-1000 and the [cats] value is equal to 10 the resulting raster map layer will contain the 10 categories:1=1-100, 2=101-200 etc. In other words, each category in the new raster map layer will represent a range of 100 values from the database column used in the rescale operation. The database column being evaluated must be numeric in type. To identify the data types of columns in a database table use the g.column.inf command with the [-*v*] flag. r.rescale.inf does not take outlying data values into account. Therefore, if the range of values for a database column contains a limited number of extreme values the resulting rescale operation will be skewed in the direction of these values.

OPTIONS
Parameters

tab=database_table_name Table containing a column linked to category values in an existing raster map.

key=database_column_name Column corresponding to category values in an existing raster map.

col=database_column_name Column to base rescale operation on which is numeric in type.

cats=value Number of categories to define in the resulting reclass map.

input=map Name of an existing raster file with category values linked to a column in the currently selected databae.

output=map Name of new raster map.

join=tab,tabkey,pkey *Tab* is the table used to develop the current SQL query. *Tabkey* is the database column used to relate information in this table with data in the table linked to the GRASS category file. *Pkey* is the associated column in the table linked to the GRASS category file which is related to tabkey in the current table.

For instance, assume that stf1_main is a table containing column values associated with category values in a the GRASS raster file blkgrp.ids. In addition, assume that stf1_main is a table containing attribute data on age in the column pop100. In this example stf1_main is the table associated with the GRASS raster map and tract_blck is the column linking stf1_main to the GRASS category file. The column pop100 in stf1_main will be the basis for the rescale effort. To specify the rescale:

r.rescale.inf tab=stf1_main key=tract_blck
col=pop100
cats=5 input=blkgrp.ids output=pop100.rescale

Specifying these conditions would insure that all rows from table stf1_main which satisfy the query criteria would be related to the spatial features in the GRASS data layer via the GRASS category values.

BUGS
None known.

NOTE
This program requires the Informix database software.

SEE ALSO
g.column.inf, g.select.inf, g.stats.inf, g.table.inf, d.rast.inf, d.site.inf, d.vect.inf, d.what.r.inf, d.what.s.inf, d.what.v.inf, r.reclass.inf, v.reclass.inf

AUTHOR
James A. Farley, Wang Song and W. Fredrick Limp University of Arkansas, CAST

r.rescale

NAME
r.rescale - Rescales the range of category values in a raster map layer.
(GRASS Raster Program)

SYNOPSIS
r.rescale
r.rescale help
r.rescale [-q] input=name [from=min,max] output=name [to=min,max] \ [title="phrase"]

DESCRIPTION
The *r.rescale* program rescales the range of category values appearing in a raster map layer. A new raster map layer, and an appropriate category file and color table based upon the original raster map layer, are generated with category labels that reflect the original category values that produced each category. This command is useful for producing representations with a reduced number of categories from a raster map layer with a large range of category values (e.g., elevation). Rescaled map layers are appropriate for use in such GRASS programs as *r.stats, r.report, and r.coin.*

r.rescale will be run non-interactively if the user specifies program arguments on the command line, using the form:

r.rescale [-q] input=name [from=min,max] output=name [to=min,max] \ [title="phrase"]

Alternately, the user can simply type:

r.rescale

on the command line without program arguments. In this case, the user will be prompted for parameter values using the standard GRASS user interface described in the manual entry for parser.

Flag:

-q Run quietly, without printing messages on program progress to the user's terminal.

Parameters:

input=name The name of the raster map layer whose category values are to be rescaled.

from=min,max The input map range to be rescaled.
Default: The full range of the input map layer.

output=name The name of the new, rescaled raster map layer.

to=min,max The output map range (after rescaling).
Default: 1,255

title="phrase" Title for new output raster map layer.

EXAMPLE
To rescale an elevation raster map layer with category values ranging from 1090 meters to 1800 meters into the range 1-255, the following command line could be used:

r.rescale input=elevation from=1090,1800 output=elevation.255 to=1,255

NOTES
The rescaled category value range is actually unlimited, but the category value range 1 to 255 is frequently used due to limitations of color graphics monitors.

Category values that fall beyond the input range will become zero. This allows the user to select a subset of the full category value range for rescaling if desired. This also means that the user should know the category value range for the input raster map layer. The user can request the r.rescale program to determine this range, or can obtain it using the *r.describe* command. If the category value range is determined using *r.rescale*, the input raster map layer is examined, and the minimum and maximum non-zero category values are selected as the input range.

SEE ALSO
r.coin, r.describe, r.mapcalcr.reclass, r.report, r.resample, r.stats, and *parser*

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.slope.aspect

NAME
r.slope.aspect - Generates raster map layers of slope and aspect from a raster map layer of true elevation values.
(GRASS Raster Program)

SYNOPSIS
r.slope.aspect
r.slope.aspect help
r.slope.aspect [-aqz] elevation=name [slope=name] [aspect=name] [format=name] [zfactor=value]

DESCRIPTION
r.slope.aspect generates raster map layers of slope and aspect from a raster map layer of true elevation values. The user must specify the input elevation file name and at least one output file name to contain the slope or aspect data. The user can also specify the format for slope (degrees, percent; default=degrees), and zfactor: multiplicative factor to convert elevation units to meters; (default 1.0).

The program will be run non-interactively if the user specifies program inputs and any desired options on the command line, using the form

r.slope.aspect [-aq] elevation=name [slope=name] [aspect=name] [slope=name] [zfactor=value]

If the user runs:

r.slope.aspect

without command line arguments, the program will prompt the user for flag settings and parameter values.

Flags:

-a Do not align the settings of the current geographic region (to which the output slope and aspect map layers will be set) to those of the elevation layer. See NOTES.

-q Run quietly, and suppress the printing of information on program operations during execution.

-z Assume that zero values in the elevation map layer represent true elevation values, not areas of “no data”.

Parameters:

elevation=name Name of the raster map layer of true elevation values to be used as input.

slope=name Name of a raster map layer of slope values created from the elevation map.

aspect=name Name of a raster map layer of aspect values created from the elevation map.

format=name Format for reporting the slope; options: degrees,percent; default: degrees.

zfactor=value Multiplicative factor to convert elevation units to meters (default: 1.0).

min_slp=value Minimum slope for which aspect is computed.

Resulting raster map layers of slope and aspect are named by the user and placed in the current mapset.

ELEVATION RASTER MAP
The raster elevation map layer specified by the user must contain true elevation values, not rescaled or categorized data.

ASPECT RASTER MAP
The raster aspect map layer which is created indicates the direction that slopes are facing. The aspect categories represent the number degrees of east.

Category and color table files are also generated for the aspect map layer.

SLOPE RASTER MAP
The resulting raster slope map layer will contain slope values, stated in degrees of inclination from the horizontal if *format=degrees* option (which is also default) is chosen, and in percent rise if *format=percent* option is chosen. The category file, but not the color table, is generated by r.slope.aspect for the raster slope map layer.

For most applications, the user will wish to use a reclassified map layer of slope that groups slope values into ranges of slope. This can be done using *r.reclass*. An example of a useful reclassification is given below:

category	range (in degrees)	category labels (in percent)
1	0-1	0-2%
2	2-3	3-5%
3	4-5	6-10%
4	6-8	11-15%
5	9-11	16-20%
6	12-14	21-25%
7	15-90	26% and higher

The following color table works well with the above reclassification.

category	red	green	blue
0	179	179	179
1	0	102	0
2	0	153	0
3	128	153	0
4	204	179	0
5	128	51	51
6	255	0	0
7	0	0	0

NOTES
To ensure that the raster elevation map layer is not inappropriately resampled, the settings for the current region are modified slightly (for the execution of the program only): the resolution is set to match the resolution of the elevation map and the edges of the region (i.e. the north, south, east and west) are shifted, if necessary, to line up along edges of the nearest cells in the elevation map. If the user really wants the elevation map resampled to the current region resolution, the *-a* flag should be specified.

The current mask, if set, is ignored.

The algorithm used to determine slope and aspect uses a 3x3 neighborhood around each cell in the elevation file. Thus, it is not possible to determine slope and aspect for the cells adjacent to the edges in the elevation map layer. These cells are assigned a “no data” value (category 0) in both the slope and aspect raster map layers.

Because Horn’s formula is used to find the derivatives in x and y directions, the aspect is biased in 0, 45, 90, 180, 225, 270, 315, and 360 directions; i.e., the distribution of aspect categories is very uneven, with peaks at 0, 45,..., 360 categories. In the next GRASS release, a different algorithm will be tried in the computation of derivatives in x and y directions; programmers will attempt to interpolate the surface and find the derivatives from the resulting equation.

Helena Mitasova of USACERL observed that most cells with a very small slope end up having category 0, 45,..., 360. By filtering out such aspects it is sometimes possible to reduce bias in these directions. The new option

min_slp=value

was added (minimum slope for which aspect is computed). The aspect for all cells with slope < min_slp is set to 0 (no value).

WARNING
Elevations of zero (as well as below sea level elevations) are valid. This means that areas assigned category value 0 may have one of two possible meanings: they may either be areas of "no data" or areas having 0 elevation. If the user wishes *r.slope.aspect* to assume that cells assigned category value zero in the elevation map layer represent true elevation values, not areas of "no data", the user should set the *-z* flag when running this program.

If the *-z* flag is not set and the raster map layer of true elevation contains areas of "no data" that are assigned to category 0, either at its edges or in its interior, incorrect (and usually quite large) slopes will result.

SEE ALSO
r.mapcalc, r.neighbors, r.reclass, r.rescale

AUTHORS
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
Olga Waupotitsch, U.S. Army Construction Engineering Research Laboratory

NAME
r.stats - Generates area statistics for raster map layers.
(GRASS Raster Program)

SYNOPSIS
r.stats
r.stats help
r.stats [-1aclmqzgx] input=name[,name,...] [fs=character|space] [output=name]

DESCRIPTION
r.stats calculates the area present in each of the categories of user-selected raster map layer(s). Area statistics are given in units of square meters and/or cell counts. This analysis uses the current geographic region and mask settings. Output can be sent to a file in the user's current working directory.

The program will be run non-interactively if the user specifies the program arguments and desired options on the command line, using the form

r.stats [-1aclmqzgx] input=name[,name,...] [fs=character|space] [output=name]

where each input name is the name of a raster map layer on which area/cell statistics are to be generated, the (optional) output name is the name of a file to contain program output (sent to the user's current working directory), the fs character or space is the field separator to be used to separate data fields in the output file (default is a space if unspecified), and the (optional) flags *-1, -a, -c, -l, -m, -q, -z, -g,* and *-x* have the meanings described in the OPTIONS section.

Alternately, the user can simply type *r.stats* on the command line, without program arguments. In this case, the user will be prompted for needed inputs and option choices using the standard GRASS parser interface described in the manual entry for parser.

OPTIONS
Flags:
-1 The data for each cell in the current geographic region will be output, one cell per line, rather than the totals for each distinct combination.

-a Print area totals in square meters.

-c Print total cell counts.

-m Report all zero values present in the input map layer(s), whether or not they fall inside or outside the current mask (see *r.mask*). When a mask is present, *r.stats* will only report zero values falling within the mask area unless the user runs *r.stats* with the *-m* option. When the user runs *r.stats* with the *-m* option, *r.stats* will report zero values falling outside the mask area, in addition to those within the mask.

-l Prints the category label(s) as well as the category number(s).

-q Run quietly, and suppress printing of percent complete messages to standard output.

-z Report only non-zero data values. Zero data will not be reported. However, for multiple map layers this means that if zero values occur in every map layer, they will not be reported; if non-zero category values occur in any map layer (along with zeros in others), the non-zero values along with the zero values will be reported.

-g Print the grid coordinates (easting and northing), for each cell. This option works only if the *-1* option is also specified.

-x Print the x and y (column and row) values, for each cell. This option works only if the *-1* option is also specified.

Parameters:

input=name The name(s) of one or more existing raster map layer(s) whose cell counts or area statistics are to be calculated.

fs=character or *fs=space* The field separator (fs) to be used to separate data fields in the output file
Options: a character or space Default: a space

output=name The name to be assigned to the ASCII output file.

NON-INTERACTIVE PROGRAM USE
If users invoke program options on the command line, *r.stats* will print out area statistics for the user-specified raster map layers in a columnar format suitable for input to UNIX programs like awk and sed. Output can be saved by specifying the name of an output file on the command line.

If a single map layer is specified on the command line, a list of areas in square meters (assuming the map's coordinate system is in meters) for each category in the raster map layer will be printed. (If the *-c* option is chosen, areas will be stated in number of cells). If multiple raster map layers are specified on the command line, a cross-tabulation table of areas for each combination of categories in the map layers will be printed.

For example, if one raster map layer were specified, the output would look like:

```
1:1350000.00
2:4940000.00
3:8870000.00
```

If three raster map layers a, b, and c, were specified, the output would look like:

```
0:0:0:8027500.00
0:1:0:1152500.00
1:0:0:164227500.00
1:0:1:2177500.00
1:1:0:140092500.00
1:1:1:3355000.00
2:0:0:31277500.00
2:0:1:2490000.00
2:1:0:24207500.00
2:1:1:1752500.00
3:0:0:17140000.00
3:1:0:11270000.00
3:1:1:2500.00
```

Within each grouping, the first field represents the category value of map layer a, the second represents the category values associated with map layer b, the third represents category values for map layer c, and the last field gives the area in square meters for the particular combination of these three map layers' categories. For example, above, combination 3,1,1 covered 2500 square meters. Fields are separated by colons.

NOTES
r.stats works in the current geographic region with the current mask.

If a nicely formatted output is desired, pipe the output into a command which can create columnar output. For example, the command:

r.stats input=a,b,c | pr -3 | cat -s

will create a three-column output

```
1:4:4:10000.00          2:1:5:290000.00         2:4:5:2090000.00
1:4:5:1340000.00        2:2:5:350000.00         3:1:2:450000.00
2:1:1:1090000.00        2:4:1:700000.00         3:1:3:5280000.00
2:1:3:410000.00         2:4:3:10000.00          3:1:5:3140000.00
```

The output from *r.stats* on more than one map layer is sorted.

Note that the user has only the option of printing out cell statistics in terms of cell counts and/or area totals. Users wishing to use different units than are available here should use the GRASS program *r.report*.

SEE ALSO
g.region, r.coinr.describe, r.mask, r.report, and *parser*

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.support

NAME
r.support - Allows the user to create and/or modify raster map layer support files.
(GRASS Raster Program)

SYNOPSIS
r.support

DESCRIPTION
The GRASS program *r.support* allows the user to create and/or modify raster map layer support files. It may be run only on raster map layers in the user's current mapset.

No non-interactive version of this program currently exists; the user runs the program by typing *r.support*, and will be queried for inputs.

Various GRASS programs depend on one or more of the following GRASS support files:

cellhd The cell header file contains information on a map's projection, zone, regional boundaries, row and column totals, cell resolution, storage format, and compression. It describes where and how this map's raster (cell) data fits in with reference to other raster map layers' data. Without it, the raster map layer could not be displayed or analyzed properly. Using *r.support*, the user can change the # of columns, # of bytes per cell, and default geographic region settings. Generally, users would not change this information. Cell header files are stored under the cellhd directory under the user's current mapset.

stats Raster map layer statistics are saved in the form of a histogram and range of the category values which occur in the map layer. Statistics files are stored in subdirectories of the cell_misc directory under the user's current mapset.

cats A category file associates each category value in the raster map layer with a category description (label). The user may add or edit the category descriptions, alter the number of categories, and add or alter the map's title. Category files associated with raster map layers are stored under the cats directory in the user's current mapset.

colr A color file associates each category value in the raster map layer with a color. Using *r.support*, the user may assign one of eight color table types to the raster map layer. Map color table files are stored under the colr and colr2 directories under the user's current mapset.

hist Historical information about the raster map layer is stored in a history file. The user may add or edit the raster map's title, data type, data source, data description, and include comments. (Note that the specification of map data type here is somewhat archaic, and should always be set to raster.) Map history files are stored under the hist directory under the user's current mapset.

NOTES
The *r.support* program attempts to verify that the information in the cell header is reasonable. The data format specified in the header is verified against the raster map layer itself. This includes checking that files which the header indicates are compressed are really compressed, and that the number of rows and columns specified in the header correspond to the actual file size.

The *r.support* program can also be used to determine the number of columns and rows of data in a raster map layer, in the event that no cell header is available. This is useful, for example, for importing raster map layers created by software other than GRASS.

If the file is not compressed, the file size should be the product of the number of rows and columns. If the file is compressed, this test cannot be performed since the file size will bear no relation to the product. The number of rows can still be verified, but the number of columns cannot.

To compute or correct the stats, the cell header must be correct, since the raster map layer is read to determine the stats.

If a new cats or colr (or colr2) file is required, the stats must be correct.

The user is allowed to change the number of categories specified in the category file. This should only be done if the user knows that the maximum category value in the raster map layer is different than that which is recorded in the category file. Changing the category value in the cats file allows the user to add more category labels, or to remove labels. It does NOT change the category values in the raster map layer itself.

The color file is unique among GRASS support files. While it is necessary to protect a user's original data from being modified by users working under other mapsets, these users need to be able to create color tables for maps that are stored under mapsets other than their own. Color table files meet both these objectives.

Color table files get stored in one of two directories, both under the user's current mapset. The color files created by a user for raster maps stored under that user's current mapset get stored in the directory $LOCATION/colr and cannot be modified or removed by other users. The color table files that the user modifies/creates for raster map layers not stored under the user's current mapset get stored in a secondary color file under the user's mapset. This secondary color table is stored under $LOCATION/colr2/<mapset> where <mapset> is the name of the mapset under which the raster map data are stored. In versions of GRASS prior to 3.0, this was also the case for color tables in the user's own mapset. Now, however, if a user modifies a color table associated with a raster map layer in his own current mapset, these changes will be made to the user's original color file (i.e., the user's color changes will overwrite whatever previous color table file existed for this map under the user's $LOCATION/colr directory). No secondary color files are created for raster map layers stored in the user's own mapset.

WARNING
In order to modify the cell header, the raster (cell) map layer under consideration must not be a reclass file. This is because the reclass file's header does not contain positional information, but rather a reference to another raster map layer. Thus it shares a cell header with the referenced raster map layer. In order to change the cell header, *r.support* must be run on the true raster file referenced.

SEE ALSO
For more information regarding the location and function of GRASS support files consult the GRASS Programmer's Manual chapter on GRASS Database Structure

d.colors, r.colors, r.reclass

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.surf.contour

NAME
r.surf.contour - Surface generation program.
(GRASS Raster Program)

SYNOPSIS
r.surf.contour
r.surf.contour help
r.surf.contour [-f] input=name output=name

DESCRIPTION
*r.surf.*contour creates a raster elevation map from a rasterized contour map. Elevation values are determined using procedures similar to a manual methods. To determine the elevation of a point on a contour map, an individual might interpolate its value from those of the two nearest contour lines (uphill and downhill).
*r.surf.*contour works in a similar way. Initially, a vector map of the contour lines is made with the elevation of each line as its label (see *v.digit*). When the program *v.to.rast* is run on the vector map, continuous "lines" of rasters containing the contour line values will be the input for *r.surf.*contour. For each cell in the input map, either the cell is a contour line cell (which is given that value), or a flood fill is generated from that spot until the fill comes to two unique values. The flood fill is not allowed to cross over the rasterized contour lines, thus ensuring that an uphill and downhill contour value will be the two values chosen. r.surf.contour interpolates from the uphill and downhill values by the true distance.
The program will be run non-interactively if the user specifies the program parameter values and desired flag settings on the command line, using the form:

r.surf.contour [-f] input=name output=name

Alternately, the user can simply type r.surf.contour on the command line, without program arguments. In this case, the user will be prompted for needed inputs and option choices using the standard GRASS user interface described in the manual entry for parser.

Flag:
-f Invoke fast, but memory-intensive program operation.

Parameters:
input=name Name of an existing raster map layer that contains a set of initial category values (i.e., some cells contain known category values (denoting contours) while the rest contain zeros (0)).

output=name Name to be assigned to new output raster map layer that represents a smooth (e.g., elevation) surface generated from the known category values in the input raster map layer.

NOTES
r.surf.contour works well under the following circumstances:1) the contour lines extend to the the edge of the current region, 2) the program is run at the same resolution as that of the input map, 3) there are no disjointed contour lines,and 4) no spot elevation data BETWEEN contour lines exist. Spot elevations at the tops of hills and the bottoms of depressions, on the other hand, improve the output greatly. Violating these constraints will cause non-intuitive anomalies to appear in the output map. Run *r.slope.aspect* on *r.surf.contour* results to locate potential anomalies.
The running of *r.surf.contour* is very sensitive to the resolution of rasterized vector map. If multiple contour lines go through the same raster, slight anomalies may occur. The speed of *r.surf.contour* is dependent on how far "apart" the contour lines are from each other (as measured in rasters). Since a flood fill algorithm is used, the program's running time will grow exponentially with the distance between contour lines.

SEE ALSO
r.surf.idw, r.surf.idw2, s.surf.idw, v.digit, v.to.rast, r.slope.aspect, and *parser*

AUTHOR
Chuck Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory

r.surf.idw

NAME
r.surf.idw - Surface interpolation utility for raster map layers.

SYNOPSIS
r.surf.idw [-e] input=name output=name [npoints=value]

DESCRIPTION
r.surf.idw fills a grid cell (raster) matrix with interpolated values generated from a set of input layer data points. It uses a numerical approximation technique based on distance squared weighting of the values of nearest data points. The number of nearest data points used to determined the interpolated value of a cell can be specified by the user (default: 12 nearest data points).

If there is a current working mask, it applies to the output raster file. Only those cells falling within the mask will be assigned interpolated values. The search procedure for the selection of nearest neighboring points will consider all input data, without regard to the mask.

The command line input is as follows:

Flag:

-e Error analysis option that interpolates values only for those cells of the input raster map which have non-zero values and outputs the difference (see NOTES below).

Parameters:

input=name Name of an input raster map layer containing an incomplete set of data values. (i.e., some grid cells contain known data values while the rest contain zero (0)).

output=name Name to be assigned to new output raster map that represents the surface generated from the known data values in the input layer.

npoints=value Number of nearest data points used to determine the interpolated value of an output raster cell.
Default: 12

NOTES
r.surf.idw is a surface generation utility which uses inverse distance squared weighting (as described in Applied Geostatistics by E. H. Isaaks and R. M. Srivastava, Oxford University Press, 1989) to assign interpolated values. The implementation includes a customized data structure somewhat akin to a sparse matrix which enhances the efficiency with which nearest data points are selected. For latitude/longitude projections, distances are calculated from point to point along a geodesic.

Unlike *r.surf.idw2*, which processes all input data points in each interpolation cycle, *r.surf.idw* attempts to minimizethe number of input data for which distances must be calculated. Execution speed is therefore a function of the search effort, and does not increase appreciably with thenumber of input data points.

r.surf.idw will generally outperform *r.surf.idw2* except when the input data layer contains few non-zero data, i.e. when the cost of the search exceeds the cost of the additional distance calculations performed by *r.surf.idw2*. The relative performance of these utilities will depend on the comparative speed of boolean, integer and floating point operations on a particular platform.

Worst case search performance by *r.surf.idw* occurs when theinterpolated cell is located outside of the region in which input data are distributed. It therefore behooves the user to employ a mask when geographic region boundaries include large areas outside the general extent of the input data.

The degree of smoothing produced by the interpolation will increase relative to the number of nearest data points considered. The utility may be used with regularly or irregularly spaced input data. However, the output result for the former may include unacceptable nonconformities in the surface pattern.

The *-e* flag option provides a standard surface-generation error analysis facility. It produces an output raster mapof the difference of interpolated values minus input values for those cells whose input data are non-zero. For each interpolation cycle, the known value of the cell under consideration is ignored, and the remaining input values are used to interpolate a result. The output raster map may be compared to the input raster map to analyze the distribution of interpolation error. This procedure may be helpful inchoosing the number of nearest neighbors considered for surface generation.

SEE ALSO
r.surf.contour, r.surf.idw2, s.surf.idw, and *parser*

AUTHOR
Greg Koerper (Oregon State University) U.S. EPA Environmental Research Laboratory

r.surf.idw2

NAME
r.surf.idw2 - Surface generation program.
(GRASS Raster Program)

SYNOPSIS
r.surf.idw2 input=name output=name [npoints=count]

DESCRIPTION
r.surf.idw2 fills a raster matrix with interpolated values generated from a set of irregularly spaced data points using numerical approximation (weighted averaging) techniques. The interpolated value of a cell is determined by values of nearby data points and the distance of the cell from those input points. In comparison with other methods, numerical approximation allows representation of more complex surfaces (particularly those with anomalous features), restricts the spatial influence of any errors, and generates the interpolated surface from the data points. It is the most appropriate method to apply to most spatial data.

The program will be run non-interactively if the user specifies the values of needed program parameters and any desired optional parameter values on the command line, using the form:

r.surf.idw2 input=name output=name [npoints=count]

Alternately, the user can simply type *r.surf.idw2* on the command line, without program arguments. In this case, the user will be prompted for parameter values using the standard GRASS user interface described in the manual entry for parser.

Parameters:

input=name Name of an input raster map layer that contains a set of irregularly spaced data values; i.e., some cells contain known data values while the rest contain zero (0).

output=name Name to be assigned to the new output raster map layer containing a smooth surface generated from the known data values in the input map layer.

npoints=count The number of points to use for interpolation. The default is to use the 12 nearest points when interpolating the value for a particular cell.
Default: 12

NOTES
The amount of memory used by this program is related to the number of non-zero data values in the input map layer. If the input raster map layer is very dense (i.e., contains many non-zero data points), the program may not be able to get all the memory it needs from the system. The time required to execute increases with the number of input data points.

If the user has a mask set, then interpolation is only done for those cells that fall within the mask. However, all non-zero data points in the input layer are used even if they fall outside the mask.

This program does not work with latitude/longitude data bases. Another surface generation program, named *r.surf.idw*, should be used with latitude/longitude data bases.

The user should refer to the manual entries for *r.surf.idw,* r.surf.contour, and *s.surf.idw* to compare this surface generation program with others available in GRASS.

SEE ALSO
r.mask, r.surf.contour, r.surf.idw, s.surf.idw, and *parser*

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.thin

NAME
r.thin - Thins non-zero cells that denote linear features in a raster map layer.
(GRASS Raster Program)

SYNOPSIS
r.thin
r.thin help
r.thin input=name output=name

DESCRIPTION
r.thin scans the named input raster map layer and thins non-zero cells that denote linear features into linear features having a single cell width.

r.thin will thin only the non-zero cells of the named input raster map layer within the current geographic region settings. The cell width of the thinned output raster map layer will be equal to the cell resolution of the currently set geographic region. All of the thinned linear features will have the width of a single cell.

r.thin will create a new output raster data file containing the thinned linear features. r.thin assumes that linear features are encoded with positive values on a background of 0's in the input raster data file.

Parameters:

input=name Name of a raster map layer containing data to be thinned.

output=name Name of the new raster map layer to hold thinned program output.

The user can run this program either non-interactively or interactively. The program will be run non-interactively if the user specifies program arguments on the command line, using the form:

r.thin input=name output=name

Alternately, the user can simply type:

r.thin

on the command line, without program arguments. In this case, the user will be prompted for needed parameter values using the standard GRASS parser interface described in the manual entry for parser.

NOTES
r.thin only creates raster map layers. You will need to run *r.line* on the resultant raster file to create a vector (*v.digit*) map layer.

r.thin may create small spurs or "dangling lines" during the thinning process. These spurs may be removed (after creating a vector map layer) by *v.trim*.

r.thin creates a 0/1 output map.

This code implements the thinning algorithm described in "Analysis of Thinning Algorithms Using Mathematical Morphology" by Ben-Kwei Jang and Ronlad T. Chin in Transactions on Pattern Analysis and Machine Intelligence, vol. 12, No. 6, June 1990. The definition Jang and Chin give of the thinning process is "successive removal of outer layers of pixels from an object while retaining any pixels whose removal would alter the connectivity or shorten the legs of the sceleton."

The sceleton is finally thinned when the thinning process converges; i.e., "no further pixels can be removed without altering the connectivity or shortening the sceleton legs" (p. 541). The authors prove that the thinning process described always

converges and produces one-pixel thick sceletons. The number of iterations depends on the original thickness of the object. Each iteration peels off the outside pixels from the object. Therefore, if the object is <= n pixels thick, the algorithm should converge in <= iterations.

SEE ALSO
g.region, r.line, v.digit, v.support, v.trim, and *parser*

AUTHOR
Olga Waupotitsch, U. S. Army Construction Engineering Research Laboratory

The code for finding the bounding box as well as input/output code was written by Mike Baba (DBA Systems, 1990) and Jean Ezell (USACERL, 1988).

r.traj.data

NAME
r.traj.data - Reviews the ammunition and weapon data base used by *r.traj*.
(GRASS Raster Program)

SYNOPSIS
r.traj.data
r.traj.data help
r.traj.data [-aw]

DESCRIPTION
r.traj uses ammunition and weapon information that can be displayed with this program. Program flags are listed below.

Flags:

-a Prints list of ammunition types usable by *r.traj*.

-w Prints list of weapons usable by *r.traj*.

SEE ALSO
r.traj, and *parser*

AUTHOR
James Westervelt, Construction Engineering Research Laboratory

r.traj

NAME
r.traj - Ballistic trajectory modeling program.
(GRASS Raster Program)

SYNOPSIS
r.traj
r.traj help
r.traj input=name output=name weapon=name coordinate=x,y elevation=value ammunition=name left.azimuth=name right.azimuth=name

DESCRIPTION
r.traj generates a raster map layer showing cells that can be hit from a firing point by shells from a user-specified weapon. Cells are marked with integer values that represent the muzzle firing angles required to hit them.

Parameters:
input=name Name of the elevation raster map.

output=name Name of new raster map containing results.

weapon=name Type of weapon used for firing. A list of weapon types and their associated attributes are kept in the file $GISBASE/weapon_data/weapons.

coordinate=x,y The coordinates of the firing point (east, north).

elevation=value Maximum weapon muzzle elevation.

ammunition=name Type of ammunition. A list of ammunition types and their associated attributes are kept in the file $GISBASE/weapon_data/ammunition.

left.azimuth=name Far left edge of allowable firing azimuth. The angle will be in the form of:

[NS]0-90[EW]

For example, S60E indicates the angle is 60 degrees East of true South.

right.azimuth=name Far right edge of allowable firing azimuth, stated in same form as left.azimuth.

Category values in the output raster map layer will program results. Category values between -89 and 89 indicate the gun elevation angle in degrees needed to hit that cell. A category value of 90 is assigned to the weapon's firing point. A category value of -90 is assigned to those points unhittable by the weapon.

EXAMPLE
r.traj input=elevation output=name weapon=M1 coordinate=600000.0,4920000.0 elevation=2.5 ammunition=M392 left.azimuth=S30E right.azimuth=S25W

WEAPON AND AMMUNITION TYPES
Weapon types and their attribute types are listed below. These can be listed by running the program *r.traj.data*.

Weapon Type	
M48	19 -9
M1	20 -9
M10166	-5
M10275	-5

Ammunition types and their attributes are listed below. These can be displayed by running the program *r.traj.data*.

Ammo Type			
M392	105	18.60	1458
M392A2	105	18.60	1458
M728	105	18.95	1426
M735	105	23.05	1501
M735A1	105	17.24	1508
M774	105	17.23	1508
M494	105	24.98	21
M456	105	21.78	1173
M416	105	20.68	737
M467	105	20.42	730
M490	105	20.41	1170
M724	105	14.51	1507
M724A1	105	14.51	1539
M737	105	9.44	1539
M393	105	20.41	732

SEE ALSO

See $GISBASE/etc/weapon_data/weapons for a list of weapon types. See, $GISBASE/etc/weapon_data/ ammunition, for a list of ammunition types.

d.rast.arrow, r.los, r.slope.aspect, r.surf.contour, r.surf.idw r.surf.idw2, r.traj.data, range.place, and *parser*

AUTHORS

Chuck Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory
Kewan Q. Khawaja, Intelligent Engineering Systems Laboratory, M.I.T.

r.transect

NAME
r.transect - Outputs raster map layer values lying along user defined transect line(s).
(GRASS Raster Program)

SYNOPSIS
r.transect
r.transect help
r.transect map=name [result=type] [width=value] line=east,north,azimuth,distance[,east, north,azimuth,distance,...]

DESCRIPTION
This program outputs, in ASCII, the values in a raster map which lie along one or more user-defined transect lines. The transects are described by their starting coordinates, azimuth, and distance. The transects may be single-cell wide lines, or multiple-cell wide lines. The output, for each transect, may be the values at each of the cells, or a single aggregate value (e.g., average or median value).

OPTIONS
Parameters:

map=name Name of an existing raster map layer to be queried.

result=type Type of results to be output.
Options: raw, median, average
Default: raw

If raw results are output, each of the category values assigned to cells falling along the transect are output. Median and average results output a single value per transect: average outputs the average category value of all cells along the transect; median outputs the median category value of these cells.

line=east,north,azimuth,distance[,east,north,azimuth,distance,...] A definition of (each) transect line, specified by the geographic coordinates of its starting point (easting, northing), the angle and direction of its travel (azimuth), and its distance (distance).

The azimuth is an angle, in degrees, measured to the east of north. The distance is in map units (meters for a metered database, like UTM).

width=value Profile width, in cells (odd number).
Default: 1

Wider transects can be specified by setting the width to 3, 5, 7, etc. The transects are then formed as rectangles 3, 5, 7, etc., cells wide.

OUTPUT FORMAT
The output from this command is printed to standard output in ASCII. The format of the output varies slightly depending on the type of result requested. The first number printed is the number of cells associated with the transect. For raw output, this number is followed by the individual cell values. For average and median output, this number is followed by a single value (i.e., the average or the median value).

The following examples use the elevation.dem raster map layer in the spearfish sample data set distributed with GRASS 4.0. (To reproduce these examples, first set the geographic region as shown:

g.region rast=elevation.dem

Single-cell transect:

r.transect map=elevation.dem line=593655,4917280,45,100

```
4 1540 1551 1557 1550
```

3-cell wide transect:

r.transect map=elevation.dem line=593655,4917280,45,100 width=3

```
22 1556 1538 1525 1570 1555 1540 1528 1578 1565 1551 1536 1523 1569 1557 1546 1533 1559 1550
1542 1552 1543 1548
```

(Output appears as multiple lines here, but is really one line)

3-cell wide transect average:

r.transect map=elevation.dem line=593655,4917280,45,100 width=3 result=average

```
22 1548.363636
```

3-cell wide transect median:

r.transect map=elevation.dem line=593655,4917280,45,100 width=3 result=median

```
22 1549.000000
```

NOTES
This program is a front-end to the *r.profile* program. It simply converts the azimuth and distance to an ending coordinate and then runs r.profile.

SEE ALSO
r.profile, and *parser*

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

r.volume

NAME
r.volume - Calculates the volume of data "clumps", and (optionally) produces a GRASS site_lists file containing the calculated centroids of these clumps.
(GRASS Raster Program)

SYNOPSIS
r.volume
r.volume help
r.volume [-fq] data=name [clump=name] [site_list=name]

DESCRIPTION
This program computes the cubic volume of data contained in user-defined clumps. *r.volume* outputs:

(1) The category value assigned to each clump formed by the clump map layer. This value is stored under the "Cat Number" column (field 1) in the output table.
(2) The average category value of the cells found in a data map that fall within the boundaries of each clump in a clump map. The table stores this value under the "Average in Clump" column (field 2).
(3) The summed total value of the category values assigned to the cells falling within each of these clumps. This value is output under the "Data Total" column (field 3).
(4) The number of cells from the data map that fall within the boundaries of each clump formed by the clump map layer. This cell count is stored under the "# Cells in Clump" column (field 4) in the output table.
(5,6) The centroid (easting and northing) of each clump. These values are output under the "Centroid Easting" and "Centroid Northing" columns (fields 5 and 6) in the output.
(7) The total "volume" of each clump. For each clump, the volume is calculated by multiplying the area of each cell by its category value, and taking the sum of this value for all cells within the clump. Since, in GRASS, each cell in the data map will have the same cell dimensions (i.e., the same area), this is equivalent to multiplying the area of one cell by the "Data Total" column (field 3). (The area of each cell is equal to the product of its east-west resolution by its north-south resolution. See g.region.)

Results are sent to standard output in the form of a table. If the user sets the *-f* flag, this table will be output in a form suitable for input to such UNIX programs as awk and sed; the table's columns are stored as colon-separated fields. The user can also (optionally) elect to store clump centroids in a GRASS site_lists file. A sample output report is shown below.

r.volume works with the current geographic region definitions and respects the current MASK.

The user can run *r.volume* non-interactively by specifying parameter values on the command line. If the user omits parameter values from the command line, the program will prompt the user for input using the standard interface described in the manual entry for parser.

OPTIONS
Flags:

-f Generate unformatted output. Output is in a form suitable for input to UNIX programs like awk; each column in the output is separated by a colon. Results are sent to standard output.

-q Run quietly, suppressing the printing of debugging messages to standard output.

Parameters:

data Name of an existing raster map layer containing the category values to be summed. The cell resolution (area) of the data map will also be used.

clump Name of an existing raster map layer that defines the boundaries of each clump. Preferably, this map should be the

output of *r.clump*. If the user has imposed a mask, the program uses this mask as the clump map layer if no other clump layer is specified by the user.

site_list The name to be assigned to a new GRASS site_lists file, in which clump centroids can be stored.

EXAMPLE OF REPORT
The following report might be generated by the command:

r.volume d=elevation c=fields.only s=field.centers

```
  Cat     Average      Data         # Cells          Centroid              Total
Number  in clump      Total        in clump   Easting       Northing       Volume
  1      1181.09      75590           64      595500.00      4927700.00     755900000.00
  2      1163.50      69810           60      597100.00      4927700.00     698100000.00
  3      1146.83      34405           30      598300.00      4927700.00     344050000.00
  4      1193.20      366311          307     599400.00      4927300.00     3663110000.00
      .....
      .....
 60      1260.08      351563          279     603100.00      4921000.00     3515630000.00
 61      1213.93      35204           29      603700.00      4921500.00     352040000.00
 62      1207.71      33816           28      604100.00      4921500.00     338160000.00
                                                             Total Volume = 67226740000.00
```

For ease of example, it is assumed that each clump in the fields.only map layer is a field, that cell category values in the elevation map layer represent actual elevation values in meters, and that the data base is a UTM data base (in meters). This means that field #1 (clump #1) contains 64 cells; the average elevation in field #1 is 1181.09 meters. The sum of all of the elevation values assigned to cells within field #1 is 75590 meters. The volume (x*y*z) of space in field #1 is equal to 755900000 cubic meters.

The "Data Total" column is the sum of the cell category values appearing in the elevation map layer, within each field of the fields.only map layer. The Total Volume is the sum multiplied by the e-w resolution times the n-s resolution.

CENTROIDS
The coordinates of the clump centroids are the same as those stored in the GRASS site_lists file (if one was requested). They are guaranteed to fall on a cell of the appropriate category; thus, they are not always the true, mathematical centroids. However, they will always fall at a cell center.

FORMAT OF SITE LIST
For each line of above table the GRASS site_lists file reads:

easting|northing|#cat v=volume a=average t=sum n=count

This can be converted directly to a raster map layer in which each point is assigned to a separate category.

APPLICATIONS
By preprocessing the elevation map layer with *r.mapcalc* and using suitable masking or clump maps, very interesting applications can be done with r.volume. For instance, one can calculate: the volume of rock in a potential quarry; cut/fill volumes for roads; and, water volumes in potential reservoirs. Data layers of other measures of real values can also be used.

NOTES
The output is sent to the terminal screen. The user can capture the output in a file using the UNIX redirection mechanism, as in the following example:

r.volume d=data_map c=clump_map s=site_list > table.out

Output can also be sent directly to the printer, as shown below:

r.volume d=data_map c=clump_map s=site_list | lpr

The user should be aware of what units of measurement the cell e-w and n-s resolution are in, and in what units the data map's cell category values are stated (since these three values will be multiplied together to produce the volume).

This program respects the current mask, and uses this mask as the clump map layer if none is specified by the user.

SEE ALSO
g.region, r.clump, r.mapcalc, r.mask, s.db.rim, s.menu, and *parser*

AUTHOR
Dr. James Hinthorne, Central Washington University GIS Laboratory

r.water.outlet

NAME
r.water.outlet - Watershed basin creation program.
(GRASS Raster Program)

SYNOPSIS
r.water.outlet
r.water.outlet help
r.water.outlet drainage=name basin=name easting=value northing=value

DESCRIPTION
r.water.outlet generates a watershed basin from a drainage direction map (from *r.watershed* or *r.water.aspect*) and a set of coordinates representing the outlet point of watershed.

OPTIONS
drainage
Input map: drainage direction. Indicates the "aspect" for each cell. Multiplying positive values by 45 will give the direction in degrees that the surface runoff will travel from that cell. The value -1 indicates that the cell is a depression area. Other negative values indicate that surface runoff is leaving the boundaries of the current geographic region. The absolute value of these negative cells indicates the direction of flow. This map is generated from either *r.watershed* or *r.water.aspect.*

basin Output map: Values of one (1) indicate the watershed basin. Values of zero are not in the watershed basin.

easting Input value: Easting value of outlet point.

northing Input value: Northing value of outlet point.

NOTES
In the context of this program, a watershed basin is the region upstream of an outlet point. Thus, if the user chooses an outlet point on a hill slope, the resulting map will be a thin silver of land representing the overland slope uphill of the point.

It is usually a good idea for the user to "find" the stream channel of the desired basin. If the user runs *r.water.accum*, *r.water.swale* with a small swale threshold, and *d.where* on the resulting map, the user can pinpoint the exact location of the outlet point with ease.

SEE ALSO
r.watershed, r.water.aspect, r.water.accum, r.water.swale, r.water.basin, d.where

AUTHOR
Charles Ehlschlaeger, U.S. Army Construction EngineeringResearch Laboratory

r.watershed

NAME
r.watershed - Watershed basin analysis program.
(GRASS Raster Program)

SYNOPSIS
r.watershed
r.watershed help
r.watershed [-m4] elevation=name [depression=name] [flow=name] [disturbed.land=name|value] [blocking=name] [threshold=value] [max.slope.length=value] [accumulation=name] [drainage=name] [basin=name] [stream=name] [half.basin=name] [visual=name] [length.slope=name] [slope.steepness=name]

DESCRIPTION
r.watershed generates a set of maps indicating: 1) the location of watershed basins, and 2) the LS and S factors of the Revised Universal Soil Loss Equation (RUSLE).

r.watershed can be run either interactively or non-interactively. If the user types

r.watershed

on the command line without program arguments, the program will prompt the user with a verbose description of the input maps. The interactive version of *r.watershed* can prepare inputs to lumped-parameter hydrologic models. After a verbose interactive session, *r.watershed* will query the user for a number of map layers. Each map layer's values will be tabulated by watershed basin and sent to an output file. This output file is organized to ease data entry into a lumped-parameter hydrologic model program. The non- interactive version of *r.watershed* cannot create this file.

The user can run the program non-interactively, by specifying input map names on the command line. Parameter names may be specified by their full names, or by any initial string that distinguish them from other parameter names. In *r.watershed*'s case, the first two letters of each name sufficiently distinguishes parameter names. For example, the two expressions below are equivalent inputs to *r.watershed*:

r.watershed el=elev.map th=100 st=stream.map ba=basin.map

r.watershed elevation=elev.map threshold=100 stream=stream.map basin=basin.map

OPTIONS
-m Without this flag set, the entire analysis is run in memory maintained by the operating system. This can be limiting, but is relatively fast. Setting the flag causes the program to manage memory on disk which allows larger maps to be processes but is considerably slower.

-4 Allow only horizontal and vertical flow of water. Stream and slope lengths are approximately the same as outputs from default surface flow (allows horizontal, vertical, and diagonal flow of water). This flag will also make the drainage basins look more homogeneous.

elevation Input map: Elevation on which entire analysis is based.

*depressio*n Input map: Map layer of actual depressions in the landscape that are large enough to slow and store surface runoff from a storm event. Any non-zero values indicate depressions.

flow Input map: amount of overland flow per cell. This map indicates the amount of overland flow units that each cell will contribute to the watershed basin model. Overland flow units represent the amount of overland flow each cell contributes to surface flow. If omitted, a value of one (1) is assumed.

disturbed.land Raster map input layer or value containing the percent of disturbed land (i.e., croplands, and construction

sites) where the raster or input value of 17 equals 17%. If no map or value is given, r.watershed assumes no disturbed land. This input is used for the RUSLE calculations.

blocking Input map: terrain that will block overland surface flow. Terrain that will block overland surface flow and restart the slope length for the RUSLE. Any non-zero values indicate blocking terrain.

threshold The minimum size of an exterior watershed basin in cells, or overland flow units.

max.slope.length Input value indicating the maximum length of overland surface flow in meters. If overland flow travels greater than the maximum length, the program assumes the maximum length (it assumes that landscape characteristics not discernible in the digital elevation model exist that maximize the slope length). This input is used for the RUSLE calculations and is a sensitive parameter.

accumulation Output map: number of cells that drain through each cell. The absolute value of each cell in this output map layer is the amount of overland flow that traverses the cell. This value will be the number of upland cells plus one if no overland flow map is given. If the overland flow map is given, the value will be in overland flow units. Negative numbers indicate that those cells possibly have surface runoff from outside of the current geographic region. Thus, any cells with negative values cannot have their surface runoff and sedimentation yields calculated accurately.

drainage Output map: drainage direction. Provides the "aspect" for each cell. Multiplying positive values by 45 will give the direction in degrees that the surface runoff will travel from that cell. The value -1 indicates that the cell is a depression area (defined by the depression input map). Other negative values indicate that surface runoff is leaving the boundaries of the current geographic region. The absolute value of these negative cells indicates the direction of flow.

basin Output map: Unique label for each watershed basin. Each basin will be given a unique positive even integer. Areas along edges may not be large enough to create an exterior watershed basin. 0 values indicate that the cell is not part of a complete watershed basin in the current geographic region.

stream Output map: stream segments. Values correspond to the watershed basin values.

half.basin Output map: each half-basin is given a unique value. Watershed basins are divided into left and right sides. The right-hand side cell of the watershed basin (looking upstream) are given even values corresponding to the watershed basin values. The left-hand side cells of the watershed basin are given odd values which are one less than the value of the watershed basin.

visual Output map: useful for visual display of results. Surface runoff accumulation with the values modified to provide for easy display. All negative accumulation values are changed to zero. All positive values above the basin threshold are given the value of the basin threshold.

length.slope Output map: slope length and steepness (LS) factor. Contains the LS factor for the Revised Universal Soil Loss Equation. Equations taken from Revised Universal Soil Loss Equation for Western Rangelands (see SEE ALSO section). Since the LS factor is a small number, it is multiplied by 100 for the GRASS output map.

slope.steepness Output map: slope steepness (S) factor for RUSLE. Contains the revised S factor for the Universal Soil Loss Equation. Equations taken from article entitled Revised Slope Steepness Factor for the Universal Soil Loss Equation (see SEE ALSO section). Since the S factor is a small number (usually less than one), it is multiplied by 100 for the GRASS output map layer.

NOTES

There are two versions of this program: *ram* and *seg*. Which is run by *r.watershed* depends on whether the *-m* flag is set. *ram* uses virtual memory managed by the operating system to store all the data structures and is faster than *seg*; *seg* uses the GRASS segment library which manages data in disk files. *seg* allows other processes to operate on the same CPU, even when the current geographic region is huge. Due to memory requirements of both programs, it will be quite easy to run out of memory. If *ram* runs out of memory and the resolution size of the current geographic region cannot be increased, either more memory needs to be added to the computer, or the swap space size needs to be increased. If *seg* runs out of memory, additional disk space needs to be freed up for the program to run.

seg uses the A^T least-cost search algorithm to determine the flow of water over the landscape (see SEE ALSO section). The algorithm produces results similar to those obtained when running *r.cost* and *r.drain* on every cell on the map.

In many situations, the elevation data will be too finely detailed for the amount of time or memory available. Running *r.watershed* will require use of a coarser resolution. To make the results more closely resemble the finer terrain data, create a map layer containing the lowest elevation values at the coarser resolution. This is done by: 1) Setting the current geographic region equal to the elevation map layer, and 2) Using the neighborhood command to find the lowest value for an area equal in size to the desired resolution. For example, if the resolution of the elevation data is 30 meters and the resolution of the geographic region for *r.watershed* will be 90 meters: use the minimum function for a 3 by 3 neighborhood. After going to the resolution at which r.watershed will be run, *r.watershed* will be taking values from the neighborhood output map layer that represents the minimum elevation within the region of the coarser cell.

The minimum size of drainage basins is only relevant for those basins that have no basins draining into them (they are called exterior basins). An interior drainage basin has the area that flows into an interior stream segment. Thus, interior drainage basins can be of any size.

The *r.watershed* program does not require the user to have the current geographic region filled with elevation values. Areas without elevation data MUST be masked out using the *r.mask* command. Areas masked out will be treated as if they are off the edge of the region. Masks will reduce the memory necessary to run the program. Masking out unimportant areas can significantly reduce processing time if the watersheds of interest occupies a small percentage of the overall area.

Zero data values will be treated as elevation data (not no_data). If there are zero data along the edges of the current region, that edge will not be able to propagate negative accumulation data to the rest of the map. This might give users a false sense of security about the quality of their data. If there are incomplete data in the elevation map layer, users should mask out those areas.

SEE ALSO

The A^T least-cost search algorithm used by r.watershed is described in Using the A^T Search Algorithm to Develop Hydrologic Modelsfrom Digital Elevation Data, in Proceedings of International Geographic Information Systems Symposium '89. pp, 275-281, (Baltimore, MD, 18-19, March 1989 by, Charles Ehlschlaeger, U.S. Army Construction Engineering, Research Laboratory.

Length slope and steepness (length.slope) factor equations were taken from Revised Universal Soil Loss Equation for Western Rangelands, presented at the U.S.A./Mexico Symposium of Strategies for Classification and Management of Native Vegetation for Food Production In Arid Zones (Tucson, AZ, 12-16 Oct 1987), by M. A. Weltz, K. G. Renard, and J. R. Simanton.

The slope steepness (slope.steepness) factor contains the revised slope steepness factor for the Universal Soil Loss Equation. Equations were taken from article entitled Revised Slope Steepness Factor for the Universal Soil Loss Equation, in Transactions of the ASAE (Vol 30(5), Sept-Oct 1987), by McCool et al.

r.cost, r.drain, r.mask

AUTHOR

Charles Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory

r.watershed4.0

NAME
r.watershed4.0 - Watershed basin analysis program.
(GRASS 4.0 Raster Program)

SYNOPSIS
r.watershed4.0
r.watershed4.0 help
r.watershed4.0 [-m4] elevation=name [depression=name] [flow=name] [disturbed.land=name|value] [blocking=name] [threshold=value] [max.slope.length=value] [accumulation=name] [drainage=name] [basin=name] [stream=name] [half.basin=name] [visual=name] [length.slope=name] [slope.steepness=name] [armsed=name]

DESCRIPTION
This is the GRASS 4.0 version of this program; the GRASS 4.1 version is available as *r.watershed*. Note also that the the armsed sedimentation calculation facility is not available.

r.watershed4.0 generates a set of maps indicating: 1) the location of watershed basins, 2) information to interface with ARMSED, a storm-water runoff and sedimentation yield model, and 3) the LS and S factors of the Revised Universal Soil Loss Equation (RUSLE).

r.watershed4.0 can be run either interactively or non-interactively. If the user types

r.watershed4.0

on the command line without program arguments, the program will prompt the user with a verbose description of the input maps. The interactive version of *r.watershed4.0* can prepare inputs to lumped-parameter hydrologic models. After a verbose interactive session, *r.watershed4.0* will query the user for a number of map layers. Each map layer's values will be tabulated by watershed basin and sent to an output file. This output file is organized to ease data entry into a lumped-parameter hydrologic model program. The non- interactive version of *r.watershed4.0* cannot create this file.

The user can run the program non-interactively, by specifying input map names on the command line. Parameter names may be specified by their full names, or by any initial string that distinguish them from other parameter names. In *r.watershed4.0*'s case, the first two letters of each name sufficiently distinguishes parameter names. For example, the two expressions below are equivalent inputs to *r.watershed4.0*:

r.watershed4.0 el=elev.map th=100 st=stream.map ba=basin.map
r.watershed4.0 elevation=elev.map threshold=100 stream=stream.map basin=basin.map

OPTIONS
-m Without this flag set, the entire analysis is run in memory maintained by the operating system. This can be limiting, but is relatively fast. Setting the flag causes the program to manage memory on disk which allows larger maps to be processes but is considerably slower.

-4 Allow only horizontal and vertical flow of water. Stream and slope lengths are approximately the same as outputs from default surface flow (allows horizontal, vertical, and diagonal flow of water). This flag will also make the drainage basins look more homogeneous.

elevation Input map: Elevation on which entire analysis is based.

depression Input map: Map layer of actual depressions in the landscape that are large enough to slow and store surface runoff from a torm event. Any non-zero values indicate depressions.

flow Input map: amount of overland flow per cell. This map indicates the amount of overland flow units that each cell will contribute to the watershed basin model. Overland flow units represent the amount of overland flow each cell contributes to surface flow. If omitted, a value of one (1) is assumed.

disturbed.land Raster map input layer or value containing the percent of disturbed land (i.e., croplands, and construction sites) where the raster or input value of 17 equals 17%. If no map or value is given, r.watershed4.0 assumes no disturbed land. This input is used for the RUSLE calculations.

blocking Input map: terrain that will block overland surface flow. Terrain that will block overland surface flow and restart the slope length for the RUSLE. Any non-zero values indicate blocking terrain.

threshold The minimum size of an exterior watershed basin in cells, or overland flow units.

max.slope.length Input value indicating the maximum length of overland surface flow in meters. If overland flow travels greater than the maximum length, the program assumes the maximum length (it assumes that landscape characteristics not discernable in the digital elevation model exist that maximize the slope length). This input is used for the RUSLE calculations and is a sensitive parameter.

accumulation Output map: number of cells that drain through each cell. The absolute value of each cell in this output map layer is the amount of overland flow that traverses the cell. This value will be the number of upland cells plus one if no overland flow map is given. If the overland flow map is given, the value will be in overland flow units. Negative numbers indicate that those cells possibly have surface runoff from outside of the current geographic region. Thus, any cells with negative values cannot have their surface runoff and sedimentation yields calculated accurately.

drainage Output map: drainage direction. Provides the "aspect" for each cell. Multiplying positive values by 45 will give the direction in degrees that the surface runoff will travel from that cell. The value -1 indicates that the cell is a depression area (defined by the depression input map). Other negative values indicate that surface runoff is leaving the boundaries of the current geographic region. The absolute value of these negative cells indicates the direction of flow.

basin Output map: Unique label for each watershed basin. Each basin will be given a unique positive even integer. Areas along edges may not be large enough to create an exterior watershed basin. 0 values indicate that the cell is not part of a complete watershed basin in the current geographic region.

stream Output map: stream segments. Values correspond to the watershed basin values.

half.basin Output map: each half-basin is given a unique value. Watershed basins are divided into left and right sides. The right-hand side cell of the watershed basin (looking upstream) are given even values corresponding to the watershed basin values. The left-hand side cells of the watershed basin are given odd values which are one less than the value of the watershed basin.

visual Output map: useful for visual display of results. Surface runoff accumulation with the values modified to provide for easy display. All negative accumulation values are changed to zero. All positive values above the basin threshold are given the value of the basin threshold.

length.slope Output map: slope length and steepness (LS) factor. Contains the LS factor for the Revised Universal Soil Loss Equation. Equations taken from Revised Universal Soil Loss Equation for Western Rangelands (see SEE ALSO section). Since the LS factor is a small number, it is multiplied by 100 for the GRASS output map.

slope.steepness Output map: slope steepness (S) factor for RUSLE. Contains the revised S factor for the Universal Soil Loss Equation. Equations taken from article entitled Revised Slope Steepness Factor for the Universal Soil Loss Equation (see SEE ALSO section). Since the S factor is a small number (usually less than one), it is multiplied by 100 for the GRASS output map layer.

NOTES

There are two versions of this program: *ram* and *seg*. Which is run by *r.watershed4.0* depends on whether the *-m* flag is set. ram uses virtual memory managed by the operating system to store all the data structures and is faster than *seg; seg* uses the GRASS segment library which manages data in disk files. *seg* allows other processes to operate on the same CPU, even

when the current geographic region is huge. Due to memory requirements of both programs, it will be quite easy to run out of memory. If *ram* runs out of memory and the resolution size of the current geographic region cannot be increased, either more memory needs to be added to the computer, or the swap space size needs to be increased. If *seg* runs out of memory, additional disk space needs to be freed up for the program to run.

seg uses the A^T least-cost search algorithm to determine the flow of water over the landscape (see SEE ALSO section). The algorithm produces results similar to those obtained when running *r.cost* and *r.drain* on every cell on the map.

In many situations, the elevation data will be too finely detailed for the amount of time or memory available. Running *r.watershed4.0* will require use of a coarser resolution. To make the results more closely resemble the finer terrain data, create a map layer containing the lowest elevation values at the coarser resolution. This is done by: 1) Setting the current geographic region equal to the elevation map layer, and 2) Using the neighborhood command to find the lowest value for an area equal in size to the desired resolution. For example, if the resolution of the elevation data is 30 meters and the resolution of the geographic region for *r.watershed4.0* will be 90 meters: use the minimum function for a 3 by 3 neighborhood. After going to the resolution at which *r.watershed4.0* will be run, *r.watershed4.0* will be taking values from the neighborhood output map layer that represents the minimum elevation within the region of the coarser cell.

The minimum size of drainage basins is only relevant for those basins that have no basins draining into them (they are called exterior basins). An interior drainage basin has the area that flows into an interior stream segment. Thus, interior drainage basins can be of any size.

The *r.watershed4.0* program does not require the user to have the current geographic region filled with elevation values. Areas without elevation data MUST be masked out using the *r.mask* command. Areas masked out will be treated as if they are off the edge of the region. Masks will reduce the memory necessary to run the program. Masking out unimportant areas can significantly reduce processing time if the watersheds of interest occupies a small percentage of the overall area.

Zero data values will be treated as elevation data (not no_data). If there are zero data along the edges of the current region, that edge will not be able to propagate negative accumulation data to the rest of the map. This might give users a false sense of security about the quality of their data. If there are incomplete data in the elevation map layer, users should mask out those areas.

SEE ALSO

The A^T least-cost search algorithm used by *r.watershed4.0*, is described in Using A^T Search Algorithm to Develop Hydrologic Models from Digital Elevation Data. *in* Proceedings of International Information Systems (IGIS) Symposium, '89, pp, 275-281, (Baltimore, MD, 18-19 1989), by Charles Ehlschlaeger U.S. Army Construction Engineering Research Laboratory.

Length slope and steepness (length.slope) factor equations were taken from Revised Universal Soil Loss Equation for Western Rangelands, presented at the U.S.A./Mexico Symposium of Strategies for Classification and Management of Native Vegetation for Food Production In Arid Zones (Tucson, AZ, 12-16 Oct 1987), by M. A. Weltz, K. G. Renard, and J. R. Simanton.

The slope steepness (slope.steepness) factor contains the revised slope steepness factor for the Universal Soil Loss Equation. Equations were taken fro article entitled Revised Slope Steepness Factor for the Universal Soil Loss Equation, in Transactions of the ASAE (Vol 30(5), Sept-Oct 1987), by McCool et al.

r.cost, r.drain, r.grass.armsed, r.mask

AUTHOR

Charles Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory

r.weight

NAME
r.weight - Raster map overlay program.
(GRASS Raster Program)

SYNOPSIS
r.weight

DESCRIPTION
r.weight is a language driven raster map overlay program. It provides a means for performing geographical analyses using several raster maps. *r.weight* asks the user to weight (assign numeric values to) the raster map categories of interest. The assignment of weighted values requires that the user intimately understand the analysis being undertaken. How important is the slope of the land in comparison with the current land use, or the depth to bedrock? The assignment of values to the land's characteristics allows *r.weight* to mix and compare apples and oranges, such as slopes and land uses, and soil types and vegetation. *r.weight* is a language-driven analysis tool. It responds to worded commands typed at the terminal. Help is always available via the one word command: *help*. Commands available in *r.weight* are listed below.
Note that raster map names appear in parentheses. The use of parentheses is now optional in *r.weight*.

list maps	List available raster maps
list categories (name)	List categories for raster map (name)
list save	List saved analyses
list analysis	Display current analysis request
print analysis	Send current analysis request to printer
choose (name)	Choose raster map (name) for analysis
assign (name)	Interactive way to assign weights for raster map (name)
assign (name) (cat) (val)	Assign weight (val) to category (cat) for raster map (name)
assign (name) (cat) (cat) (val)	Assign weight (val) to categories (cat) (cat) for raster map (name)
save	Save the current analysis
recover	Recover old analysis
add	Request that weights be added (this is the default)
mult	Request that weights be multiplied
execute	Run analysis
erase	Erase the screen
color grey	Set the graphics montior colors to a grey scale (this is the default)
color wave	Set the graphics monitor colors to a color wave.
color ramp	Set the graphics monitor colors to a color ramp.
quit	Leave *r.weight*

A more detailed explanation of a command can be obtained by typing:

help (command)

SUGGESTED APPROACH

In order for *r.weight* to generate raster maps useful for analysis, the user must make a reasonable and defensible request. While much more powerful than *r.combine, r.weight* is also more dangerous. The user provides the necessary value judgements which are registered as weights. Only well-conceived value judgements will result in defensible results. We suggest the following approach to a weighted overlay analysis:

STEP 1: BEFORE RUNNING WEIGHT

a) Define the question to be answered. e.g., "Locate sites suitable for building apartments."

b) Determine what mapped information is useful for answering the question. e.g., geology, soils, land_use, flood_potential.

c) Based on professional judgement, statistical inference, and engineering principals, assign weights to the categories in the chosen raster maps.

STEP 2: CHOOSE CELL MAPS

In *r.weight,* use the command choose to identify up to six raster maps of interest.

STEP 3: ASSIGN WEIGHTS

Using the *r.weight* command assign, assign specific weights to raster map categories.

STEP 4: SAVE ANALYSIS

Use the save command to save a copy of the analysis requested for later use.

STEP 5: RUN ANALYSIS

Use execute to run the analysis.

STEP 6: EVALUATE RESULTS

To modify an existing analysis request, use the recover command to retrieve the analysis and then go to STEP 3.

SEE ALSO

GRASS Tutorial: *r.weight, r.infer, r.combine, r.mapcalc*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

r.weight2

NAME
r.weight2 - Weighted overlay raster map layer analysis program.
(GRASS Raster Program)

SYNOPSIS
r.weight2 [output] [action] [color]
r.weight2 [output=option] [action=option] [color=option]

DESCRIPTION
r.weight2 is the non-interactive version of *r.weight.* Both programs allow the user to assign numeric values (i.e., "weights") to individual category values within raster map layers. These weights are then distributed locationally throughout a raster map layer based on the distribution of the categories with which they are associated. The user can weight the categories of several raster map layers in a data base. Such weighted raster map layers can then be overlain. *r.weight2* will combine weights in the overlain map layers by cell location.

A resulting output raster map layer depicts the combination of map layer weights across a landscape. These values represent a hierarchy of suitability for some user-defined purpose. To obtain a more detailed description, see the manual entry for *r.weight.*

Output raster map must be specified (no default) Action must be either (add or mult) (default: add) Color table for the new raster map (grey | wave | ramp) (default: grey)

Once the *r.weight2* command line is entered, the user will need to enter a raster map layer name and assign numeric values to its categories. Values can be assigned to the categories of up to six raster map layers within r.weight2 in a single analysis. Help is available to the user by typing *r.weight2* help.

EXAMPLE
The following is the format in which data should be entered to *r.weight2*:

Raster_layer1
[Reclass rule 1a]
[Reclass rule 1b]
Raster_layer2
[Reclass rule 2a]
etc.
end

Raster_layer: raster_map OR "raster_map in mapset"
Reclass rule: (example) 1 = 5 OR 20 thru 50 = 10 (must leave spaces between the category, =, and value entries)

Example: (the prompts are shown in bold)

> *r.weight2 sites add wave*
> *soils*

soils> *1 thru 20 = 5*
soils> *21 thru 30 = 10*
soils> *landcover*

landcover> *1 = 2*
landcover> *2 = 4*
landcover> *3 thru 8 = 6*
landcover> *end*

NOTES
The user must be knowledgeable about r.weight to run *r.weight2*. *r.weight2* does not provide the user with a listing of raster map layers or map layer categories. Users unsure about raster map layer names should run the GRASS program *g.list*. To obtain a listing of the categories for a raster map layer run r.report.

The user can create an input file containing the data needed to run *r.weight2*. This file must list the raster map layer and reclass rules in the format shown in the above example. The prompts must not be included in the file. This file can be directed into *r.weight2* at the command line by typing *r.weight2* output action color < input_file

BUGS
When entering data for the reclass rules, if the user does not include spaces between the category, =, and value, the program will assume that the entry is a raster map layer.

SEE ALSO
g.list, r.combine, r.infer, r.report, r.weight

AUTHOR
David Gerdes, U.S. Army Construction Engineering Research Laboratory

r.what

NAME
r.what - Queries raster map layers on their category values and category labels.
(GRASS Raster Program)

SYNOPSIS
r.what
r.what help
r.what [-f] input=name[,name,...]
r.what [-f] input=name[,name,...] [< inputfile]

DESCRIPTION
r.what outputs the category values and (optionally) the category labels associated with user-specified locations on raster input map(s). Locations are specified as geographic x,y coordinate pairs (i.e., pair of eastings and northings); the user can also (optionally) associate a label with each location.

The program will be run non-interactively if the user specifies the program parameter values and (optionally) the flag setting on the command line, using the form:

r.what [-f] input=name[,name,...]

where each input name is the name of a raster map layer whose category values are to be queried, and the (optional) flag -f directs r.what to also output category labels. The user can also redirect a user-created ASCII input file containing a list of geographic coordinate pairs and (optionally) user-named labels, into r.what using the form:

r.what [-f] input=name[,name,...] [< inputfile]

If the user does not redirect an input file containing these coordinates into the program, the program will query the user for point locations and labels.

Alternately, the user can simply type:

r.what

on the command line, without program arguments. In this case, the user will be prompted for the flag setting and parameter values using the standard GRASS parser interface described in the manual entry for parser.

Flag:

-f Also output the category label(s) associated with the cell(s) at the user-specified location(s).

Parameters:

input=name[,name,name,...] The name(s) of one or more existing raster map layers to be queried.

EXAMPLES
The contents of the ASCII inputfile to *r.what* can be typed in at the keyboard, redirected from a file, or piped from another program (like d.where). Each line of the input consists of an easting, a northing, and an optional label, which are separated by spaces. The word end is typed to end input of coordinates to *r.what*. For example:

635342.21 7654321.09 site 1
653324.88 7563412.42 site 2
end

r.what output consists of the input geographic location and label, and, for each user-named raster map layer, the category value, and (if *-f* is specified) the category label associated with the cell(s) at this geographic location. Sample input (in bold) to and output (in plain text) from *r.what* are given below.

r.what input=soils,aspect
635342.21 7654321.09 site 1
653324.88 7563412.42 site 2
end

```
635342.21|7654321.09|site 1|45|21
653324.88|7563412.42|site 2|44|20
```

r.what -f input=soils,aspect
635342.21 7654321.09 site 1
653324.88 7563412.42 site 2
end

```
635342.21|7654321.09|site 1|45|NaC|21|30 degrees NW
653324.88|7563412.42|site 2|44|NdC|20|15 degrees NW
```

NOTES
The maximum number of raster map layers that can be queried at one time is 14.

SEE ALSO
d.sites, d.where, r.cats, r.report, r.stats, sites, parser

AUTHOR
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory