GRASS 4.1 User's Reference Manual

Spring, 1993

United States Army Corps of Engineers
Construction Engineering Research Laboratories
Champaign, Illinois

GRASS is a product of the US Army Corps of Engineers Construction Engineering Research Laboratories (USACERL), in Champaign, IL. It is an integrated set of programs designed to provide digitizing, image processing, map production, and geographic information system capabilities to its users.

Other agencies have also written programs for distribution with GRASS. System development continues to be conducted by the Environmental Laboratory of USACERL.

Individual program authors are cited within.

CONTENTS

1 INTRODUCTION

Background

This reference manual details the use of programs distributed with version 4.1 of the Geographical Resources Analysis Support System (GRASS), a public domain, image processing and geographic information system (GIS) originally developed by researchers in the Environmental Division of the US Army Construction Engineering Research Laboratories (USACERL) in Champaign, Illinois. The GRASS system is used to input, manipulate, analyze, and output geographic data by users in both military and nonmilitary, public and private agencies based in North America, Europe, and other parts of the world.

Although most GRASS development has been conducted at USACERL, system integration, development, testing, distribution, training, and support are performed by numerous publicly and privately operated sites throughout the world.

Reference Manual

The manual explains the capabilities, syntax, and use of current GRASS programs. Chapters contain program entries which are ordered alphabetically by program name. Each entry contains:

* the program name
* a brief description
* a synopsis of files associated with the program
* a full description of the program's capabilities and uses
* a summary of the command-line operations associated with the program
* a list of associated or related programs
* a list of program authors

Each entry may optionally include a description of the program's operation in interactive mode, and a discussion of "bugs," program limitations, or special notes of interest.

Throughout the manual, program input to be explicitly entered by the user is printed in bold text. User-specified input (i.e., variables set by the user) is set in italic type. Boxed text shows information output to the user's terminal screen.

GRASS Categories

The GRASS code has been divided into the following categories according to reliability, testing, and application:

* Main and Alpha
* UNIX Shell Scripts
* Contributed
* Garden
* Related

Descriptions of Main and Alpha, Shell Script, and Contributed categories are presented at the introduction of each appropriate chapter in this manual.

Other software distributed with GRASS but not documented in this manual includes:

(1) Hardware- and software-specific code: SG3d (specific to Silicon Graphics computers) and XGRASS (specific to *Motif*).

(2) Garden and Related programs contained in the *src.garden* and *src.related* directories, which the user must compile separately (see the GRASS *Installation Guide* for compilation instructions): Any original documentation for these programs is compiled on-line with the directories. Garden includes program interfaces that link GRASS 4.1 with other related software. Related includes programs to which GRASS has been interfaced and that program developers have allowed to be distributed with GRASS.

2 MAIN PROGRAMS and PROGRAMS in ALPHA-TESTING

This chapter describes the syntax and use of GRASS 4.1 programs that have undergone (and survived) alpha- and beta-testing (i.e., initial testing by program developers, and subsequent testing by out-of-house users). These programs constitute the bulk of GRASS and are respectively referred to in the documentation as "Main Programs" and "Alpha Programs." Main programs, which receive both alpha- and beta-testing prior to their release and are located in the *src* directory. Alpha programs receive only alpha-testing prior to release and are located in the *src.alpha* directory. For the GRASS 4.1 release, both types of programs compile automatically during general GRASS compilation. The GRASS *Installation Guide* distributed with GRASS 4.1 software explains the compilation procedure.

NAME

d.3d **–** Displays three-dimensional images based on raster map layers.
(GRASS Display Program)

SYNOPSIS

d.3d
d.3d help
d.3d [**–l0a**] **map=***name* **elevation=***name* **[from_coordinate=***x,y,z]*
[to_coordinate=*x,y,z]* **[exaggeration=***value]* **[lines=***value]* **[field=***value]*
[color=*name]* **[box=***name]*

DESCRIPTION

d.3d displays three-dimensional graphic images based on GRASS raster map layers. The user identifies the viewing point, the line of sight, a vertical exaggeration factor, the viewing angle (field of view), the frequency and color of vector grid lines to appear in the display, the map to be displayed in 3-d, and the map whose category values are to be used as elevation values in the 3-dimensional image.

The program will be run non-interactively if the user specifies all needed parameter values and flag settings on the command line. Alternately, the user can simply type **d.3d** on the command line; in this case, the program will prompt the user for parameter values and flag settings using the standard interface described in the manual entry for *parser*.

Three-dimensional images can also be generated through the GRASS *d.display* program. enter short description of d.3d here

COMMAND LINE OPTIONS

-l Display *lines* only, without displaying the raster base *map*.

-0 Show zero elevations.

-a Each grid-cell is rendered using four elevation coordinates: one for each corner. By default, each corner takes the value of the grid-cell to the lower right. With the -a option each corner is calculated to be an average of the four bordering cell elevation values.

Parameters:

map=*name* The raster map layer used to generate the color shown in the 3-d output (i.e., the map whose x and y values will be displayed, using the z values present in the *elevation* map layer.)

elevation=*name* The raster map used to generate the texture in the 3-d image (i.e., the map whose category values will become the elevation values in the 3-d map displayed). This need not be an elevation map layer, although elevation is commonly used.
Default: elevation

from_coordinate=*x,y,z*
Coordinates of the viewing point, given as: *northing,easting,elevation*. The default *from_coordinate* value is appropriate for the spearfish sample data set.
Default: Calculated at run-time just south-west of the south-west corner.

to_coordinate=*x,y,z* Coordinates of center of view, given as: *northing,easting,elevation*. The default *to_coordinate* value is appropriate for the spearfish sample data set.
Default: Calculated at run-time as the center of the image

exaggeration=*value* Vertical exaggeration factor. This value is multiplied by the "elevation" values (category values) in the *elevation* map layer. As the vertical coordinates are exaggerated, the elevation (z) value of the center of view (the *to_coordinate*) should be increased.

Default: 2.0

lines=*value* North-south and east-west trending lines can be drawn on the 3-d image output, to enhance the 3-d effect. The *lines* value specifies the number of rows and columns to skip between lines. 0 means "display no lines". Lines will be drawn in the *color* stated. The user can elect to only display these lines, without displaying the *map*, by setting the -l flag.
Default: 1

field=*value* The field of view (viewing angle) in the image, stated in degrees.
Default: 30

color=*name* Color of vector *lines* drawn on the output.
Options: color, white, red, orange, yellow, green, blue, indigo, violet, gray, black
Default: gray

The resulting three-dimensional image is drawn in the active frame on the graphics monitor. The user should select and erase the full graphics monitor frame before running *d.3d* to prepare the screen for graphics. Refer to the GRASS program *3d.view.sh* for a demonstration of the *d.3d* program.

INTERACTIVE MODE

The interactive mode prompts for the raster map to be rendered, the raster map to be queried for elevation information, and an optionally saved set of saved viewing parameters. The user is then presented with a form in which the options can be modified.

It is suggested that the following procedure be used to efficiently find the correct viewing parameters. Start with a gross grid (by default, a resolution is chosen which limits the total number of rows and columns), display lines only, set line colors to "color" (lines take their color from the map), set the box color to something other than "none", and use the default viewing coordinates. Experiment with the viewing coordinates until the map is positioned in a desireable configuration. When the map is lined up correctly, set the display resolution as low as you like for a final image.

BUGS

Several additions to this program have been suggested. These include the following:
1) Border the image with a visually appealing curtain.
2) Add x,y,z coordinates and scale information to the displayed 3-d image.
3) Provide a graphic-oriented user interface for identifying viewing parameters.
4) Display output more quickly by doing mathematics in integers.
5) Add the option of plotting vector map layers.

SEE ALSO

3d.view.sh, d.display, d.erase, d.frame, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.ask – Prompts the user to select a GRASS data base file from among files displayed in a menu on the graphics monitor.
(GRASS File Management Program)

SYNOPSIS

d.ask help
d.ask element=*name,description* [**prompt="***message*"]

DESCRIPTION

d.ask is designed for shell scripts that need to prompt the user for the name of a data base file in the user's current GRASS mapset search path. After *d.ask* is invoked with needed parameters, the mouse becomes active and a menu containing files of the specified *element* type is displayed on the user's graphics monitor. The user is prompted by the prompt "*message*" to select one of the listed file names with the mouse. Specifically, the query that appears to the user takes the form:

> **Double click on the** *prompt "message"*
> **Double click here to cancel**
>
> (A list of files of the specified *element* type from each of the mapsets
> listed in the user's mapset search path is also displayed.)

After the user responds, the mouse is deactivated and the displayed menu is erased from the screen (leaving any underlying materials on display intact). Three lines are written to standard output (the user's terminal screen):

> **name**=*file_name*
> **mapset**=*mapset_name*
> **fullname**=*file_name@mapset_name*

Parameters:

element=*name,description*
: Name of a GRASS data base element, followed by a one word description of the element. GRASS data base elements are mapset subdirectories; these include: bdlg, cats, cell, cell_misc, cellhd, colr, colr2, dig, dig_ascii, dig_att, dig_cats, dig_plus, dlg, group, hist, icons, noise, paint, site_lists, and windows. The *description* will be used to display an error message to the screen if no files of the named element type exist in the user's mapset search path. The prompt "*message*" will appear in the pop-up menu displayed on the user's graphics monitor. The files listed in the menu will be of the specified *element* type, and exist in mapsets listed in the user's current mapset search path.

prompt="*message*"
: A brief message with which the user will be prompted. If this message contains more than one word, it should be enclosed within double quotes (""). It is a non-selectable message displayed in the menu bar of a pop-up menu. Selectable items in the displayed menu include a cancel option and any of the files in the mapset subdirectory *name* in the user's mapset search path.

EXAMPLE

Given the following input,

> **d.ask element=cell,raster prompt="raster map layer to be used"**

d.ask will prompt the user to select a raster (cell) file from among those listed in the "cell" directories of the mapsets listed in his current mapset search path. A pop-up menu will be displayed on the user's graphics monitor, containing the prompt:

> **Double click on the** *raster map layer to be used*
> **Double click here to cancel**

OUTPUT

If the user selects a file name from the displayed menu, *d.ask* writes three lines to standard output:

name=*file_name*
mapset=*mapset_name*
fullname=*file_name@mapset_name*

The output is in the form of /bin/sh commands to set the variable *name* to the file name specified by the user (of the *element* type requested by *d.ask*), *mapset* to the GRASS mapset in which this file resides, and *fullname* to the name of the file and its mapset.

If the user elects not to select a file but instead chooses the "Double click here to cancel" option, the mouse will be deactivated and no variable assignments will be returned to standard output.

If no files of the specified *element* type are found in the user's current mapset search path, the mouse is activated and the following message is displayed on the user's graphics monitor:

No *element description* **files found**
Click here to continue

NOTES

Parameter values (the *element* name and description, and *prompt* message) cannot be supplied to *d.ask* interactively; they must be supplied on the command line or from a file.

SEE ALSO

g.ask, g.filename, g.findfile, g.gisenv, g.mapsets

BUGS

This program calls other GRASS programs which may generate confusing error messages. Example: an error message from the program d.menu will appear if a nonexistent map element is chosen.

AUTHOR

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

NAME

d.colormode − Allows the user to establish whether a map will be displayed using its own color table or the fixed color table of the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.colormode
d.colormode help
d.colormode mode=*name*

DESCRIPTION

d.colormode establishes what color table will be used to display maps to the graphics monitor.

Parameters:

mode=*name* Options: *fixed* or *float*

Each time a new raster map layer is drawn on the screen, a color table associated with that map is loaded into the graphics display monitor. The command **d.colormode mode**=**fixed** requests that the monitor's color look-up table be *fixed* (i.e., static). The effect is that colors drawn on the screen by graphics calls will not change when subsequent maps are drawn to the screen. When the *fixed* option is used, the colors in the map's color table are mapped to the nearest colors available in the monitor's fixed color table. When the *float* option is used, the map's color table is loaded directly into the monitor's color look-up table.

There are advantages to using each color display mode. The *d.display* and *d.colors* programs allow the user to interactively modify the color tables of maps displayed to the graphics monitor. This is done by allowing the user to directly modify the monitor's color look-up table. Hence, to accommodate this option in *d.display* and *d.colors*, the *d.colormode mode*=*float* option is chosen. However, this option has the sometimes undesirable effect of changing the colors in which other maps are subsequently displayed to the graphics monitor (although it does not actually change the *color tables* of these latter maps). The *float* colormode is therefore best used when the user wishes to interactively change a map's color table, or when the user wishes to display one or more maps having the same color table. The *fixed* option allows any number of maps to be displayed to the screen, where each map uses different colors, but all use the same fixed color look-up table. You cannot toggle a map's color table when running in fixed mode.

This program will be run non-interactively if the user specifies the color *mode* on the command line (e.g., by typing **d.colormode mode**=**fixed** or **d.colormode mode**=**float**). Alternately, if the user simply types **d.colormode** on the command line, the program will prompt the user for the color mode using the standard GRASS parser interface described in the manual entry for *parser*.

NOTES

Some devices support only a fixed color look-up; e.g., ink-jet printers, plotters, and the AT&T 6300 (running a DEB board). The number of color categories capable of being displayed is also device-dependent; if your colors fall outside this range when in *float* mode, they may not be *displayed* in the colors indicated in the map's color table file. However, the color table files themselves will accurately represent the user's changes.
Color table files associated with raster map layers are stored in the user's current mapset under the *colr* and *colr2* directories.

BUGS

It is strongly recommended that the user erase the graphics monitor screen (e.g., by running *d.erase*) immediately after changing the mode between *fixed* and *float.*

SEE ALSO

d.colors, d.colortable, d.display, d.erase, d.rast and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.colors − Allows the user to interactively change the color table of a raster map layer displayed on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.colors
d.colors help
d.colors map=*name*

DESCRIPTION

A color table file associates specific colors with the categories of a raster map layer. The user can change these map category color assignments (i.e., change the map's color table) interactively, by first displaying the raster map to the graphics monitor and then running the program *d.colors*. If you have first set *d.colormode* to "float" before displaying the map to the graphics monitor, any color changes that you subsequently make while in *d.colors* will immediately (and interactively) appear on the graphics display.

While running *d.colors*, the cursor must be inside the window where the execution takes place. When in "float" colormode, the correct colors appear on the graphics monitor only if the cursor is inside the monitor. To see the correct colors, move the cursor from the execution window to inside the graphics monitor, then move back to the window to continue making changes.

If *d.colormode* is "fixed", any color changes made using *d.colors* will not immediately be shown on the graphics display; however, any color changes saved will still alter the map's color table and will appear next time the raster map layer is redisplayed (see *d.colormode* and *d.colortable*). Note that some graphics display monitors or drivers cannot support interactive color change.

The user must first display the relevant raster map layer to the active frame on the graphics monitor (e.g., using *d.rast* or *d.display*) before running *d.colors*. The user can then either enter the name of the raster map layer whose color table is to be changed on the command line (e.g., by typing: **d.colors map=soils**), or type **d.colors** without program arguments. If the user simply types **d.colors** without program arguments on the command line, *d.colors* will ask the user to enter the name of an existing raster map layer using the standard GRASS interface described in the manual entry for *parser*.

In either case, the user is then presented with the *d.colors* command menu, shown below. This menu is the same as the category and color changing portion of the *d.display* menu. The *d.colors* commands are listed beneath the Category Pointer Movement, Color Modification, Replotting Screen, and Quitting sections below. Commands are invoked by typing in the single-key response shown to the left below. (Longer descriptions of these commands appear to the right.) Results from invoking these commands will be reflected in the Category and Category Number sections of the *d.colors* screen. On the *d.colors* screen menu, commands appear in the right half of the screen, and the current status of categories appears in the left half of the screen.

```
CATEGORIES
  0   No Data
  1   (Category 1 description)
  2   (Category 2 description)
  .   ....
  .   ....
CATEGORY NUMBER:
  RED       0   0%
  GREEN     0   0%
  BLUE      0   0%
Shift Incr:  10   3%
CATEGORY POINTER MOVEMENT
  D/d  down (cats)                    Move pointer to next category
```

U/u up (cats)	Move pointer to previous category
COLOR MODIFICATIONS	
R/r RED	Increase/decrease RED intensity
G/g GREEN	Increase/decrease GREEN intensity
B/b BLUE	Increase/decrease BLUE intensity
I/i increment	Increase/decrease increment (of intensity shift)
h highlight	Highlight current color
+/- shift colors	Shift entire color table (up/down)
c save color	Save color table
t toggle table	Toggle to different color table
REPLOTTING SCREEN	
* Replot screen	Replots the screen
QUITTING	
Q quit	Quits program

Changing categories - The keys **d, D, u,** and **U** are used to move to a different category. The lower case letters move up, **u,** and down, **d,** the category list one category at a time. The upper case letters move 10 categories at a time for fast movement. The cursor does wrap between the first and last categories. The current category is noted on the text screen with an arrow, and is indicated on the graphics screen by a box around the current color.

Changing colors - The color associated with the current category can be changed with the **R, r, G, g, B,** and **b** keys. The upper case letters increase the intensities of red **R,** green **G,** and blue **B** for the current category; the lower case letters decrease the intensities of these same colors for the current category. Video devices make all the colors of the spectrum by mixing red, green, and blue. For those accustomed to red, yellow, and blue being the primary colors, this can be confusing. For starters, yellow is made by mixing red and green. The intensities are listed on the text screen in as percentages.

Keys **I** and **i** increase and decrease the percentage change that each keystroke of one of the color keys (*R, r, G, g, B, b*) causes in its respective color. The increase increment is initially set to 10%. Thus, pressing the *R* key would will increase the red component of the current category by 10%.

Highlight - The **h** key toggles between the current category color and the current highlight color. This color is initially black but can be modified as above while in highlight mode. Blinking can be accomplished by repeatedly striking the **h** key. When changing to different categories using the movement keys as described above, while in highlight mode the category colors will be always left showing their actual colors. Only one category is highlighted at any one time.

Saving the current color table - Pressing the **c** key will save the current color table as you have modified it. This table will then be used next time you display or paint this raster map layer.

Color table toggle - Different types of color tables are suitable for different raster map layers. The key **t** flips between the following color tables: red, green, blue color ramp; gray scale; smooth changing color wave; random colors; and the saved color table.

Color table shift - The entire table is shifted up and down using the **+** and **-** keys.

Quitting the *d.colors* program - Pressing the **Q** key will cause you to quit the *d.colors* program. If colors have been modified but not saved, *d.colors* will ask:
Colors changed
Save the changes? (y/n)
The user should type in **y** to save changes, or **n** to not save changes, before quitting the program. If the user types *n*, the program will ask:

Quit anyway? (y/n)

NOTES

To see map color changes reflected on the display monitor as you make them, you must put the monitor in *float* color mode before running *d.colors*. It is also wise to erase the display screen after changing the color mode.

The map whose color table is to be altered with *d.colors* must already be on display in the active display frame on the graphics monitor before *d.colors* is run. This can be done using the command **d.rast map**=*name* (where *name* is a raster map layer whose color table the user wishes to alter).

The user might type the following sequence of commands to interactively change the colors of a raster *soils* map:

> **d.colormode mode=float**
> **d.erase**
> **d.rast map=soils**
> **d.colors**

After the user has saved any color changes made with *d.colors* and exited the program, the user could then redisplay the *soils* map with the new colors by typing:

> **d.colormode mode=fixed**
> **d.erase**
> **d.rast map=soils**

Some color monitors may not support the full range of colors required to display all of the map's categories listed in the map's color table. However, regardless of whether the user can see the color changes he is effecting to a map's color table, any changes to a map's color table made with *d.colors* that are saved will appear in the map's color table.

Some monitors may not support an interactive color change capability.

SEE ALSO

d.colors.boxes, d.colormode, d.colortable, d.display, d.rast, p.colors, r.colors and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.colortable **−** To display the color table associated with a raster map layer.
(GRASS Display Program)

SYNOPSIS

d.colortable
d.colortable help
d.colortable map=*name* [**color**=*name*] [**lines**=*value*] [**cols**=*value*]

DESCRIPTION

The GRASS program *d.colortable* is used to display the color table associated with a raster map layer in the active frame on the graphics monitor. The map *name* should be an available raster map layer in the user's current mapset search path and location.

Parameters:

map=*name* Name of a raster map layer in the user's current mapset search path whose color table is to be displayed.

color=*name* Color of the lines separating the colors of the color table.
Options: red, orange, yellow, green, blue, indigo, violet, gray, white, and black.
Default: *white*

lines=*value* Number of lines to appear in the color table.
Options: 1 - 1000

cols=*value* Number of columns to appear in the color table.
Options: 1 - 1000

If the *value*s of both *lines* and *cols* are not specified by the user, *d.colortable* divides the active frame equally among the number of categories present in the named raster map layer. If one option is specified, the other is automatically set to accommodate all categories. If both are specified, as many categories as possible are displayed.

The user can specify all needed parameters on the command line using the form:

d.colortable map=*name* [**color**=*name*] [**lines**=*value*] [**cols**=*value*]

If the user specifies the name of a map on the command line but does not specify the values of other parameters, parameter default values will be used. Alternately, if the user types simply **d.colortable** on the command line without any program arguments, the program will prompt the user for parameter settings using the standard GRASS parser interface described in the manual entry for *parser*.

EXAMPLE

The user running the command:

d.colortable map=soils color=red lines=1 cols=3

would see the active graphics frame divided into three columns extending the full frame height. The lines dividing the color table associated with the *soils* map would be displayed in red. The user would see, at most, only three of the colors from the *soils* color table displayed in the active frame (because the user requested that this frame be divided into three sections).

NOTES

If the user wishes to display the entire color table associated with a map, the user should either stipulate a number of lines (rows) and columns (cols) sufficient to accommodate the number of categories in the map's color table, or fail to assign values to one or both of *lines* and/or *cols*. If the user runs *d.colortable* using the default number of lines and columns (the full graphics frame), all categories from the map's color table will be displayed. However, if the user requests that the color table associated with a map which has 10 data categories be displayed in a graphics frame with only 3 lines (rows) and 2 columns (a total of six cells), only six of the ten map categories will be displayed.

The user should run the command **d.colormode mode=float** before running *d.colortable*, for accurate display of the map's color table.

The user should run the GRASS program *d.erase* between runs of *d.colortable* to avoid confusion.

SEE ALSO

d.colormode, d.colors, d.display, d.erase, d.rast and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.display – A menu-driven, highly interactive display program for viewing maps and producing final map products
(GRASS Display Program)

SYNOPSIS

d.display

DESCRIPTION

The GRASS program *d.display* is used to display maps on a graphics monitor and prepare final map products for printing. It is advisable to first run *d.erase* before each run of *d.display* to prepare the screen for graphics.

After invoking the command **d.display**, the DISPLAY MAIN MENU appears on the monitor. Options in this and subsequent *d.display* menus are selected through use of the pointing device (mouse). When the mouse cursor is over the desired menu selection, the user simply presses any button on the mouse.

d.display is a macro that allows the user to access a wide range of GRASS display functions in a menu-driven and highly interactive environment. Most of the functions accessible through *d.display* can also be run independently of one another by running the GRASS display (*d.*) commands listed in the SEE ALSO section, below. However, in many cases, these other *d.* commands give the user greater flexibility and make available more options than are available through the *d.display* menus. For example, the *d.display label file* option runs the *d.label* command using defaults. Users running *d.label* directly from the command line have the choice of stipulating specific label background and text colors, as well as text sizes and font types. Users running the *label file* option through *d.display* cannot specify these options; instead, default values are used (see *d.label* for default values used).

The *d.display* tool is best learned by playing with its options. Refer to the individual *d.* and other commands in the SEE ALSO section for a detailed examination of these functions.

NOTES

The user can also modify the current region settings from the DISPLAY MAIN MENU. Using the mouse, the user can choose to *zoom in* on a portion of the displayed raster map layer to create a smaller geographic region. Once the user has defined the corners of this area using the mouse, the smaller area is enlarged and replotted on the screen. The user can similarly choose to *zoom out*. In this case, the user uses the mouse to encompass that portion of the reduced map that the user wishes to make the current geographic region and display in the center of the screen. The larger map is then replotted and displayed. Finally, the user can enter geographic coordinates for the region's corners from the keyboard using the GRASS command *g.region*. This is done by selecting the *type coordinates* option.

The main menu also offers the user the option of hiding the main menu for ten seconds (*hide menu for 10 seconds*).

The *d.display* command must be run in conjunction with a program which converts the GRASS graphics commands generated by *d.display* to the graphics commands that the current device understands. This device will almost always be a graphics monitor. On most systems the user is required to start up this other program separately from (and prior to) *d.display*. In such cases, a locally-defined program, usually "monitor", is used (see your GRASS manager). (For example, on MASSCOMP machines, the *d.mon* program must be running in the foreground on the graphics monitor. On SUN machines, the *d.mon* program can be started from any terminal and run in the background.)

d.display sends program output to the current graphics frame (set, by default, to the full graphics screen unless modified by the user). You can use *d.frame* to create and choose a smaller display frame on the graphics monitor for use by *d.display*.

See the GRASS macro *3d.view.sh* stored under $GISBASE/scripts for an example of how you can create a *d.display*-like macro using a shell script.

SEE ALSO

See *3d.view.sh* or *slide.show.sh* under $GISBASE/scripts for examples of *d.display*-like macros.

d.3d, d.colormode, d.colors, d.frame, d.label, d.legend, d.measure, d.mon, d.rast, d.scale, d.sites, d.vect, d.zoom, g.region, p.menu, and *v.digit*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.erase – Erases the contents of the active display frame on the user's graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.erase
d.erase help
d.erase [**color**=*name*]

DESCRIPTION

d.erase erases the contents of the active graphics frame, and replaces it with the color black (by default) or by whatever color is specified by the user. *d.erase* will not alter the assignment of the active frame.

Parameter:

color=name	Color with which active frame will be erased. Options: red, orange, yellow, green, blue, yellow, indigo, violet, black, white, gray, brown, and magenta Default: black

SEE ALSO

d.frame, d.mon

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.font – Selects the font in which text will be displayed on the user's graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.font
d.font help
d.font font=*name*

DESCRIPTION

d.font allows the user to select use of a specific text font for display of text on the graphics monitor. The GRASS program *show.fonts.sh* is a UNIX Bourne shell macro which names and displays the fonts that can be selected using *d.font*. If the user does not specify a font when using other GRASS programs that display text, the font type *romans* is used by default.

The user can run this program either non-interactively or interactively. If the user specifies a font type name on the command line the program will run non-interactively. Alternately, the user can simply type **d.font** on the command line; in this case, the program will prompt the user for a display text font type using the standard GRASS interface described in the manual entry for *parser*.

Parameter:

font=*name* Name of a font type, from among the font types italicized below.
Default: *romans*
Options: (italized)

cyrilc	- Cyrillic
gothgbt	- Gothic Great Britain triplex
gothgrt	- Gothic German triplex
gothitt	- Gothic Italian triplex
greekc	- Greek complex
greekcs	- Greek complex script
greekp	- Greek plain
greeks	- Greek simplex
italicc	- Italian complex
italiccs	- Italian complex small
italict	- Italian triplex
romanc	- Roman complex
romancs	- Roman complex small
romand	- Roman duplex
romanp	- Roman plain
romans	- Roman simplex
romant	- Roman triplex
scriptc	- Script complex
scripts	- Script simplex

NOTES

The font type *romans* is the fastest font type to display to the graphics monitor.

SEE ALSO

d.INTRO, d.text, d.title, show.fonts.sh, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

d.font uses the public domain version of the Hershey Fonts created by Dr. A.V. Hershey while working at the U.S. National Bureau of Standards.

NAME

d.frame − Manages display frames on the user's graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.frame
d.frame help
d.frame [**−cepsD**] [**frame**=*name*] [**at**=*bottom,top,left,right*]

DESCRIPTION

This program manages display frames on the user's graphics monitor. GRASS display programs at run-time connect with graphics rendering programs. While the display programs are identical on every hardware platform, the graphics rendering programs are (essentially the only GRASS programs) designed for individual hardware devices. These rendering programs are managed with the GRASS program *d.mon*. Graphics are displayed in rectangular frames on whatever graphics monitor the user is currently directing GRASS display output to. These frames are created and managed with this program, **Note that GRASS frame contents are not retained when one frame covers another.** You cannot shuffle frames from top to bottom and then back again. They simply define rectangular areas on the screen where subsequent drawing will occur. *d.frame.*

Flags:

-c Creates a new display frame on the graphics monitor.

-e Removes all existing display frames and reinitializes the entire graphics screen (the full-screen display frame).

-p Prints the name of the active frame, in which GRASS display output will appear.

-s Selects a frame for the display of GRASS graphics. This frame is then known as the "active frame".

-D Prints the status of the user's graphics monitor and active display frame to standard output. Information includes the name and the dimensions of the current frame on the graphics monitor, given in the form *bottom top left right*. This function is useful for debugging output, and for determining display screen coordinates.

Parameters:

frame=*name* The name of the display frame to be created/selected.

at=*bottom,top,left,right*
Where to place the frame (implies -c). Frame coordinates are stated in the form: *bottom,top,left,right*. The lower-left corner of the graphics monitor always is at location 0,0 while the monitor's upper-right corner is always at 100,100.

NOTES

If the user has created multiple display frames that overlap one another, whatever the user displays in the active frame will overwrite those portions of the underlying frame where these frames overlap.

SEE ALSO

d.erase, d.mon

AUTHOR

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

NAME

d.geodesic – Displays a geodesic line, tracing the shortest distance between two geographic points along a great circle, in a longitude/latitude data set.
(GRASS Display Program)

SYNOPSIS

d.geodesic
d.geodesic help
d.geodesic [**coor**=*lon1,lat1,lon2,lat2*] [**lcolor**=*name*] [**tcolor**=*name*]

DESCRIPTION

d.geodesic displays a geodesic line in the active frame on the user's graphics monitor. This line traces the shortest distance between two user-specified points on the curved surface of a longitude/latitude data set. The two coordinate locations named must fall within the boundaries of the user's current geographic region.

COMMAND LINE OPTIONS

This program can be run either interactively or non-interactively. If the user types **d.geodesic** on the command line without other program parameters, the mouse will be activated; the user is asked to use the mouse to indicate the starting and ending points of each geodesic line to be drawn. The default line color (white) and text color (red) will be used.

Alternately, the user can specify the starting and ending coordinates of the geodesic, line color, and text color on the command line, and run the program non-interactively.

Once the user indicates the starting and ending coordinates of the geodesic, the line and its length (in miles) are displayed to the user's graphics monitor.

Parameters:

coor=*lon1,lat1,lon2,lat2*
Starting and ending coordinates, in longitude and latitude values, of the geodesic line to be drawn.

lcolor=*name* Line color in which geodesic will be displayed.
Options: red, orange, yellow, green, blue, magenta, indigo, violet, gray, white, black
Default: white

tcolor=*name* Text color in which the length of the geodesic will be displayed.
Options: red, orange, yellow, green, blue, magenta, indigo, violet, gray, white, black.
Default: red

NOTES

This program works only on GRASS data bases using a longitude/latitude coordinate system.

SEE ALSO

d.rhumbline

AUTHOR

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

NAME

d.graph – Program for generating and displaying simple graphics to the graphics display monitor.
(GRASS Display Program)

SYNOPSIS

d.graph
d.graph help
d.graph [**input**=*name*] [**color**=*name*]

DESCRIPTION

d.graph draws graphics that are described either from standard input (default), or within a file (if an input *file* name is identified on the command line). If graphics commands are entered from standard input, a *ctrl-d* is used to signal the end of input to *d.graph*.

The program can be run interactively or non-interactively. The user can run the program completely non-interactively by specifying the name of a graphics *file* containing *d.graph* graphics commands and the values of all needed parameters on the command line. The user can instead elect to run the program partially interactively, by specifying any/all of the parameters *except* the graphics file=*name* parameter on the command line. In this case, *d.graph* will expect the user to input *d.graph* graphics commands from standard input (i.e., the keyboard) and will (silently) prompt the user for these graphics commands.

Alternately, the user can simply type **d.graph** on the command line, and be prompted for the values of all parameters (the user can still input graphics commands from either an input file or standard input using this form). In this case, the user is presented with the standard GRASS parser interface described in the manual entry for *parser*.

Parameters:

file=*name* Name of file containing graphics commands. This is a UNIX file name; the file must be located in the user's current working directory or be specified by its full path name. If no filename is specified commands are taken from the standard input.

color=*name* Starting color desired for graphics.
Options: red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white, and black.
Default: *white*

The coordinate system used is 0-100 in x and 0-100 in y, regardless of the graphics monitor display frame size and aspect. The (0,0) location is the lower left corner of the active graphics monitor display frame. All values may be floating point.

The graphics language is simple, and uses the following commands:

*comment*
A line of comment which is ignored in the processing.

move *xpos ypos*
The current location is updated to *xpos ypos*. Values are stated as a percent of the active display frame's horizontal (*xpos*) and vertical (*ypos*) size, and may be floating point values. Values are between 0-100. **Note.** A space must separate *xpos* and *ypos*.

draw *xpos ypos*
A line is drawn in the current color from the current location to the new location *xpos ypos*, which then becomes the current location. Values are stated as a percent of the active display frame's horizontal (*xpos*) and vertical (*ypos*) size, and may be floating point values. Values are between 0-100. **Note.** A space must separate *xpos* and *ypos*.

color *color*
Sets the current color to that stated; subsequent graphics will be drawn in the stated color, until the current color is set to a different color. Options are *red, orange, yellow, green, blue, indigo, violet, brown, magenta, gray, white*, and *black*.

size *xper yper*
Subsequent text will be drawn such that the text is *xper* percent of the graphics monitor display frame wide and *yper* percent of the frame high. By default, the text size is set to 1 percent of the active frame's width and 1 percent of the frame's height if unspecified by the user (this may be too small to be seen by the user).
Note. A space must separate *xper* and *yper*.

text *line-of-text*
The stated text is drawn at the current location using the current color, and the new current location is then positioned at the end of the text string.

icon *type size x y*
Draws an icon of types o, x, or + with specified size at location x,y. Note: type o designates a square.

polygon
xpos ypos
xpos ypos
.
.
. The coordinates appearing beneath the word *polygon*, one pair per line, circumscribe a polygon that is to be filled with the current color.

EXAMPLE

For an example use of *d.graph*, examine the contents of the macro command *$GISBASE/bin/grass.logo.sh* located in the GRASS command bin (which draws a GRASS logo by inputting *d.graph* graphics commands that are stored in a shell file). Note that the coordinates in the *grass.logo.sh* macro were taken directly off an image drawn by hand on graph paper.

NOTES

d.graph remembers the last screen location (xpos ypos) to which the user moved, even after the user erases the display frame. If the user runs *d.graph* repeatedly, and wishes to start anew with the default (xpos ypos) screen location, the user should *clear* the display frame between runs of *d.graph*.

LIMITATIONS

There are no automated ways of generating graphic images. It is anticipated that GRASS user sites will write programs to convert output from a resident graphics editor into GRASS *d.graph* format.

SEE ALSO

d.INTRO, d.font, d.mapgraph, d.text, grass.logo.sh, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.grid – Overlays a user-specified grid in the active display frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.grid
d.grid help
d.grid size=*value* **[color=***name]* **[origin=***easting,northing]*

DESCRIPTION

d.grid overlays a grid of user-defined size and color in the active display frame on the graphics monitor. The grid will overlay, not overwrite, the contents of the active display frame.

d.grid can be run non-interactively or interactively. If the user specifies the grid *size* and (optionally) the grid *color* on the command line the program will run non-interactively; if no grid *color* is given the default will be used. Alternately, the user may simply type **d.grid** on the command line; in this case, the program will prompt the user for parameter values using the standard GRASS user interface described in the manual entry for *parser*.

Parameters:

size=*value*	Size of grid to be drawn in the active display frame, in current map coordinate system units. It should be noted that, depending on the scale of the map layer displayed in this frame, the grid may fall outside the user's viewing range. Options: 0-100000
color=*name*	Sets the current grid color to *name*.
origin	Lines of the grid pass through this coordinate. The coordinate need not be within the current frame. Options: red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white, and black. Default: *gray*

NOTES

d.grid will not erase grids already displayed in the active graphics display frame by previous invocations of *d.grid*; multiple invocations of *d.grid* will therefore result in the drawing of multiple grids inside the active graphics frame. (A command like *d.erase*, which erases the entire contents of the active display frame, must be run to erase previously drawn grids from the display frame.)

Currently, the grid size can only be drawn in the units of the map coordinate system used by the current GRASS location.

SEE ALSO

d.display, d.erase, d.frame, d.legend, d.rast, d.scale, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.his − Produces and displays a raster map layer combining hue, intensity, and saturation (his) values from user-specified input raster map layers.
(GRASS Display Program)

SYNOPSIS

d.his
d.his help
d.his h_map=*name* [**i_map**=*name*] [**s_map**=*name*] [**out**=*name*]

DESCRIPTION

his stands for hue, intensity, and saturation. This program produces a raster map layer providing a visually pleasing combination of hue, intensity, and saturation values from two or three user-specified raster map layers.

The human brain automatically interprets the vast amount of visual information available according to basic rules. Color, or *hue,* is used to categorize objects. Shading, or *intensity,* is interpreted as three-dimensional texturing. Finally, the degree of haziness, or *saturation,* is associated with distance or depth. This program allows data from up to three raster map layers to be combined into one new raster map layer which retains the original information in terms of *hue, intensity,* and *saturation.*

OPTIONS

This program can be run non-interactively or interactively. It will run non-interactively if the user specifies on the command line the name of a map containing hue values (*h_map*), and the name(s) of map(s) containing intensity values (*i_map*) and/or saturation values (*s_map*). If the user also names an output map (*out*) on the command line, the combined hue, saturation, and intensity values will be saved as a raster map layer in the user's current mapset; otherwise, the resulting image will only be displayed in the active display frame on the graphics monitor.

Alternately, the user can run the program interactively by typing **d.his** without naming parameter values on the command line. In this case, the program will prompt the user for parameter values using the standard GRASS parser interface described in the manual entry for *parser*.

Parameters:

h_map=*name*	Name of input raster map layer to be used for HUE values.
i_map=*name*	Name of input raster map layer to be used for INTENSITY values.
s_map=*name*	Name of input raster map layer to be used for SATURATION values.
out=*name*	Name of the output raster map layer combining hue, intensity, and saturation values from input layers.

While any raster map layer can be used to represent the hue information, map layers with a few very distinct colors work best. Only raster map layers representing continuously varying data like elevation, aspect, weights, intensities, or amounts can suitably be used to provide intensity and saturation information.

For example, a visually pleasing raster map layer can be made by using a watershed map for the *hue* factor, an aspect map for the *intensity* factor, and an elevation map for *saturation.* (The user may wish to leave out the elevation information for a first try.) Ideally, the resulting image should resemble the view from an aircraft looking at a terrain on a sunny day with a bit of haze in the valleys.

THE PROCESS

Each map cell is processed individually. First, the working color is set to the color of the corresponding cell in the map layer chosen to represent *HUE.* Second, this color is multiplied by the *red* intensity of that cell in the *INTENSITY* map layer. This map layer should have an appropriate gray-scale color table associated with it. You can ensure this by using the color manipulation capabilities of *d.display* or *d.colors*. Finally, the color is made somewhat gray-based on the *red* intensity of that cell in the *SATURATION* map layer. Again, this map layer should have a gray-scale color table associated with it.

NOTES

This program produces an image and (optionally) a raster map layer with 1000 colors (10 intensities each of red, green, and blue). The resulting image and raster map layers will not display properly if the graphics display monitor does not have at least 1000 colors while the user is running in *float* colormode. Thus, unless the display device has 1000 colors, it is necessary to run the GRASS command

d.colormode mode=fixed

before running *d.his*. Otherwise, the colors will be incorrectly displayed on the graphics monitor.

Either (but not both) of the intensity or the saturation map layers may be omitted. This means that it is possible to produce output images that represent combinations of *his, hi,* or *hs.*

SEE ALSO

d.3d, d.colormode, d.colors, d.colortable, d.display hsv.rgb.sh, rgb.hsv.sh and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.histogram **–** Displays a histogram in the form of a pie or bar chart for a user-specified raster file.
(GRASS Display Program)

SYNOPSIS

d.histogram
d.histogram help
d.histogram [**-zq**] **map**=*name* [**color**=*name*] [**style**=*name*]

DESCRIPTION

d.histogram displays the category-value distribution for a user-specified raster map layer, in the form of a bar chart or a pie chart. The display will be displayed in the active display frame on the graphics monitor, using the colors in the raster map layer's color table. The program determines the raster file's category value distribution by counting cells.

OPTIONS

The user can run this program 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) any other desired parameters and flags on the command line, using the form:

d.histogram [**-zq**] **map**=*name* [**color**=*name*] [**style**=*name*]

If a map name is given 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 **d.histogram** 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:

-z Display zero-data information (cells with category value zero) in the histogram. If the -z flag is set, then cells with category value 0 in the named raster map layer will be included in the pie or bar chart. If this flag is not set, then cells with category value 0 will be excluded from the bar chart.

-q Gather the histogram quietly, without printing messages on program progress to the user's terminal.

Parameters:

map=*name* The name of an existing raster map layer in the user's current mapset search path for which a histogram is to be displayed.

color=*name* The name of the color to be used for the axis, text-labels, and tic-marks on the pie or bar chart.
Options: white, red, orange, yellow, green, blue, indigo, magenta, violet, brown, gray, and black.
Default: **color=white**

style=*name* The style of the histogram. If **style=bar**, then *d.histogram* will displays the statistics in the form of a bar chart. If **style=pie**, then *d.histogram* will display the statistics in the form of a pie chart.
Options: *bar* or *pie*
Default: **style=bar**

NOTES

d.histogram uses the current geographic region settings and the current mask (if one exists).

d.histogram uses the colors in the map's color look-up table (i.e., the map's *colr* or *colr2* file). To view results correctly on the display monitor, make sure the color mode is correctly set before running *d.histogram* (i.e., set **d.colormode=float**).

d.histogram does not erase the active frame before displaying output.

SEE ALSO

d.colormode, d.colors, d.colortable, d.display, d.erase, g.region, r.mask, r.stats, and *parser*

AUTHOR

Dave Johnson
DBA Systems, Inc.
10560 Arrowhead Drive
Fairfax, Virginia 22030

NAME

d.icons − Displays points, as icons, at user-defined locations in the active display frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.icons
d.icons help
d.icons [**-r**] **icon**=*name* [**color**=*name*] [**size**=*value*] [**points**=*name*]

DESCRIPTION

d.icons graphically displays point (site) locations as icons in the active frame on the graphics monitor. Geographic coordinates are read either from standard input or from an input file whose name is stated by the user. At each site location, a user-defined icon is displayed.

OPTIONS

The user must enter at least the name of an *icon* file storing a graphic representation of the icon to be displayed, and the geographic coordinates of the points at which they will appear.

The user can run this program either non-interactively or interactively. The program will be run non-interactively if the user specifies the name of an *icon* file and (optionally) any other desired parameters on the command line, using the form:

d.icons [**-r**] **icon**=*name* [**color**=*name*] [**size**=*value*] [**points**=*name*]

If the user fails to specify the name of a *points* file on the command line, the program will prompt the user to enter geographic coordinates from standard input. If the user specifies at least the name of an *icon* file on the command line, any other parameter values left unspecified by the user will be set to their default values (see below).

Alternately, the user can simply type:

d.icons

on the command line, without program arguments. In this case, the user will be prompted for needed parameter values using the standard GRASS user interface described in the manual entry for *parser*. The geographic coordinates at which icons are to be displayed can still be input from an input file (*points*) in interactive use.

Flag:

-r Coordinates are input in reverse order (i.e., as *northing easting*).

Parameters:

icon=*name* The name of an existing file containing a graphic representation of the icon to be drawn. *icon* files can be created by the user using the *p.icons* command and are stored under the *icons* directory under the user's current mapset.

color=*name* Sets the current icon color to the *name* stated.
Options: red, orange, yellow, green, blue, indigo, violet, gray, white, and black
Default: **color=white**

size=*value* The icon scaling factor.
Options: 1-1000
Default: **size=1**

points=*name* The name of a UNIX file containing the geographic coordinates of sites at which icons are to be drawn. Since this is a UNIX file, the user should specify the name using standard UNIX file naming conventions. If this file is not in the user's current working directory, its pathname should be specified. The file contents should consist of a series of geographic coordinates that fall within the boundaries of the current geographic region. Each site location should be stated on a separate line as an easting and northing (in that order) separated by a single blank space. If no *points* file is specified by the user, input is taken from standard input and should be given in the same form; to end standard input, type *end* (or press control-d).

EXAMPLE

An *icons* file contains a graphic representation of the icon to be displayed. Here, spaces represent areas of no color, and x's represent areas of color. For example, the user might construct an icon resembling a cross in the following way:

```
  x
  x
xxxxx
  x
  x
  x
```

This icon might be stored in a file called *cross* (under the user's $LOCATION/icons directory). If the user specified that this be the *icon* file used while running *d.icons*, this cross would then appear in the *color* and *size* specified by the user, at each site location named in the *points* file.

Note: icons are created with the *p.icons* command.

The *points* file lists the geographic coordinates of site locations (at which icons will be displayed). This file should take the form:

```
easting northing
easting northing
easting northing
 ...
```

If the user sets the -r flag, the order of these coordinates should be reversed (i.e., coordinates should be given as *northing easting*).

If the points file is not specified, then the corrdinates are read from the keyboard or across a pipe. This feature allows users to enter the coordinates by hand, or, more usefully, to get then from another program. For example, to display icons at locations specified in a sites list:

```
s.out.ascii name | d.icons icon=cross
```

where *name* is the name of a site list, and *cross* is the name of an icon.

SEE ALSO

d.points, d.sites, d.where, p.icons, s.db.rim, s.out.ascii, v.db.rim, and *parser*

AUTHOR

Contributed by:

Dave Johnson
DBA Systems, Inc.
10560 Arrowhead Drive
Fairfax, Virginia 22030

Modified by:

Michael Shapiro
U.S. Army Construction Engineering Research Laboratory

NAME

d.label **–** Creates and displays text labels in the active display frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.label
d.label help
d.label [**size**=*value*] [**backcolor**=*name*] [**textcolor**=*name*] [**font**=*name*]

DESCRIPTION

d.label allows the user to create and display text labels in the active frame on the graphics monitor. It interactively requests that the user type in text and use the pointing device (mouse) to identify where this text is to be placed within the active graphics frame. The program will prompt the user for label size, background color, text color, and text font type, if the user fails to specify these values on the command line. Program prompts use the standard GRASS parser interface described in the manual entry for *parser*.

Parameters:

size=*value* Sets the label text size to the specified number. Values are stated as a percentage of the frame height; e.g., a size of 10 will make each line of text equal to one-tenth the height of the display frame.
Options: 0-1000
Default: 10

backcolor=*name* Sets the color of the label background to the *name* stated.
Options: red, orange, yellow, green, blue, indigo, violet, gray, white, and black.
Default: *black*

textcolor=*name* Sets the color of the label text to the *name* stated.
Options: same as for *backcolor*.
Default: *white*

font=*name* Sets the font type used for the label's text to *name*.
Options: romand, romanp, romant, romans, scriptc, scripts, romancs, italicc, italiccs, gothitt, gothgrt, and gothgbt. The user can view these available fonts by running *show.fonts.sh*.
Default: *romans* (Roman simplex font type), or whatever font the user has set the font type to before entering *d.label*.

NOTES

This program will allow the user to display labels anywhere in the active display frame, even in areas lying outside of the current geographic region.

d.label will only allow the user to type in one line of text. The fully interactive program *d.labels* can be used to create text labels having as many as four lines.

d.label does not create paint labels files displayable by *d.paint.labels* -- *d.labels* does. These programs should be integrated in future GRASS releases.

BUGS

If the user is running GRASS under X Windows, and the user enters a text label longer than 80 characters (i.e., allows text to wrap onto a second line), X Windows will stop the user's GRASS graphics monitor without throwing the user out of GRASS. If this happens, the user should simply restart a graphics monitor, using *d.mon*.

SEE ALSO

d.display, d.font, d.labels, d.legend, d.mon, d.paint.labels, d.rast, d.scale, d.text, d.title, d.where, show.fonts.sh and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.labels – To create/edit GRASS paint label files for display on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.labels

DESCRIPTION

The *d.labels* program allows the user to interactively create and modify paint-labels files for use with the *p.map* and *d.paint.labels* programs. The *paint labels* files created will be stored in the user's current mapset under the */paint/labels* directory.

d.labels is fully interactive, and requires use of a graphics monitor. After the user has typed

d.labels

and has typed in the name of a new or existing paint-labels file, the user is asked to indicate label positions on the display frame using the mouse-pointer, and to edit label parameters that define the label's size, color, and other characteristics through use of a visual-ask (VASK) data entry screen. The mouse can also be used to indicate the geographic coordinates of locations within the current geographic region in the active frame. It is helpful to display a relevant data file in the active display frame before running *d.labels* to provide visual clues for label placement.

USER PROMPTS

You will first be prompted to enter the name of either an old paint-labels file that you wish to modify, or a new paint-labels file that you wish to create. The following is the file name prompt:

```
OPTION: Label text file name

key: name
required: YES
enter option >
```

If you enter the name of a new (non-existent) paint-labels file, you are asked to indicate a location falling within the current geographic region in the active display frame at which the label is to be placed.

```
Mouse Buttons
 Left:    Place Label Here
 Middle:  Where Am I ?
 Right:   Quit
```

Once you have indicated a position, you will be presented with a data-entry screen (also known as a VASK screen) so that you can enter the paint-label parameters that define how the label will appear on the display or on the color printer. Once you finish entering the label-parameters, and you hit the <ESC> key to proceed, the label will be displayed on the screen. If you like the appearance of the label, enter "y" to the "Look OK?" prompt; if not, enter "n" and you will again be presented with the data entry screen. Once you approve the label's appearance you will be asked to indicate the position of the next label.

If you choose to display an already-existing paint-labels file, you will be shown the labels. You will then have the option of editing a label or adding more labels.

```
Buttons
 Left:    Edit Existing Labels
 Middle:  Quit
 Right:   Add more Labels
```

On-line help is available. If you are puzzled about what should go in a data entry field, just enter "help" in that field, hit the <ESC> key, and you will be presented with a short description of the field and what you may enter.

The following is a list of the different fields in the data entry screen and a short description of each.

TEXT: The user can enter up to four lines of text.

SKIP: yes/no. If *no*, label will be printed. If *yes*, the label will be retained in the file but not printed.

LOCATION: Determines where the text will be located. The user specifies the easting and northing in map units, and (optionally) specifies a vertical and horizontal offset (in printer pixels) from the specified easting/northing. (The vertical (y) offset will shift the location to the south if positive, north if negative. The horizontal (x) offset will shift the location east if positive, west if negative.) These offsets are provided to allow finer placement of labels.

RESET LOCATION: yes/no. If *yes*, the user will be allowed to use the mouse to select a new location for the label. The mouse buttons will function as follows:

Mouse Buttons
Left: Place Label Here
Middle: Where Am I ?
Right: Quit

PLACEMENT: Determines which part of the label to which the location refers. This may be specified as:

lower left	(lower left corner of the text)
lower right	(lower right corner of the text)
lower center	(bottom center of the text)
upper left	(upper left corner of the text)
upper right	(upper right corner of the text)
upper center	(top center of the text)
center	(center of the text)

TEXT SIZE: Determines the size of the lettering used in the label. Text size (height) is stated in units of (ground) meters; e.g., a text size of 500 sets the height of each line of text equal to 500 ground meters. Thus, text will appear to grow or shrink when displayed, depending on the user's current geographic region settings.

TEXT COLOR: This selects the text color. This color can be specified in one of three ways:

1) By color name:
aqua black blue brown cyan gray green grey indigo magenta orange purple red violet white yellow

2) As red, green, blue percentages. For example: .5 .4 .7
(This form is not supported by *d.paint.labels*, but see *p.map* and *p.labels*.)

3) By printer color number, to get the exact printer color.

(This form is not supported by *d.paint.labels*, but see *p.map* and *p.labels*.)

Note. If the user sets the text color equal to the background color, the text will not appear on the graphics monitor.

WIDTH: This determines the line thickness of the letters. The normal text width should be set to 1. Larger numbers can be used to simulate bold face. (*d.paint.labels* ignores this value and always uses 1.)

HIGHLIGHT COLOR: The text can be highlighted in another color so that it appears to be in two colors. The text is drawn first in this color at a wider line width, and then redrawn in the text color at the regular line width. Highlight colors are not shown on the graphics display, they are only shown in the output from *p.labels* and *p.map*.

HIGHLIGHT WIDTH: Specifies how far from the text lines (in pixels) the highlight color should extend. Highlight colors are not shown on the graphics display, they are only shown in the output from *p.labels* and *p.map*.

BACKGROUND COLOR: Text may be boxed in a solid color by specifying a background color (see TEXT COLOR above to specify a color). Specify *none* to use no background.

OPAQUE TO VECTORS: *yes/no*. This field only has meaning if a background color is selected. *yes* will prevent vector lines from entering the background. *no* will allow vector lines to enter the background.

BORDER COLOR: Select a color for the border around the background. Specify *none* to suppress the border.

NOTES

This program is fully interactive and requires no command line arguments.

If the user types the word **help** in a data entry field while entering label information, acceptable field choices will be listed.

Labels must be placed within the current geographic region. If the user attempts to place a label outside of this region, the program will appear to loop endlessly. It is simply failing to display a label positioned beyond the boundaries of the current region; in this event, the user should simply reposition the label.

Labels created by this program are stored in ASCII files under the directory $LOCATION/paint/labels.

SEE ALSO

d.label, d.paint.labels, d.rast, d.vect, d.where, p.labels, p.map

AUTHOR

Dave Johnson
DBA Systems, Inc.
10560 Arrowhead Drive
Fairfax, Virginia 22030

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

d.legend **–** Displays a legend for a raster map layer in the active frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.legend
d.legend help
d.legend map=*name* [**color**=*name*] [**lines**=*value*]

DESCRIPTION

d.legend displays a legend for a user-specified raster map layer in the active frame on the graphics monitor. The legend's size is based on the height of the active frame. The user should therefore take care to create a display frame of suitable dimensions before running *d.legend*. *d.legend* will only obscure those portions of the active frame that directly underlie the legend.

The user can run *d.legend* either non-interactively or interactively. If the user specifies the name of a raster *map* layer on the command line, the program will run non-interactively. Default legend text *color* and number of *lines* will be used unless the user specifies other values on the command line.

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

Parameters:

map=*name* Name of a raster map layer whose legend is to be displayed in the active display frame.

color=*name* Sets the legend text color to the *name* stated.
Options: red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white, and black.
Default: *white*

lines=*value* Number of lines to appear in the map legend.
The number of lines refers to the maximum number of lines of type that can be displayed given the height of the active display frame. If unspecified by the user, the program will divide the display frame into the number of lines required to display all of the category labels and colors associated with the named *map*. To decrease the size of the displayed text, increase the number of lines.
Options: 1 - 1000
Default: Set *lines* value equal to the number of *map* categories

NOTES

The legend text size is based on the number of *lines* requested (or, by default, on the number of lines needed to display the legend). If the user attempts to display a very long legend in a relatively short display frame, the legend may appear in unreadably small text.

SEE ALSO

d.colormode, d.colors, d.colortable, d.display, d.erase, d.font, d.frame, d.grid, d.label, d.labels, d.rast, d.scale, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.mapgraph – Generates and displays simple graphics on map layers drawn in the active graphics monitor display frame.
(GRASS Display Program)

SYNOPSIS

d.mapgraph
d.mapgraph help
d.mapgraph [**input**=*name*] [**color**=*name*] [**vsize**=*value*] [**hsize**=*value*]

DESCRIPTION

d.mapgraph draws graphics that are described in standard input (default) or the UNIX input file *name*. If commands are entered via standard input, a *ctrl-d* is used to signal the end of input to *d.mapgraph*. This program performs essentially the same function as *d.graph*; however, point locations are specified to *d.mapgraph* in the geographic coordinate system of the user's current mapset and location (i.e., in map coordinates), rather than in graphics display screen coordinates.

The program can be run interactively or non-interactively. The user can run the program completely non-interactively by specifying the name of a file containing *d.mapgraph* graphics commands and the values of all needed parameters on the command line. The user can instead elect to run the program partially interactively, by specifying any/all of the parameters *except* the graphics input=*name* parameter on the command line. In this case, *d.mapgraph* will expect the user to input *d.mapgraph* graphics commands from standard input (i.e., the keyboard) and will (silently) prompt the user for these graphics commands. Alternately, the user can simply type **d.mapgraph** on the command line, and be prompted for the values of all parameters (the user can still input graphics commands from an input file using this form.) In this case, the user is presented with the standard GRASS parser interface described in the manual entry for *parser*.

Parameters:

input=*name* Name of a UNIX file containing graphics instructions. Specify the full path name of the file if not in the current directory. If no file name is given, commands are taken from standard input.

color=*name* Starting color desired for graphics.
Options: red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white, and black.
Default: *white*

d.mapgraph is used for drawing simple graphics on top of map layers. The coordinate system used by *d.mapgraph* is the same as that of the map layer displayed in the active display frame on the graphics monitor (or that of the user's current region, if no map is displayed).

The graphics language is simple and uses the following commands:

*comment*

A line of comment which is ignored in the processing.

move *xpos ypos*

The current location is updated to *xpos ypos* (where these, respectively, are the easting and northing of geographic coordinates stated in the map coordinate system of the user's current GRASS location, falling within the current region and active frame). If unspecified by the user, the current location becomes (0,0). If, as most likely, the point (0,0) falls outside of the user's current region, graphics drawn there will not appear in the graphics frame.
Note: use *g.region* to obtain the coordinates of current location. Use *d.where* to obtain specific map coordinates of various points on the raster map displayed in the active frame.
Note: there must be a space between *xpos* and *ypos*.

draw *xpos ypos*

A line is drawn in the current color from the current location to the new location *xpos ypos*, which then becomes the current location. *xpos* and *ypos* are (respectively) an easting and northing stated in the map coordinate system of the user's current GRASS location, and located within the user's

current geographic region and active frame.
Note: there must be a space between *xpos* and *ypos*.

color *color*
: Sets the current color to that stated. Color options are: *red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white*, and *black*.

size *xper yper*
: Subsequent text will be drawn such that the text is *xper* percent of the display frame's width and *yper* percent of the display frame height. If not specified by the user, the text size becomes 5 percent of the active frame's width and 5 percent of the frame's height. This is equivalent to entering **size 5 5**.

text *line-of-text*
: The stated text is drawn at the current location using the current color and the current size.

icon *type size x y*
: Draws an icon of types o, x, or + with specified size at location x,y. Note: type o designates a square.

polygon

xpos ypos
xpos ypos
.
.
.

The map coordinates appearing on lines beneath the word *polygon*, one pair per line, circumscribe a polygon which is to be filled with the current color.

NOTES

d.mapgraph is identical to the *d.graph* command, except for the difference in coordinate systems used.
d.mapgraph will complain if the user enters something to standard input that it does not understand. Blank lines in the input file will result in this error message.

SEE ALSO

d.INTRO, d.frame, d.graph, d.rast, d.zoom, g.region, grass.logo.sh, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.measure – Measures the lengths and areas of features drawn by the user in the active display frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.measure
d.measure help
d.measure [**c1**=*name*] [**c2**=*name*]

DESCRIPTION

d.measure provides the user with an interactive way to measure the lengths and areas of lines and polygons drawn by the user in the active frame on the graphics monitor. Lines and polygons are drawn using a pointing device (mouse). Each line segment is drawn in colors *c1* and *c2*. A mouse button menu indicates that the user can find out the geographic coordinates of the cursor location, draw line segments between user-specified vertices, and quit *d.measure*. Lines and polygons are defined by the series of vertices marked by the user. If more than two successive vertices are drawn, *d.measure* prints the area encompassed (*d.measure* will assume the area is closed even if the user has not joined the first and last vertices). Line lengths are stated in the same units as those of the current LOCATION. Areas are stated in hectares, square miles, and square meters.

Lines and polygons drawn using *d.measure* will overlay (not overwrite) whatever display currently appears in the active frame on the graphics monitor. The user can, for example, run *d.rast* or *d.vect* prior to running *d.measure*, and use raster and/or vector maps as a backdrop.

OPTIONS

The user can specify the colors in which line segments will be drawn by setting the values of *c1* and *c2* on the command line. Default line colors (see below) will be used if the user does not specify the values of *c1* and *c2* on the command line.

Parameters:

c1=*name* The first color in which each line segment is drawn, while being positioned.
Options: red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white, and black
Default: *white*

c2=*name* The second color in which each line segment is drawn, after its vertices are fixed.
Options: Same as *c1*.
Default: *white*

NOTES

d.measure uses all segments drawn by the user when computing area. If the user draws an area within another area, the combined area of both regions will be output.

SEE ALSO

d.display, d.frame, d.graph, d.mapgraph, d.rast, d.vect, d.where

AUTHOR

Jim Westervelt, U.S. Army Construction Engineering Research Laboratory
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory, 4.1 Release

NAME

d.menu – Creates and displays a menu within the active frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.menu
d.menu help
d.menu [**bcolor**=*name*] [**tcolor**=*name*] [**dcolor**=*name*] [**size**=*value*]

DESCRIPTION

d.menu allows the user to create a menu containing a title and options, and to display this menu in the active frame on the graphics monitor. After the menu is displayed in the active frame, the pointing device (mouse) is activated and must be used to select one of the menu options. The number associated with the selected menu option is then printed to standard output and the program exits. This program provides GRASS macro writers with a mouse interface for users.

Parameters can be stated on the command line, from within standard input, or (as illustrated in EXAMPLE) from within a script file.

The user can specify the menu's background, text, and line colors (*bcolor*, *tcolor*, and *dcolor*) and the menu size (*size*) on the command line. If the user sets at least one of these values on the command line, any remaining values that are not specified on the command line will be set (automatically) to their default values (see below). Alternately, the user can simply type **d.menu** on the command line; in this case, the program will prompt the user for the menu colors and text size using the standard GRASS parser interface described in the manual entry for *parser*. The user will then be (silently) prompted to enter the menu's location and contents through standard input (see Menu Information, below).

Parameters:

bcolor=*name* Sets the color of the menu background (the rectangle on which the text sits).
Options: red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white, and black.
Default: *black*

tcolor=*name* Sets the color of the menu text.
Options: Same as those for *bcolor*.
Default: *white*

dcolor=*name* Sets the line color used to divide text lines in the menu.
Options: Same as those for *bcolor*.
Default: *white*

size=*value* Sets the menu's text size to the specified value. Sizes are stated as percentages of the height of the active graphics display frame.
Options: 1 - 100
Default: 3 (i.e., 3%)

Menu Information:

After the user has (optionally) specified menu colors and size, the program expects the user to enter information about the menu's location and content. The menu will be placed in the lower left corner of the active display frame by default if the user does not position it elsewhere using the .T or .L commands.
The user specifies the menu contents by typing in a menu title followed by the option choices to appear in the menu when displayed. The user *must* enter a menu title and at least one menu option.

.T *value* Specifies the menu's distance from the active display frame's top edge (as a percentage of the active frame's height).
Note: Not required

.L *value* — Specifies the menu's distance from the active display frame's left edge (as a percentage of the active frame's width).
Note: Not required

menu title — A title that describes the type of options listed in the menu, and that will appear at the top of the menu when it is displayed.

option name(s) — The options that will appear in the menu when displayed. Each menu option should appear on a separate line. The user may enter as many options as desired, but must enter at least one menu option.
Note: The user should choose a menu size and location that will allow all menu options to be displayed in the active frame.

If the user enters the menu title and option(s) from standard input (i.e., at the keyboard rather than from a file), the user should enter *control-d* to end input and display the menu in the active frame on the graphics monitor. (Note: The *d.menu* program can also be incorporated into UNIX Bourne shell script macros. The below example shows how this might be done.)

EXAMPLE

In the following example, the shell script *menu2* calls the shell script *color.select* which contains *d.menu* commands to display a menu in the current frame on the graphics monitor. After the user selects an option from the display menu, the selection number is available for use by *menu2*.

Contents of file *menu2*:

```
#! /bin/csh -f
set option = 0
set colors = (red green blue black white )
@ option = `color.select`

if ($option <= 5) then
        set color = $colors[$option]
        echo $color

endif

exit
```

Contents of file *color.select*:

```
#! /bin/csh -f
d.menu bcolor=red tcolor=green dcolor=yellow size=5 << EOF
.T 25
.L 25
Color Choices
Option 1
Option 2
Option 3
Option 4
Option 5
EOF
```

If the user runs *menu2*, a menu will be displayed on the graphics monitor that has red background, green text, with menu options divided by yellow lines, and a text size of 5% of the active display frame height. The mouse cursor will become active, allowing the user to select (by pointing with the mouse) one of the displayed menu options. Here, these menu options are called *Option 1*, *Option 2*, and *Option 3*, etc. The first line of text (here, the words *Color Choices*) contains the title of the menu; this line is **not** a menu option that can be chosen by the user with the mouse. When the user presses one of the mouse buttons while pointing to the desired menu choice, the number of the option chosen will be available for capture by the shell script *menu2*. *menu2* is a simple example that takes this information and only echoes it to the screen.

NOTES

Although the user can vary text size, all text within the same menu is displayed in a single text size (and font). If the user specifies that items included in the menu's text be displayed in different sizes, all text will be displayed in the size stated last.

SEE ALSO

d.display, d.font, d.frame, d.grid d.label, d.legend d.paint.labels, d.text, d.title, show.fonts.sh and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.mon – To establish and control use of a graphics display monitor.
(GRASS Display Program)

SYNOPSIS

d.mon
d.mon help
d.mon [**-lLprs**] [**start**=*name*] [**stop**=*name*] [**select**=*name*] [**unlock**=*name*]

DESCRIPTION

The GRASS program *d.mon* allows the user to start, select, list, query the status of, release control of, stop, and unlock control of, available graphics monitors. The user can run this program either interactively (through a series of menus), or non-interactively by typing the name of the monitor to start, stop, select, or unlock, and any desired flags on the command line.

Flags:

-l List all monitors

-L List all monitors (with current status)

-p Print name of currently selected monitor

-r Release currently selected monitor

-s Do not automatically select when starting

Parameters:

start=*name* Name of graphics monitor to start

stop=*name* Name of graphics monitor to stop

select=*name* Name of graphics monitor to select

unlock=*name* Name of graphics monitor to unlock

When the user enters the command **d.mon** without specifying parameter values,
the below menu appears on the screen:

MONITOR MENU
Making sure that the graphics monitor is running:

1 - Start a graphics monitor
(also automatically selects this monitor)
2 - Stop a graphics monitor

Choosing a graphics monitor for your graphics:

3 - Select a graphics device for output
(currently selected monitor: *name*)
4 - Release control of the graphics driver
(let someone else use it)
(option 4 appears only after selection of a monitor)

RETURN quit

These parameters perform the following functions:

1 **Start a Monitor.**
In order to display on-screen GRASS graphics, the user must *start* and *select* a graphics monitor. By default, the *start* command actually runs two commands, to both start and select whatever monitor is named by the user. (The user can get a list of available monitors by setting the -l or -L flag on the command line.)

When a monitor is *started*, it is therefore also (automatically) *selected* for output, unless the *-s* flag is set by the user; the user can also explictly *select* a monitor that has been started (see (3) below). After a monitor is started, a blank graphics frame should appear on whatever terminal the user is using to display graphics.

The desired monitor should be started once and need not be restarted unless it is stopped (option *2*) for some reason. A monitor may continue to run for any length of time, even when no GRASS session is being run. The monitor program runs in the background.

2 **Stop a Monitor.**
Sometimes the monitor program needs to be stopped (terminated). Choosing option 2 will terminate a user-specified monitor program.

A graphics monitor has two different types of status: monitor program *not running*, and monitor *running*. A monitor that has been started and/or selected will be listed as running; a monitor that has been stopped (or not started) will be listed as not running. The -L (list status) flag will list the status of each monitor connected to the system.

3 **Select a Monitor.**
When the user *starts* a monitor, it is also (automatically) *selected* for graphics output unless the user sets the *-s* flag. In order to use (direct graphics output to) a monitor, the user must *select* that monitor for use, either by simply starting the monitor without the -s flag or by explictly selecting the monitor for output using option 3. Only running monitors can be selected for graphics output. Once the user has *selected* a monitor for output, no other user can use this monitor for graphics output until the monitor driver is either *released* (by the user) or *unlocked* (by any user on the system).
The user can run multiple graphics monitors by simply starting each of the graphics monitors drivers he wishes to direct output to.

4 **Release a Monitor.**
Once a user has *selected* a monitor for graphics output, it is locked for use by that user until either: (1) the user voluntarily releases control of the monitor for use by another (option 4), or (2) another GRASS user *unlocks* the user's control of the monitor. Menu option 4 appears only to the person who has selected the monitor (since only that user can release control of his selected graphics monitor.) If another user wishes to *unlock* the user's control of the monitor, that user must run *d.mon* from the command line and set the unlock=*name* parameter.

You may choose multiple options within the *d.mon* program. To leave (exit) the *d.mon* menu, press the <RETURN> key.

NOTES

The *d.mon* program can regulate control of graphics monitors both in systems using multiple monitors and in systems using a single graphics monitor.

SEE ALSO

d.erase, d.frame, monitorcap

AUTHOR

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

NAME

d.paint.labels − Displays text labels formatted for use with GRASS paint (*p.labels*, *p.map*) output to the active frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.paint.labels
d.paint.labels help
d.paint.labels file=*name*

DESCRIPTION

d.paint.labels displays the *paint* label file *name* in the active display frame on the graphics monitor. This file can be generated by the *labels* option in the *p.labels* program, the *d.labels* program, or simply created by the user as an ASCII file (using a text editor like *vi*) and placed in the appropriate */paint/labels* directory under the user's current mapset and location (i.e., under $LOCATION/paint/labels).

OPTIONS

Parameter:

file=*name* Name of ASCII file containing paint labels.

This program can be run either non-interactively or interactively. If the user specifies the name of a paint labels file on the command line, the program will run without interacting with the user. Alternately, the user can type simply **d.paint.labels** on the command line; in this case, the program will prompt the user for the name of a paint labels file using the standard GRASS parser interface described in the manual entry for *parser*.

NOTES

Some of the information stored in the label file is unused by *d.paint.labels*. The unused information found in a label file is printed to standard output. This extra information is used by such programs as *p.labels* and *p.map*.

SEE ALSO

d.display, d.font, d.label, d.labels, d.text, d.title, p.labels, p.map and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.points – Displays point graphics in the active frame on the graphics display monitor.
(GRASS Display Program)

SYNOPSIS

d.points
d.points help
d.points [**color**=*name*] [**size**=*value*] [**type**=*name*] [**file**=*name*]

DESCRIPTION

d.points displays points whose coordinate locations are specified by the user and located within the current geographic region in the active display frame on the graphics monitor. Point coordinates are read either from standard input or from a file stated by the file=*name* option. When entering coordinates through standard input, the user presses *control-d* to signal the end of input to *d.points*.

The program can be run interactively or non-interactively. The user can run the program completely non-interactively by specifying the name of a points *file* (containing the geographic coordinates of icons to be sited) and the values of all needed parameters on the command line. If the name of an input *file* to *d.points* is given on the command line, any other parameter values left unspecified on the command line (e.g., the icon *color*, *size*, and/or *type*) are set to their default values.

The user can instead elect to run the program partially interactively, by specifying any/all of the parameters *except* the points file=*name* parameter on the command line. In this case, *d.points* will expect the user to enter icon locations from standard input (i.e., at the keyboard) and will prompt the user for these locations. Each icon location should be entered as an easting and northing pair separated by a space, followed by a carriage return. Coordinates should be stated in whatever map coordinate system is used by the user's current GRASS location. The user presses *control-d* to signal the end of standard input to *d.points*.

Alternately, the user can simply type **d.points** on the command line, and be prompted for the values of all parameters (the user can still input icon coordinates from an input file using this form). In this case, the user is presented with the standard GRASS parser interface described in the manual entry for *parser*.

Parameters:

color=*name* Sets the current color to the *name* stated.
Options: red, orange, yellow, green, blue, indigo, violet, gray, white, and black
Default: *gray*

size=*value* Size of icon in pixels.
Options: 0-1000 (pixels)
Default: 5 (pixels)

type=*name* Sets the shape (type) of the icon to be drawn to *name*.
Options: x, diamond, box, +
Default: +

file=*name* Take input from the UNIX file *name*. (Note: This file must be in the user's current directory or given by its full path name.)
Default: Standard input.
The file contents should consist of a series of geographic coordinates that fit within the boundaries of the current geographic region falling within the active display frame (refer to *d.where* or *g.region* for these coordinates). Each icon location should be specified by an easting and northing (in that order) separated by a single blank space, and followed by a carriage return. Eastings and northings should be stated in whatever map coordinate system is being used by the user's current GRASS location.

NOTES

d.points will display a point at the location specified in the active frame, even if the point lies outside the boundaries of current geographic region.

SEE ALSO

d.frame, d.where, d.sites, g.region, s.out.ascii, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.profile – Displays profiles of a user-specified raster map layer.
(GRASS Display Program)

SYNOPSIS

d.profile

DESCRIPTION

This command works only interactively. It clears the entire graphics screen and provides a graphical interaction allowing the selection of transects for which profiles are then generated.

USER PROMPTS

First, you will be presented with a prompt asking you to choose a raster map layer to be profiled. Once you specify a valid raster map layer name, the map layer will be displayed in the left half of the graphics display, and the right half of the dispay will be divided into four display frames. There will also be two frames along the top of the display: a mouse-button menu frame on the left, and a status frame on the right.

The mouse-button menu initially offers you three options:

```
Mouse Buttons:
  Left:      Where am I?
  Middle:    Mark FIRST Point of Profile Line.
  Right:     QUIT this.
```

You may query the displayed raster map layer by indicating points with the left mouse-button. The coordinates and category value of each point that you indicate will be displayed on in the status frame. If you mark the first point of the profile line you will be presented with the following mouse-button menu:

```
Mouse Buttons:
  Left:      Where am I?
  Middle:    Mark SECOND Point of Profile Line.
  Right:     QUIT this.
```

Once you mark the second point of the profile line, the profile line will be labeled (with a letter from A to D) and displayed in one of the four display frames on the right hand side of the screen. You will then be presented with a third mouse-button menu:

```
Mouse Buttons:
  Left:      DO ANOTHER
  Middle:    CLEAR DISPLAY
  Right:     QUIT this.
```

If you would like to view another profile, click on the left mouse-button. If you would like to redisplay the raster map layer and clear out the four profile frames, click on the middle mouse-button. If you would like to quit, then click on the right button.

NOTES

Useful enhancements to *d.profile* would include:

1) Adding an option to display profiles using category colors, like a bar-chart.
2) Allowing profile lines to be defined by a series of points, not just two.
3) Allowing profiles to be saved in a file, for later viewing.
4) Allowing the user to enter profile line points by typing coordinates.

AUTHOR

Dave Johnson
DBA Systems, Inc.

10560 Arrowhead Drive
Fairfax, Virginia 22030

NAME

d.rast – Displays and overlays raster map layers in the active display frame on the graphics monitor. *(GRASS Display Program)*

SYNOPSIS

d.rast
d.rast help
d.rast [**-o**] **map**=*name*

DESCRIPTION

d.rast displays the raster map layer *name* in the active display frame on the graphics monitor.

The program can be run either non-interactively or interactively. If the user specifies the name of a raster map layer (map=*name*) and (optionally) the -o option on the command line, the program will run non-interactively. Alternately, the user can type simply **d.rast** on the command line; in this case, the program will prompt the user for the flag setting and parameter value using the standard GRASS parser interface described in the manual entry for *parser*.

Flag:

-o Overlay the named raster map layer onto whatever is already displayed in the active graphics frame. Any zero category value data areas in the named raster map will seem transparent, and reveal the underlying image previously displayed in the graphics frame. If the -o flag is set, only cells containing non-zero category values will be displayed from the *overlaid* raster map. All other areas (i.e., the portions of the overlaid map that contain category value 0) will leave the underlying display untouched. If the -o flag is not set by the user, *d.rast* will (by default) completely overwrite whatever appears in the active graphics display frame.

Parameter:

map=*name* Name of the raster map to be displayed. If the active graphics frame already contains text or graphics, and the user does not wish to use the -o option, it is wise to first invoke *d.erase* to clear the active graphics frame before running *d.rast*. After running *d.rast*, other *d.* programs like *d.vect* and *d.grid* can be used to enhance the plot.

NOTES

The *d.rast* raster map overlay option (-o) only works when the color look-up table for the graphics monitor is set to **d.colormode fixed**.

SEE ALSO

d.choose, d.colormode, d.colors, d.colortable, d.display, d.erase, d.grid, d.overlay, d.vect, d.what.rast, r.reclass and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.rast.arrow – Draws arrows representing cell aspect direction for a raster map layer.
(GRASS Display Program)

SYNOPSIS

d.rast.arrow
d.rast.arrow help
d.rast.arrow [map=*name*] [**type**=*name*] [**arrow_color**=*name*] [**grid_color**=*name*] [**x_color**=*name*] [**unknown_color**=*name*]

DESCRIPTION

d.rast.arrow is designed to help users better visualize surface water flow direction, as indicated in an aspect raster map layer. There are two ways to specify the aspect layer the program is to use. The first is to display the aspect map layer on the graphics monitor before running *d.rast.arrow*. The second method involves setting the *map* parameter to the name of the desired aspect map. This allows the arrows to be drawn over any other maps already displayed on the graphics monitor.

d.rast.arrow will draw an arrow over each displayed cell to indicate in which direction the cell slopes. An arrow can point in one of eight directions. If the aspect layer has a category value denoting locations of "unknown" aspect, *d.rast.arrow* draws a question mark over the displayed cells of that category. Cells of category 0 (no data) will be marked with an "X".

Note: Arrows are cell size and cannot be seen if the raster map is relatively close in scale. From a command line mode, the user cannot run d.rast.arrow and then run d.rast.zoom and still see the arrows. The zooming action erases the arrows and the user must run .arrow again.

COMMAND LINE OPTIONS

Parameters:

map=*name* Name of an existing raster map layer to be displayed.

type=*name* Type of existing raster data to be displayed.
Options: grass, agnps, answers
Default: grass
Using this information, *d.rast.arrow* uses internal information to convert category values into appropriate arrow directions.

arrow_color=*name* Color in which arrows will be drawn.
Options: white, red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, black
Default: green

grid_color=*name* Color in which grid outlines will be drawn.
Options: white, red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, black
Default: gray

x_color=*name* Color in which x's will be drawn.
Options: white, red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, black
Default: white

unknown_color=*name*
Color in which unknown information will be displayed.
Options: white, red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, black
Default: red

SEE ALSO

d.frame, d.rast, d.rast.edit, d.rast.num, d.rast.zoom, d.zoom, g.region, r.mask, r.slope.aspect, and *parser*

AUTHOR

Chris Rewerts (rewerts@ecn.purdue.edu), Agricultural Engineering, Purdue University.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

d.rast.edit − Program allowing users to interactively edit the cell category values of raster map layers displayed on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.rast.edit
d.rast.edit help
d.rast.edit [**grid_color**=*name*]

DESCRIPTION

The *d.rast.edit* program allows users to interactively edit cell category values in a raster map layer displayed to the graphics monitor using a mouse cursor. This program determines the name of the raster map layer currently displayed in the active frame on the selected graphics monitor (if none, program will abort). The user is then prompted for the name of a new raster layer to be created in his mapset. *d.rast.edit* does not modify the user's original raster map layer.

OPTIONS

Parameter:

grid_color=*color* Sets the color to be used for a grid drawn during edit mode. A well-selected grid color helps to identify individual cells in the output.
Options: red, orange, yellow, green, blue, indigo, violet, gray, white, and black
Default: black

Geographic Region Concerns

d.rast.edit reads the region definition for the raster map layer being edited from its cell header file. The new, edited copy of the raster layer will be created with the same resolution and region dimensions. If the current region resolution does not match the raster map layer's resolution, the program will abort. The north, south, east, and west geographic region settings of the current region can be set to any view, as long as they fall within the boundaries of the raster map layer described in its cell header. This is important, since the current view must be such that individual cell locations are easily visible and sufficiently large on the graphics monitor to be pointed at with the mouse cursor. Any mask in place will be ignored when writing the new raster map layer.

Mouse Menus

Cell editing is done using a mouse cursor to choose menu selections and indicate areas on the displayed raster map that are to be edited.

Selections available via the *Main Menu* and the subsequent functions or sub-menus are described below:

1. Edit

invokes edit mode, during which no graphic menus are used, so that none of the displayed map is hidden. At the start of the edit mode, a grid is drawn over displayed cells to help distinguish cell boundaries. Users can select the color of this grid by setting the *grid_color* parameter. Interaction with the program during edit mode is done by using the three mouse buttons as follows:

Left Button

what's here. Identify the category value of the cell under the mouse cursor when the button is pressed. The category value of this cell is printed in the text frame.

Middle Button

edit. The current category value of the cell under the mouse cursor is iterated and a prompt for a new cell value appears on the textual command frame. (Remember to move the mouse to the text frame). Enter a new cell category value which is within the range of current category values for the map layer. Note: the edited cell is displayed in the color of the newly-assigned category value, but is hatched with lines of the grid color to indicate it has been edited, since, if the value of the cell is polled (before leaving edit mode), the old category value will be reported.

Right Button

exit edit mode. Command is returned to the *Main Menu* on the graphics monitor. If cell value changes were made during edit mode, they will be saved upon exit.

2. Redraw

redisplays the raster map layer on the graphics monitor.

3. Zoom

calls the *d.rast.zoom* program, to allow changing the view of the raster layer. **Zoom's** instructions will appear on the text command window. Interaction is conducted using mouse keys.

4. Arrow

This function is available for users editing aspect maps. The *d.rast.arrow* program is called, which draws arrows over the displayed cells to indicate the downslope direction of the cell. After selecting the *arrow* selection from the main menu, sub-menus will appear, allowing input options to be set for the *d.rast.arrow* program. The first sub-menu selects the type of aspect data displayed. (regular "GRASS" format, as produced by *r.slope.aspect*; a format prepared as input to the "AGNPS" program; and a format prepared for the "ANSWERS" program. The *d.rast.arrow* program can accept the name of a layer not drawn on the display for use as input, and an option is given to input the name of that layer, if desired. Otherwise, the currently displayed map layer will be used. Next, the color options of *d.rast.arrow* may be set via separate menus, or a choice for using default settings may be used.

5. Number

calls the *d.rast.num* program, which will print the cell category values over the displayed cells on the graphic monitor.

5. Options

change the program options setting for *grid_color*.

6. Exit

quit the *d.rast.edit* program. If edits have been made, the new raster map layer will be created. Support files are constructed. Category labels and color maps (if any) are copied from the original layer. *d.rast.edit* redisplays the original raster map file on the monitor as it exits.

NOTES

d.rast.edit will not create a new raster map layer if the user makes no cell edits while running the program.

Be careful not to cover the graphics monitor window with another frame during the editing process. In some cases, the *redraw* option will be able to remove traces of other frames; otherwise, the graphics monitor will not be refreshed until after *d.rast.edit* exits. Further note, however, that this has only been tested on *SUN* workstations, using a modified *SUNVIEW* graphics monitor driver.

The primary bane of the *d.rast.edit* program involves large map layers (with lots of rows and columns) and/or slow computers, since the program must read and write raster map layers row by row for the full size of the map layer as dictated by its region size and resolution. (The current region settings of north, south, east, and west will not limit the size of the edited copy of the map layer, since by use of the *d.rast.zoom* program, these values may change several times during the editing session). Their effects could be lessened if the program were to create a table of changes that it could incorporate into the new raster file on a forked process started when the user exits, or otherwise allow the user to issue a "save" command when he has made all desired edits to the raster file. Currently, for instance, if the user needs to use *d.rast.zoom* to access a different area of the map, he must wait for a read and write for each entrance and egress of the edit mode.

There is no "undo" command or way to exit edit mode without saving changes.

It would be nice to incorporate a scrollable version of *d.legend* (such that one could see a legend for files with many categories on a standard size sub-frame). It would be even nicer to be able to select the category values from a graphical legend when editing cell values (thereby saving a trip to the text frame to type in the new value).

Perhaps method(s) for multiple or mass cell edits would be useful. This could be done by providing modes in which the user may: 1) edit a block of cells to a given value by drawing a box; 2) be able to choose a given value which is automatically used as the new value on each cell chosen until a different value is desired.

There is no interrupt handling. This could leave files in .tmp or (rarely) result in half-baked raster files. The original file would survive unscathed by an interrupt at most any point in execution, but the graphics monitor may be left in an indeterminate state (try *d.erase* or *d.mon select=monitor_name* to bring it back into shape). Beware of exiting the program by means other than using *exit* on the *Main Menu*.

Perhaps a *grid_color* option allowing the grid to be turned off is needed, since it may be inappropriately displayed (i.e., not over the cell edges) in some circumstances.

The program has no method to enter new values beyond the current range of categories, but additional programming could make it so.

SEE ALSO

d.erase, d.frame, d.mon, d.rast, d.rast.arrow, d.rast.num, d.rast.zoom, d.zoom, g.region, r.mask, r.slope.aspect,

AUTHOR

Chris Rewerts (rewerts@ecn.purdue.edu), Agricultural Engineering, Purdue University. June 1991.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

d.rast.num – Overlays cell category values on a raster map layer displayed to the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.rast.num
d.rast.num help
d.rast.num [**map**=*name*] [**grid_color**=*name*]

DESCRIPTION

d.rast.num overlays cell category values onto a raster map layer displayed to the user's graphics monitor. Category values will be displayed in white and/or black, based on the colors in which underlying cells are displayed. A grid outlining each map cell will also be overlain in a user-specified color.

The user should run *d.rast* to display the desired map layer on the graphics monitor before running this program.

d.rast.num can be run non-interactively or interactively. If the user specifies the map whose category values are to be displayed and/or the grid color on the command line, the program will run non-interactively. Alternately, the user may simply type **d.rast.num** on the command line; in this case, the program will prompt the user for parameter values using the standard GRASS user interface described in the manual entry for *parser*. If no grid color is given the default will be used. If no map layer is specified, the program will use whatever raster map layer is currently displayed in the active frame on the graphics monitor.

COMMAND LINE OPTIONS

Parameters:

map=*name* Name of existing raster map layer whose category values will be displayed.
Default: (whatever raster map layer is currently displayed)

grid_color=*name* Color in which an overlain grid will be displayed.
Options: white, red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, black
Default: gray

NOTES

The user is advised to set the current region to a relatively small area (i.e., less than 100 rows by 100 columns); otherwise, the individual cells being displayed will be small and the category value associated with each will be difficult to see.

SEE ALSO

d.erase, d.frame, d.grid, d.mon, d.rast, d.rast.arrow, d.rast.edit, d.rast.num d.rast.zoom d.zoom, g.region, r.mask, r.slope.aspect

AUTHORS

Raghavan Srinivasan (srin@ecn.purdue.edu), and Chris Rewerts (rewerts@ecn.purdue.edu), Agricultural Engineering, Purdue University

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

d.rast.zoom – To interactively zoom in or zoom out of regions on a raster map displayed in the active frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.rast.zoom
d.rast.zoom help

DESCRIPTION

d.rast.zoom allows the user to zoom in or out of a region of interest on the raster map layer currently displayed in the active frame of the graphics monitor. The user zooms in and out interactively, using a mouse to establish the zoom region. To zoom, click the left mouse button to establish a corner, then drag the mouse to estabilsh the second (diagonal) corner and hit the right mouse button to accept the region. "Unzoom" will move back one step (i.e., to the last modified region). If the user tries to unzoom once again, the zoom region will be set equal to the default region settings. The user can zoom or unzoom any number of times until satisfied.

NOTES

d.rast.zoom is sensitive to the current mask (see *r.mask*.

This program runs only interactively.

SEE ALSO

d.frame, d.rast, d.rast.arrow, d.rast.edit, d.rast.num, d.zoom, g.region, r.mask, and *parser*

AUTHOR

Raghavan Srinivasan, Agricultural Engineering, Purdue University

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

d.rgb – Displays three user-specified raster map layers as red, green, and blue overlays in the active graphics frame.
(GRASS Display Program)

SYNOPSIS

d.rgb
d.rgb help
d.rgb [**red**=*name*] [**green**=*name*] [**blue**=*name*] [**out**=*name*]

DESCRIPTION

RGB stands for red, green, and blue. This program visually overlays up to three raster map layers, each displayed in either red, green, or blue. As each overlay is displayed in a single band, the intensity at each point is the average intensity of the red, green, and blue components of the currently active color table. For example, any cell that carries 100% intensity for either red, green, or blue and 0% intensity for the other two colors will be represented at 33% intensity.

WARNING: Maps that are using the "color wave" color table appear to be solid grey when using *d.rgb*. The average intensity of the colors being used is always 33%.

This program sacrifices spatial resolution to provide full color information. Any color that cannot be fully represented at a particular pixel passes the extra color to the next cell right and down. For example, if a cell is to show 50% red, but the closest color available (without showing more than 50%) is 40%, the pixle to the right and the pixle below will be given an extra 5% red each. In this way all the color is provided at the small cost of a slight amount of blurring. The result is very pleasing, especially on high resolution screens.

COMMAND LINE OPTIONS

Parameters:

red=name	Name of raster map layer to be used for RED component.
green=name	Name of raster map layer to be used for GREEN component.
blue=name	Name of raster map layer to be used for BLUE component.
out=name	Name of raster map layer to contain results.

NOTES

It produces an image and (optionally) a raster map layer with 1000 colors (10 intensities each of red, green and blue). The image and raster map layers will not display properly if the graphics device does not have at least 1000 colors while the user is running in *float* colormode. Thus, unless the display device has 1000 colors, it is necessary to run the GRASS program

d.colormode mode=fixed

before running *d.rgb*. Otherwise, the colors will be incorrect.

The intensity or the saturation layers may be left out. This means that it is possible to have *his, hi,* or *hs* images.

SEE ALSO

blend.sh, d.colormode, d.colors, d.colortable, d.his, hsv.rgb.sh, r.mapcalc, rgb.hsv.sh

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.rhumbline – Displays the rhumbline joining two user-specified points, in the active frame on the user's graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.rhumbline
d.rhumbline help
d.rhumbline coor=*lon1,lat1,lon2,lat2* [**lcolor**=*name*]

DESCRIPTION

A rhumbline is a line following a constant angle of the compass (i.e., a line of constant direction). There are 32 points on the compass (points are roughly 11 degrees 15 minutes apart). *d.rhumbline* displays the rhumbline joining any two user-specified points in the active frame on the user's graphics monitor. The named coordinate locations must fall within the boundaries of the user's current geographic region.

The user can run this program either interactively or non-interactively. If the user simply types **d.rhumbline** on the command line without specifying parameter values, the mouse will be activated and the user will be asked to use the mouse to indicate the two endpoints of the rhumbline. The rhumbline is then drawn in the default color (white). The program also outputs the coordinate locations of the two endpoints on the user's terminal and the number associated with the mouse button depressed by the user in a format useful for input to other programs.

Alternately, the user can specify the starting and ending longitude/latitude coordinates of the rhumbline and (optionally) the color in which the rhumbline will be displayed; in this case, the program will run non-interactively.

COMMAND LINE OPTIONS

Parameters:

coor=*lon1,lat1,lon2,lat2*
: Starting and ending coordinates of the rhumblines to be displayed, given as longitudes and latitudes.

lcolor=*name*
: Rhumbline color.
Options: red, orange, yellow, green, blue, magenta, indigo, violet, gray, white, black
Default: white

NOTES

This program works only with longitudelatitude data bases.

SEE ALSO

d.geodesic

AUTHOR

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

NAME

d.save **–** Creates a list of commands for recreating screen graphics.
(GRASS Display Program)

SYNOPSIS

d.save
d.save help
d.save [**-ca**] [**frame**=*name*[,*name*,...]]

DESCRIPTION

When GRASS display (d.) commands are used to generate graphical images on the graphics monitor, some commands are noted in lists that the current graphics driver (see manual entry for *d.mon*) maintains. This command, *d.save*, uses this information to create a shell script that can be used to recreate graphics in another frame at another time. The shell script is sent to standard output (the user's terminal). If you wish to save the shell script created by *d.save*, its output should be redirected to a file; for example:

d.save > script.out

The user can run the program non-interactively by specifying flag settings and parameter values on the command line. If the user types **d.save** without specifying these program arguments, the user will be prompted for inputs through the standard user interface described in the manual entry for *parser*.

COMMAND LINE OPTIONS

Flags:

-c Shell script will contain only information for regenerating graphics from the active (current) frame. Use of this flag overrides the *frame* option (below).

-a Save information needed to regenerate all the frames on the graphics monitor. The shell script can be used to completely reconstruct the contents of the graphics screen.

Parameter:

frame=*name*[,*name*,...]
Name(s) of those display frame(s) to be saved.
Options: All available frames
Default: Active frame

NOTES

Display commands used interactively, and those that take their graphic instructions from standard input, will not be saved by *d.save*.

SEE ALSO

d.frame, d.mon, d.savescreen, g.region, and *parser*

AUTHOR

David Satnik, Central Washington University

NAME

d.scale – Overlays a bar scale and north arrow for the current geographic region at a user-defined location in the active display frame.
(GRASS Display Program)

SYNOPSIS

d.scale
d.scale help
d.scale [**-m**] [**bcolor**=*name*] [**tcolor**=*name*] [**at**=*x,y*]

DESCRIPTION

d.scale overlays a bar scale and north arrow for the current geographic region at a user-defined location in the active display frame. The scale and north arrow are proportioned to fit in the active frame when placed along the frame's left edge.

The user can specify the location of the scale and north arrow interactively (using a mouse), or provide the geographic coordinates at which they will be placed. The scale and north arrow will only overwrite those portions of the graphics display that lie directly beneath them.

The user can also specify the colors in which the scale and north arrow will be drawn. By default, if unspecified by the user, a white scale and north arrow will be displayed on a black background.

OPTIONS

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

d.scale [**-m**] [**bcolor**=*name*] [**tcolor**=*name*] [**at**=*x,y*]

If at least one parameter value is given on the command line and the -m flag is not set, the program will be run non-interactively; any other parameter values left unspecified on the command line will be set equal to their default values (see below). Alternately, the user can simply type **d.scale** on the command line, without program arguments. In this case, the user will be prompted for needed parameter values and the flag setting using the standard GRASS parser interface described in the manual entry for *parser*. If the user sets the -m flag, *d.scale* will expect the user to designate the scale's location interactively, using the mouse.

Flag:

-m Use the mouse to interactively place the location of the scale and north arrow.

Parameters:

bcolor=*name* Set the background color underlying the scale and north arrow to *name*.
Options: *red, orange, yellow, green, blue, indigo, violet*, *gray, brown, magenta, white*, and *black*
Default: *black*

tcolor=*name* Set the foreground color in which the text, scale, and north arrow are displayed to *name*.
Options: Same as *bcolor* colors.
Default: *white*

at=*x,y* The map easting (x) and northing (y) geographic coordinates designating the location at which the upper left corner of the scale is to be placed. The user should choose geographic coordinates located within the current geographic region (see *d.where* and *g.region* for these coordinates).
Default: 0.0,0.0

NOTES

d.scale assumes that layer units are in meters.

The scale that *d.scale* generates is probably not suitable for very tiny maps (in small display frames).

SEE ALSO

d.legend, d.measure, d.what.rast, d.what.vect, d.where, g.region, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.sites – Displays site markers in the active display frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.sites
d.sites help
d.sites sitefile=*name* [**color**=*name*] [**size**=*value*] [**type**=*name*]

DESCRIPTION

d.sites displays a GRASS *site_lists* map using site markers of a color, size, and type specified by the user. Output is displayed in the active frame on the graphics monitor.

This program can be run non-interactively or interactively. If the user gives the name of a GRASS *site_lists* map and (optionally) specifies other parameter values on the command line the program will be run non-interactively; any parameter values left unspecified are set to their default values (see below). Alternately, the user can type simply **d.sites** on the command line; in this case, the program will prompt the user for parameter values using the standard GRASS parser interface described in the manual entry for *parser*.

Parameters:

sitefile=*name* Name of a GRASS *site_lists* map in the user's current mapset search path.

color=*name* Sets the current color to *name*.
Options: red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white, and black
Default: *gray*

size=*value* Size, in pixels, in which the site icons are to be drawn.
Options: 0-1000
Default: 5

type=*name* The type of icon to be displayed at site locations.
Options: x, diamond, box, +
Default: x

SEE ALSO

d.display, d.icons, d.points, d.rast, d.sites, d.vect, g.region and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.text – Draws text in the active display frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.text
d.text help
d.text [**size**=*value*] [**color**=*name*] [**line**=*value*]

DESCRIPTION

d.text draws text in the active display frame on the graphics monitor. Text can be provided through standard input or redirected from a file (using the UNIX redirection mechanism).

Parameters:

size=*value*	Height of letters, stated as a percent of the available display frame height. Options: 0 - 100 Default: 5
color=*name*	Sets display text color to *name*. Options: red, orange, yellow, green, blue, indigo, violet, gray, white, and black Default: *gray*
line=*value*	The screen line number on which the first line of text will be drawn. (Line 1 is at the top of the active display frame.) Options: 1 - 1000 Default: 1

In addition to the options provided on the command line, colors, text size, font type, and boldness, can be adjusted with commands in the standard input (i.e., if the user invokes *d.text* without options on the command line, and then assigns values to these options on lines within the standard input). In this case, the user also sees the standard GRASS parser interface described in the manual entry for *parser*.

Commands:

.C *color*
(where *color* is one of the available colors) causes text appearing on subsequent lines to be drawn in that color.

Text size can be adjusted with the command:
.S *size*
(where *size* is a percentage within the range 0 to 100). Note that a size of 10 would allow 10 lines to be drawn in the active display frame, 5 would allow the drawing of 20 lines, and 50 would allow the drawing of 2 lines.

Font type can be manipulated using the command:
.F *font*
(where *font* is one of the fonts known by the GRASS program *d.font*). Available fonts are listed in the GRASS manual entry for *d.font*. The default font type used (if unspecified by the user) is *romans*. Run the GRASS macro *show.fonts.sh* to see what these fonts look like.

The user can also stipulate that text be printed in **bold** on lines beneath the command:
.B 1
This command means *bold on* .
Similarly, the command:
.B 0
turns *bold off* of all text appearing on lines beneath it. (*Bold off* is used by default, if unspecified by the user.)

EXAMPLE

The following command will print the short phrase "This is a test of d.text" in the active display frame using the color yellow, in bold, and using 4/100'ths (4%) of the active frame's vertical space per line:

> **d.text**
> **.C yellow**
> **.S 4**
> **.B 1**
> **This is a test of d.text**

The user presses *control-d* (the "ctrl" and "d" keys) to end input to *d.text*.

NOTES

Note that the GRASS command *d.title* creates map titles in a format suitable for input to *d.text*.

d.text needs escape sequences that can be used within lines to change colors, boldness, and perhaps size.

SEE ALSO

d.font, d.title, show.fonts.sh, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.title – Outputs a title for a raster map layer in a form suitable for display by *d.text*.
(GRASS Display Program)

SYNOPSIS

d.title
d.title help
d.title [**-f**] **map**=*name* [**color**=*name*] [**size**=*value*]

DESCRIPTION

d.title generates to standard output a string which can be used by *d.text* to draw a title for the raster map layer *name* in the active display frame on the graphics monitor. Output created by *d.title* can be redirected into a file, or piped directly into *d.text* to display the map title created by *d.title*. The map title created will include the map layer's name, title, MAPSET, LOCATION_NAME, geographic region boundary coordinates, and cell resolution.

The user can state program arguments on the command line, or type simply **d.title** on the command line. In the latter case, the program will prompt the user for the parameter values and flag setting using the standard GRASS parser interface described in the manual entry for *parser*.

Flag:

-f Displays a fancier title.

Parameters:

map=*name* Name of an existing raster map layer in the user's mapset search path.

color=*name* Sets the current color to the *name* stated.
Options: red, orange, green, blue, indigo, violet, black, white, gray, yellow, brown, and magenta
Default: *white*

size=*value* Sets the text size as a percentage of the active display frame's height, to *value*. Floating point values can be used.
Options: 0 - 100
Default: 15.0

EXAMPLE

For example, a user wishing to create a suitable title for the Spearfish, SD *soils* map layer and to display this title in the active display frame on the graphics monitor might type the following:

d.title map=soils color=red size=5 > title.file

d.text < title.file

Alternately, the user might pipe *d.title* output directly into *d.text*:

d.title map=soils color=red size=5 | d.text

A file created by *d.title* can be displayed with *d.text*. Information contained in this file takes precedence over the *color* and *size* parameters for *d.text*.

NOTES

The text created with *d.text* will not necessarily fit within the active display frame on the graphics monitor; the user should choose a text size appropriate to this frame.

SEE ALSO

d.font, d.text and *parser*

AUTHORS

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

d.vect – Displays GRASS vector data in the active frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.vect
d.vect help
d.vect map=*name* [**color**=*name*]

DESCRIPTION

d.vect displays the user-named binary vector file in the active display frame on the graphics monitor, in the user-named color. The vector file must already exist under a mapset listed in the user's current mapset search path.

This program can be run either non-interactively or interactively. If the user gives the name of a binary vector map to be displayed and (optionally) specifies a color for vector display, the program will run non-interactively. (If the user gives a vector map name but no vector color on the command line, the named vector map layer will be displayed in the default color *white*.) Alternately, the user can simply type **d.vect** on the command line; in this case, the program will prompt the user for parameter values using the standard GRASS parser interface described in the manual entry for *parser*.

Parameters:

map=*name* Vector map layer to be displayed in the active display frame.
Must be binary vector file in the user's current mapset search path.

color=*name* Sets vector display color to *name*.
Options: white, red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, and black
Default: *white*

NOTES

This program needs to be upgraded with features that would:
1) Allow different line colors, weights, and textures to be assigned to vectors with different category values.
2) Allow different fill patterns and colors to be assigned to areas.
3) Be linked with an associated program identifying these options.

SEE ALSO

d.rast, d.sites, d.vect.dlg, d.what.vect, g.mapsets, and *parser*

AUTHOR

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

NAME

d.what.rast – Allows the user to interactively query the category contents of multiple raster map layers at user-specified locations within the current geographic region.
(GRASS Display Program)

SYNOPSIS

d.what.rast
d.what.rast help
d.what.rast [**-1t**] [**map**=*name*[,*name*,...]] [**fs**=*name*]

DESCRIPTION

d.what.rast outputs the category values and labels associated with cell(s) at user-specified location(s) on user-named raster map layer(s).

The program will query the contents of raster map layer(s) named by the user on the command line. These map layers must exist in the user's current mapset search path. If the user does not name any raster map layers on the command line, *d.what.rast* will query the category contents of whatever raster map layer is already displayed in the active frame on the graphics monitor.

The program activates the mouse, and expects the user to indicate the cell location(s) to be queried by depressing a mouse button over desired location(s) within the current geographic region in the active display frame on the graphic monitor.

Flags:

-1 Identify and query just one point location. Only one mouse click is executed. This option is provided for shell scripts and programs that want to obtain only one point from the user.

-t Provide only terse output. This option is provided to simplify the parsing of output by other programs.

Parameters:

map=*name* Name of existing raster map layer(s). Limit: 15 maps
Default: Query map currently displayed in the active graphics frame.

fs=*name* Output field separator to be used (in terse mode only).
Default: **:**

d.what.rast output consists of the geographic coordinates of the location pointed to, and, for each map layer, the map layer name, the category value, and category label in the named raster map layers at this location.

EXAMPLE

It is helpful, but not necessary, to first display a map to be used for reference in the active display frame before running *d.what.rast*. For example, the user might type the following series of commands and receive the output below.

d.rast map=soils
To first display the *soils* map in the active frame.

d.what.rast map=soils,aspect
User then moves the mouse to desired location on the displayed *soils* map layer, and presses the left mouse button to query the category contents of the *soils* and *aspect* maps at this geographic location. The program then outputs the below information to the user's terminal.

```
617112(E) 3732014(N)
soils in PERMANENT (44)Nunn clay loam, NdC
aspect in PERMANENT (20)15 degrees north of west
```

The first line of output gives the easting (E) and northing (N) coordinates of the geographic location at which the user clicked the mouse. The subsequent two lines give the map name and mapset, map category value (within parentheses), and map category label corresponding to this user-selected map location, for each of the maps queried by the user.

The query may be repeated as often as desired using the left mouse button. The right button on the mouse is used to quit the *d.what.rast* session.

Users can set the -t flag to obtain a terse output from *d.what.rast*. This is useful when the user wishes output to be parsed by another program (like *awk*). If the -t flag is set, users can also select the field separator used (with the fs=*name* option), or elect to use the default **:** field separator. In this case, the command

d.what.rast -t map=soils,aspect

produces output in the form shown below. The first line of output gives the easting, northing, and the mouse button return value (see NOTES, below). Each subsequent line gives the map name and its mapset, the category value, and category label of the point specified on the user-named raster map layers. The default output field separator, a colon, is used since none was specified on the command line.

```
617112:3732014:1
soils@PERMANENT:44:Nunn clay loam, NdC
aspect@PERMANENT:20:15 degrees north of west
```

Users can also use this program inside of shell scripts that require as input a map category value and a mouse button depressed. Users can choose an option to run *d.what.rast* only once, and return only the map category value found and the number of the mouse button depressed.

NOTES

Mouse button return values are as follows: 0 indicates no button was pressed, 1 indicates that the left mouse button was pressed, 2 indicates the middle button was pressed, and 3 indicates that the right mouse button was pressed.

d.what.rast will always print its output to the terminal screen. *d.what.rast* output can be redirected into a file; however, if it is, the output will go both to the screen and to the file. For example:

d.what.rast map=soils,aspect > what.out

will both send *d.what.rast* output to the screen and capture its output in the file named *what.out*.

The maximum number of raster map layers that can be queried at one time is 15.

d.what.vect allows the user to interactively query the category contents of multiple vector map layers at user-specified locations.

SEE ALSO

d.rast, d.what.vect, d.where, g.region, and *parser*

AUTHOR

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

NAME

d.what.vect – Allows the user to interactively query the category contents of a (binary) vector map layer at user-selected locations within the current geographic region.
(GRASS Display Program)

SYNOPSIS

d.what.vect
d.what.vect help
d.what.vect [**-1**] **map**=*name*

DESCRIPTION

d.what.vect outputs the category value(s) associated with user-specified location(s) in a user-specified vector map layer. This program currently returns only category values for line types in the user-specified vector file.

The program will query the contents of the vector map layer named by the user on the command line. This map layer must exist in the user's current mapset search path.

The program activates the mouse, and expects the user to indicate the location(s) to be queried by depressing a mouse button over desired location(s) within the current geographic region in the active display frame on the graphic monitor.

Flag:

-1 Identify and query just one point location. Only one mouse click is executed. This option is provided for shell scripts and programs that want to obtain only one point from the user.

Parameter:

map=*name* Name of an existing binary vector map in the user's mapset search path.

EXAMPLE

A sample *d.what.vect* session is given below. Although it is not necessary that the user first display a vector map to be queried in the active display frame, it is helpful to have a map displayed there for reference.

d.vect map=roads.24000

Displays the 1:24,000 scale *roads* vector map layer on the graphics monitor.

d.what.vect map=roads.24000

After typing this, the user moves the mouse to a desired location on the displayed *roads* map layer, and presses the left mouse button to query the category value of the *roads* vector map at this location. The program then outputs the category value of a line type corresponding to this user-selected map location, for the vector map queried by the user.

The query may be repeated as often as desired using the left mouse button. The right button on the mouse is used to quit the *d.what.vect* session.

Users can also use this program inside of shell scripts that require as input a map category value and a mouse button depressed. Users can set the -1 flag to run *d.what.vect* only once, and return only the map category value found and the number of the mouse button depressed. (Mouse button return values are as follows: 0 indicates no button was pressed, 1 indicates that the left mouse button was pressed, 2 indicates the middle button was pressed, and 3 indicates that the right mouse button was pressed.)

NOTES

Currently, *d.what.vect* only outputs category values for lines. It does not output category labels for lines, nor output category values or category labels for areas in a vector file.

d.what.vect will always print its output to the user's terminal screen. *d.what.vect* output can be redirected into a file; however, if it is, the output will go both to the screen and to the file. For example:

d.what.vect map=roads > what.out

will both send *d.what.vect* output to the screen and capture its output in the file named *what.out*.

d.what.rast can be used to interactively query the map category contents of multiple raster map layers at user-selected locations.

SEE ALSO

d.rast, d.vect, d.what.rast, g.region, and *parser*

AUTHOR

Jim Hinthorn, Central Washington University
Upgrades: Dennis Finch, National Park Service

NAME

d.where – Identifies the geographic coordinates associated with point locations in the active frame on the graphics monitor.
(GRASS Display Program)

SYNOPSIS

d.where
d.where help
d.where [**-1**] [**spheroid**=*name*]

DESCRIPTION

d.where is an *interactive* program that allows the user, using the pointing device (mouse), to identify the geographic coordinates associated with point locations within the current geographic region in the active display frame on the graphics monitor.

If the user runs *d.where* without specifying the name of a spheroid on the command line, each mouse click will output the UTM easting and northing of the point currently located beneath the mouse pointer. A mouse-button menu is presented so the user knows which mouse buttons to use. The output is always printed to the terminal screen; if the output is redirected into a file, it will be written to the file as well.

Flag:

-1 Only one mouse click is executed.
This option is provided for shell scripts and programs which want to obtain only one point from the user. The output is only written to stdout, unless redirected into a file. The geographic location and mouse button pressed are output.

Parameter:

spheroid=*name* Name of a spheroid (for latitude/longitude coordinate conversion).
Options: australian, bessel, clark66, clark80, everest, international, wgs72, wgs84

NOTES

This program uses the current geographic region setting and active frame. It is not necessary, although useful, to have displayed a map in the current frame before running *d.where*.

SEE ALSO

d.what.rast, d.what.vect, g.region

AUTHORS

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

NAME

d.zoom – Allows the user to change the current geographic region settings interactively, with a mouse. *(GRASS Display Program)*

SYNOPSIS

d.zoom
d.zoom help
d.zoom [**-q**] [**action**=*name*]

DESCRIPTION

d.zoom allows the user to interactively adjust the settings of the current geographic region using a pointing device such as a mouse. Like *g.region*, *d.zoom* re-defines the settings of the geographic region. However, *d.zoom* allows the user to change the current region settings interactively, by either outlining the new region perimeter with a mouse or "rotating" latitudelongitude data into the current region. The user should run *d.erase* after *d.zoom* is run for the new region settings to affect the graphics display.

Flag:

-q Run quietly, suppressing output of some program messages to standard output.

Parameter:

action=*name* Allows the user to change the geographic region settings by "rotating" the globe to move a different portion of the globe into the stationery boxed region outline (making this new portion of the globe the new current region), rather than by moving the boxed region outline. This option can only be used with latitudelongitude data bases (although *d.zoom* will not complain if the user attempts to set this parameter while running on a non-latitudelongitude data base).
Options: zoom, rotate

After the user types the command **d.zoom** and (optionally) sets the -q flag and type of zoom to be performed, a mouse-button menu will appear, directing the user to: establish the corners of the new geographic region, check its coordinates, and confirm any changes made. When the user accepts new geographic boundaries, *d.zoom* asks,

Accept new region? Y / N >

If the user clicks the mouse over the "Y" (yes), the mouse-drawn geographic region is saved as the user's current geographic region. The user is warned that *d.erase* should be run after *d.zoom* to make new current region settings affect the graphics display.

If the user clicks the mouse over the "N" (no), the first mouse-drawn geographic region is not saved; instead, *d.zoom* asks whether or not the user wishes to

Try Again? Y / N >

The user can then opt to draw a new region with the mouse ("Y"), or opt to exit *d.zoom* ("N") and leave current region settings unchanged.

To zoom-out to a larger area than was windowed-in on with *d.zoom* (i.e., to enlarge the current geographic region), the user may run *g.region*. The user may also alter the current geographic region by running the "region" option of the *d.display* program.

NOTES

Although it is not necessary that the user display a map in the active display frame before running *d.zoom*, it is helpful to do this for reference.

Currently, the rubber-banded boundaries of the zoom region drawn by the user are not clearly visible on the graphics monitor. However, whether or not these boundaries are clearly visible when drawn, the drawn region is still correctly zoomed-in on.

SEE ALSO

d.display, d.erase, d.rast, g.region

AUTHOR

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

NAME

exit – Exits the user from the current GRASS session.

SYNOPSIS

exit

DESCRIPTION

The *exit* command ends the user's current GRASS session and returns the user to the directory in which he was working prior to entering GRASS. When the user exits GRASS, he is asked whether he wishes to save the files and data stored under his current mapset, and (if maps are present) whether the user wishes to selectively remove map layers, before exiting the system. By default, if the user presses RETURN without responding to these questions, all maps in the user's current mapset are saved. However, because such maps can consume much storage space on the computer, the user should remove any unneeded files before exiting. (The user can also remove data before attempting to *exit* GRASS using the *g.remove* command.)

Before typing *exit*, the polite user should remember to release the graphics display monitor (using the command **d.mon -r**) for use by other GRASS users. Otherwise, the display device may be locked for use by the user, even though the user has exited GRASS, until another user runs *d.mon* and unlocks this monitor for others' use.

Each time the user re-enters GRASS, the variables described in *g.gisenv* are re-set. If the user wishes to change the mapset, location, or data base on which he is working (i.e., those affected by any GRASS programs running during the user's current GRASS session), the user must *exit* GRASS and then re-enter GRASS and specify a different mapset, location, and/or data base location on the GRASS start-up page. When the user re-enters GRASS, these variable settings (the current mapset, location, and data base) are set by default to those used in the user's previous GRASS session, unless changed by the user.

NOTES

This program requires no command line arguments; the user simply types **exit** on the command line to exit GRASS.

SEE ALSO

d.mon, g.gisenv, g.remove

NAME

g.access – Controls user access to the current GRASS mapset.
(GRASS File Management Program)

SYNOPSIS

g.access

DESCRIPTION

This program allows the user to control access to the current mapset. Normally, any user can read data from any GRASS mapset. But sometimes it is desirable to prohibit access to certain sensitive data. The *g.access* command allows a user to restrict read and execute access to the current mapset (see UNIX *chmod* command). *g.access* will not modify write access to the current mapset.

The user may, for example, allow only users in the same UNIX group to read data files in the mapset, or restrict the mapset to personal use only.

After typing

g.access

the user will be presented with a screen page which reflects the current mapset permissions. The user can then change them. The screen page looks like:

```
LOCATION: spearfish                                    MAPSET: demo

This program allows you to control access to your mapset by other users.
Access may be granted/removed for everyone, or for everyone in your group.

Mark an 'x' to allow access;  erase the field to restrict access.

   GROUP: _x_
   OTHER: _x_

          AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
                         (OR <Ctrl-C> TO CANCEL)
```

If you remove the x (using the space bar), access will be denied to that class of user (group or other). If you type an x, access will be granted to that class of user.

NOTES

There is no non-interactive version of *g.access*.

Under GRASS version 4.0, access to the mapset PERMANENT must be open to all users. This is because GRASS looks for the user's default geographic region definition settings and the location title in files that are stored under the PERMANENT mapset directory. The *g.access* command, therefore, will not allow you to restrict access to the PERMANENT mapset.

The *g.mapsets* command isn't smart enough to tell if access to a specified mapset is restricted, and the user is therefore allowed to include the names of restricted mapsets in his search path. However, the data in a restricted mapset is still protected; any attempts to look for or use data in a restricted mapset will fail. The user will simply not see any data listed for a restricted mapset.

SEE ALSO

UNIX manual entries for *chmod* and *group*
g.mapsets

AUTHOR

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

NAME

g.ask – Prompts the user for the names of GRASS data base files.
(GRASS File Management Program)

SYNOPSIS

g.ask help
g.ask type=*name* [**prompt**=*"string"*] **element**=*name* [**desc**=*"string"*] **unixfile**=*name*

DESCRIPTION

g.ask is designed for shell scripts that need to prompt the user for the name of a data base file in the user's current GRASS location. After *g.ask* is invoked with needed parameters, it will query the user for a file name of the specified *type* and *element*. After the user responds to this query, the program will write four lines to the UNIX output file specified by *unixfile*.

Parameters:

type=*name* The type of query. Options for *name* are *old*, *new*, *any*, and *mapset*; their functions are given below. "New", "any", and "mapset" only look in the user's current mapset.

old For existing data files anywhere in the user's mapset search path.

new Used to create a new file in the current mapset, which must not already exist there (if a file with this name exists there, it will not be overwritten).

any Creates a file in the current mapset, which may or may not already exist there. If a file with this name exists in the current mapset, it will be overwritten; if not, a new file with this name will be created.

mapset For files that must exist in the current mapset The shell write wants the name of a file which exists in the user's current mapset. This type would used instead of "old" if the file is to be modified.

prompt="*string*" The prompt to be displayed to the user. If more than one word, the prompt should be enclosed within double quotes ("").

element=*name* The name of the GRASS data base element (i.e., directory under a GRASS mapset) whose files are to be queried.

desc="*string*" A short description of the data base element which is meaningful to the user. If description contains more than one word, it should be enclosed within double quotes ("").

unixfile=*name* The name of a UNIX file to store the user's response. See next section for what is written to this file and how it can be used by shell scripts.

OUTPUT

Upon receiving the user's response to its request for a file name, *g.ask* writes four lines to the specified *unixfile*; this output file is placed in the user's current working directory, unless otherwise specified, and contains the following lines:

name='*some_name*'
mapset='*some_mapset*'
fullname='*some_fullname*'
file='*some_fullpath*'

The output is /bin/sh commands to set the variable *name* to the file name specified by the user (of the *element* and *type* requested by *g.ask*), *mapset* to the GRASS mapset in which this file resides (or will be created), *fullname* is the name with the mapset embedded in it, and *file* to the full UNIX path name identifying this file. These variables may be set in the /bin/sh as follows:

. *unixfile*

The . is a shell command which means read the *unixfile* and execute the commands found there. It is NOT part of the *unixfile* name and MUST be followed by a space.

NOTES

The user may choose to simply hit the return key and not enter a file name. If this happens the variables will be set as follows:

```
name=
mapset=
fullname=
file=
```

The following is a way to test for this case:

```
if [ ! "$file" ]
then
        exit
fi
```

SEE ALSO

d.ask, g.filename, g.findfile, g.gisenv

AUTHOR

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

NAME

g.copy – Copies available data files in the user's current mapset search path and location to the appropriate element directories under the user's current mapset.
(GRASS File Management Program)

SYNOPSIS

g.copy
g.copy help
g.copy [**rast**=*from,to*] [**vect**=*from,to*] [**icon**=*from,to*] [**labels**=*from,to*] [**sites**=*from,to*]
[**region**=*from,to*] [**group**=*from,to*]

DESCRIPTION

A user may access data stored under the other mapsets listed in his mapset search path. However, the user may only modify data stored under his own current mapset. *g.copy* allows the user to copy existing data files *from* other mapsets *to* the user's current mapset ($MAPSET). The files to be copied must exist in the user's current mapset search path and location; output is sent to the relevant data element directory(ies) under the user's current mapset.

The user specifies the type(s) of data files he wishes to copy (raster, vector, etc.), the name of the existing file to be copied (i.e., the *from* file name), and the name of the new file copy to be placed in the user's current mapset (the *to* file name). This information can be given either (non-interactively) on the command line, or entered in response to program prompts given via the standard parser interface described in the manual entry for *parser*.

Information can be entered on the command line in the following format:

> **g.copy** [**rast**=*from,to*] [**vect**=*from,to*] [**icon**=*from,to*] [**labels**=*from,to*] [**sites**=*from,to*]
> [**region**=*from,to*] [**group**=*from,to*]

For example, if the user wished to copy the existing raster file *soils* to a file called *soils.ph* and to copy an existing vector file *roads* to a file called *rds.old*, the user could type:

> **g.copy rast=soils,soils.ph vect=roads,rds.old**

Data files can also be specified by their mapsets. For example, the below command copies the raster file named *soils* from the mapset *wilson* to a new file called *soils* to be placed under the user's current mapset:

> **g.copy rast='soils@wilson',soils**

If no mapset name is specified, *g.copy* searches for the named *from* map in each of the mapset directories listed in the user's current mapset search path in the order in which mapsets are listed there (see *g.mapsets*).

If the user does not enter parameter values but instead types only **g.copy** on the command line the program will prompt the user for a data type, the name of a file of this data type to copy, and the name of a new file to hold the copy. After both file names have been entered, the copy is created and the user is again prompted for a data element to be copied, until the user hits RETURN. When prompted for file names, the user may enter 'list' to see a list of existing files, or hit RETURN to end the file listing.

Parameters:

rast=*from,to*	where *from* is an existing raster map layer to be copied, and *to* is the name given to the copy.
vect=*from,to*	where *from* is an existing binary vector map layer to be copied, and *to* is the name given to the copy.
icon=*from,to*	where *from* is an existing paint icon file to be copied, and *to* is the name given to the copy.
labels=*from,to*	where *from* is an existing /paint/labels file to be copied, and *to* is the name given to the copy.
sites=*from,to*	where *from* is an existing *site_lists* file to be copied, and *to* is the name given to the copy.
region=*from,to*	where *from* is an existing region definition (*windows*) file to be copied, and *to* is the name given to the copy.
group=*from,to*	where *from* is an existing imagery group file to be copied, and *to* is the name given to the copy.

NOTE

If a file has support files (e.g., as do raster data files), these support files will also be copied.

SEE ALSO

g.access, g.list, g.mapsets, g.remove, g.rename, and *parser*

AUTHOR

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

NAME

g.filename – Prints GRASS data base file names.
(GRASS File Management Program)

SYNOPSIS

g.filename
g.filename help
g.filename element=*name* **mapset**=*name* **file**=*name*

DESCRIPTION

g.filename is designed for Bourne shell scripts that need to know the full UNIX file name for raster map layers, vector files, site list files, geographic region definition (windows) files, imagery group files, etc., in the GRASS data base. If the user runs *g.filename* without command line arguments (i.e., simply types **g.filename**), this program will prompt the user for input using the standard parser interface described in the manual entry for *parser*.

Parameters:

element=*name*	The name of a GRASS data base element (i.e., directory within the GRASS mapset location).
mapset=*name*	The name of a GRASS data base mapset. As a convenience, a single dot (.) can be used to designate the current mapset.
file=*name*	The name of a GRASS data base file.

OUTPUT

g.filename writes one line to standard output:

file=*'full_file_pathname'*

The output is a /bin/sh command to set the variable specified by the file *name* to the full UNIX path name for the data base file. This variable may be set in the /bin/sh as follows:

**eval `g.filename element**=*name* **mapset**=*name* **file**=*name*`

NOTES

This routine generates the filename, but does not care if the file (or mapset or element) exists or not. This feature allows shell scripts to create new data base files as well as use existing ones.

If the mapset is the current mapset, *g.filename* automatically creates the *element* specified if it doesn't already exist. This makes it easy to add new files to the data base without having to worry about the existence of the required data base directories. (This program will not create a new mapset, however, if that specified does not currently exist.)

The program exits with a 0 if everything is ok; it exits with a non-zero value if there is an error, in which case file=*'full_file_pathname'* is not output.

SEE ALSO

g.ask, g.findfile, g.gisenv, and *parser*

AUTHOR

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

NAME

g.findfile – Searches for GRASS data base files and sets variables for the shell.
(GRASS File Management Program)

SYNOPSIS

g.findfile
g.findfile help
g.findfile element=*name* [**mapset**=*name*] **file**=*name*

DESCRIPTION

g.findfile is designed for Bourne shell scripts that need to search for raster map layer files, vector files, site list files, geographic region definition (windows) files, and imagery group files in the GRASS data base. If the user runs *g.findfile* without command line arguments, he will be prompted for the names of a GRASS element, file, and mapset, through the standard parser interface (see manual entry for *parser*).

Parameters:

element=*name* The data base element (i.e., directory within a GRASS mapset) to be searched.

mapset=*name* The mapset in which to search for the specified file *name*. If not specified, all mapsets in the user's GRASS search path are searched. Otherwise, the specified mapset is searched. As a convenience, if specified as a single dot (.) only the current mapset is searched.

file=*name* The name of a GRASS data file (of the stated *element* type) for which to search.

OUTPUT

g.findfile writes four lines to standard output:

name='*file_name*'
mapset='*mapset_name*'
file='*unix_filename*'
fullname='*grass_fullname*'

The output is /bin/sh commands to set the variable *name* to the GRASS data base file name, *mapset* to the mapset in which the file resides, and *file* to the full UNIX path name for the named file. These variables may be set in the /bin/sh as follows:

**eval `g.findfile element**=*name* **mapset**=*name* **name**=*name*`

NOTES

If the specified file does not exist, the variables will be set as follows:

```
name=
mapset=
fullname=
file=
```

The following is a way to test for this case:

```
if [ ! "$file" ]
then
        exit
fi
```

SEE ALSO

g.ask, g.filename, g.gisenv, g.mapsets, and *parser*

AUTHOR

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

NAME

g.gisenv – Outputs the user's current GRASS variable settings.
(GRASS File Management Program)

SYNOPSIS

g.gisenv
g.gisenv [*variable_name*]

DESCRIPTION

When a user runs GRASS, certain variables are set specifying the GRASS data base, location, mapset, peripheral device drivers, etc., being used in the current GRASS session. These variable name settings are recognized as long as the user is running a GRASS session.

No prompts are given to the user when running *g.gisenv*. If run without arguments, *g.gisenv* lists all of the user's current GRASS variable settings. Results are sent to standard output, and may look like this:

```
GISDBASE=/usr/grass4/data
LOCATION_NAME=spearfish
MAPSET=PERMANENT
```

In this example, the full path name of the user's current location (i.e., $LOCATION_NAME) is /usr/grass4/data/spearfish, and the full path name of the user's current mapset (i.e., $MAPSET) is /usr/grass4/data/spearfish/PERMANENT.

If the user specifies a *variable_name* on the command line (e.g., **g.gisenv MAPSET**), only the value for that particular GRASS variable is output to standard output. Possible variable names depend on the user's system.

While other variables may be associated with each GRASS session (e.g., DIGITIZER, PAINTER, DISPLAY, and other variables), those stated below are essential.

GISDBASE - The $GISDBASE is a directory in which all users' GRASS data are stored. Within the $GISDBASE, data are segregated into subdirectories (called "locations") based on the map coordinate system used and the geographic extent of the data. Each "location" directory itself contains subdirectories called "mapsets"; each "mapset" stores "data base elements" -- the directories (e.g., the *cell*, *cellhd*, *dig*, etc., directories) in which GRASS data files are actually stored.

LOCATION_NAME - The user must choose to work with the data under a single GRASS location within any given GRASS session; this location is then called the *current GRASS location*, and is specified by the variable $LOCATION_NAME. The $LOCATION_NAME is the GRASS data base location whose data will be affected by any GRASS commands issued during the user's current GRASS session, and is a subdirectory of the current $GISDBASE. Each "location" directory can contain multiple "mapset" directories (including the special mapset "PERMANENT"). Maps stored under the same GRASS LOCATION_NAME (and/or within the same MAPSET) must use the same coordinate system and typically fall within the boundaries of the same geographic region (aka, "location").

MAPSET - Each "mapset" contains a set of maps relevant to the LOCATION_NAME directory in which it appears. Each LOCATION_NAME can contain multiple mapsets. (Mapsets which fall under the same LOCATION_NAME all contain data geographically relevant to the LOCATION_NAME, and all store data in the same map coordinate system. Frequently, maps are placed into different mapsets to distinguish file ownership -- e.g., each user might have his own mapset, storing any maps that he has created and/or are relevant to his work.) During each GRASS session, the user must choose one mapset to be the *current mapset*; the current mapset setting is given by

$MAPSET, and is a subdirectory of $LOCATION_NAME. During a single GRASS session, the user can use available data in any of the mapsets stored under the current LOCATION_NAME directory that are in the user's mapset search path and accessible by the user. However, within a single GRASS session, the user only has *write* access to data stored under the *current mapset* (specified by the variable $MAPSET).

Each "mapset" stores GRASS data base elements (i.e., the directories in which GRASS data files are stored). Any maps created or modified by the user in the current GRASS session will be stored here. The MAPSET directory "PERMANENT" is generally reserved for the set of maps that form the base set for all users working under each LOCATION_NAME.

Once within a GRASS session, GRASS users have access only to the data under a single GRASS data base directory (the *current GRASS data base*, specified by the variable $GISDBASE), and to a single GRASS location directory (the *current location*, specified by the variable $LOCATION_NAME). Within a single session, the user may only *modify* the data in the *current mapset* (specified by the variable $MAPSET), but may *use* data available under other mapsets under the same LOCATION_NAME.

All of these names must be legal names on the user's current system. For UNIX users, names less than 14 characters and containing no non-printing or space codes are permissible. Examples of permissible names include: *one, mymap*, *VeGe_map*, and *1_for_me*. The underscore character can safely be used in place of a blank for multiple-word names.

NOTES

The output from *g.gisenv* when invoked without arguments is directly usable by /bin/sh. The following command will cast each variable into the UNIX environment:

eval `g.gisenv`

This works only for /bin/sh. The format of the output is not compatible with other UNIX shells.

SEE ALSO

g.access, g.ask, g.filename, g.findfile, g.mapsets

AUTHOR

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

NAME

g.help – GRASS help facility.
(GRASS Help Program)

SYNOPSIS

g.help

DESCRIPTION

g.help provides the user with functional information on GRASS programs, a glossary, and access to on-line *User's Reference Manual* entries. The help facility is accessed by simply typing **g.help** on the command line. The user can then wend his way through a series of menus (organized by functional area) and keyword searches.

On-line reference manual entries can also be accessed directly, through the GRASS *g.manual* command.

SEE ALSO

GRASS User's Reference Manual

g.manual

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory
Deb Brinegar, U.S. Army Construction Engineering Research Laboratory
Mary Martin, U.S. Army Construction Engineering Research Laboratory

Note. The help facility uses the *hyper* program written by Jim Westervelt.

NAME

g.list – Lists available GRASS data base files of the user-specified data type to standard output.
(GRASS File Management Program)

SYNOPSIS

g.list
g.list help
g.list [**-f**] **type**=*datatype* [**mapset**=*name*]

DESCRIPTION

g.list allows the user to list user-specified, available and accessible files from *mapsets* under the user's current location. When invoked simply as **g.list**, the program prompts the user for the type of data to be listed from all *mapsets* in the user's current mapset search path. The user can list files from a mapset not listed in the current mapset search path by running the program non-interactively, specifying the (optional) flag setting and parameter values on the command line. Program flag and parameters are described below.

Flag:

-f Returns a verbose file listing, that includes map titles.

Parameters:

type=*datatype* The type of data to be listed.
Options: (listed in bold)

rast	Raster files
vect	Binary vector files
icon	Paint icon files
labels	Paint labels files
sites	Site list files
region	Region definition files
group	Imagery group files

mapset=*name* The name of a mapset to be searched for files of the specified *type*. Any mapset name under the current location, whether or not it is listed in the user's current mapset search path, can be specified.
Default: If unspecified, files of the specified *type* from all mapsets in the user's current search path will be listed to standard output.

NOTES

If the user requests that files from a mapset to which access has been restricted (see *g.access* be listed, no files from this mapset will be listed.

SEE ALSO

g.access, g.mapsets

AUTHOR

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

NAME

g.manual – Accesses GRASS User's Reference Manual entries.
(GRASS File Management Program)

SYNOPSIS

g.manual
g.manual help
g.manual [**-1aefs**] [**entries**=*name*[,*name,...*]]

DESCRIPTION

The *g.manual* command provides user access to the on-line **GRASS User's Reference Manual** entries. The user may request a list of available manual entries, and may get a manual entry printed to the terminal screen and/or to the line printer.

g.manual can be run either interactively or non-interactively. If the user types

g.manual

on the command line without program arguments, the program will prompt the user for a manual entry to display. The user may enter "list" to get a section-by-section listing of the manual entries available. Once the user has viewed the desired information it may be printed by responding to the questions appropriately.

The user can run the program non-interactively, by specifying the appropriate options and/or the *name*(s) of the manual entries to displayed.

OPTIONS

-1 This option will list all manual entries, one per line.

-a This option will list all manual entries in a more appealing format. The manual page list will be separated by manual section.

-e This option tells *g.manual* to ignore empty manual sections when printing the listings from the *-1* or *-a* options.

-f This option will add formfeeds to output listing when using the *-a* option.

-s This option will cause *g.manual* to run silently. Instead of displaying the manual page it will simply set the exit status to:
0 if entry exists, or
1 if it does not exist.

These entries may also be accessed through the *g.help* command.

SEE ALSO

GRASS User's Reference Manual

g.help

AUTHOR

Kurt Buehler, U.S. Army Construction Engineering Research Laboratory

NAME

g.mapsets – Modifies the user's current mapset search path, affecting the user's access to data existing under the other GRASS mapsets in the current location.
(GRASS File Management Program)

SYNOPSIS

g.mapsets
g.mapsets help
g.mapsets [**-lp**] [**mapset**=*name*[,*name*,...]]

DESCRIPTION

A *mapset* holds a distinct set of data layers, each relevant to the same (or a subset of the same) geographic region, and each drawn in the same map coordinate system. At the outset of every GRASS session, the user identifies a GRASS data base, location, and mapset that are to be the user's *current data base*, *current location*, and *current mapset* for the duration of the session; any maps created by the user during the session will be stored under the *current mapset* ($MAPSET) set at the session's outset.

The user can add, modify, and delete data layers that exist under his *current mapset*. Although the user can also *access* (i.e., use) data that are stored under *other* mapsets in the same GRASS location, the user can only make permanent changes (create or modify data) located in the *current mapset*. The user's *mapset search path* lists the order in which other mapsets in the same GRASS location can be searched and their data accessed by the user. The user can modify the listing and order in which these mapsets are accessed by modifying the mapset search path; this can be done using the *g.mapsets* command. This program allows the user to use other's relevant map data without altering the original data layer, and without taking up disk space with a copy of the original map.

g.mapsets shows the user available mapsets under the current GRASS location, lists mapsets to which the user currently has access, and lists the order in which accessible mapsets will be accessed by GRASS programs searching for data files. The user is then given the opportunity to add or delete mapset names from his search path, or modify the order in which mapsets will be accessed.

When the user specifies the name of a data base element file (e.g., a particular vector file, raster file, imagery group file, etc.) to a GRASS program, the program searches for the named file under each of the mapsets listed in the user's mapset search path in the order listed there until the program finds a file of the given name. (Users can also specify a file by its mapset, to make explicit the mapset from which the file is to be drawn; e.g., the command:

g.copy rast='soils.file@PERMANENT',my.soils

ensures that a new file named *my.soils* is to be a copy of the file *soils.file* from the mapset PERMANENT.)

It is common for a user to have the special mapset **PERMANENT** included in his mapset search path, as this mapset typically contains finished base maps relevant to many applications. Often, other mapsets which contain sets of interpreted map layers will be likewise included in the user's mapset search path. Suppose, for example, that the mapset *Soil_Maps* contains interpreted soils map layers to which the user wants access. The mapset *Soil_Maps* should then be included in the user's *search path* variable.

The *mapset search path* is saved as part of the current mapset. When the user works with that mapset in subsequent GRASS sessions, the previously saved mapset search path will be used (and will continue to be used until it is modified by the user with *g.mapsets*).

Flags:

-l List all available mapsets under the user's current location.

-p Print the user's current mapset search path to standard output.

Parameter:

mapset=*name*[,*name*,...] Name(s) of existing GRASS mapset(s) under the current location.

g.mapsets sets the current *mapset search path* to the *mapsets* named on the command line. If **g.mapsets** is typed but no *mapset* names are specified by the user on the command line, the program will print the user's current mapset search path, list available mapsets, and prompt the user for a new mapset search path listing.

NOTES

Users can restrict others' access to their mapset files through use of the GRASS program *g.access*. Mapsets to which access is restricted can still be listed in another's mapset search path; however, access to these mapsets will remain restricted.

SEE ALSO

g.access, g.copy, g.gisenv, g.list

AUTHOR

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

Greg Koerper, ManTech Environmental Technology, Inc.

NAME

g.region − Program to manage the boundary definitions for the geographic region.
(GRASS Region Management Program)

SYNOPSIS

g.region
g.region help
g.region [-dgpu] [region=*name*] **[raster**=*name*] **[vector**=*name*] **[sites**=*name*] **[3dview**=*name*] **[n**=*value*] **[s**=*value*] **[e**=*value*] **[w**=*value*] **[res**=*value*] **[nsres**=*value*] **[ewres**=*value*] **[zoom**=*name*] **[align**=*name*] **[save**=*name*]

DESCRIPTION

The *g.region* program allows the user to manage the settings of the current geographic region. These regional boundaries can be set by the user directly and/or set from a region definition file (stored under the *windows* directory in the user's current mapset). The user can create, modify, and store as many geographic region definitions as desired for any given mapset. However, only one of these geographic region definitions will be current at any given moment, for a specified mapset; i.e., GRASS programs that respect the geographic region settings will use the current geographic region settings.

INTERACTIVE PROGRAM USE: MAIN MENU

The main menu consists of an information section listing the current GRASS data base LOCATION, MAPSET, and CURRENT REGION, followed by user options:

```
----------------------------------------------------------
|                  REGION FACILITY                        |
| LOCATION: sample                        MAPSET: grass   |
|                                                         |
| CURRENT REGION: N=5167600  S=5156755  RES=50  ROWS=216  |
|                 E=729314   W=705924   RES=50  COLS=467  |
| PROJECTION: 1 (UTM)                                     |
| ZONE: 13                                                |
|                                                         |
|                                                         |
| Please select one of the following options              |
|                                                         |
|     Current Region             Region Database          |
|                                                         |
| 1 Modify current region     6 Save current region in    |
|   directly                    the database             |
|                                                         |
| 2 Set from default region   7 Create a new region       |
|                                                         |
| 3 Set from a database       8 Modify an existing region |
|   region                                                |
|                                                         |
| 4 Set from a raster map                                 |
|                                                         |
| 5 Set from a vector map                                 |
|                                                         |
| RETURN to quit                                          |
----------------------------------------------------------
```

DEFINITIONS

Region:

Here, a *region* refers to a geographic area with some defined boundaries, based on a specific map coordinate system and map projection. Each region also has associated with it the specific east-west and north-south resolutions of its smallest units (rectangular units called "cells").

The region's boundaries are given as the northernmost, southernmost, easternmost, and westernmost points that define its extent. The north and south boundaries are commonly called *northings*, while the east and west boundaries are called *eastings*.

The region's cell resolution defines the size of the smallest piece of data recognized (imported, analyzed, displayed, stored, etc.) by GRASS programs affected by the current region settings. The north-south and east-west cell resolutions need not be the same, thus allowing non-square data cells to exist.

Default Region:

Each GRASS LOCATION_NAME has a fixed geographic region, called the default geographic region (stored in the region file DEFAULT_WIND under the special mapset PERMANENT), that defines the extent of the data base. While this provides a starting point for defining new geographic regions, user-defined geographic regions need not fall within this geographic region.

Current Region:

Each mapset has a current geographic region. This region defines the geographic area in which all GRASS displays and analyses will be done. Data will be resampled, if necessary, to meet the cell resolutions of the current geographic region setting.

Region Data Base:

Each GRASS MAPSET may contain any number of pre-defined, and named, geographic regions. These region definitions are stored in the user's current mapset location under the *windows* directory (also referred to as the user's data base of region definitions). Any of these pre-defined geographic regions may be selected, by name, to become the current geographic region. Users may also access saved region definitions stored under other mapsets in the current location, if these mapsets are included in the user's mapset search path.

REGION EDIT PROMPT

Most of the options will require the user to edit a geographic region, be it the current geographic region or one stored in the user's data base of region definitions (the *windows* directory). A standard prompt is used to perform this edit. An example is shown below:

```
-------------------------------------------------------------
|                      IDENTIFY REGION                      |
|                                                           |
|          ======== DEFAULT REGION ========                 |
|          |  Default North: 3402025.00        |            |
|          |                                   |            |
|          |         ===YOUR REGION===         |            |
|          |         |  NORTH EDGE   |         |            |
|          |         |  3402025.00_  |         |            |
|          |         |               |         |            |
| Def West: |WEST EDGE |             |EAST EDGE | Def.East:  |
| 233975.00 |233975.00_|             |236025.00_| 236025.00  |
|          |         |  SOUTH EDGE   |         |            |
|          |         |  3399975.00_  |         |            |
|          |         =================         |            |
|          |                                   |            |
|          |  Default South: 3399975.00        |            |
|          =====================================            |
|                                                           |
|            Default   GRID RESOLUTION   Region             |
|             50.00    --- East-West ---   50.00__          |
|             50.00    -- North-South --   50.00__          |
|                                                           |
|                                                           |
|    AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE    |
-------------------------------------------------------------
```

The fields NORTH EDGE, SOUTH EDGE, WEST EDGE and EAST EDGE, are the boundaries of the geographic region that the user can change. The fields Default North, Default South, Def West and Def East are the boundaries of the default geographic region that are displayed for reference and *cannot* be changed. The two GRID RESOLUTION Region fields (east-west, and north-south) are the geographic region's cell resolutions that the user can change. The two GRID RESOLUTION Default fields list the resolutions of the default geographic region; these are displayed for reference and cannot be changed here by the user.

REGION MANAGEMENT MENU OPTIONS

1 Modify the current geographic region directly
: Allows the user to edit the current region.

2 Set current geographic region from default region
: Copies the default region to the current geographic region, and then lets the user edit the current geographic region.

3 Set current geographic region from a data base geographic region
: Allows the user to select a geographic region by name from the data base of geographic regions to become the current geographic region, and then lets the user edit the current geographic region.

  Note: geographic region definition files may be selected from other mapsets as well, if accessible and in the user's mapset search path.

4 Set current geographic region from a raster (cell) map layer
: Allows the user to select a raster map layer, copies the cell header for this map layer to the current geographic region, and then lets the user edit the current geographic region. This option is useful when subsequent GRASS operations will be used to produce a raster map layer from one input raster map layer and it is necessary that the result coincide with the input raster map layer.

5 Save the current geographic region (window) in the data base
: Allows the user to save the current geographic region settings in the user's data base of such settings. These files are stored in the *windows* directory under the user's current mapset. This option is useful when the current geographic region is set directly using option 2, or even by another GRASS program (e.g., *d.display*). This option installs an otherwise temporary geographic region setting into the geographic region definition data base for recall when needed.

6 Create a new data base geographic region setting
: Creates a new geographic region definition in the user's data base of such settings in the *windows* directory under the current mapset, using the geographic region edit prompt described above. After the geographic region definition is created, the user is asked if this geographic region setting should also be used as the current geographic region.

7 Modify a data base geographic region setting
: Modifies a geographic region setting (in the data base of such settings in the *windows* directory of the current mapset), using the geographic region edit prompt. After the changes have been made, the user is asked if this geographic region setting should also be used as the current geographic region.

NON-INTERACTIVE PROGRAM USE

Alternately, the user can modify the settings of the current geographic region by specifying all needed parameters on the command line. The user enters the command **g.region** *parms*, where *parms* are the following parameters and/or flags:

Flags:

-d	Set current region settings equal to default region settings.
-g	Print the current region settings (shell script style) in a format that can be given back to *g.region* on its command line.
-p	Print the current region settings.
-u	Do not update the current region file settings. Allows the user to temporarily use a different region setting, without saving this setting.

Parameters:

region=*name*	Make current region settings same as the named region file settings
raster=*name*	Make current region settings same as those in the named raster map's cell header. But see **zoom**=*name* option, below.
vector=*name*	Make the current region settings the same as those of the named vector map.
sites=*name*	Set the current region to the smallest region encompassing all coordinates in the named site_lists file, aligned with the current region.
3dview=*name*	Make current region settings same as those in the named 3dview file, which holds the region that was current when the 3dview was saved.
n=*value*	Set map coordinate value for the region's northern edge to x
s=*value*	Set map coordinate value for the region's southern edge to x
e=*value*	Set map coordinate value for the region's eastern edge to x
w=*value*	Set map coordinate value for the region's western edge to x
res=*value*	Set grid resolution (both north-south and east-west) to x
nsres=*value*	Set north-south grid resolution value to x
ewres=*value*	Set east-west grid resolution value to x
zoom=*name*	Set current region settings to the smallest region encompassing all non-zero data in the named raster map layer that fall inside the user's current region.

If the user also includes the **raster**=*name* option on the command line, **zoom**=*name* will set the current region settings to the smallest region encompassing all non-zero data in the named **zoom** map that fall inside the region stated in the cell header for the named **raster** map.

align=*name*
: Set the current resolution equal to that of the named raster map, and align the current region to a row and column edge in the named map. Alignment only moves the existing region edges outward to the edges of the next nearest cell in the named raster map -- not to the named map's edges. To perform the latter function, use the **raster**=*name* option.

save=*name*
: Save current region settings in the named region file

EXAMPLES

g.region n=7360100 e=699000

will reset the northing and easting for the current region, but leave the south edge, west edge, and the region cell resolutions unchanged.

g.region −dp s=698000

will set the current region from the default region for the GRASS data base location, reset the south edge to 698000, and then print the result.

g.region n=n+1000 w=w500

The n=*value* may also be specified as a function of its current value: n=n+*value* increases the current northing, while n=n*value* decreases it. This is also true for s=*value*, e=*value*, and w=*value*. In this example the current region's northern boundary is extended by 1000 units and the current region's western boundary is decreased by 500 units.

g.region n=s+1000 e=w+1000

This form allows the user to set the region boundary values relative to one another. Here, the northern boundary coordinate is set equal to 1000 units larger than the southern boundary's coordinate value, and the eastern boundary's coordinate value is set equal to 1000 units larger than the western boundary's coordinate value. The corresponding forms s=n-*value* and
w=e-*value* may be used to set the values of the region's southern and western boundaries, relative to the northern and eastern boundary values.

g.region raster=soils

This form will make the current region settings exactly the same as those given in the cell header file for the raster map layer *soils*.

g.region raster=soils zoom=soils

This form will first look up the cell header file for the raster map layer *soils*, use this as the current region setting, and then shrink the region down to the smallest region which still encompasses all non-zero data in the map layer *soils*. Note that if the parameter *raster=soils* were not specified, the zoom would move to encompass all non-zero data values in the soils map that were located within the current region setting.

g.region −up raster=soils

The −u option suppresses the re-setting of the current region definition. This can be useful when it is desired to only extract region information. In this case, the cell header file for the soils map layer is printed without changing the current region settings.

g.region −u raster=soils zoom=soils save=soils

This will zoom into the smallest region which encompasses all non-zero soils data values, and save the new region settings in a file to be called *soils* and stored under the *windows* directory in the user's current mapset. The current region settings are not changed.

g.region −p
This will print the current region in the format:

projection:	1 (UTM)
zone:	15
north:	4294050.00
south:	4249950.00
east:	526050.00
west:	500950.00
nsres:	100.00
ewres:	100.00
rows:	441
cols:	251

g.region −g
The −g option prints the region in the following format:

```
n=4294050.00
s=4249950.00
e=526050.00
w=500950.00
nsres=100.00
ewres=100.00
```

This format does not have the rows and columns, but can be fed back into *g.region* on its command line.

The −p (or −g) option is recognized last. This means that all changes are applied to the region settings before printing occurs.

NOTE

After all updates have been applied, the current region's southern and western boundaries are (silently) adjusted so that the north/south distance is a multiple of the north/south resolution and that the east/west distance is a multiple of the east/west resolution.

SEE ALSO

d.display, d.zoom, g.access, g.mapsets

AUTHOR

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

NAME

g.remove – Removes data base element files from the user's current mapset.
(GRASS File Management Program)

SYNOPSIS

g.remove
g.remove help
g.remove [**rast**=*name*[,*name*,...]] [**vect**=*name*[,*name*,...]] [**icon**=*name*[,*name*,...]] [**labels**=*name*[,*name*,...]] [**sites**=*name*[,*name*,...]] [**region**=*name*[,*name*,...]] [**group**=*name*[,*name*,...]]

DESCRIPTION

g.remove allows the user to remove specified data base element files from the current mapset. If **g.remove** is invoked without arguments on the command line, a menu will appear listing possible data element types, as below:

1 raster maps
2 vector maps
3 paint icon files
4 paint label files
5 site list files
6 region files
7 imagery group files
RETURN to exit

Once the element type is selected, the user is prompted to name a specific file of this element type for removal. (This list will vary, depending on what files currently exist in the user's mapset.) The specified file is removed, and the user is again prompted for the name of a file of this element type to be removed. When prompted for a file name, the user may enter **list** to see a list of existing files of this element type, or hit RETURN to get back to the above menu.

Alternately, the user can specify the data base element type and file(s) to be removed on the command line. Data base element types are specified by the names to the left, below.

Parameters:

rast=*name*[,*name*,...]	Name(s) of raster file(s) to be removed.
vect=*name*[,*name*,...]	Name(s) of vector file(s) to be removed.
icon=*name*[,*name*,...]	Name(s) of paint icon file(s) to be removed.
labels=*name*[,*name*,...]	Name(s) of paint labels file(s) to be removed.
sites=*name*[,*name*,...]	Name(s) of site list file(s) to be removed.
region=*name*[,*name*,...]	Name(s) of region file(s) to be removed.
group=*name*[,*name*,...]	Name(s) of imagery group file(s) to be removed.

The data base element file(s) named by the user on the command line are subsequently removed from the user's current mapset.

EXAMPLE

For example, the below command will cause the raster files named *soils*, *slope*, and *temp*, the vector files named *roads* and *rail*, and the imagery group files named *nhap.1* and *nhap.2*, and these files' associated support files (e.g., cell header files, category files, etc.), to be removed from the user's current mapset.

g.remove rast=soils,slope,temp vect=roads,rail group=nhap.1,nhap.2

NOTE

If a particular data base element file has support files associated with it (e.g., as is commonly the case with raster files), *g.remove* will remove these support files along with the data base element file specified.

The user can only use *g.remove* to remove data files existing under the user's *current mapset*.

FILES

$GISBASE/etc/element_list lists the element types whose files can be removed by the user.

SEE ALSO

g.copy, g.list, g.rename

AUTHOR

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

NAME

g.rename – To rename data base element files in the user's current mapset.
(GRASS File Management Program)

SYNOPSIS

g.rename
g.rename help
g.rename [rast=*old,new*] [**vect**=*old,new*] [**icon**=*old,new*] [**labels**=*old,new*] [**sites**=*old,new*]
[**region**=*old,new*] [**group**=*old,new*]

DESCRIPTION

g.rename allows the user to rename data base element files in the user's current mapset. The user can specify all necessary information to *g.rename* on the command line, by specifying: the type of data base element to be renamed (one or more of: **rast, vect, icon, labels, sites, region,** and **group**); the specific file element in the current mapset to be renamed (*old*); and the new name to be assigned to this file element (*new*) in the current mapset. The file element *old* is then renamed to *new*.

Users can also simply type **g.rename** without arguments on the command line, to receive a menu of existing data base element types and files from which to choose for possible renaming:

1	raster maps
2	binary vector maps
3	paint icon files
4	paint label files
5	site list files
6	region definition files
7	imagery group files

RETURN to exit

NOTE

If a data base element has support files (e.g., as is commonly the case with raster files), these support files also are renamed.

If the user attempts to rename a file to itself by setting the *new* file name equal to the *old* file name (e.g., **g.rename rast=soils,soils**), *g.rename* will not execute the rename, but instead state that no rename is needed. However, *g.rename* will allow the user to overwrite other existing files in the current mapset by making the *new* file name that of an already existing file.

SEE ALSO

g.copy, g.list, g.remove

AUTHOR

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

NAME

g.setproj – Allows the user to create the PROJ_INFO and the PROJ_UNITS files to record the projection information associated with a current location.

SYNOPSIS

g.setproj

DESCRIPTION

Allows a user to create a PROJ_INFO file in the PERMANENT mapset of the current location. PROJ_INFO file is used to record the projection information associated with the specified mapset.

Notes:

User running *g.setproj* must own PERMANENT mapset. It is highly recommended to run *g.setproj* after creating a new location so that conversion programs (such as *v.proj can be run.*
The current location must not contain a PROJ_INFO or PROJ_UNITS file.
The user will be prompted for the projection name.
The specification of any projection other than ll will generate a request to the user for a name of a standard ellipse.
The projections of aea, lcc, merc, and tmerc will generate a request to the user for the prime meridian and standard parallel for the output map.
The projection of stp will generate a request to the user for the state abbreviation and choice of zone for the output map.
The user will be prompted for the spheroid and zone of the UTM projection.

SEE ALSO

v.proj m.proj

AUTHOR

Irina Kosinovsky, US Army CERL, Champaign, Il

NAME

g.tempfile – Creates a temporary file and prints the file name
(GRASS File Management Program)

SYNOPSIS

g.tempfile help
g.tempfile pidfR=*value*

DESCRIPTION

g.tempfile is designed for shell scripts that need to use large temporary files. GRASS provides a mechanism for temporary files that does not depend on /tmp. GRASS temporary files are created in the data base with the assumption that there will be enough space under the data base for large files. GRASS periodically removes temporary files that have been left behind by programs that failed to remove them before terminating.

g.tempfile creates an unique file and prints the name. The user is required to provide a process-id which will be used as part of the name of the file. Most Unix shells provide a way to get the process id of the current shell. For /bin/sh and /bin/csh this is $$. It is recommended that $$ be specified as the process-id for *g.tempfile.*

EXAMPLE

For /bin/sh scripts the following syntax should be used:

```
temp1=`g.tempfile pid=$$`
temp2=`g.tempfile pid=$$`
```

For /bin/csh scripts, the following can be used:

```
set temp1=`g.tempfile pid=$$`
set temp2=`g.tempfile pid=$$`
```

NOTES

Each call to *g.tempfile* creates a different (i.e. unique) name.

Although GRASS does eventually get around to removing tempfiles that have been left behind, the programmer should make every effort to remove these files. They often get large and take up disk space. If you write /bin/sh scripts, learn to use the /bin/sh *trap* command. If you write /bin/csh scripts, learn to use the /bin/csh *onintr* command.

AUTHOR

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

NAME

g.version – Outputs the GRASS version number and date.
(GRASS File Management Program)

SYNOPSIS

g.version
g.version help

DESCRIPTION

g.version prints to standard output the GRASS version number and date, in the form:

GRASS 4.0 (Summer 1991)

NOTES

This program requires no command line arguments; the user simply types **g.version** on the command line to see the version number and date of the GRASS software currently being run by the user.

AUTHOR

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

NAME

i.cca – Canonical components analysis (cca) program for image processing.
(GRASS Image Processsing Program)

SYNOPSIS

i.cca
i.cca help
i.cca group=*name* **subgroup**=*name* **signature**=*name* **output**=*name*

DESCRIPTION

i.cca is an image processing program that takes from two to eight (raster) band files and a signature file, and outputs the same number of raster band files transformed to provide maximum separability of the categories indicated by the signatures. This implementation of the canonical components transformation is based on the algorithm contained in the LAS image processing system.

Typically the user will use the *i.class* program to collect a set of signatures and then pass those signatures along with the raster band files to *i.cca*. The raster band file names are specified on the command line by giving the group and subgroup that were used to collect the signatures.

The output raster map names are built by appending a ".1", ".2", etc. to the output raster map name specified on the command line.

Parameters:

group=*name*	Name of the imagery group to which the 2 to 8 raster band files used belong.
subgroup=*name*	Name of the imagery subgroup to which the 2 to 8 raster band files used belong.
signature=*name*	Name of an ASCII file containing spectral signatures.
output=*name*	Output raster file prefix name. The output raster map layer names are built by appending a ".1", ".2", etc. onto the *output* name specified by the user.

NOTES

i.cca respects the current geographic region definition and the current mask setting while performing the transformation.

SEE ALSO

Schowengerdt, Robert A. **Techniques for Image Processing and Classification in Remote Sensing**, Academic Press, 1983.

i.class, i.pca, r.covar, r.mapcalc

AUTHORS

David Satnik, GIS Laboratory, Central Washington University
Ali R. Vali, Space Research Center, University of Texas, Austin

NAME

i.class – An imagery function that generates spectral signatures for an image by allowing the user to outline regions of interest. The resulting signature file can be used as input for *i.maxlik* or as a seed signature file for *i.cluster*.
(GRASS Image Processing Program)

SYNOPSIS

i.class

DESCRIPTION

i.class performs the first pass in the GRASS two-pass supervised image classification process; the GRASS program *i.maxlik* executes the second pass. Both programs must be run to generate a classified map in GRASS raster format.

i.class is an interactive program that allows the user to outline a region on the screen and calculate the spectral signature based on the cells that are within that region. During this process the user will be shown a histogram of the region for each image band. The user can also display the cells of the image bands which fall within a user-specified number of standard deviations from the means in the spectral signature. By doing this, the user can see how much of the image is likely to be put into the class associated with the current signature.

The spectral signatures that result are composed of region means and covariance matrices. These region means and covariance matrices are used in the second pass (*i.maxlik*) to classify the image.

Alternatively, the spectral signatures generated by *i.class* can be used for seed means for the clusters in the *i.cluster* program.

USER INPUTS

The first screen in the program *i.class* asks the user for the imagery *group* and *subgroup* to be analyzed:

```
LOCATION: location      SUPERVISED CLASSIFIER      MAPSET: demo

          Please select the group and subgroup to be analyzed

     GROUP:       spot_______     (list will show available groups)
     SUBGROUP:    123________     (list will show available subgroups)


        AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
                       (OR <Ctrl-C> TO CANCEL)
```

The *group* should contain the imagery bands that the user wishes to classify. The *subgroup* is a subset of this group. The user must create a group and a subgroup by running the GRASS program *i.group* before running *i.class*. The subgroup should contain only the image bands that the user wishes to classify. Note that this subgroup must contain more than one band.

After the first screen, the program asks the user for the name of the resulting signature file. The signature file is both the output file for *i.class* and the required input file for the GRASS program *i.maxlik*. It contains the region means and covariance matrices that are used to classify an image in *i.maxlik*.

After entering the resulting signature file name, the user is asked to enter the name of a seed signature file. This is optional. A "seed" signature file is a previously created signature file. Such a seed signature file may be the result of an earlier run of *i.class*. The seed signature file is copied into the new resulting signature file before any new signatures are added by *i.class*. In this way, you can collect the work from several sessions with *i.class* into one signature file.

Next, the user is asked to enter the name of the raster map to be displayed during the process of outlining regions. Typically, the user will want to enter the name of a color composite previously created by *i.composite*. However, the user can enter the name of any existing raster map. This leaves the potential for using a raster map not directly derived from the image as a backdrop on which the user can outline the classes of interest.

At this point the *i.class* graphics screen will be drawn on the graphics monitor and the user will be directed to use the mouse. From this point on the user will primarily work with the mouse, selecting options from the menus and outlining regions on the screen. The only time that the user will need to return to the text terminal is to enter names for the signatures created.

THE DISPLAY FRAMES

The display frame layout that *i.class* uses is represented below for reference.

```
+--------------------------+--------------------------+
|                          |                          |
|                          |   Map Display Frame      |
|                          |                          |
|                          |                          |
|   Histogram Display      |                          |
|        Frame             |                          |
|                          |                          |
|                          |                          |
|                          |                          |
|                          +--------------------------+
|                          |                          |
|                          |   Zoom Display Frame     |
|                          |                          |
|                          |                          |
|                          |                          |
|                          |                          |
|                          |                          |
|                          |                          |
+--------------------------+--------------------------+
|       Menu Frame                                    |
+-----------------------------------------------------+
```

THE MENUS

All of the menus in the *i.class* program are displayed across the bottom of the graphics monitor in the Menu Frame. To select an option from one of these menus, simply place the cursor over your selection and press any button on the mouse. Each of the menus is discussed in the following paragraphs.

The Command Menu

The Command Menu includes the following selections:

Zoom - This command allows the user to outline a rectangular region in either the Map or Zoom Display Frames and the region is displayed, magnified, to fit in the Zoom Display Frame. A red rectangle is drawn in the Map Display Frame, indicating what area the Zoom Display Frame shows.

To outline the rectangular region simply use any mouse button to anchor the first corner of the border and then use any button to choose the other corner.

Define region - This selection takes the user to the Region Menu. This menu includes the options that allow the user to outline a region of interest on the displayed raster map.

Redisplay map - This selection takes the user to the Redisplay Menu. The Redisplay Menu allows the user to redraw map display frames.

Analyze region - This selection starts the process of analyzing the currently defined region. A histogram of the defined region will be displayed for each band. On the histogram for each band, the mean, standard deviation, minimum cell value and maximum cell value are marked. The histograms are automatically scaled in an attempt to fit the data into the space available, but it is possible that all of the data will not fit. In this case, as much of the data as possible, centered around the mean, will be displayed. After the histograms are displayed, the user will be given the Signature Menu.

Quit - The user should make this selection to end the session with *i.class*.

The Region Menu

The Region Menu contains the following selections:

Erase region - This selection erases any currently defined region.

Draw region - This selection allows the user to use the mouse to draw a region on either the Map or Zoom Display Frame. An explanation of which mouse buttons to use is displayed in the Menu Frame. The user does not need to try to complete the region boundary. The last line of the region will be added when the user selects the Complete region option on the Region Menu.

Restore last region - This selection restores the last region that was drawn. After a region is completed, it will be saved to be restored later. Only one previous region is saved.

Complete region - This selection completes the region that is currently being drawn. As noted above, it saves the complete region to be restored later, if needed. Once the user has made a complete region, it can be analyzed with the Analyze Region selection on the Command Menu.

Done - Use this selection to return to the Command Menu.

The Redisplay Map Menu

The Redisplay Map Menu has the following selections, which are useful to redraw the raster maps displayed in the Map and Zoom Display Frames.

Map geographic region - This selection causes the raster map in the Map Display Frame to be redrawn.

Zoom region - This selection causes the Zoom Display Frame to be redrawn.

Both - This selection causes both the Map and Zoom Display Frames to be redrawn.

Cancel - Use this selection if you do not want to redisplay either of the above regions. The user will be returned to the Command Menu.

The Analyze Region Menu

The Analyze Region Menu contains the Signature Menu, which allows the user to set the number of standard deviations and the display color, and then to display (as an overlay) the cells that match the signature within the number of standard deviations specified. Note that once the matching cells are displayed, the Map Display Frame must be redisplayed to see only the original raster map again. The following selections are available on the Signature Menu:

Set std dev's - This selection allows the user to set the number of standard deviations from the mean for the maximum and minimum range. The maximum and minimum range is used when finding the cells that "match" the signature. The user is presented with a menu of typical choices and an "Other" option. If the "Other" option is selected, enter the number of standard deviations from the keyboard on the text terminal.

Otherwise, the selected option will be used. When the number of standard deviations is set, the histograms for each band will be redrawn with the maximum and minimum range marked.

Note that the number in parentheses on this selection is the current number of standard deviations.

Set color - This selection allows the user to set the color for the display of cells that "match" the current signature. The user is presented with a menu of color choices. The color selected will be used when the Display Matches Menu selection is made.

Note that the color in parentheses on this selection is the current color for display.

Display matches - This selection displays the cells that "match" the current signature in the current color. A cell "matches" the current signature if the cell value in each band is between the minimum range and maximum range for that band defined by the number of standard deviations currently set.

Done - When this selection is chosen, the user will be asked whether or not he/she would like to save the current signature. If the user answers with the "Yes" selection, he/she will be asked to enter a description for the resultant signature file on the text terminal keyboard. The saved signature file description will be used by *i.maxlik* to name the category that is created from the current signature. After either a "No" answer or the signature description is entered, the user is returned to the Command Menu.

NOTES

i.class uses the current MASK to generate the overlay for cells that match a signature. As a result, if a MASK already exists it will be removed during the execution of this program.

The cell values in the image bands cannot fall outside of the range of 0 to 255. *i.class* will report an error if they do.

i.class, like some of the other imagery programs, does not use the standard GRASS display frames. After running *i.class*, you will need to create a display frame (e.g., using *d.frame* or *d.erase*) before you can use most of the GRASS display (d.) commands.

i.group must be run before *i.class* to create an imagery group and a subgroup containing the image bands to be classified.

The user can perform a supervised image classification by running *i.class* followed by *i.maxlik*. The user can perform an unsupervised classification by running *i.cluster* followed by *i.maxlik*.

i.class is interactive and requires no command line arguments. The user must be running a graphics display monitor (see *d.mon*) to run this program.

SEE ALSO

GRASS Tutorial: Image Processing

d.frame, d.mon, g.region, i.cca, i.cluster, i.composite, i.group, i.maxlik, r.mapcalc, r.mask

AUTHOR

David Satnik, Central Washington University GIS Laboratory

NAME

i.cluster − An imagery function that generates spectral signatures for land cover types in an image using a clustering algorithm. The resulting signature file is used as input for *i.maxlik*, to generate an unsupervised image classification.
(GRASS Image Processing Program)

SYNOPSIS

i.cluster
i.cluster help
i.cluster [**-q**] **group**=*name* **subgroup**=*name* **sigfile**=*name* **classes**=*value* [**seed**=*name*] [**sample**=*row_interval,col_interval*] [**iterations**=*value*] [**convergence**=*value*] [**separation**=*value*] [**min_size**=*value*] [**reportfile**=*name*]

DESCRIPTION

i.cluster performs the first pass in the *GRASS* two-pass unsupervised classification of imagery, while the GRASS program *i.maxlik* executes the second pass. Both programs must be run to complete the unsupervised classification.

i.cluster is a clustering algorithm that reads through the (raster) imagery data and builds pixel clusters based on the spectral reflectances of the pixels. The pixel clusters are imagery categories that can be related to land cover types on the ground. The spectral distributions of the clusters (which will be the land cover spectral signatures) are influenced by six parameters set by the user. The first parameter set by the user is the initial number of clusters to be discriminated. *i.cluster* starts by generating spectral signatures for this number of clusters and "attempts" to end up with this number of clusters during the clustering process. The resulting number of clusters and their spectral distributions, however, are also influenced by the range of the spectral values (category values) in the image files and the other parameters set by the user. These parameters are: the minimum cluster size, minimum cluster separation, the percent convergence, the maximum number of iterations, and the row and column sampling intervals.

The cluster spectral signatures that result are composed of cluster means and covariance matrices. These cluster means and covariance matrices are used in the second pass (*i.maxlik*) to classify the image. The clusters or spectral classes result can be related to land cover types on the ground.

OPTIONS

The program can be run either non-interactively or interactively. It will be run non-interactively if the user specifies the name of group file, the name of subgroup file, the name of a file to contain result signatures, the initial number of clusters to be discriminated, and optionally other parameters (see below) on the command line using the form:

> **i.cluster** [**-q**] **group**=*name* **subgroup**=*name* **sigfile**=**name** **classes**=*value* [**seed**=*name*] [**sample**=*row_interval,col_interval*] [**iterations**=*value*] [**convergence**=*value*] [**separation**=*value*] [**min_size**=*value*] [**reportfile**=*name*]

where the *group* should contain the imagery files that the user wishes to classify. The *subgroup* is a subset of this group. The user must create a group and subgroup by running the *GRASS* program *i.group* before running *i.cluster*. The subgroup should contain only the imagery band files that the user wishes to classify. Note that this subgroup must contain more than one band file. The purpose of the group and subgroup is to collect map layers for classification or analysis. The *sigfile* is the file to contain result signatures which can be used as input for *i.maxlik*. The classes value is the initial number of clusters to be discriminated; any parameter values left unspecified are set to their default values. Alternatively, the program will be run interactively if the user types only *i.cluster*; in this case the program will prompt the user for parameter values using the standard *GRASS* parser interface described in the manual entry for parser.

Flags:

-q Run quietly. Suppresses output of program percent-complete messages and the time elapsed from the beginning of the program. If this flag is not used, these messages are printed out.

Parameters:

group=*name* The name of the group file which contains the imagery files that the user wishes to classify.

subgroup=*name* The name of the subset of the group specified in group option, which must contain only imagery band files and more than one band file. The user must create a group and a subgroup by running the *GRASS* program *i.group* before running *i.cluster*.

sigfile=*name* The name assigned to output signature file which contains signatures of classes and can be used as the input file for the *GRASS* program *i.maxlik* for an unsupervised classification.

classes=*value* The number of clusters that will initially be identified in the clustering process before the iterations begin.

seed=*name* The name of a seed signature file is optional. The seed signatures are signatures that contain cluster means and covariance matrices which were calculated prior to the current run of *i.cluster*. They may be acquired from a previously run of *i.cluster* or from a supervised classification signature training site section (e.g., using the signature file output by *i.class*). The purpose of seed signatures is to optimize the cluster decision boundaries (means) for the number of clusters specified.

sample=*row_interval,col_interval*
These numbers are optional with default values based on the size of the data set such that the total pixels to be processed is approximately 10,000 (consider round up).

iterations=*value* This parameter determines the maximum number of iterations which is greater than the number of iterations predicted to achieve the optimum percent convergence. The default value is 30. If the number of iterations reaches the maximum designated by the user; the user may want to rerun *i.cluster* with a higher number of iterations (see *reportfile*).
Default: 30

convergence=*value* A high percent convergence is the point at which cluster means become stable during the iteration process. The default value is 98.0 percent. When clusters are being created, their means constantly change as pixels are assigned to them and the means are recalculated to include the new pixel. After all clusters have been created, *i.cluster* begins iterations that change cluster means by maximizing the distances between them. As these means shift, a higher and higher convergence is approached. Because means will never become totally static, a percent convergence and a maximum number of iterations are supplied to stop the iterative process. The percent convergence should be reached before the maximum number of iterations. If the maximum number of iterations is reached, it is probable that the desired percent convergence was not reached. The number of iterations is reported in the cluster statistics in the report file (see *reportfile*).
Default: 98.0

separation=*value* This is the minimum separation below which clusters will be merged in the iteration process. The default value is 0.0. This is an image-specific number (a "magic" number) that depends on the image data being classified and the number of final clusters that are acceptable. Its determination requires experimentation. Note that as the minimum class (or cluster) separation is increased, the maximum number of iterations should also be increased to achieve this separation with a high percentage of convergence (see *convergence*).
Default: 0.0

min_size=*value* This is the minimum number of pixels that will be used to define a cluster, and is therefore the minimum number of pixels for which means and covariance matrices

will be calculated.
Default: 17

reportfile=*name* The reportfile is an optional parameter which contains the result, i.e., the statistics for each cluster. Also included are the resulting percent convergence for the clusters, the number of iterations that was required to achieve the convergence, and the separability matrix.

NOTES

Running in command line mode, *i.cluster* will overwrite the output signature file and reportfile (if required by the user) without prompting if the files existed.

SEE ALSO

GRASS Tutorial: Image Processing

i.class, i.group, i.gensig, i.maxlik

AUTHORS

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
Tao Wen, University of Illinois at Urbana-Champaign, Illinois

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.colors – An imagery function that creates colors for imagery groups.
(GRASS Image Processing Program)

SYNOPSIS

i.colors

DESCRIPTION

i.colors allows the user to interactively assign red, green, and blue colors to the band files in an imagery group while viewing the display of the combined bands on the graphics monitor.

The user is asked to select an imagery group. Band files in the group that are currently assigned to the red, green, and blue color bands are displayed along the left edge of the user's graphics display frame, and a composite image of these bands is displayed in the right portion of the display frame. A small matrix appears in the lower left portion of the display frame, and reflects current red, green, and blue band file color assignments. Red, green, and blue color boxes appear next to the names of the image band files to which they are currently assigned. The user can change current color assignments by clicking the mouse cursor beside desired color assignments.

A menu along the bottom of the graphics display frame indicates that the user can also replot and enlarge the composite image on display, or quit the program, by clicking the mouse cursor on the desired option.

The GRASS program *i.group* can also be used to assign red, green, and blue colors to imagery group band files.

NOTES

This program is not yet complete. To assign colors to an imagery group use the GRASS program *i.group*.

This program is interactive and requires no command line arguments. The user must be running a graphics monitor to use this program.

SEE ALSO

GRASS Tutorial: Image Processing

d.mon, hsv.rgb.sh, i.composite, i.grey.scale, i.group, i.his.rgb, i.rgb.his, r.mapcalc, rgb.hsv.sh

AUTHOR

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

NAME

i.composite – An imagery function that creates a color composite image from three imagery band files specified by the user.
(GRASS Image Processing Program)

SYNOPSIS

i.composite

DESCRIPTION

i.composite creates a color composite image from three band files specified by the user. The user specifies the bands to be used by assigning a red, blue, and/or green color to each band. The resulting image is a raster map layer of raw spectral data composed of the three bands chosen by the user. The color composite can then be displayed, painted, or manipulated as would any raster map layer in GRASS.

The first prompt asks the user for the imagery group whose files are to be used.

The following menu is then displayed:

Please indicate which files to use for red, green, and blue colors. You may leave any color out. You may specify more than one color per file. However, each color may only be specified once. For example, to get a full color image, specify r,g,b for 3 different files. To get a grey scale image, specify rgb for a single file.

b__ spot.1
g__ spot.2
r__ spot.3
___ spotclass
___ spotreject

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The user is then allowed to check the choice of bands:

Colors assigned as follows:

RED: spot.3@mapsetname
GREEN: spot.2@mapsetname
BLUE: spot.1@mapsetname

Look ok? (y/n) [y]

The color table that is created has 1000 colors (10 saturation levels (or shades) per primary color (blue, green, red)). The number of colors that can be displayed at one time on a color graphics monitor will depend on the graphics monitor being used. For example, if the graphics monitor can only display 512 colors at one time, then the user must run the GRASS command **d.colormode mode=fixed** before displaying the raster map layer. The colors that cannot be displayed will be assigned to the nearest displayable color, and the raster map layer will retain its relative color accuracy. If the user is in *float* colormode, however, the raster map layer displayed on the graphics monitor will not accurately reflect the map's real color assignments.

The user is then asked to name the composite image raster map layer. The percentage completed is echoed to the screen and *r.support* files are created automatically.

NOTES

The user should always check the geographic region settings before running most imagery commands. It is very easy for the boundaries of the geographic region to be completely off the image. Before running *i.composite*, or other imagery commands, the user should probably set the geographic region to match that of the raster map layers to be read. This can be accomplished using option 4 of the *g.region* command.

This program is interactive and requires no command line arguments.

SEE ALSO

GRASS Tutorial: Image Processing

d.colormode, d.his, d.rast, g.region, i.colors, i.grey.scale, i.group, r.support

AUTHOR

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

NAME

i.fft – Fast Fourier Transform (FFT) for image processing.
(GRASS Image Processing Program)

SYNOPSIS

i.fft
i.fft help
i.fft input_image=*name* **real_image**=*name* **imaginary_image**=*name* [**range**=*value*]

DESCRIPTION

i.fft is an image processing program based on the algorithm given by Press (1988), with enhancements by Vali (1990), that processes a single input raster map layer (*input_image*) and constructs the real and imaginary Fourier components in frequency space.

Parameters:

input_image=*name*	Input raster map layer on which the fast Fourier transform is run.
real_image=*name*	Output real part arrays stored as raster map layer.
imaginary_image=*name*	Output imaginary part arrays stored as raster map layer.
range=*value*	Range of values used during fast Fourier transformation.

The real and imaginary components are stored as arrays of doubles in the *cell_misc* directory (for use in the inverse transform program, *i.ifft*), and are also scaled and formatted into the *real_image* and *imaginary_image* raster map layers for inspection, masking, etc. In these raster map layers the low frequency components are in the center and the high frequency components are toward the edges. The *input_image* need not be square; before processing, the X and Y dimensions of the *input_image* are padded with zeroes to the next highest power of two in extent (i.e., 256 x 256 is processed at that size, but 200 x 400 is padded to 256 x 512). The cell category values for viewing, etc., are calculated by taking the natural log of the actual values then rescaling to 255, or whatever optional range is given on the command line, as suggested by Richards (1986). A color table is assigned to the resultant map layer.

The current geographic region and mask settings are respected when reading the input file. The presence of a mask will, in general, make the resulting fast Fourier transform invalid, or at least difficult to interpret.

SEE ALSO

Numerical Recipes in C, by William H. Press, Cambridge University Press, 1988.

Remote Sensing Digital Image Analysis, by John A. Richards, Springer-Verlag, 1986.

Personnal communication, between progam author and Ali R. Vali, Space Research Center, University of Texas, Austin, 1990.

i.cca, i.class, i.ifft, i.pca

AUTHOR

David Satnik, GIS Laboratory, Central Washington University

NAME

i.gensig – Generates statistics for *i.maxlik* from raster map layer.
(GRASS Imagery Program)

SYNOPSIS

i.gensig
i.gensig help
i.gensig trainingmap=*name* **group=***name* **subgroup=***name* **signaturefile=***name*

DESCRIPTION

i.gensig is a non-interactive method for generating input into *i.maxlik.* It can be used as the first pass in the GRASS two-pass classification process (instead of i.cluster or i.class). It reads a raster map layer, called the training map, which has some of the pixels or regions already classified. *i.gensig* will then extract spectral signatures from an image based on the classification of the pixels in the training map and make these signatures available to *i.maxlik*.

The user would then execute the GRASS program *i.maxlik* to actually create the final classified map.

COMMAND LINE OPTIONS

Parameters

trainingmap
ground truth training map

This map must be prepared by the user in advance. Programs like *v.digit* or *r.digit* can be used to define representative areas of the classes the user defines to be in the image. Of course other methods could be devised by the user for creating this training map - *i.gensig* makes no assumption about the origin of this map layer. It simply creates signatures for the classes defined in the training map for the image to be classified (the image is specified in other options - see below).

group imagery group

This is the name of the group that contains the band files which comprise the image to be analyzed. The *i.group* command is used to construct groups of raster layers which comprise an image.

subgroup
subgroup containing image files

This names the subgroup within the group that selects a subset of the bands to be analyzed. The *i.group* command is also used to prepare this subgroup. The subgroup mechanism allows the user to select a subset of all the band files that form an image.

signaturefile
resultant signature file

This is the resultant signature file (containing the means and covariance matrices) for each class in the training map that is associated with the band files in the subgroup select (see above).

INTERACTIVE MODE

If none of the arguments are specified on the command line, *i.gensig* will interactively prompt for the names of these maps and files.

It should be noted that interactive mode here only means interactive prompting for maps and files. It does not mean visualization of the signatures that result from the process.

SEE ALSO

i.group for creating groups and subgroups.

v.digit, r.digit for interactively creating the training map.

i.cluster for unsupervised clustering as an alternative to i.gensig to create signatures.

i.class for a graphic/interactive as an alternative to i.gensig to create signatures.

AUTHOR

Michael Shapiro, U.S. Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.gensigset – generate statistics for *i.smap* from raster map layer.
(GRASS Imagery Program)

SYNOPSIS

i.gensigset
i.gensigset help
i.gensigset trainingmap=*name* **group=***name* **subgroup=***name* **signaturefile=***name* **[maxsig=***value]*

DESCRIPTION

i.gensigset is a non-interactive method for generating input into *i.smap.* It is used as the first pass in the a two-pass classification process. It reads a raster map layer, called the training map, which has some of the pixels or regions already classified. *i.gensigset* will then extract spectral signatures from an image based on the classification of the pixels in the training map and make these signatures available to *i.smap.*

The user would then execute the GRASS program *i.smap* to create the final classified map.

COMMAND LINE OPTIONS

Parameters

trainingmap
ground truth training map

This raster layer, supplied as input by the user, has some of its pixels already classified, and the rest (probably most) of the pixels unclassified. Classified means that the pixel has a non-zero value and unclassified means that the pixel has a zero value.

This map must be prepared by the user in advance. The user must use *r.digit,* a combination of *v.digit* and *v.to.rast,* or some other import/developement process (e.g., *v.in.transects)* to define the areas representative of the classes in the image.

At present, there is no fully-interactive tool specifically designed for producing this layer.

group imagery group

This is the name of the group that contains the band files which comprise the image to be analyzed. The *i.group* command is used to construct groups of raster layers which comprise an image.

subgroup
subgroup containing image files

This names the subgroup within the group that selects a subset of the bands to be analyzed. The *i.group* command is also used to prepare this subgroup. The subgroup mechanism allows the user to select a subset of all the band files that form an image.

signaturefile
resultant signature file

This is the resultant signature file (containing the means and covariance matrices) for each class in the training map that is associated with the band files in the subgroup selected.

maxsig maximum number of sub-signatures in any class
default: 10

The spectral signatures which are produced by this program are "mixed" signatures (see NOTES). Each signature contains one or more subsignatures (represeting subclasses). The algorithm in this program starts with a maximum number of subclasses and reduces this number to a minimal number of subclasses which are spectrally distinct. The user has the option to set this starting value with this option.

INTERACTIVE MODE

If none of the arguments are specified on the command line, *i.gensigset* will interactively prompt for the names of these maps and files.

It should be noted that interactive mode here only means interactive prompting for maps and files. It does not mean visualization of the signatures that result from the process.

NOTES

The algorithm in *i.gensigset* determines the parameters of a spectral class model known as a Gaussian mixture distribution. The parameters are estimated using multispectral image data and a training map which labels the class of a subset of the image pixels. The mixture class parameters are stored as a class signature which can be used for subsequent segmentation (i.e., classification) of the multispectral image.

The Gaussian mixture class is a useful model because it can be used to describe the behavior of an information class which contains pixels with a variety of distinct spectral characteristics. For example, forest, grasslands or urban areas are examples of information classes that a user may wish to separate in an image. However, each of these information classes may contain subclasses each with its own distinctive spectral characteristic. For example, a forest may contain a variety of different tree species each with its own spectral behavior.

The objective of mixture classes is to improve segmentation performance by modeling each information class as a probabilistic mixture with a variety of subclasses. The mixture class model also removes the need to perform an initial unsupervised segmentation for the purposes of identifying these subclasses. However, if misclassified samples are used in the training process, these erroneous samples may be grouped as a separate undesired subclass. Therefore, care should be taken to provided accurate training data.

This clustering algorithm estimates both the number of distinct subclasses in each class, and the spectral mean and covariance for each subclass. The number of subclasses is estimated using Rissanen's minimum description length (MDL) criteria [1]. This criteria attempts to determine the number of subclasses which "best" describe the data. The approximate maximum likelihood estimates of the mean and covariance of the subclasses are computed using the expectation maximization (EM) algorithm [2,3].

REFERENCES

[1] J. Rissanen, "A Universal Prior for Integers and Estimation by Minimum Description Length," *Annals of Statistics,* vol. 11, no. 2, pp. 417-431, 1983.

[2] A. Dempster, N. Laird and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Roy. Statist. Soc. B,* vol. 39, no. 1, pp. 1-38, 1977.

[3] E. Redner and H. Walker, "Mixture Densities, Maximum Likelihood and the EM Algorithm," *SIAM Review,* vol. 26, no. 2, April 1984.

SEE ALSO

i.group for creating groups and subgroups *v.digit, r.digit* for interactively creating the training map.

i.smap for creating a final classification layer from the signatures generated by *i.gensigset.*

AUTHOR

Charles Bouman, School of Electrical Engineering, Purdue University
Michael Shapiro, U.S. Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.grey.scale – An interactive imagery function that assigns a histogram contrast stretch grey scale color table to a raster map layer.
(GRASS Image Processing Program)

SYNOPSIS

i.grey.scale

DESCRIPTION

i.grey.scale is an interactive imagery function that assigns a histogram contrast stretch grey scale color table to a raster map layer. The histogram contrast stretch expands the original range of digital (category) values to utilize the full range of the user's graphics monitor.

The user is asked for the name of the raster map layer that needs a grey scale. When the raster map layer is displayed it will be displayed with a grey scale color scheme.

NOTES

This program is interactive and requires no command line arguments.

SEE ALSO

GRASS Tutorial: Image Processing

d.colormode, d.colors, d.colortable, i.colors, i.composite

AUTHOR

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

NAME

i.group – An imagery function that creates and edits groups and subgroups of (raster) imagery files. *(GRASS Image Processing Program)*

SYNOPSIS

i.group

DESCRIPTION

i.group allows the user to collect raster map layers in an imagery group by assigning them to user-named subgroups or other groups. This enables the user to run analyses on any combination of the raster map layers in a group. The user creates the groups and subgroups and selects the raster map layers that are to reside in them. Imagery analysis programs like *i.points*, *i.rectify*, *i.ortho.rectify* and others ask the user for the name of an imagery group whose data are to be analyzed. Imagery analysis programs like *i.cluster* and *i.maxlik* ask the user for the imagery group and imagery subgroup whose data are to be analyzed.

The first menu in the *i.group* program asks the user to select a *group*. If the group does not exist, the user will be asked if he or she would like to create a new group.

This program edits imagery groups. You may add raster map layers to, or remove such layers from, an imagery group. You may also create new groups.

Please enter the group to be created/modified

GROUP: _____________ (enter 'list' for a list of groups)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO EXIT)

If the word *list* is entered, groups that have already been created in the user's current LOCATION_NAME and MAPSET(S) will be listed. The second menu in *i.group* provides the user with the following options:

1. Select a different group
2. Edit group title
3. Include new raster (cell) files in the group or remove raster (cell) files from the group
4. Assign colors to the group
5. Create a new subgroup within the group

RETURN to exit

The options are described as follows:

Select a different group

If option number 1 is chosen, the following menu is displayed:

Please enter the group to be created/modified

GROUP: _____________ (enter 'list' for a list of groups)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO EXIT)

If the word *list* is entered, groups that have already been created in the current LOCATION_NAME and MAPSET(S) will be displayed.

Edit group title

If option number 2 is selected, an entry space is provided to type in the group title. This title is useful in identifying each group:

TITLE____

This option offers an opportunity to go back and change the entry if it is not correct by asking: **Look ok? (y/n)**.

Include new raster (cell) files in the group or remove raster (cell) files from the group

When choosing option number 3, the following menu is displayed:

LOCATION: *location* **GROUP:** *spot* **MAPSET:** *demo*

If you wish to delete a file from group [*spot*], remove the 'x' from in front of the file name.

x_ spot.1 in demo
x_ spot.2 in demo
x_ spot.3 in demo

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR Ctrl-C> TO CANCEL)

Next,a menu listing all the other raster map layers present in the current MAPSET(S) will be displayed:

LOCATION: *location* **GROUP:** *spot* **MAPSET:** *demo*

Please mark an 'x' by the files to be added in group [*spot*]

MAPSET: *demo*

x_ composite1
x_ spotclass1
__ spotclass2

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR Ctrl-C> TO CANCEL)

If more than one MAPSET is selected, menus for those mapsets will also be displayed. All raster map layers selected with an 'x' will be included in the group being updated.

The user will then have the opportunity to check the contents of the group that was just modified:

Group [*spot*] references the following raster files

spot.1	**in demo**
spot.2	**in demo**
spot.3	**in demo**
composite1	**in demo**
spotclass1	**in demo**

Look ok? (y/n)

If the user responds with the letter *y* then the following sentence is displayed on the screen:

Group [*spot*] updated!

And the main menu for *i.group* returns.

If the user responds *n*, the menu containing the group files after it was modified will be displayed and the user will be asked to place an *x* in front of those raster map layers that are to be removed from the group. Then, a menu listing all of the other raster map layers in the current MAPSET will be displayed again, and the user will be again asked to place an *x* in front of raster map layers to be included in the group. This gives the user the opportunity to correct mistakes or make changes in the choice of raster map layers to be selected in a group without exiting *i.group*.

Assign colors to the group

Option number 4 provides the following menu:

Please indicate which files to use for red, green, and blue colors. You may leave any color out. You may specify more than one color per file. However, each color may only be specified once. For example, to get a full color image, specify r,g,b for 3 different files. To get a grey scale image, specify rgb for a single file.

b__	**spot.1**
g__	**spot.2**
r__	**spot.3**
___	**composite1**
___	**spotclass1**

<<< r,g,b can only be specified once >>>

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

This menu allows you to select a color for each imagery band or for each file for display. Note, however, that composite images and classified images are already assigned colors during their creation.

An opportunity to change the choice of colors is offered after escaping the menu by:

Look ok? (y/n)

Create a new subgroup within the group

The following menu enables the user to create a subgroup out of any combination of raster map layers in the group. Any number of subgroups may be created by repeating the option.

LOCATION: *location* **MAPSET:** *spot*

GROUP: *spot1*
SUBGROUP: ___________ ('list' will show available subgroups)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

After selecting or creating a subgroup, this menu is displayed:

Mark an 'x' by the files to form subgroup [*123*]

x_	**spot.1**
x_	**spot.2**
x_	**spot.3**
__	**composite1**
__	**spotclass1**

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The user is then given the opportunity to check the contents of the subgroup:

Subgroup [*123*] references the following raster (cell) files

spot.1	**in demo**
spot.2	**in demo**
spot.3	**in demo**

Look ok? (y/n)

If the user responds with the letter *n*, the group menu will appear again enabling the user to select raster map layers to form the subgroup.

NOTES

The *i.group* options are only available for imagery map layers in the current LOCATION_NAME.

Subgroup names may not contain more than 12 characters.

This program is interactive and requires no command line arguments.

SEE ALSO

GRASS Tutorial: Image Processing

i.cluster, i.maxlik, i.points, i.rectify

AUTHOR

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

NAME

i.his.rgb – Hue-intensity-saturation (his) to red-green-blue (rgb) raster map color transformation function. *(GRASS Image Processing Program)*

SYNOPSIS

i.his.rgb
i.his.rgb help
i.his.rgb hue_input=*name* **intensity_input**=*name* **saturation_input**=*name* **red_output**=*name* **green_output**=*name* **blue_output**=*name*

DESCRIPTION

i.his.rgb is an image processing program that processes three input raster map layers as hue, intensity and saturation components and produces three output raster map layers representing the red, green and blue components of this data. The output raster map layers are created by a standard hue-intensity-saturation (his) to red-green-blue (rgb) color transformation. Each output raster map layer is given a linear gray scale color table. The current geographic region and mask settings are respected.

Parameters:

hue_input=*name*	Name of input raster map layer representing *hue*.
intensity_input=*name*	Name of input raster map layer representing *intensity*.
saturation_input=*name*	Name of input raster map layer representing *saturation*.
red_output=*name*	Output raster map layer representing the *red* component in the data.
green_output=*name*	Output raster map layer representing the *green* component in the data.
blue_output=*name*	Output raster map layer representing the *blue* component in the data.

NOTES

It is not possible to process three bands with *i.his.rgb* and then exactly recover the original bands with *i.rgb.his*. This is due to loss of precision because of integer computations and rounding. Tests have shown that more than 70% of the original cell values will be reproduced exactly after transformation in both directions and that 99% will be within plus or minus 1. A few cell values may differ significantly from their original values.

SEE ALSO

hsv.rgb.sh, i.colors, i.grey.scale, i.rgb.his, rgb.hsv.sh

AUTHOR

David Satnik, GIS Laboratory, Central Washington University

with acknowledgements to Ali Vali, Univ. of Texas Space Research Center, for the core routine.

NAME

i.ifft – Inverse Fast Fourier Transform (ifft) for image processing.
(GRASS Image Processing Program)

SYNOPSIS

i.ifft
i.ifft help
i.ifft real_image=*name* **imaginary_image**=*name* **output_image**=*name*

DESCRIPTION

i.ifft is an image processing program based on the algorithm given by Press (1988), and modified by Vali (1990), that converts real and imaginary frequency space images (produced by *i.fft*) into a normal image.

Parameters:

real_image=*name*	Input raster map layer for inversion fast Fourier transform, real part.
imaginary_image=*name*	Input raster map layer for inversion fast Fourier transform, imaginary.
output_image=*name*	Output inversion raster map layer after fast Fourier transformation.

The current mask is respected when reading the real and imaginary component files; thus, *r.mask* become a primary program for selecting the portion of the frequency space data to be included in the inverse transform. The GRASS program *d.digit* can be used to create masks while viewing the real or imaginary component image. When *i.ifft* is executed, it (automatically) uses the same GRASS region definition setting that was used during the original transformation done with *i.fft*.

The real and imaginary components are read from arrays of doubles in the *cell_misc* directory (produced by the forward transform program, *i.fft*), and the reconstructed image will preserve the cell value scaling of the original image processed by *i.fft*. No color table is assigned to the output map; one should be created before viewing the *output_image*.

SEE ALSO

Numerical Recipes in C, by William H. Press, Cambridge University Press, 1988.

Remote Sensing Digital Image Analysis, by John A. Richards, Springer-Verlag, 1986.

Personal communication, between program author and Ali R. Vali, Space Research Center, University of Texas, Austin, 1990.

i.cca, i.class, i.fft, i.pca

AUTHOR

David Satnik, GIS Laboratory, Central Washington University

NAME

i.in.erdas −Creates raster files from ERDAS files.
(GRASS Raster Program)

SYNOPSIS

i.in.erdas
i.in.erdas help
i.in.erdas [**−l**] **input=***name* **output=***name* [**trailer=***name*] [**bands=***value[,value,...*]] [**srow=***value*] [**scol=***value*] [**rows=***value*] [**cols=***value*]

DESCRIPTION

This program creates raster map files from ERDAS files. It creates one raster file for each selected band, up to a maximum of seven bands. GRASS color and category support files are created if an ERDAS trailer file is specified.

Only ERDAS version 7.4 or later files in 4-bit, 8-bit, or 16-bit formats are supported.
GRASS raster files will be named *prefix.band*

Note:

Remember that it is necessary to run: *r.support* to create the histogram or change the color table. *i.group* to associate the individual raster files as an image group.

COMMAND LINE OPTIONS

Flags:

-l List the ERDAS header only (no raster files created)

Parameters:

input ERDAS input file name

output output prefix of the GRASS raster files to be created.

trailer ERDAS trailer input file name

bands Selected bands to extract. (defaults to all bands)

srow Starting row. (defaults to first row in file)

scol Starting column. (defaults to first column in file)

rows Number of rows to extract. (defaults to all rows)

cols Number of columns to extract. (defaults to all columns)

SEE ALSO

r.support,
i.group

AUTHOR

M.L. Holko, USDA, SCS, NHQ-CGIS
Paul H. Fukuhara, USDA, SCS, NCG-GSS

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.maxlik – An imagery function that classifies the cell spectral reflectances in imagery data based on the spectral signature information generated by either *i.cluster*, *i.class*, or fIi.gensig.
(GRASS Image Processing Program)

SYNOPSIS

i.maxlik
i.maxlik help
i.maxlik [**-q**] **group**=*name* **subgroup**=*name* **sigfile**=*name* **class**=*name* [**reject**=*name*]

DESCRIPTION

i.maxlik is a maximum-likelihood discriminant analysis classifier. It can be used to perform the second step in either an unsupervised or a supervised image classification.

Either image classification methods are performed in two steps. The first step in an unsupervised image classification is performed by *i.cluster*; the first step in a supervised classification is executed by the *GRASS* program *i.class*. In both cases, the second step in the image classification procedure is performed by *i.maxlik*.

In an unsupervised classification, the maximum-likelihood classifier uses the cluster means and covariance matrices from the *i.cluster* signature file to determine to which category (spectral class) each cell in the image has the highest probability of belonging. In a supervised image classification, the maximum-likelihood classifier uses the region means and covariance matrices from the spectral signature file generated by *i.class*, based on regions (groups of image pixels) chosen by the user, to determine to which category each cell in the image has the highest probability of belonging.

In either case, the raster map layer output by *i.maxlik* is a classified image in which each cell has been assigned to a spectral class (i.e., a category). The spectral classes (categories) can be related to specific land cover types on the ground.

The program will run non-interactively if the user specifies the names of raster map layers, i.e., group and subgroup names, seed signature file name, result classification file name, and any combination of non-required options in the command line, using the form

i.maxlik[**-q**] **group**=*name* **subgroup**=*name* **sigfile**=*name* **class**=*name* [**reject**=*name*]

where each flag and options have the meanings stated below.

Alternatively, the user can simply type *i.maxlik* in the command line without program arguments. In this case the user will be prompted for the program parameter settings; the program will run foreground.

OPTIONS

Flags:

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

Parameters:

group=*name* The imagery group contains the subgroup to be classified.

subgroup=*name* The subgroup contains image files, which were used to create the signature file in the program *i.cluster*, *i.class*, or *i.gensig* to be classified.

sigfile=*name* The name of the signatures to be used for the classification. The signature file contains the cluster and covariance matrices that were calculated by the *GRASS* program *i.cluster* (or the region means and covariance matrices generated by *i.class*, if the user runs a supervised classification). These spectral signatures are what determine the categories (classes) to which image pixels will be assigned during the classification process.

class=*name* The name of a raster map holds the classification results. This new raster map layer will contain categories that can be related to land cover categories on the ground.

reject=*name* The optional name of a raster map holds the reject threshold results. This is the result of a chi square test on each discriminant result at various threshold levels of confidence to determine at what confidence level each cell classified (categorized). It is the reject threshold map layer, and contains one calculated confidence level for each classified cell in the classified image. One of the possible uses for this map layer is as a mask, to identify cells in the classified image that have the lowest probability of being assigned to the correct class.

NOTES

The maximum-likelihood classifier assumes that the spectral signatures for each class (category) in each band file are normally distributed (i.e., Gaussian in nature). Algorithms, such as *i.cluster, i.class,* or *i.gensig*, however, can create signatures that are not valid distributed (more likely with *i.class).* If this occurs, *i.maxlik* will reject them and display a warning message.

This program runs interactively if the user types *i.maxlik* only. If the user types *i.maxlik* along with all required options, it will overwrite the classified raster map without prompting if this map existed.

SEE ALSO

Grass Tutorial: Image Processing

i.class, i.cluster, i.gensig i.group, r.mask

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
Tao Wen, University of Illinois at Urbana-Champaign, Illinois

NAME

i.ortho.photo - An interactive imagery function for the ortho-rectification of imagery group files.
(GRASS Image Processing Program)

SYNOPSIS

i.ortho.photo

DESCRIPTION

i.ortho.photo allows the user to ortho-rectify imagery group files. An imagery group consists of several scanned aerial photographs (raster files) of a common area. Imagery groups can be created or modified using the GRASS Image Processing Program *i.group*, or using the first menu option described below. *i.ortho.photo* guides the user through the steps required to ortho-rectify the raster files in a single imagery group.

The first menu in *i.ortho.photo* provides the user with the following options:

Initialization Options
1) Select/Modify imagery group
2) Select/Modify imagery group target
3) Select/Modify target elevation model
4) Select/Modify imagery group camera

Transformation Parameter Computation
5) Compute image-to-photo transformation parameters
6) Compute photo-to-target transformation parameters
7) Initialize exposure station parameters

Ortho-rectification Option
8) Ortho-rectify imagery group raster files

RETURN to exit
>

The options are described as follows:

Select/Modify imagery group

The current imagery group is display at the top of the previous menu. You may select another (new or existing) imagery group for the ortho-rectification program using option (1). After choosing option (1) you will be prompted for the name of a new or existing imagery group. Option (1) using the GRASS Image Processing Program *i.group*, for creation or modification of imagery groups. For more information on imagery group creation or modification please the the GRASS manual page for *i.group*.

Select/Modify imagery group target

The target location and mapset may be selected or modified using option (2). After choosing option (2) you will be prompted for the names of the target location and mapset, where the ortho-rectified raster files will reside. The target location is also the location from which the elevation model (raster file) will be selected -- see option (3). Option (2) uses the GRASS Image Processing Program *i.target*, for selection or modification of the imagery group target location and mapset. For more information on imagery group selection or modification please the the GRASS manual page for *i.target*.

Select/Modify imagery group

Option (3) allows you to select the raster file from the target location to be used as the elevation model. The elevation model is required for both the computation of photo-to-target parameters (option 6) and for the ortho-rectification of the imagery group files (option 8). The raster file select for the elevation model should cover the entire area of the image group to be ortho-rectified. Currently, the elevation model raster file is expected to be in units of meters. DTED and DEM files are suitable for use as the elevation model in the ortho-rectification program. After selection option (3) you will be prompted for the name of the raster file in the target location that you want to use as the elevation model.

Select/Modify imagery group camera

Using option (4) you may select or create an camera reference file that will be used with the current imagery group. A camera reference file contains information on the internal characteristics of the aerial camera, as well as the geometry of the fiducial or reseau marks. The most important characteristic of the camera is its focal length. Fiducial or reseau marks locations are required to compute the scanned image to photo-coordinate transformation parameter (option 5). For a more detailed description of option (4) please see the GRASS manual page for *photo.camera*.

Compute image-to-photo transformation parameters

The scanned image to photo-coordinate transformation parameters are computed using option (5). In this interactive option you associate scanned reference points (fiducials, reseau marks, etc.) with their known photo coordinates from the camera reference file. Complete documentation for this option is available under the manual entry *photo.2image*.

Compute photo-to-target transformation parameters

The photo to target transformation parameters are compute using option(6). Here control points are marked on one or more imagery group files and associated with there known UTM and elevation coordinates. Complete documentation for this option is available under the manual entry *photo.2target*.

Select initial exposure station parameters

If option (7) is selected, initial camera exposure station parameters and initial variances may be selected or modified. Complete documentation for this option is available under the manual entry *photo.init*.

Ortho-photo imagery group files

Option (8) is used to perform the actual image ortho-rectification after all of the transformation parameters have been computed. Ortho-rectified raster files will be created in the target location for each selected imagery group file. You may select either the current window in the target location or the minimal bounding window for the ortho-rectified image. Complete documentation for this option is available under the manual entry *photo.rectify*.

NOTES

i.ortho.photo currently requires the elevation model to be in meters, and the target location to be a UTM coordinate system.

SEE ALSO

photo.camera[2], *photo.2image[2]*, *photo.2target[2]*, *photo.init[2]*, *photo.rectify[2]*

AUTHOR

Mike Baba DBA Systems, Inc.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.pca – Principal components analysis (pca) program for image processing.
(GRASS Image Processing Program)

SYNOPSIS

i.pca
i.pca help
i.pca input=*name,name*[,*name,name,...*] **output**=*name* **rescale**=*min,max*

DESCRIPTION

i.pca is an image processing program based on the algorithm provided by Vali (1990), that processes n (2 >= n) input raster map layers and produces n output raster map layers containing the principal components of the input data in decreasing order of variance ("contrast"). The output raster map layers are assigned names with .1, .2, ...n suffixes. The current geographic region definition and mask settings are respected when reading the input raster map layers. When the rescale option is used, the output files are rescaled to fit the min,max range.

Parameters:

input=*name,name*[,*name,name,...*]
Name of two or more input raster map layers.

output=*name* The output raster map layer name to which suffixes are added. Each output raster map layer is assigned this user-specified *name* with a numerical (.1, .2, ...n) suffix.

rescale=*min,max* The optional output category range. (Default: 1,255) If rescale=0,0, no rescaling is performed on output files.

Richards (l986) gives a good example of the application of principal components analysis (pca) to a time series of LANDSAT images of a burned region in Australia.

SEE ALSO

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

Vali, Ali R., Personal communication, Space Research Center, University of Texas, Austin, 1990.

i.cca, i.class, i.fft, i.ifft, m.eigensystem, r.covar, r.mapcalc

AUTHOR

David Satnik, GIS Laboratory, Central Washington University
Major modifications for GRASS 4.1 were made by Olga Waupotitsch and Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

NAME

i.points − An imagery function that enables the user to mark coordinate system points on an image to be rectified and then input the coordinates of each point for creation of a coordinate transformation matrix. The transformation matrix is needed as input for the GRASS program *i.rectify*.
(GRASS Image Processing Program)

SYNOPSIS

i.points

DESCRIPTION

i.points is an imagery function that enables the user to mark points on a (raster) image to be rectified and then input the geographic coordinates of each point for calculation of a coordinate transformation matrix. *i.points* must be followed by use of the GRASS program *i.rectify*, which rectifies the image using the transformation matrix coefficients calculated by *i.points*.

Rectification is the mapping (transformation) of an image from one coordinate system to another. The geometry of an image extracted into a GRASS LOCATION having an x,y coordinate system is not planimetric. To create a planimetric image, that is, to convert the x,y coordinate system into a standard coordinate system (for example, the UTM coordinate system or the State Plane coordinate system), points from a map having the standard coordinates must be associated with the same points on the image to be rectified. *i.points* enables the user to mark points on an image and input the standard coordinates for that point. *i.points* then calculates a least squares regression using the two coordinate systems (x,y and standard) for the marked points. A matrix containing transformation coefficients is the output file for *i.points*.

During the process of marking points and entering map coordinates, the user can compute the RMS (root mean square) error for each point entered. *i.points* does this by calculating the transformation equation (the same one that is calculated in the GRASS program *i.rectify*), and then plugging these results into an equation for RMS error.

i.points offers a zoom option to locate precisely the point to be marked on an image. This program also offers the user the option of acquiring standard coordinates for a marked point from a map layer in the target data base.

i.target must be run before running *i.points* to enable the PLOT RASTER option to be used and to identify a target data base LOCATION_NAME and MAPSET for the rectified image. To run *i.points*, a graphics monitor is required.

The procedure for marking points, entering coordinates, and calculating RMS error is described below.

The first prompt in the program asks the user for the imagery group to be registered. Note that if *i.target* is not run before *i.points*, the *i.points* program will display the following error message:

ERROR: Target information for group [*spot*] missing
Please run *i.target* for group [*spot*]

After entering the group to be registered the terminal screen displays the message:

Use mouse now...

The graphics monitor displays the following screen:

imagery filename (mag)	*target filename (mag)*
QUIT ZOOM PLOT RASTER ANALYZE	

A pop-down menu like that shown below will be superimposed on the left half of the screen:

Double click on raster map layer to be plotted Double click here to cancel

Mapset *demo*	
spotclass	**spot.1**
composite	**spot.2**
spot.3	

Any single raster map layer in the imagery group may be used on which to mark points, and the user can mark points on more than one raster map layer in the imagery group to accumulate the suggested minimum number of 12 points. Any raster map layer in the imagery group can be rectified (using *i.rectify*) based on the transformation matrix computed from these points.

The imagery file chosen by the user is displayed in the upper left quadrant of the screen.

ZOOM

To magnify the displayed file, the user must place the mouse cross hairs on the word ZOOM. The following menu will then be displayed at the bottom of the screen:

Cancel	**Box**	**Point**	**Select type of ZOOM**

The user has the option of identifying the zoom region either by using the mouse to make a box, or by using the mouse to mark the two diagonal points of the desired region. The terminal screen will display a mouse button menu to guide the user in identifying the corner points of the region.

MARKING POINTS

To mark the points on the image that correspond to the points on a standard coordinate system map, the user must place the mouse cross hairs on the corresponding location on the image to be marked and press the left hand button on the mouse. A diamond shaped symbol will be marked on the image. The user's terminal will display the following menu:

Point 1 marked on the image at **East: 1023.77** **North: -164.41**	
Enter coordinates as east north:	

The user then enters the easting and northing (separated by a space) for the point marked on the image. If the user wishes not to enter a coordinate, he or she may simply hit RETURN to return control to the mouse; the marked point then disappears.

PLOT RASTER

In addition to acquiring reference points from a standard map, the user has the option of acquiring the reference points from a raster map layer in the target data base LOCATION_NAME. The data base raster map layer is displayed by placing the mouse cross hairs on the words PLOT RASTER. The following line is then displayed at the bottom of the graphics monitor:

Cancel	**Indicate which side should be plotted**

Which side of the graphics monitor is to be plotted is indicated by placing the mouse cross hairs on the half of the graphics monitor screen that the user would like to use, and pressing the left mouse button. The following pop-down menu will be superimposed on the half of the screen that was chosen:

Double click on raster (cell) map layer **to be plotted** **Double click here to cancel**

Mapset *demo*	
tm.rectified	
tm.classified	
Mapset PERMANENT	
elevation	**geology**
slope	**soils**
aspect	
roads	
streams	
airfields	

After the raster map layer is displayed the following message appears at the bottom of the graphics monitor:

input method -->	**keyboard**	**screen**

If the user wishes to use the plotted raster map layer only as a comparative reference, then the keyboard can be chosen as the means to input coordinates corresponding to the marked points on the image. This is done by placing the mouse cross hairs on the word KEYBOARD and pressing the left button on the mouse.

If the user selects the SCREEN option, then points marked on the image will automatically be associated with the coordinates from the corresponding points on the target data base map layer. In this option, when the user marks a point on the image, the following menu is displayed at the terminal:

Point 5 marked on the image at **East: 1023.77** **North: -164.41** **Point located at** **East: 679132.57** **North: 4351080.67**	
use mouse now...	

The user then uses the mouse to mark a corresponding point on the displayed image from the target data base. The coordinates for the target data base map layer are automatically saved as the coordinates corresponding to the marked point on the image.

ANALYZE

After a number of points have been marked (4 to 7), the user can check the RMS error of the points marked on the image. This is done by placing the mouse cross hairs on the word ANALYZE at the bottom of the graphics monitor. An error report resembling that shown below is superimposed on the graphics monitor:

	error			image		target	
#	**row**	**col**	**target**	**east**	**north**	**east**	**north**
1	**0.0**	**-0.9**	**1.0**	**1048.5**	**-144.8**	**679132.5**	**4351080.6**
2	**0.4**	**1.0**	**1.3**	**2153.1**	**-567.2**	**684314.7**	**4399001.4**
3	**-1.2**	**-0.5**	**.6**	**1452.8**	**-476.5**	**567841.4**	**3457682.8**
4	**1.1**	**0.5**	**1.3**	**1034.0**	**-109.2**	**677573.8**	**4352626.4**
5	*-2.7*	*14.0*	*14.2*	*1048.6*	*-144.9*	*679132.6*	*4351080.7*
	overall	**rms**	**error:**	**4.46**			

The following menu then appears at the bottom of the graphics monitor:

DONE	PRINT FILE	Double click on point to be included/excluded

The RMS error for the image is given under the column titled "error" and subtitled "row" and "col". In the above report, point number 1 is 0.0 rows and -0.9 columns from the predicted location calculated from the transformation equation. The RMS error for the target raster map layer is listed under the heading "target". This is the RMS error for the east and north coordinates of the target map layer, but it is presented in the table using one general value. The overall RMS error is displayed at the bottom of the screen in meters. Points that create high RMS error are displayed in red on the graphics monitor (represented here in italics).

The location of the point marked on the imagery group file is given under the heading "image" and the subheadings "east" and "north". The location of the point in the target data base is given under the heading "target" and the subheadings "east" and "north". If the user would like to exclude or include a point, this can be accomplished by placing the mouse cross hairs on the point number to be included (if the point is absent) or excluded (if the point is displayed) and pressing the left button on the mouse twice. When a point is excluded, it is not afterwards included in the calculation of the RMS error, or included in the final transformation matrix. However, it can be retrieved within *i.points* at any time by double clicking with the mouse as described above.

QUIT

To end the *i.points* program place the mouse cross hairs on the word QUIT; the marked points (including coordinates) will be saved.

NOTES

A good rule of thumb is to mark at least 12 to 15 points which are evenly distributed over the entire imagery group file in order to obtain an accurate transformation equation for the rectification process. The RMS error may increase with more points added, but the transformation equation will be more accurate.

An RMS error of less than or equal to approximately one resolution unit (pixel or cell) for the image being rectified is generally considered acceptable.

In order to use a digitizer with *i.points*, at least one digitizer driver besides "none" (the on-screen digitizer) must be available in the digitcap file.

This program is interactive and requires no command line arguments.

SEE ALSO

GRASS Tutorial: Image Processing

g.mapsets, i.group, i.rectify, i.target

AUTHOR

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

NAME

i.quantize − An interactive imagery function that creates a raster map layer whose color table is based on the red, green, and blue color values present in existing, user-specified imagery group files.
(GRASS Image Processing Program)

SYNOPSIS

i.quantize

DESCRIPTION

i.quantize is an interactive imagery function that allows the user to create a new raster map layer whose color table values are based on the red, green, and blue color values present in (as many as) three raster map layers in an imagery group.

The user is first asked to enter the name of the imagery group from whose map layers red, green, and blue color values are to be extracted. The user is then shown a listing of imagery files in the specified group, and is asked to indicate which files in the group are to be used for red, green, and blue colors in the new map layer. Each color (red, green, and blue) must be specified once and only once.

The user is also asked to assign a name to the new raster map layer output.

NOTES

This program is interactive and requires no command line arguments.
The input files must ALL be within the range of 0-255.

SEE ALSO

GRASS Tutorial: Image Processing

d.colormode, d.colors, d.colortable, hsv.rgb.sh, i.colors, i.composite, i.grey.scale, i.group, i.his.rgb, i.rgb.his, r.mapcalc, rgb.hsb.sh, r.colors

AUTHOR

David Gerdes, U.S. Army Construction Engineering Research Laboratory
Ref. Paul Heckbert "Image Quantization" Siggraph Proceedings 1982

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.rectify – An imagery function that rectifies an image by computing a coordinate transformation for each cell (pixel) in the image using the transformation coefficient matrix created by the GRASS program *i.points*.
(GRASS Image Processing Program)

SYNOPSIS

i.rectify

DESCRIPTION

i.rectify rectifies an image by using the transformation coefficient matrix created by *i.points*. Rectification is the process by which the geometry of an image is made planimetric. This is accomplished by mapping (transforming) an image from one coordinate system to another. In *i.rectify*, the coefficients computed by *i.points* are used in an equation to convert x,y coordinates to standard map coordinates for each cell in the image. The result is an image with a standard map coordinate system. Upon completion of the program the rectified image is deposited in a previously targeted GRASS LOCATION_NAME and MAPSET.

The first prompt in the program asks the user for the name of the group containing the raster map layers to be rectified.

The user is then asked to select the map layer(s) within the group to be rectified:

Please select the file(s) to rectify by naming an output file

spot.1 in demo	**.........**
spot.2 in demo	**.........**
spot.3 in demo	**.........**
spotclass in demo	*spotrectify*..
spotreject in demo	**.........**

(Enter list by any name to get a list of existing raster files)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

More than one raster map layer may be rectified at a time. Each raster map layer should have a unique output file name.

Next the user is asked to select one of two geographic region settings:

Please select one of the following options
1. Use the current region in the target location
2. Determine the smallest region which covers the image

i.rectify will only rectify that portion of the image that occurs within the chosen geographic region setting. Only that portion will be relocated in the target database. It is therefore important to check the current geographic region settings in the target location if choice number one is selected.

NOTES

i.rectify uses nearest neighbor resampling during the transformation.

i.rectify uses a linear affine transformation:

x' = ax + by +c
y' = Ax + Bt +C

The a,b,c,A,B,C are determined by least squares regression based on the control points entered. This transformation applies scaling, translation and rotation. It is NOT a general purpose rubber-sheeting, nor is it ortho-photo rectification using a DEM, not second order polynomial, etc. It can be used if (1) you have geometrically correct images, and (2) the terrain or camera distortion effect can be ignored.

Use *i.ortho.photo* for photos for which you have camera information and a DEM, or *i.rectify2* if you want a 2nd or 3rd order polynomial transformation.

i.rectify will run in the background and notify the user by mail when it is finished. The process may take an hour or more depending on the size of the image, the number of files, and the size of the geographic region definition.

The rectified (raster) image will be located in the target LOCATION when the program is completed. The original unrectified raster map layers are not modified or removed.

This program is interactive and requires no command line arguments.

SEE ALSO

GRASS Tutorial: Image Processing

i.group, i.points, i.target

AUTHOR

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

NAME

i.rectify2 **–**

The i.rectify2 routine is the imagery function that rectifies an image by computing a coordinate transformation for each pixel in the image based on the control points created by the GRASS program i.points or i.vpoints. The i.rectify2 program uses the control points identified in i.points or i.vpoints to calculate a transformation matrix based on a first, second , or third order polynomial and then converts x,y cell coordinates to standard map coordinates for each pixel in the image. The result is a planimetric image with a transformed coordinate system (i.e., a different coordinate system than before it was rectified).

The i.vpoints or i.points program must be run before i.rectify2, and both programs are required to rectify an image. An image must be rectified before it can reside in a standard coordinate LOCATION, and therefore be analyzed with the other map layers in the standard coordinate LOCATION. Upon completion of i.rectify2, the rectified image is deposited in the target standard coordinate LOCATION. This LOCATION is selected using the i.target program of GRASS.

(GRASS Image Processing Program)

SYNOPSIS

i.rectify2

DESCRIPTION

Program Prompts

The first prompt in the program asks for the name of the group containing the files to be rectified.

```
Enter the group containing files to be rectified
Enter 'list' for a list of existing imagery groups
Enter 'list -f' for a verbose listing
Hit RETURN to cancel request
>
```

This is the same imagery group that was selected in i.points or i.vpoints and the group that contains the cell files with the marked points and their associated map coordinates. You are then asked to select the cell file(s) within the group to be rectified:

```
Please select the file(s) to rectify by naming an output file

        spot1.1 in mapsetname    ............
        spot1.2 in mapsetname   ............
        spot1.3 in mapsetname    ............
          spotclass1 in mapsetname spotrectify1..

        spotreject1 in mapsetname     ............

(enter list by any name to get a list of existing cell files)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE (OR<Ctrl-C> TO CANCEL
```

More than one cell file may be rectified at a time. Each cell file should be given a unique output file name.

Next, you are asked to select one of two windows regions:

Please select one of the following options

1. Use the current window in the target location
2. Determine the smallest window which covers the image

>

The i.rectify2 program will only rectify that portion of the image or cell file that occurs within the chosen window region, and only that portion of the cell file will be relocated in the target database. It is important therefore, to check the current mapset window in the target LOCATION if choice number one is selected.

If you are rectifying a file with plans to patch it to another file using the GRASS program patch[1] or patch[2], choose option number one, the current window in the target location. This window, however, must be the default window for the target LOCATION. When a file being rectified is smaller than the default window in which it is being rectified, zeros are added to the rectified file. Patching files of the same size that contain 0/non-zero data, eliminates the possibility of a no-data line the patched result. This is because, when the images are patched, the zeros in the image are "covered" with non-zero pixel values. When rectifying files that are going to be patched, rectify all of the files using the same default window.

By selecting the 1st ORDER option, the user may select the order of transformation desired:

Select order of transformation --> 1st Order 2nd Order 3rd Order

The program will immediately recalculate the RMSE and the number of points required.

Polynomial Transformation Matrix

The ANALYZE function has been changed to support calculating the registration coefficients using a first, second, or third order transformation matrix. The number of control points required for a selected order of transformation (represented by þnþ) is ((n + 1) ∗ (n + 2) / 2) or 3, 6, and 10 respectively. It is strongly recommended that one or more additional points be identified to allow for an overly- determined transformation calculation which will generate the Root Mean Square (RMS) error values for each included point. The RMS error values for all the included control points are immediately recalculated when the user selects a different transformation order from the menu bar. The polynomial equations in the new i.rectify2 program are performed using a modified Gaussian elimination method instead of the Cramerþs rule method used in the i.rectify routine.

Program Execution

i.rectify2 will run in the background and notify you by mail when it is finished.

Note: The rectified image or rectified cell files will be located in the target LOCATION when the program is completed. The original unrectified files are not modified or removed.

This program is interactive and requires no command line arguments.

SEE ALSO

GRASS Tutorial: Image Processing

i.group, i.points, i.vpoints, i.rectify, i.target

AUTHOR

William R. Enslin, Michigan State University Center for Remote Sensing

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of

GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.rgb.his – Red-green-blue (rgb) to hue-intensity-saturation (his) function for image processing.
(GRASS Image Processing Program)

SYNOPSIS

i.rgb.his
i.rgb.his help
i.rgb.his red_input=*name* **green_input**=*name* **blue_input**=*name*
hue_output=*name* **intensity_output**=*name* **saturation_output**=*name*

DESCRIPTION

i.rgb.his is an image processing program that processes three input raster map layers as red, green, and blue components and produces three output raster map layers representing the hue, intensity, and saturation of the data. The output raster map layers are created by a standard red-green-blue (rgb) to hue-intensity-saturation (his) color transformation. Each output raster map layer is given a linear gray scale color table. The current geographic region definition and mask settings are respected.

Parameters:

red_input=*name*	Input raster map layer representing the red component.
green_input=*name*	Input raster map layer representing the green component.
blue_input=*name*	Input raster map layer representing the blue component.
hue_output=*name*	Output raster map layer representing hue.
intensity_output=*name*	Output raster map layer representing intensity.
saturation_output=*name*	Output raster map layer representing saturation.

SEE ALSO

hsv.rgb.sh, i.his.rgb, rgb.hsv.sh

AUTHOR

David Satnik, GIS Laboratory, Central Washington University,

with acknowledgements to Ali Vali, Univ. of Texas Space Research Center, for the core routine.

NAME

i.smap – An imagery function that performs contextual image classification using sequential maximum a posteriori (SMAP) estimation.
(GRASS Imagery Program)

SYNOPSIS

i.smap
i.smap help
i.smap [**−mq**] **group=***name* **subgroup=***name* **signaturefile=***name* **[blocksize=***value]* **output=***name*

DESCRIPTION

The *i.smap* program is used to segment multispectral images using a spectral class model known as a Gaussian mixture distribution. Since Gaussian mixture distributions include conventional multivariate Gaussian distributions, this program may also be used to segment multispectral images based on simple spectral mean and covariance parameters.

i.smap has two modes of operation. The first mode is the sequential maximum a posteriori (SMAP) mode [1,2]. The SMAP segmentation algorithm attempts to improve segmentation accuracy by segmenting the image into regions rather than segmenting each pixel separately (see NOTES).

The second mode is the more conventional maximum likelihood (ML) classification which classifies each pixel separately, but requires somewhat less computation. This mode is selected with the -m flag (see below).

COMMAND LINE OPTIONS

Flags

-m Use maximum likelihood estimation (instead of smap). Normal operation is to use SMAP estimation (see NOTES).

-q Run quietly, without printing messages about program progress. Without this flag, messages will be printed (to stderr) as the program progresses.

Parameters

group imagery group

The imagery group that defines the image to be classified.

subgroup
imagery subgroup

The subgroup within the group specified that specifies the subset of the band files that are to be used as image data to be classified.

signaturefile
imagery signaturefile

The signature file that contains the spectral signatures (i.e., the statistics) for the classes to be identified in the image. This signature file is produced by the program *i.gensigset* (see NOTES).

blocksize
size of submatrix to process at one time
default: 128

This option specifies the size of the "window" to be used when reading the image data.

This program was written to be nice about memory usage without influencing the resultant classification. This option allows the user to control how much memory is used. More memory may mean faster (or slower) operation depending on how much real memory your machine has and how much virtual memory the program uses.

The size of the submatrix used in segmenting the image has a principle function of controlling memory usage; however, it also can have a subtle effect on the quality of the segmentation in the smap mode. The smoothing parameters for the smap segmentation are estimated separately for each submatrix. Therefore, if the image has regions with qualitatively different behavior, (e.g., natural woodlands and man-made agricultural fields) it may be useful to use a submatrix small enough so that different smoothing parameters may be used for each distinctive region of the image.

The submatrix size has no effect on the performance of the ML segmentation method.

output output raster map.

The name of a raster map that will contain the classification results. This new raster map layer will contain categories that can be related to landcover categories on the ground.

INTERACTIVE MODE

If none of the arguments are specified on the command line, *i.smap* will interactively prompt for the names of the maps and files.

NOTES

The SMAP algorithm exploits the fact that nearby pixels in an image are likely to have the same class. It works by segmenting the image at various scales or resolutions and using the course scale segmentations to guide the finer scale segmentations. In addition to reducing the number of misclassifications, the SMAP algorithm generally produces segmentations with larger connected regions of a fixed class which may be useful in some applications.

The amount of smoothing that is performed in the segmentation is dependent of the behavior of the data in the image. If the data suggests that the nearby pixels often change class, then the algorithm will adaptively reduce the amount of smoothing. This ensures that excessively large regions are not formed.

REFERENCES

[1] C. Bouman and M. Shapiro," "Multispectral Image Segmentation using a Multiscale Image Model," *Proc. of IEEE Int'l Conf. on Acoust., Speech and Sig. Proc.,* pp. III-565 - III-568, San Francisco, California, March 23-26, 1992.

[2] C. Bouman and M. Shapiro, "A Multiscale Random Field Model for Bayesian Image Segmentation," submitted to the *IEEE Trans. on Image Processing.*

SEE ALSO

i.group for creating groups and subgroups

i.gensigset to generate the signature file required by this program

AUTHOR

Charles Bouman, School of Electrical Engineering, Purdue University
Michael Shapiro, U.S. Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.tape.mss – An imagery function that extracts Multispectral Scanner (MSS) imagery data from half-inch tape.
(GRASS Image Processing Program)

SYNOPSIS

i.tape.mss

DESCRIPTION

i.tape.mss is a program that extracts Multispectral Scanner (MSS) imagery data from tape.

This program must be run in a LOCATION_NAME with a x,y coordinate system (i.e., a coordinate system with projection 0). For further information regarding this LOCATION_NAME refer to the manual entry for *imagery*.

The first prompt in *i.tape.mss* asks the user for the tape device name. This is sometimes /dev/rmt0 (for a half-inch tape with a tape density of 1600), but this varies with each machine.

The next prompt is:

Please mount and load tape, then hit RETURN -->

IMAGE IDENTIFICATION MENU

The first menu in the program asks the user for information about the data.

please enter the following information

Tape Identification: ______________

Image Description: ______________

Title for the Extracted Raster (Cell) Files: ______________

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

This program automatically enters the scene ID number and the date of the image into the field for Tape Identification. The sun angles are automatically entered into the field for Image Description.

The second menu is:

MSS TAPE EXTRACTION
please select the desired tape window (geographic region definition) to extract

first row: _______(1-2984)
last row: _______(1-2984)

first col: _______(1-3548)
last col: _______(1-3548)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The numbers in parentheses are the total number of rows and columns on the tape including filler (zeros). This information and additional information can also be obtained by running the GRASS program *i.tape.mss.h* which reads the header information on an MSS tape. Any subset of the image on tape may be extracted. For a discussion of row and column extraction see the subheading titled ROW AND COLUMN EXTRACTION below.

The next menu is:

please make an x by the bands you want extracted

_____ 1
_____ 2
_____ 3
_____ 4

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

MSS imagery has 4 bands, but the user may want to extract only a subset of these bands. See the subheading in this entry titled ROW AND COLUMN EXTRACTION.

The user then is asked to enter the prefix/group for the band files to be created. This name will precede each band file extracted into GRASS. For example, if three bands are extracted the following three (raster) band files will result:

prefixname.1
prefixname.2
prefixname.3

Whatever *prefixname* is specified will also automatically become the name for the imagery group file being created. Each image (i.e., each run of *i.tape.mss*) should be given a unique prefix/group name.

The extraction process will begin by first skipping the number of specified files, advancing to the starting row, and then reading the tape. The percent completion of the extraction is displayed on the screen. If more than one tape is required to store the image, the program will pause and inform the user to mount the next tape.

The extracted (raster) band files will be listed as raster map layers available in the current MAPSET and may be displayed using the GRASS commands *d.display*, *d.rast* or *i.points*.

NOTES

After extracting an image from tape, the geographic region definition in the x,y coordinate LOCATION_NAME will be set based upon the extracted rows and columns from the tape. The relationship between the image rows and columns and the geographic coordinates of the region is discussed in the manual entry for *imagery*.

This program is interactive and requires no command line arguments.

ROW AND COLUMN EXTRACTION

The display options in GRASS allow the user to locate rows and columns on the digital image. If enough disk space is available, one band of an entire image, or one band of a portion of an image known to contain the area of interest, can be extracted and displayed. The *measurements* option in *d.display*, or *d.where* (following the use of *d.rast*) will echo x and y coordinates to the screen. (These coordinates will display negative numbers in the north-south direction, but ignoring the negative sign will yield the row number.) See the *imagery* manual entry for further explanation.

If a photograph of the digital image is available, the rows and columns to be extracted can be determined from it by associating inches with the total number of known rows and columns in the scene. For example, if the total length of the photograph is 12 inches, the total number of rows on the tape is 2000, and the northwest corner of the area of interest begins 2 inches from the top of the photo, then:

$$12'' / 2000 \text{ rows} = 2'' / x \text{ rows}$$
$$x = 333.333$$

The northwest corner of the area of interest starts at row 333. The starting row, ending row, starting column, and ending column can be calculated in this manner.

SEE ALSO

GRASS Tutorial: Image Processing

d.display, d.rast, d.where, i.group, i.points, i.tape.mss.h, i.tape.other, i.tape.tm, imagery

AUTHOR

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

NAME

i.tape.mss.h **–** An imagery function that extracts header information from LANDSAT Multispectral Scanner (MSS) imagery data stored on half-inch tape.
(GRASS Image Processing Program)

SYNOPSIS

i.tape.mss.h
i.tape.mss.h help
i.tape.mss.h *tape_drive_name*

DESCRIPTION

i.tape.mss.h reads the header information on a Multispectral Scanner (MSS) tape. This program reads the specified input file (the computer-compatible tape *tape_drive_name*), and by default displays the output to the user's terminal. The user may redirect output to a file by using the UNIX redirection mechanism. For example:

i.tape.mss.h /dev/rmt0 > *h.out*

The name of the tape drive depends on the computer being used.

This program can be run either non-interactively or interactively. The user can run the program by specifying program arguments on the command line. Alternately, the user can simply type **i.tape.mss.h** on the command line, without program arguments. In this event, the program will prompt the user to enter a tape device name using the standard user interface described in the manual entry for *parser*.

SEE ALSO

GRASS Tutorial: Image Processing

i.tape.mss and *parser*

AUTHOR

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

NAME

i.tape.other – An imagery function that extracts scanned aerial imagery (NHAP, etc.) and satellite imagery (TM, SPOT, etc) from half-inch or 8mm tape.
(GRASS Image Processing Program)

SYNOPSIS

i.tape.other

DESCRIPTION

i.tape.other is a generic program that extracts imagery from tape using the tape description that is input by the user.

This program must be run in a LOCATION_NAME with a x,y coordinate system (i.e., a coordinate system with projection 0). For further information regarding this LOCATION_NAME refer to the manual entry for *imagery*.

The first prompt in *i.tape.other* asks the user for the tape device name. This is sometimes /dev/rmt0 (for a density of 1600), but this varies with each machine.

The next prompt is:

Please mount and load tape, then hit RETURN -->

IMAGE IDENTIFICATION MENU

The first menu in the program asks the user for information about the data.

please enter the following information

Tape Identification: ______________

Image Description: ______________

Title for the Extracted Raster (Cell) Files: ______________

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

TAPE LAYOUT MENU

The next menu asks for the physical layout of the tape.

GENERIC TAPE EXTRACTION

```
tape layout
    0__     number of tape files to be skipped
    0__     number of records in the remaining files to be skipped
band files
    0__     number of bands on the tape
data format
    ___     band sequential (BSQ) | mark one with an x
    ___     band interleaved (BIL) |
    0__     if you select BSQ format and all the bands are in a single file,
            enter the total number of records in the file. Otherwise enter 0

    0__     length (in bytes) of the longest record on the tape
    1__     blocking factor of data in the file

     AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
                   (OR <Ctrl-C> TO CANCEL)
```

number of tape files to be skipped

If there are files at the beginning of the tape which are not image data, they can be skipped. Sometimes information that comes with a tape will indicate the number of header files or records on the tape. The GRASS utility *m.examine.tape* will also provide this information. The *record length* is the number of columns in the image, while the *number of records* is the number of rows in the image (not always correct, see *blocking factor of data in the file* below). NHAP imagery and usually most scanned aerial imagery do not have tape header files, but this should be checked. TM imagery has one header file that contains imagery format of data files and parameters of data aquisition. SPOT imagery has two files that should be skipped on the first tape, and one file to be skipped on the second tape (of a two-tape set).

number of records in the remaining files to be skipped

If the files which contain the image begin with non-image data, these records can also be skipped. This is usually 0 for most data types. SPOT imagery stored in 1600bpi has one header record in the image file on each tape that should be skipped.

number of bands on the tape

Most aerial imagery have 3 bands, but satellite simulator data may have more. TM data has seven bands and SPOT has three bands as a standard, respectively. The total number of bands on the tape should be specified here, not just the number that will be extracted.

data format

The two formats that imagery data are most commonly stored in are called *band interleaved* format (BIL) and *band sequential* format (BSQ). In BIL format, each record on the tape contains one line for one band of data. If the data contains three bands, then the first five records will look like this:

band 1, line 1
band 2, line 1
band 3, line 1
band 1, line 2
band 2, line 2

In BSQ format, all lines of one band are stored together on a tape, followed by all lines of another band, followed by all lines of the next band, etc. These data are stored as if they were in a one band BIL format:

band 1, line 1
band 1, line 2
band 1, line 3
.
.
.
band 2, line 1
band 2, line 2
.
.
band 2, line 156
band 2, line 157

Each pixel contains one byte and there is one line per record. BSQ format is the format that is usually created by optical scanning devices when they scan photographs, but not all digitized aerial imagery are stored in this format. The format of the data is usually written on the exterior of the tape; this should be checked.

length (in bytes) of the longest record on the tape
This must be set to the number of bytes in the longest data record. It is used to determine how large a buffer to use for reading the tape. This value can be obtained using *m.examine.tape.*

blocking factor of data in the file
This is the number of lines combining into one physical record on tape. It is usually one for most of imageries, i.e., one line per physical record. However, considering on data compressing and tape memory saving, sometimes more than one lines are combined into one physical record on the tape. This number may be written on the exterior of the tape, otherwise the user need to experiment on this number by running of *i.tape.other*.

BAND EXTRACTION MENU
The user is then asked to mark an *x* beside the bands to be extracted. See the subheading in this entry entitled ROW AND COLUMN EXTRACTION.

please mark an x by the bands you want extracted

_____1
_____2
_____3
_____4

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

PREFIX/GROUP NAME
The user is asked to enter the prefix/group for the (raster) band files to be created. This name will precede each band file extracted into GRASS. For example, if three bands are extracted the following three band files will result:

prefixname.1
prefixname.2
prefixname.3

The specified *prefixname* will also automatically become the name for the imagery group file being created. Each image (i.e., each run of *i.tape.other)* should be given a unique prefix/group name.

ROW AND COLUMN MENU

Finally, the starting row, ending row, starting column and ending column are required. This allows the user to extract any subset of the image from the tape.

EXTRACT
please select desired tape window (geographic region definition) to extract

start row:_0_
end row:_0_

start col:_0_
end col:_0_

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The extraction process will begin by first skipping the number of specified files, advancing to the starting row, and then reading the tape. The percent completion of the extraction is displayed on the screen.

Following the extraction, the extracted band files will be listed as raster map layers available in the current MAPSET. These raster map layers may be displayed individually using the GRASS commands *d.display*, *d.rast* or *i.points*.

NOTES

This program can be used for extraction of TM, SPOT and other types of data from tape; however, the user must supply information to the program on how the image data is laid out on the tape. For example, the image data may be padded with surrounding extra rows and/or columns; the user may wish to skip over these rows and columns and extract only the actual image data from the tape. *i.tape.other* does not know where image data actually begins on the tape; the user must tell the program what portion of the tape data to extract. Often, information on the orientation and layout of the image data on tape will be included on a printout accompanying any tape data received by the user; however, this may not always be the case. The user may need to experiment with various runs of *i.tape.other* before extracting the portions of an image actually desired.

After extracting an image from tape, the geographic region in the x,y coordinate LOCATION_NAME will be set, based upon the extracted rows and columns from the tape. The relationship between the image rows and columns and the coordiates bounding the geographic region is discussed in the *imagery* manual entry.

This program is interactive and requires no command line arguments.

ROW AND COLUMN EXTRACTION

The display options in GRASS allow the user to locate rows and columns on the digital image. If enough disk space is available, one band of an entire image, or one band of a portion of an image known to contain the area of interest, can be extracted and displayed. The *measurements* option in *d.display*, or *d.where* (following a run of *d.rast*) will echo x and y coordinates to the screen. (These coordinates will display negative numbers in the north-south direction but ignoring the negative sign will yield the row number. See the *imagery* manual entry for further explanation.)

If a photograph of the digital image is available, the rows and columns to be extracted can be determined from it by associating inches with the total number of known rows and columns in the scene. For example, if the total length of the photograph is 12 inches, the total number of rows on the tape is 2000, and the northwest corner of the area of interest begins 2 inches from the top of the photo, then:

12" / 2000 rows = 2" / x rows
x = 333.333

The northwest corner of the area of interest starts at row 333. The starting row, ending row, starting column, and ending column can be calculated in this manner.

SEE ALSO

GRASS Tutorial: Image Processing

d.display, d.rast, d.where, i.group, i.points, i.tape.mss, i.tape.mss.h, i.tape.tm, i.tape.tm.fast, imagery, m.examine.tape

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
Tao Wen, University of Illinois at Urbana-Champaign, Illinois

NAME

i.tape.spot – An imagery function that extracts SPOT imagery from half-inch tape.
(GRASS Image Processing Program)

SYNOPSIS

i.tape.spot

DESCRIPTION

i.tape.spot is a program that extracts SPOT imagery from 9-track, half-inch tape.

This program must be run in a LOCATION_NAME with an x,y coordinate system (i.e., a coordinate system with projection 0). For further information regarding this LOCATION_NAME type refer to the *imagery* manual entry.

The first prompt in *i.tape.spot* asks the user for the tape device name.

Enter tape device name:

This is sometimes /dev/rmt0 (for a half-inch tape having a density of 1600 bpi), but this varies with each machine.

The next prompt is:

Mount SPOT tape and hit RETURN -->

IMAGE IDENTIFICATION MENU

The first menu in the program asks the user for information about the data.

Please enter the following information

TAPE IDENTIFICATION: ______

IMAGE DESCRIPTION: ______

TITLE FOR THE EXTRACTED CELL (RASTER) FILES: ______

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

This program automatically reads the satellite name, tape product code, instrument name, interleaving indicator, spectral mode, preprocessing level and work order number into the field for TAPE IDENTIFICATION. The mission, path, row, scene shift, scene center date and time, orientation, incidence, azimuth, elevation angle, and absolute calibration coefficients and offsets are automatically entered into the field for IMAGE DESCRIPTION. User can type in any other messages into the two sections and a description as the title of raster map layer. The second menu is:

SPOT IMAGE EXTRACT
Please select region of the image to extract

start row: 0______(1-3002)
end row: 0______(1-3002)

start col: 0______(1-3166)
end col: 0______(1-3166)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The numbers in parentheses are the total number of rows and columns on the tape including zeros (filler). This information and additional information can also be obtained by running the program *m.examine.tape*. *m.examine.tape* will read any tape and provide the user with the number of files on a tape, the number of records on a tape, and the record lengths. Any subset of the image on the tape may be extracted. For a discussion of row and column extraction see the subheading entitled ROW AND COLUMN EXTRACTION below.

The next menu is:

Please mark an x by the bands you want extracted

_____ 1
_____ 2
_____ 3

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

SPOT imagery has three bands, but the user may want to extract a subset of these bands. See the subheading in this entry entitled ROW AND COLUMN EXTRACTION.

The user then is asked to enter the prefix/group name for the raster band files to be created. This name will precede each band file extracted into GRASS. For example, if two bands are extracted the following two band files will result:

prefixname.1
prefixname.2

The specified *prefixname* will also automatically become the name for the imagery group file being created. Each image or subset (i.e., each run of *i.tape.spot*) should be given a unique prefix/group name.

The extraction process will begin by first skipping a number of files which are not data or not requested, advancing to the first band requested, forwarding to the requested column, and then reading the data. After extracting the requested rows and columns from each band, the program creates support files for the raster band map layer. The percent completion of the extraction is displayed on the screen. Because sometimes SPOT imagery is very large and is stored in multiple tape sets, the program is designed to read image by pausing when the tape need to be changed and inform the user to mount and load next tape. The number of tapes required to store one scene depends on the number of bytes per inch (bpi) in which the data are stored.

The extracted band files will be listed as raster map layers available in the current MAPSET and may be displayed using the GRASS commands *d.display*, *d.rast*, or *i.points*.

ROW AND COLUMN EXTRACTION

The display options in GRASS allow the user to locate rows and columns on the digital image. If enough disk space is available, one band of an entire image or one band of a portion of an image known to contain the area of interest, can be extracted and displayed. The *measurements* option in *d.display*, or *d.where* (following use of *d.rast*) will echo x and y coordinates to the screen. (These coordinates will display negative numbers in the north-south direction, but ignoring the negative sign will yield the row number.) See the *imagery* manual entry for further explanation.

Each scene of a SPOT image contains filler on both the left and right sides of the quad. The user may want to identify the row and column numbers in order to exclude the filler. This filler will otherwise be extracted with the image and take up unnecessary disk space.

If a photograph of the digital image is available, the rows and columns to be extracted can be determined from it by associating inches with the total number of known rows and columns in the scene. For example, if the total length of the photograph is 12 inches, the total number of rows on the tape is 2000, and the northwest corner of the area of interest begins 2 inches from the top of the photo, then:

$$12'' / 2000 \text{ rows} = 2'' / x \text{ rows}$$
$$x = 333.333$$

The northwest corner of the area of interest starts at row 333. The starting row, ending row, starting column, and ending column can be calculated in this manner.

NOTES

After extracting an image from tape the geographic region definition in the x,y coordinate LOCATION_NAME will be set based upon the extracted rows and columns from the tape. The relationship between the image rows and columns and the coordinates of the geographic region is discussed in the *imagery* format manual entry.

This program is interactive and requires no command line arguments.

ROW AND COLUMN EXTRACTION

SEE ALSO

GRASS Tutorial: Image Processing

d.display, d.rast, d.where, i.group, i.points, i.tape.mss, i.tape.mss.h, i.tape.other, i.tape.tm, imagery, m.examine.tape

AUTHOR

Tao Wen, University of Illinois at Urbana-Champaign, Illinois

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.tape.tm – An imagery function that extracts LANDSAT Thematic Mapper (TM) imagery from half-inch tape.
(GRASS Image Processing Program)

SYNOPSIS

i.tape.tm

DESCRIPTION

i.tape.tm is a program that extracts LANDSAT Thematic Mapper (TM) imagery from half-inch tape.

This program must be run in a LOCATION_NAME with a x,y coordinate system (i.e., a coordinate system with projection 0). For further information regarding this LOCATION_NAME refer to the *imagery* manual entry.

The first prompt in *i.tape.tm* asks the user for the tape device name. This is sometimes /dev/rmt0 (for a half-inch tape having a density of 1600 bpi), but this varies with each machine.

The next prompt is:

Please mount and load tape, then hit RETURN -->

IMAGE IDENTIFICATION MENU

The first menu in the program asks the user for information about the data.

please enter the following information

Tape Identification: ______________

Image Description: ______________

Title for the Extracted Raster (Cell) Files: ______________

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

This program automatically enters the scene ID number into the field for Tape Identification. The mission, path, row, quadrant, date, and whether the image is corrected is automatically entered into the field for Image Description.

The second menu is:

THEMATIC MAPPER EXTRACT
please select the desired tape window (geographic region definition) to extract

first row: _______(1-2984)
last row: _______(1-2984)

first col: _______(1-4220)
last col: _______(1-4220)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The numbers in parentheses are the total number of rows and columns on the tape including zeros (filler). This information and additional information can also be obtained by running the program *m.examine.tape*. *m.examine.tape* will read any tape and provide the user with the number of files on a tape, the number of records on a tape, and the record lengthes. Any subset of the image on the tape may be extracted. For a discussion of row and column extraction see the subheading entitled ROW AND COLUMN EXTRACTION below.

The next menu is:

please make an x by the bands you want extracted

_____ 1
_____ 2
_____ 3
_____ 4
_____ 5
_____ 6
_____ 7

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

TM imagery has 7 bands, but the user may want to extract only a subset of these bands. See the subheading in this entry entitled ROW AND COLUMN EXTRACTION.

The user then is asked to enter the prefix/group for the raster band files to be created. This name will precede each band file extracted into GRASS. For example, if three bands are extracted the following three band files will result:

prefixname.1
prefixname.2
prefixname.3

The specified *prefixname* will also automatically become the name for the imagery group file being created. Each image or quad (i.e., each run of *i.tape.tm*) should be given a unique prefix/group name.

The extraction process will begin by first skipping the number of specified files, advancing to the first band requested, and then reading the tape. After extracting the requested rows and columns for each band, the program creates support files for the raster band map layer. The percent completion of the extraction is displayed on the screen. Because TM imagery is divided into four quads and is stored in multiple tape sets, the program is designed to read one quad at a time. The number of tapes required to store one quad depends on the number of bytes per inch in which the data is stored. If more than one tape is required to store one quad, the program will pause and inform the user to mount the next tape.

The extracted band files will be listed as raster map layers available in the current MAPSET and may be displayed using the GRASS commands *d.display*, *d.rast* or *i.points*.

NOTES

After extracting an image from tape the geographic region definition in the x,y coordinate LOCATION_NAME will be set based upon the extracted rows and columns from the tape. The relationship between the image rows and columns and the coordinates of the geographic region is discussed in the *imagery* manual entry.

This program is interactive and requires no command line arguments.

ROW AND COLUMN EXTRACTION

The display options in GRASS allow the user to locate rows and columns on the digital image. If enough disk space is available, one band of an entire image or, one band of a portion of an image known to contain the area of interest, can be extracted and displayed. The *measurements* option in *d.display*, or *d.where* (following use of *d.rast*) will echo x and y coordinates to the screen. (These coordinates will display negative numbers in the north-south direction, but ignoring the negative sign will yield the row number.) See the *imagery* manual entry for further explanation.

Each quad of a TM image contains filler on both the left and right sides of the quad. The user may want to identify the row and column numbers in order to exclude the filler. This filler will otherwise be extracted with the image and take up unnecessary disk space.

If a photograph of the digital image is available, the rows and columns to be extracted can be determined from it by associating inches with the total number of known rows and columns in the scene. For example, if the total length of the photograph is 12 inches, the total number of rows on the tape is 2000, and the northwest corner of the area of interest begins 2 inches from the top of the photo, then:

12" / 2000 rows = 2" / x rows
x = 333.333

The northwest corner of the area of interest starts at row 333. The starting row, ending row, starting column, and ending column can be calculated in this manner.

SEE ALSO

GRASS Tutorial: Image Processing

d.display, d.rast, d.where, i.group, i.points, i.tape.mss, i.tape.mss.h, i.tape.other, imagery, m.examine.tape

AUTHOR

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

NAME

i.tape.tm.fast – An imagery function that extracts Thematic Mapper (TM) imagery from tape media *(GRASS Image Processing Program)*

SYNOPSIS

i.tape.tm.fast
i.tape.tm.fast help
i.tape.tm.fast [**-q**] **input**=*name* **group**=*name* **bands**=*value*[,*value*,...] [**rows**=*firstrow-lastrow*] [**cols**=*firstcol-lastcol*] [**title**=*name*]

DESCRIPTION

i.tape.tm.fast is a program that extracts TM imagery from tape media with different blocking factors (its value indicates how many rows are combined into one physical record on the tape).

i.tape.tm.fast must be run in a LOCATION_NAME with a (x,y) coordinate system (i.e., a coordinate system with projection 0). For further information regarding the LOCATION_NAME type, please refer to the *imagery* manual entry.

i.tape.tm.fast reads the blocking factor from the header file as well as other parameters, such as gains and offsets for each band, map projection, sun elevation and azimuth, etc., and writes into history file (depending upon the contents of the header file).

OPTIONS

This program can be run either non-interactively or interactively. It will be run non-interactively if the user specifies the name of input device, the name of output group file, bands to be extracted, and optionally other parameters (see below) on the command line using the form:

> **i.tape.tm.fast** [**-q**] **input**=*name* **group**=*name* **bands**=*value*[,*value*,...] [**rows**=*firstrow-lastrow*] [**cols**=*firstcol-lastcol*] [**title**=*name*]

where the *input* should be the device name on which the tape media are mounted. The *group* is an imagery group which will store the extracted TM imagery. The *bands* value in a list separated by commas is the bands the user wants to extract from the imagery. The *rows* and *cols* represent the region which the user wishes to extract, where the default is whole imagery. At last an optional *title* is for information only. Alternatively, the program will be run interactively if the user types only *i.tape.tm.fast*; in this case the program will prompt the user for parameter values 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.

Parameters:

input=*name* The name of the device on which the tape media containing the imagery files mounted.

group=*name* The name of the group, which will store the imagery extracted from tape media.

bands=*value*[,*value*,...]
The bands the user wishes to extracted from the tape media.
Options: 1-7

rows=*firstrow-lastrow*
The values of first and last row of the extracting region.
Default: full imagery

cols=*firstcol-lastcol* The values of first and last column of the extracting region.

Default: full imagery

title=*name* The title of the extracting imagery.
Default: TM Imagery File Extracted from Tape

NOTES

Running in command line mode, *i.tape.tm.fast* will overwrite the group file and support files without prompting if the files existed.

SEE ALSO

GRASS Tutorial: Image Processing

i.group, i.tape.mss, i.tape.mss.h, i.tape.other, i.tape.tm, i.tape.spot, imagery m.examine.tape

AUTHOR

Tao Wen, University of Illinois at Urbana-Champaign, Illinois

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.target – An interactive imagery function that establishes a GRASS target location and mapset for an imagery group.
(GRASS Image Processing Program)

SYNOPSIS

i.target

DESCRIPTION

i.target targets an imagery group to a GRASS data base location name and mapset. During the imagery program *i.rectify*, a location name and mapset are required into which to transfer the rectified file just prior to completion of the program; *i.target* enables the user to specify this location. *i.target* must be run before *i.points* and *i.rectify*.

The first prompt in the program asks the user for the name of the imagery group that needs a target. The imagery group must be present in the user's current mapset.

The following menu asking for the target LOCATION NAME and MAPSET is displayed:

Please select the target LOCATION and MAPSET for group <*spot*>

CURRENT LOCATION: *location*______
CURRENT MAPSET: *demo*__________

TARGET LOCATION:_________
TARGET MAPSET:___________

(Enter list for a list of location names or mapsets within a location)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL

An imagery group may be targeted to any GRASS location.

NOTES

This program is interactive and requires no command line arguments.

SEE ALSO

Grass Tutorial: Image Processing

i.group, i.points, i.rectify

AUTHOR

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

NAME

i.vpoints – Identifies coordinate pairs of points from a vector map or keyboard entry and corresponding points in an image.

DESCRIPTION

This program enables the user to identify coordinate pairs of points from a vector map or keyboard entry and corresponding points in an image to be rectified. The map coordinate values of each point are used to calculate a transformation matrix. The operator may then use the *i.rectify* or *i.rectify2* program to rectify the image using the transformation matrix coefficients calculated from the control point file created in *i.vpoints*. The *i.rectify* program performs only a first order transformation of the image whereas *i.rectify2* can rectify an image using either a first, second, or third order polynomial equation.

The first step is to display the unrectified image and corresponding vector map data. The operator would then mark corresponding control point locations on the image and map. To identify the precise location of a point to be marked, i.vpoints has a zoom option. In addition to marking control points on an image to be rectified and inputting their world coordinate values using the keyboard, i.vpoints has the option to simultaneously display vector map data available in the targeted database, and identify on the vector map the location of the corresponding marked points. When this option is chosen, the coordinate values are input automatically. Any GRASS map layer or vector map in the targeted database LOCATION can be displayed using i.vpoints. The i.vpoints program also has the capability of overlaying (i.e., warping) the vector data onto the raster image to visually check the accuracy of the registration based on the current set of active control points. During the process of marking points and entering map coordinates, the user can compute the RMS (root mean square) error for each point entered. The i.vpoints program does this by calculating a transformation equation (the same one that is calculated in the GRASS program i.rectify2). Coefficients are computed for the equation. The coefficients are then used in the equation along with the x,y coordinates of the marked points. The results are plugged into an equation for RMS error. The interpretation of RMS error is described in the ANALYZE subsection.

The procedures for marking control points (registration points), displaying vector map layers, overlaying vector maps onto the raster image, and calculating RMS error are described in the following sections .

To enter the program (the i.vpoints program requires the use of a graphics monitor) type i.vpoints . . .

GRASS - GRID > i.vpoints

the first prompt in the program asks for the imagery group to be registered . . .

Enter imagery group to be registered
Enter 'list' for a list of existing imagery groups
Enter 'list -f' for a verbose listing
Hit RETURN to cancel request
>

for example, >list might produce the following response

<list>
Available groups
- -
test
- -
hit RETURN to continue -->

whereas, >list -f might produce the following response

```
<list -f>
Available groups
- - - - - - - - - - - - - - - - - - - - - - - - -
test
    photo in PERMANENT
    res2 in tifftest
- - - - - - - - - - - - - - - - - - - - - - - - -
hit RETURN to continue -->
```

The imagery group entered above should contain the files that you wish to rectify. After entering the group to be registered, the terminal screen displays the message:

```
>test
<test>
Use mouse now . . .
```

And the color graphics monitor displays the following screen:

Any single file in the imagery group may be used to mark points, and points can be marked on more than one file in the imagery group to accumulate the suggested minimum number of points (3 for a 1st order transformation, 6 for a 2nd order, and 10 for a 3rd order). Any file in the imagery group can be subsequently rectified (using i.rectify2) based on the transformation matrix computed from these points.

The chosen file is displayed in the upper left quadrant of the monitor at a default magnification based on the extent of the current active window.

RASTER IMAGE

The raster image option on the menu at the bottom of the window allows the user to display any single file in the imagery group in the upper right quadrant of the window screen. The option provides the same file selection pick list as is presented when you first enter the i.vpoints program. When you select this option, the program will erase the data contained in all of the four quadrant windows and will reinitialize all program values.

VECTOR MAPS

The vector maps option on the menu at the bottom of the screen allows the user to display vector map data in the upper right quadrant of the screen. After selecting the vector map layer to display, a menu selection bar appears along the bottom on the screen. This pick list is used to select the line color (blue, gray, green, red, white, or yellow) for the selected vector data layer.

Refresh

The refresh option on the main menu allows the user to "refresh" or re-draw the displayed vector data. This function will erase all outlines showing the limits of previously zoomed areas. A "yes/no" roompt will appear:

```
Refresh Map ?     NO   YES
```

Zoom

To enlarge a raster or vector image, place the mouse cross hairs on the word zoom on the main menu and press the left button. The following menu will be displayed at the bottom. of the screen:

CANCEL BOX POINT Select type of zoom

You have the option to identify the map extent of the zoom window using either the mouse to define a box, or the mouse to mark a center point from which to enlarge the image. The box option first prompts you to identify a starting corner for the zoom region and then allows you to define the area to be zoomed using a rubber band box. The prompts appear as follows...

CANCEL Mark the first corner of region

CANCEL Define the region

After marking the first corner of the region to be enlarged, hold down the left button and move the mouse to change the size and shape of the rubber band box. After defining the area to be enlarged, press the right button to accept it.

The point method for enlarging an image will display a mouse menu to guide you in selecting the appropriate enlargement. To enlarge or reduce the magnification factor, place the cursor on the "+" or "-" box and press the left button on the mouse.

You may zoom either the raster or the vector display.

Upon accepting the new region limits, the raster or vector data are redisplayed in either the lower left (raster) or lower right (vector) windows.

The extent of the zoomed area is outlined on the unzoomed image in the main window area.

While the main menu is displayed, you can mark corrosponding control points on the raster and vector images or enter map coordinates from the keyboard. If you are using coordinates taken from a reference map, circle these points and then use whatever means you have available to identify as precisely as possible the coordinate values for these points. Digitizing software is recommended, especially GRASS 3,0 program digit/1/. Once you have determined the standard coordinates (for example, UTM's) of each circled point, you are ready to mark the points on the displayed image.

To mark the points on the image, that correspond to the points on the standard coordinate map, place the mouse cross hairs on the point on the image to be marked (you will probably have to ZOOM to find the exact spot) and press the left hand button on the mouse. A diamond shaped symbol will be marked on the image. The text monitor will display the following screen:

Point 1 marked on the image at
East: 1023.77
North: -164.41

Enter coordinates as east north:

Analyze

After a number of points have been marked (a minimum of 4 for a 1st order transformation, 7 for a 2nd order, and 11 for a 3rd order), the RMS error of the points marked on the image can be checked. This is done by placing the cross hairs on the word ANALYZE on the main menu at the bottom of the monitor. The following error report is superimposed on the monitor:

```
     error              image              target
 #   col  row  target        east     north        east
 north

 1   -0.9 0.0  1.0      1048.5      -144.8 679132.5      4351080.6
 2   1.0  0.4  1.3      2153.1      -567.2 684314.7      4399001.4
 .
 .
 .

     Overall rms error  76.85
```

The RMS error for the image being rectified is recorded under the column "error" and subtitled "row' and "col". In the above report, the marked point number 1 is 0.0 rows and -0.9 columns from the predicted location calculated by the transformation equation.

The RMS error for the target database map is recorded under the heading "error" and the subheading "target". This is the RMS error for the east and the north coordinate values of the target map, but it is represented in the table using one general value.

The overall RMS error for the image is displayed at the bottom of the screen in meters. Points that generate a high RMS error are displayed in red on the monitor.

The x,y coordinate values of the point marked on the image being rectified are recorded under the heading "image" and the subheadings "east" and "north".

The standard coordinate values of the point in the target database are recorded under the heading "target" and the subheadings "east" and "north".

If the user would like to exclude or include a point, this can be accomplished by placing the mouse cross hairs on the point number to be included (if the point is absent) or excluded (if the point is displayed) and then pressing the left button on the mouse twice. When a point is excluded, it is not included in the calculation of the RMS error, or included in the final transformation matrix. However, it can be retrieved within i.vpoints at any time by double clicking with the mouse as described above.

The following menu appears at the bottom of the monitor:

DONE PRINT FILE OVERLAY DELETE ON Transformation - -> 1st ORDER Double click on point to be DELETED

Selecting DELETE ON will toggle the option to DELETE OFF, the toggle option is used to allow the user to physically remove a control point from the POINTS file instead of just flagging it as an non-active reference point.

Overlay

Overlay allows the user to overlay the vector map(s) onto the raster image. Overlay can be used to warp (register) and display the selected vector file data on top of the raster image contained in the upper left window of the color screen. An inverse coordinate transformation is performed using the currently active order of transformation (i.e., first, second, or third).

Overlay vectors on raster image NO YES

By selecting the 1st ORDER option, the user may select the order of transformation desired:

Select order of transformation --> 1st Order 2nd Order 3rd Order

The program will immediately recalculate the RMSE and the number of points required.

Quit

To exit the i.vpoints program, place the mouse cross hairs on the word QUIT at the bottom of the monitor and all of the marked points (including coordinates) will be saved.

SEE ALSO

GRASS Tutorial: Image Processing

g.mapsets, i.group, i.points, i.rectify, i.rectify2, i.target

AUTHOR

William R. Enslin, Michigan State University Center for Remote Sensing

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

i.zc – Zero-crossing "edge detection" raster function for image processing.
(GRASS Image Processing Program)

SYNOPSIS

i.zc
i.zc help
i.zc input_map=*name* **zc_map**=*name* [**width**=*value*] [**threshold**=*value*] [**orientations**=*value*]

DESCRIPTION

i.zc is an image processing program used for edge detection. The raster map produced shows the location of "boundaries" on the input map. Boundaries tend to be found in regions of changing cell values and tend to run perpendicular to the direction of the slope. The algorithm used for edge detection is one of the "zero-crossing" algorithms and is discussed briefly below.

This program will be run interactively if the user types **i.zc** without program arguments on the command line. In this event, the program will prompt the user for parameter values using the standard interface described in the manual entry for *parser*. Alternately, the user can run the program non-interactively by specifying program parameter values on the command line.

Parameters:

input_map=*name*	Name of input raster map layer.
zc_map=*name*	Name of raster map layer to be used for zero-crossing values.
width=*value*	This parameter determines the x-y extent of the Gaussian filter. The default value is 9; higher and lower values can be tested by the user. Increasing the width will result in finding "edges" representing more gradual changes in cell values. Default: 9
threshold=*value*	This parameter determines the "sensitivity" of the Gaussian filter. The default value is 10; higher and lower values can be tested by the user. Increasing the threshold value will result in fewer edges being found. Default: 10
orientations=*value*	This value is the number of azimuth directions the cells on the output raster map layer are categorized into (similar to the aspect raster map layer produced by the *r.slope.aspect* program). For example, a value of 16 would result in detected edges being categorized into one of 16 bins depending on the direction of the edge at that point. Default: 1

The current region definition and mask settings are respected when reading the input map.

NOTES

The procedure to find the "edges" in the image is as follows: 1) The Fourier transform of the image is taken, 2) The Fourier transform of the Laplacian of a two-dimensional Gaussian function is used to filter the transformed image, 3) The result is run through an inverse Fourier transform, 4) The resulting image is traversed in search of places where the image changes from positive to negative or from negative to positive, 5) Each cell in the map where the value crosses zero (with a change in value greater than the threshold value) is marked as an edge and an orientation is assigned to it. The resulting raster map layer is output.

SEE ALSO

Personal communication between program author and Ali R. Vali, Space Research Center, University of Texas, Austin, 1990.

i.fft, i.ifft, r.mapcalc, r.mfilter, r.slope.aspect and *parser*

AUTHOR

David Satnik, GIS Laboratory, Central Washington University

NAME

m.datum.shift – Datum shift program.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.datum.shift lat=*dd.mm.ss{n| s}* lon=*dd.mm.ss{e| w}* is=*input_spheroid* os=*output_spheroid* dx=*xshift* dy=*yshift* dz=*zshift*

DESCRIPTION

m.datum.shift returns geographic coordinates based on a different spheroid (and datum) than the one used to obtain the original coordinates.

The input and output spheroids, *is* and *os*, are the spheroids for two different datums. The input spheroid is the one on which the original coordinates are based. The output spheroid is that on which the resultant coordinates will be based. The "shifting" occurs between the two datums. The shift values, *dx*, *dy*, and *dz*, are constants. They indicate the mean differences between points in the second datum versus the first as measured in meters.

The list of spheroids available is somewhat dynamic. It may not contain exactly the ones listed below. To determine the current list of possible spheroids, type in the command:

m.datum.shift lat=0n lon=0w dx=0 dy=0 dz=0 is=help os=help

A list of available spheroids will be printed on the screen. If the spheroid desired is not on the list, the values for the semi-major axis and the eccentricity squared for the spheroid may be entered in place of a spheroid name in the following format:

s=a=*semi-major_axis*,**e=***eccentricity_squared*

SOME POSSIBLE SPHEROIDS
(*The on-line listing includes only the spheroid names*)

Spheroid	Commonly used for:	Semi-major axis	Eccentricity sqrd
australian	Australia	6378160.0	0.0066945419
bessel	Japan	6377739.155	0.0066743722
clark66	N. America	6378206.4	0.006768658
clark80	France, Africa	6378249.145	0.0068035113
everest	India, Burma	6377276.345	0.0066378466
international	Europe	6378388.0	0.00672267
wgs72	worldwide coverage	6378135.0	0.006694317778

EXAMPLE

m.datum.shift lat=0n lon=175w is=clark66 os=wgs72 dx=-22 dy=157 dz=176

Results:
lat=0.00.05.72999N
lon=174.59.55.004133W

NOTES

Essentially, the program follows these steps. The original point, as defined by a latitude and a longitude, is converted to geocentric coordinates. The shift values are added to the geocentric coordinates. The summed values are then converted to latitude and longitude based on the output spheroid.

For a brief discussion of spheroids and datums see *m.ll2u*. For a brief discussion of geocentric coordinates see *m.ll2gc*.

This remains under testing is still an experimental program. It is part of an initial effort to incorporate geographic coordinates into GRASS.

SEE ALSO

m.gc2ll, m.ll2gc, m.ll2u, m.u2ll

AUTHOR

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

NAME

m.dem.examine – Provides a terse description of USGS Digital Elevation Model (DEM) data files stored on 1/2-inch magnetic tape.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.dem.examine input=*name* **blocksize**=*value*

DESCRIPTION

m.dem.examine reads Digital Elevation Model (DEM) terrain elevation data produced and supplied by the USGS from the input file specified by *name*, in the block size specified by *value*. If no input file is named on the command line, the program assumes that /dev/rmt0 is the input file. The information provided to the user comes from each file included on tape. Included are the maximum and minimum elevation values and the approximate UTM boundaries of each file. The program assumes unlabeled tapes in ASCII format with no header or trailer files.

SEE ALSO

g.region, m.dem.extract, m.examine.tape, r.rescale, r.support

AUTHOR

Andrew Heekin, U.S. Army Construction Engineering Research Laboratory

NAME

m.dem.extract – Extracts USGS Digital Elevation Model (DEM) data from 1/2-inch magnetic tape.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.dem.extract
m.dem.extract help
m.dem.extract input=*name* **output**=*name* **blocksize**=*value* **start**=*value* **end**=*value*

DESCRIPTION

m.dem.extract extracts USGS Digital Elevation Model (DEM) elevation data that fits into the user's current geographic region from the input file *input*, in blocks of *blocksize* bytes. If no *input* file is specified by the user, input is taken from /dev/rmt0, by default. Results are placed in the named *output* file, and stored beneath the *cell* directory of the user's current mapset. *m.dem.extract* will only extract data that fall within the boundaries of the user's current geographic region. Data falling outside this region will be ignored. *m.dem.extract* will not complain if the input file does not cover the entire geographic region. The user should run *m.dem.extract* without specifying output file (which only examines the input file) prior to extracting the data to determine the size of the geographic region needed. If the block size is unknown, run the command *m.examine.tape*.

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:

m.dem.extract input=*name* [**output**=*name*] [**blocksize**=*value*] [**start**=*value*] [**end**=*value*]

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

Parameters:

input=*name* The full path name of tape device or file from which DEM data are to be extracted.
Default: /dev/rmt0

output=*name* The name to be assigned to the output file containing raster DEM data extracted from 1/2-inch tape.

blocksize=*value* The physical block size (record length) of each each record, in bytes. *m.examine.tape* can be used to determine block size.

start=*value* First DEM quad to look at (default 1).

end=*value* Last DEM quad to look at (default 9999999).

NOTES

The user should check the boundaries and resolution of the current region setting (see *g.region*) **BEFORE** extracting data, since *m.dem.extract* will only extract the data that falls within these boundaries and only use the set resolution during extraction.

Warning: This program assumes that the tape has **ONLY** DEM data. If the tape contains other files (DLG, for example) the program will not skip over them. In this case, forward the tape to the first DEM file. Use the appropriate UNIX commands (**mt** to forward the tapes past the non-DEM files, and **dd** to extract and copy the correct files) on the tape on another machine first to extract the non-DEM data from the tape.

USGS Tapes: This version of *m.dem.extract* is sensitive to formatting errors. If the format on a tape is wrong, the program exits with the error message: "The file with incorrect data format encountered ..." Nevertheless, the program continues. In this case, run the program without specifying input files. If the program exits again with the same error message, there is an inconsistency in the data format.

The format error: Each USGS Dem file consists of A and B, and sometimes C, records--

type A record	header
type B record	collection of profiles with various numbers of elevations
type C record	accuracy

This new version of *m.dem.extract* cannot skip over any records to the end of a DEM file. It first determines how many records there are in a file, reads those records, and then proceeds to the header (record A) of the next file. Thus it is very important for the program to know whether or not there is a C record to be read at the end of each file. This information is given in the file's header record (type A) which has a field set to 1 when a C-record exits, and to 0 when a C-record does not exist.

In some DEM files, the accuracy field is set to 0, even though a C-record does exist. Because of this error, when the program tries to read the data, it will not try to read the C-record, but instead attempts to go on to the next file and read its header--a type A record. This results in the format error because type A records and type C records have different formats, and the program is trying to read a type A record when the tape is presenting a type C record.

When this format error in a file is encountered, forward the tape to the next file.

SEE ALSO

g.region, m.examine.tape, r.rescale, r.support and *parser*

AUTHOR

Andrew Heekin, U.S. Army Construction Engineering Research Laboratory

Improvements to program code were made for GRASS 4.0 by
David Satnik, Central Washington University

Major enhancements for GRASS 4.1 to allow the input to be a file or tape were made by
Olga Waupotitsch, U.S. Army Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.dmaUSGSread − Extracts digital terrain elevation data (DTED) produced by the Defense Mapping Agency (DMA) but supplied by the USGS (in a different tape format) on 1/2-inch magnetic tape.
(GRASS Data Import/Processing Program)

SYNOPSIS

dd [**if**=*tapedev*] **ibs**=*input-block-size* | **m.dmaUSGSread** **top**=*value* **bottom**=*value* **left**=*value* **right**=*value* **output**=*name* **logfile**=*name*

DESCRIPTION

m.dmaUSGSread extracts digital terrain elevation data (DTED) that was produced by the Defense Mapping Agency (DMA) but bought from the USGS; these two agencies distribute the same data in a slightly different tape formats. This program requires the use of the UNIX command *dd* to read the 1/2-inch magnetic tape. *m.dmaUSGSread* should be used prior to using the GRASS programs *m.rot90* and *r.in.ll*.

Parameters for the UNIX *dd* command are listed below.

if=*tapedev* The path name of the tape drive from which input will be taken (usually /dev/rmt0).

ibs=*input-block-size* The physical blocking value of the data on tape, usually written on the tape spool. If this value is unknown, run *m.examine.tape* prior to running *m.dmaUSGSread*.

Parameters for *m.dmaUSGSread* are listed below.

top=*value* Beginning row number of data.
Options: 1-1201

bottom=*value* Ending row number of data.
Options: 1-1201

left=*value* Beginning column number of data.
Options: 1-1201

right=*value* Ending column number of data.
Options: 1-1201

output=*name* Name of the output file to hold the extracted DEM data.

logfile=*name* Name of a file to hold related information about the extracted data.

EXAMPLE:

For example, the command:

```
dd if=/dev/rmt0 ibs=10240 | \
m.dmaUSGSread top=1 bottom=400 left=1 right=500 \
   output=dem logfile=log
```

will extract data from /dev/rmt0 and put the first 400 rows and first 500 columns into the file *dem* located in the user's current directory.

SEE ALSO

Options for Acquiring Elevation Data, by Stuart Bradshaw (ADP Report N-87/22, USACERL, 1988).

DTED and DEM Elevation Data Extraction, by Stuart Bradshaw, Mary Martin, and Chester Kos (ADP Report N-87/22, USACERL, 1988).

and the UNIX manual entry for dd[1]

g.region, m.dted.examine, m.dted.extract, m.examine.tape, m.region.ll, m.rot90, r.describe, r.in.ll, r.rescale, r.slope.aspect

AUTHORS

James Farley, Arkansas Archeological Survey, University of Arkansas
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

NAME

m.dted.examine − Provides a terse description of level 1 and 2 digital terrain elevation data (DTED) files produced and distributed by the Defense Mapping Agency (DMA) on 1/2-inch magnetic tapes.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.dted.examine [*tapedev*]

DESCRIPTION

m.dted.examine scans digital terrain elevation data (DTED - Levels 1 and 2) that was produced and supplied by the Defense Mapping Agency (DMA) on 1/2-inch magnetic tape from the tape device specified by *tapedev*, and prints to the screen a terse description of each file.

If the user specifies no *tapedev* on the command line the program assumes that input is coming from /dev/rmt0. The information provided by *m.dted.examine* includes the UTM borders, cell resolution, and number of elevation profiles in each file.

BUGS

The header file for DTED Level 1 and 2 data was changed in 1987. *m.dted.examine* and *m.dted.extract* operate only on DTED data containing pre-1987 headers. DTED data containing the pre-1987 headers may be purchased from the Defense Mapping Agency (DMA) upon request.

SEE ALSO

Pursuance of Elevation Data, by Stuart Bradshaw, USACERL
DEM and DTED Elevation Extraction, by Stuart Bradshaw, Mary Martin, and Chester Kos, USACERL

g.region, m.dmaUSGSread, m.dted.extract, m.examine.tape, m.region.ll, m.rot90, r.describe, r.in.ll, r.rescale, r.slope.aspect

AUTHORS

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

NAME

m.dted.extract – Extracts digital terrain elevation data (DTED - levels 1 and 2) produced and supplied by the Defense Mapping Agency (DMA) on 1/2-inch magnetic tapes.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.dted.extract if=*tapedev* of=*outfile* hf=*headfile* n=*lat* s=*lat* e=*lon* w=*lon*

DESCRIPTION

m.dted.extract extracts DTED (Levels 1 and 2) digital terrain elevation data produced by the Defense Mapping Agency (DMA) from 1/2-inch magnetic tapes obtained from DMA. Data is read from the input file (if) specified by *tapedev*. If the user does not specify any input file name on the command line, the program assumes that input is coming from /dev/rmt0. The extracted data is placed in the output file (of) specified by *outfile*. This program should be used in conjunction with the programs *m.rot90* and *r.in.ll* to convert DTED data to GRASS raster format.

if The pathname of the tape device where the raw DTED data exists (default is /dev/rmt0)

of The full pathname of the output file into which the extracted tape data is to be copied

hf The full pathname of a file to contain descriptive information about the extracted data; should be placed in the same directory as the output file

n North latitude value defining the boundaries of the extraction geographic region (format: dd.mm.ss[n|s])

s South latitude value defining the extraction geographic region (format: dd.mm.ss[n|s])

ea East longitude value defining the extraction geographic region (format: dd.mm.ss[e|w])

w West longitude value defining the extraction geographic region (format: dd.mm.ss[e|w])

The *n s e* and *w* parameters define a geographic region that should completely encompass the data set. *dd.mm.ss* are degree, minute, and second values. Only data that falls within this defined geographic region will be extracted from the tape.

EXAMPLE

The command line:

m.dted.extract if=/dev/rmt0 of=dted.out hf=dted.head n=37.30.00n s=37.15.00n e=103.30.00w w=103.45.00w

will extract DTED data from /dev/rmt0, store it in a file named *dted.out*, and store some supporting information in the file *dted.head*. Only data that falls within the geographic region defined by the coordinates *n,s,e* and *w* will be extracted.

NOTE

The user should examine the contents of the header file produced by this program; it contains information needed as inputs to the data rotation and raster file import programs *m.rot90* and *r.in.ll,* respectfully.

BUGS

The format of the header file for DTED Level 1 and 2 data was changed in 1987. *m.dted.extract* and *m.dted.examine* only operate on DTED data containing pre-1987 headers. DTED data containing the pre-1987 headers may be purchased from the DMA upon request.

DIAGNOSTICS

Invalid requests are so designated.

SEE ALSO

Pursuance of Elevation Data, by Stuart Bradshaw, USACERL
DEM and DTED Elevation Extraction, by Stuart Bradshaw, Mary Martin, and Chester Kos, USACERL

g.region, m.dmaUSGSread, m.dted.examine, m.examine.tape, m.region.ll, m.rot90, r.describe, r.in.ll, r.rescale, r.slope.aspect

AUTHOR

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

NAME

m.examine.tape – Provides a description of the files on a 1/2-inch magnetic tape.
(GRASS File Management Tool)

SYNOPSIS

m.examine.tape

DESCRIPTION

m.examine.tape is a convenient tool for previewing a tape and obtaining block size values. After the user enters the command

m.examine.tape

the program will prompt the user for the full path name of the tape device on which the tape is mounted. In most cases, this will be /dev/rmt0. However, the user must enter this, as there is no default value for the tape device name. The user is then prompted to enter the buffer size to be used when reading the tape. The buffer size is the amount of memory allocated to read one physical record of the tape. If the user hits RETURN, the program will assume a default buffer size of 32767 bytes.

m.examine.tape reads the tape specified by the user and reports back the block size (record length) and the number of blocks for each file contained on the tape. *m.examine.tape* can be used on any 1/2-inch magnetic tape. The user has the option of sending the output into a file or viewing the output on the terminal screen.

NOTES

The buffer size is the amount of memory allocated to read one physical record on the tape. Magnetic tape devices will accept a request to read more bytes than actually exist in any given record on a tape. However, *m.examine.tape* reads only as many bytes as physically exist on the tape and returns the number of bytes actually read. The user is allowed to alter the buffer size in order to request smaller reads for tape devices unable to handle requests this large (32767 bytes), or to request larger reads for tapes with larger record sizes read on drives able to handle larger record sizes.

Note that *m.examine.tape* cannot be used to examine the content of either ordinary files or 1/4-inch tape cartridges.

SEE ALSO

m.dem.examine, m.dem.extract, m.dmaUSGSread, m.dted.examine, m.dted.extract, m.lulc.USGS, m.lulc.read

AUTHOR

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

NAME

m.flip – Flips elevation data extracted from systems that retrieve data by rows from south to north.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.flip
m.flip help
m.flip [**-q**] **input**=*name* **output**=*name* **rows**=*value* **cols**=*value* **bpc**=*value*

DESCRIPTION

m.flip is similar to *m.rot90*. However, rather than rotating a raster file, *m.flip* flips the contents of the raster generated by tape extraction programs about the east-west axis. Flipping may be necessary to compensate for the orientation of the data read from tape. This program can be used in conjunction with the program *r.in.ll* to convert this rotated data into GRASS raster format.

m.flip requires five inputs to be entered by the user. These parameters and the optional flag setting are described below.

Flag:

-q Run quietly, suppressing output of messages on program progress to standard output.

Parameters:

input=*name* The full pathname of an existing file containing the data to be flipped.

output=*name* The full pathname of the output file in which the rotated data are to be stored.

rows=*value* The number of rows of data in the *input* file. Values must be positive integers.

cols=*value* The number of columns of data in the *input* file. Values must be positive integers.

bpc=*value* The number of bytes per cell (i.e., per data value) in the *input* file. Values must be positive integers.

EXAMPLE

The following command:

m.flip input=/tmp/foo.out output=/tmp/flip.out rows=301 cols=358 bpc=2

will rotate the file /tmp/foo.out, and place the rotated file in /tmp/flip.out. Here, the input file is 301 rows by 358 columns, at 2 bytes per data value.

SEE ALSO

Pursuance of Elevation Data, by Stuart Bradshaw, USACERL
DEM and DTED Elevation Extraction, by Stuart Bradshaw, Mary Martin, and Chester Kos, USACERL

g.region, m.dmaUSGSread, m.dted.examine, m.dted.extract, m.examine.tape, m.region.ll, m.rot90, r.describe, r.in.ll, r.rescale, r.slope.aspect

AUTHORS

Donald Newcomb, U.S. Naval Oceanographic Office,
borrowing heavily from the *m.rot90* program written by Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.gc2ll – Converts geocentric to geographic coordinates.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.gc2ll x=*#* y=*#* z=*#* s=*spheroid*

DESCRIPTION

m.gc2ll returns geographic coordinates for geocentric ones supplied by the user. It performs the reverse operation of the GRASS program *m.ll2gc*. The **x**, **y** and **z** values are the three dimensions needed to locate a point in three-dimensional space. The values that are printed include the latitude, the longitude and the height above (or distance below) the spheroid.

The list of spheroids available is somewhat dynamic. It may not contain exactly the ones listed below. To determine the current list of possible spheroids, simply type in:

m.gc2ll x=0 y=0 z=0 s=help

A list of available spheroids will be printed on the screen. If the spheroid desired is not on the list, the values for the semi-major axis and the eccentricity squared for the spheroid may be entered in place of a spheroid name in the following format:

s=a=*semi-major_axis*,e=*eccentricity_squared*

SOME POSSIBLE SPHEROIDS
(*The on-line listing includes only the spheroid names*)

Spheroid	Commonly used for:	Semi-major axis	Eccentricity sqrd
australian	Australia	6378160.0	0.0066945419
bessel	Japan	6377739.155	0.0066743722
clark66	N. America	6378206.4	0.006768658
clark80	France, Africa	6378249.145	0.0068035113
everest	India, Burma	6377276.345	0.0066378466
international	Europe	6378388.0	0.00672267
wgs72	worldwide coverage	6378135.0	0.006694317778

EXAMPLE

m.gc2ll x=0.0 y=0.0 z=6356750.520017 s=wgs72

Results:
lat=90N
lon=90W
h=0.0000

NOTES

For a brief discussion of spheroids see *m.ll2u*.

For a brief discussion of geocentric coordinates see *m.ll2gc*.

This is an experimental program. It is part of an initial effort to incorporate geographic coordinates into GRASS.

SEE ALSO

m.datum.shift, m.ll2gc, m.ll2u, m.u2ll

AUTHOR

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

NAME

m.in.pl94.db3 – Import Demographic records from Census PL94-171 dBase3 (CDROM) Files
(GRASS Sites Program)

SYNOPSIS

m.in.pl94.db3 help
m.in.pl94.db3 [-f] [-p] [-n] sc=string [sc=str . .] in=/cdrom/pl9417ss.dbf > result_file
This command may be run in command line mode only. Parameters used must be in the order specified.

PARAMETERS

-f Output to stdout the list of all field name and starting positions for the Identification Section of the PL94 file. Other parameters are not necessary and will be ignored.

-p Print several information items to stderr input file is opened.

-n Output only the sequential number and PL94 logical record number (LOGRECNU) of matching records to stdout.

sc=string sc is starting column or a capitilized reference name from the Identification Section of the PL94 Data Dictionary; string is characters to match there. A negative test may also be made with: sc!=string. If more than one test is specified the results are "anded" resulting in a conjunction of the results of the expressions. (See *m.in.stf1.tpe* for complete details.)

in Path/file for dBase files for state *ss*.

DESCRIPTION

This program is provided to read the Census PL94 data records from a file for one state. The file is usually on a CD-ROM, but this is not necessary providing the proper path is given for the *in=* parameter and the contents of the file duplicate that on CD-ROM.

The output consists of records of 517 characters extracted from the input file with a suitable line termination character added (0 to create a normal text file.

Users of GRASS 4.1 and the Census PL94 files on CD-ROM will generally use this program to select appropriate subsets of PL94 records; write them to one or more files; and then either process these files directly with *v.apply.census* to print demographics, create site maps or label vector maps, or use *m.in.stf1.tpe* to extract further subsets.

The PL94 CD-ROM data file for most states is very large, 100+ MB. This program may take 20 to 60 minutes for a single extraction, so plan carefully and don't discard useful subsets that you create on disk.

EXAMPLES

Insert the appropriate CD-ROM and mount it (see your system administrator). (We assume here that it is mounted as the file system /cdrom.)

To extract all Census tract summary records for the state of South Dakota:

m.in.pl94.db3 -p 11=050 in=/cdrom/pl9417sd.dbf > counties.sd

The 11=050 parameter selects those records which have the character string Summary Level (SUMLEV) field which is used in almost every selection from CD-ROM.

To extract all block records for Lawrence County, South Dakota:

m.in.pl94.db3 -p SUMLEV=100 CNTY=081 in=/cdrom/pl9417sd.dbf >

100 is the SUMLEV for blocks and 081 is the county code for Lawrence Co.

SEE ALSO

m.in.stf1.tpe is used as a post-processor to select further subsets of PL94 records. NOTE: Unix utility programs, such as *grep* or *awk*, can also be used for the for processing the extracted PL94 files.

AUTHOR

Dr. James Hinthorne, GIS Laboratory, Central Washington University, August 1992.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.in.stf1.db3 – Import Demographic records from Census STF1A dBase3 (CDROM) Files
(GRASS Sites Program)

SYNOPSIS

m.in.stf1.db3 help
m.in.stf1.db3 [-f] [-p] [-n] sc=string [sc=str . .] in=/cdrom/stf1a0ss.dbf > result_file
This command may be run in command line mode only. Parameters used must be in the order specified.

PARAMETERS

-f Output to stdout the list of all field name and starting positions for the Identification Section of the STF1A file. Other parameters are not necessary and will be ignored.

-p Print several information items to stderr as the 10 input files are opened.

-n Output only the sequential number and STF1A logical record number (LOGRECNU) of matching records to stdout.

sc=string sc is starting column or a capitilized reference name from the Identification Section of the STF1A Data Dictionary; string is characters to match there. A negative test may also be made with: sc!=string. If more than one test is specified the results are "anded" resulting in a conjunction of the results of the expressions. (See *m.in.stf1.tpe* for complete details.)

in Path/file for dBase files for state *ss*.

DESCRIPTION

This program is provided to read and assemble the Census STF1A data records from a set of 10 files for one state. The files are usually on a CD-ROM, but this is not necessary providing the proper path is given for the *in=* parameter and all 10 files are in the same directory with their normal names.

The output consists of data from the 10 input files assembled into records identical to the STF1A tape distribution format. The output records are each about 4800 characters and a pair constitute a logical STF1A record.

Users of GRASS 4.1 and the Census STF1A files on CD-ROM will generally use this program to select appropriate subsets of STF1A records; write them to one or more files (in the STF1A tape format); and then either process these files directly with *v.apply.census* to print demographics, create site maps or label vector maps, or use *m.in.stf1.tpe* to extract further subsets.

The STF1A CD-ROM data sets for most states are very large, 100+ MB. This program may take 20 to 60 minutes for a single extraction, so plan carefully and don't discard useful subsets that you create on disk.

EXAMPLES

Insert the appropriate CD-ROM and mount it (see your system administrator). (We assume here that it is mounted as the file system /cdrom.)

To extract all Census tract summary records for the state of South Dakota:

m.in.stf1.db3 -p 11=140 in=/cdrom/stf1a0sd.dbf > tracts.SD

The 11=140 parameter selects those records which have the character string Summary Level (SUMLEV) field which is used in almost every selection from CD-ROM.

To extract all block group records for Lawrence County, South Dakota:

m.in.stf1.db3 -p SUMLEV=150 CNTY=081 in=/cdrom/stf1a0sd.dbf >

150 is the SUMLEV for block groups and 081 is the county code for Lawrence Co.

SEE ALSO

m.in.stf1.tpe is used as a post-processor to select further subsets of STF1A records. NOTE: Unix utility programs, such as *grep* or *awk*, cannot be used for the for processing the STF1 files due to their very long record lengths.

AUTHOR

Dr. James Hinthorne, GIS Laboratory, Central Washington University, May 1992.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.in.stf1.tape – Filter to extract lines from a text file based on column contents, especially for Bureau of the Census STF1 files.
(GRASS Support Program)

SYNOPSIS

m.in.stf1.tape
m.in.stf1.tape help
m.in.stf1.tape -f
m.in.stf1.tape [-n] sc=str [sc=str . . .] < infile > outfile

This program must be run in command mode only.

PARAMETERS

sc is a starting column number of a desired field in the input file, or is the name of one of the Identification Section field names for the STF1 records (all upper case letters), and *str* is a string to match against input lines starting at column *sc*. *sc=str* may be repeated resulting in a conjunction (anding) of the results of each *sc=str* expression. A '?' may be used as a single character wild card in *str*; if *sc=str* contains '?' or other shell interpreted characters, it should be protected in quotes. Preceeding *sc* by 'N', or preceeding the = by '!' reverses the sense of the test ("not equals").

Examples:
m.in.stf1.tape 11=050 < infile > outfile
m.in.stf1.tape SUMLEV=050 < infile > outfile
m.in.stf1.tape 1=T450 '7=Bu??s' < infile > outfile
m.in.stf1.tape 51=tract N37=9753 < infile > outfile
m.in.stf1.tape 51=tract 37!=9753 < infile > outfile

Running the program with the *-f* flag generates the list of STF1 Identification Section field names to stdout.

DESCRIPTION

This is a text filter program written in C especially to work with the Census STF1 files, but useful for selecting subsets of lines from any text file. It will work with arbitrarily long input lines, up to 10,000 characters. Input lines may be of variable length.

A *sc=str* condition which refers to columns beyond the end of the input line is assumed to be true.

Multiple tests are 'anded' into a single test; that is, lines which pass all tests are written to the output file.

Null characters are always filtered out; thus, this program can be used as a null filter by specifying a *sc=str* condition which will always be true (e.g., '1=?'). In the same way, files with lines terminated with <LF><CR> or just <CR> can be "fixed" to have the standard Unix <LF> terminator.

NOTE: One special property of *m.in.stf1.tape* is that it will pad (with 0) its output lines to 4806 characters if the line begins with the characters "STF1" and is greater than 4000 characters long. Experience has shown that some STF1 records are a few characters short (but no data has been omitted), and this corrects them so that other programs will be able to read full lines.

BUGS/FEATURES

Input lines must be terminated with <LF>, <CR> or <LF><CR>. Output lines will be terminated with <LF> only.

Input characters lexically less than 'space' (32 decimal; the "control" characters) which are not line terminators will be perceived as line terminators and thus cause improper functioning.

SEE ALSO

For extracting lines from files, Unix programs such as *grep*, or *awk* are somewhat more flexible than *m.in.stf1.tape*, but have line length limitations and do not adapt to lines terminated with <LF> and <CR>.

This program was created as a preprocessor for Census STF1 files, to produce input files for the program *v.apply.census*. The program *m.in.stf1.db3* performs the same preprocessing function for the Census STF1 files distributed in "dBase 3" format on CD-ROM.

AUTHOR

Dr. James Hinthorne, GIS Laboratory, Central Washington University.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.ll2gc – Converts geographic coordinates to geocentric coordinates.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.ll2gc lat=*dd.mm.ss{n| s}* lon=*dd.mm.ss{e| w}* [h=*height*] s=*spheroid*

DESCRIPTION

m.ll2gc returns geocentric coordinates for geographic coordinates (latitude and longitude). Geographic coordinates are in degrees, minutes, and seconds and must include designation of north or south {n| s} and east or west {e| w}. The spheroid on which they are based must also be entered. The height (in meters) above the spheroid is optional.

The list of spheroids available is somewhat dynamic. It may not contain exactly the ones listed below. To determine the current list of possible spheroids, type in:

m.ll2gc lat=0n lon=0w s=help

A list of available spheroids will be printed on the screen. If the spheroid desired is not on the list, the values for the semi-major axis and the eccentricity squared for the spheroid may be entered in place of a spheroid name in the following format:

s=a=*semi-major_axis*,e=*eccentricity_squared*

SOME POSSIBLE SPHEROIDS
(*The on-line listing includes only the spheroid names*)

Spheroid	Commonly used for:	Semi-major axis	Eccentricity sqrd
australian	Australia	6378160.0	0.0066945419
bessel	Japan	6377739.155	0.0066743722
clark66	N. America	6378206.4	0.006768658
clark80	France, Africa	6378249.145	0.0068035113
everest	India, Burma	6377276.345	0.0066378466
international	Europe	6378388.0	0.00672267
wgs72	worldwide coverage	6378135.0	0.006694317778

EXAMPLE

m.ll2gc lat=0n lon=90w s=wgs72

Results:
x=0.0
y=6378135.0
z=0.0

The distances designated by "x," "y" and "z" are in meters from the center of the earth. Because the sample point is on the equator and is at 90 degrees west, two of the three parameters have a value of zero (i.e., the point is at the origin of those two dimensions).

NOTES

For a brief discussion of spheroids see *m.ll2u*.

Geographic coordinates (latitude/longitude) describe the location of a point on the earth relative to the equator in the north/south directions and relative to Greenwich in the east/west directions. The same point will have different coordinates depending on the spheroid used as the approximation of the shape of the earth. The geocentric coordinates given here describe the point on the basis of the same spheroid, but they use the center of the spheroid as their origin. This can be thought of as being the center of the earth, which is why they are called "geocentric." Describing the location of a point in a spheroid requires three points: x, y, and z; these are measured out from the center along three different axes. The distances given are in meters.

This is an experimental program. It is part of initial efforts to incorporate geographic coordinates into GRASS.

SEE ALSO

m.datum.shift, m.gc2ll, m.ll2u, m.u2ll

AUTHOR

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

NAME

m.ll2u – Converts geographic coordinates to Universal Transverse Mercator (UTM) coordinates. (GRASS Data Import/Processing Program)

SYNOPSIS

m.ll2u
m.ll2u help
m.ll2u [**-rwoz**] **spheroid**=*name* [**zone**=*value*] [**input**=*name*] [**output**=*name*]

DESCRIPTION

m.ll2u converts geographic coordinates (i.e., latitudes and longitudes) to Universal Transverse Mercator (UTM) eastings and northings. The user must specify the spheroid on which to base the UTM conversion. The user may optionally specify the UTM zone; however, the program can determine this from the geographic coordinates submitted.

The list of spheroids available is somewhat dynamic. At the time of this release, available spheroids included: airy, australian, bessel, clark66, everest, grs80, hayford, international, krasovsky, wgs66, wgs72, and wgs84 (see table below).

This command can be run either non-interactively or interactively. The user can run the program non-interactively by entering desired flag settings and parameter values on the command line using the following format:

m.ll2u [**-rwoz**] **spheroid**=*name* [**zone**=*value*] [**input**=*name*] [**output**=*name*]

Alternately, the user can simply type:

m.ll2u

on the command line. In this case, the user will be prompted for parameter values and flag settings through the standard interface described in the manual entry for *parser*.

Input can be entered from the keyboard or from an input file. In either case, input should be entered with one longitude and latitude pair per line, in either of the below forms:

```
degrees:minutes:seconds{E|W} degrees:minutes:seconds{N|S}
degrees:minutes:seconds{E|W} degrees:minutes:seconds{N|S}
degrees:minutes:seconds{E|W} degrees:minutes:seconds{N|S}
degrees:minutes:seconds{E|W} degrees:minutes:seconds{N|S}
  .
  .
end
```

```
degrees.decimal{E|W} degrees.decimal{N|S}
degrees.decimal{E|W} degrees.decimal{N|S}
degrees.decimal{E|W} degrees.decimal{N|S}
degrees.decimal{E|W} degrees.decimal{N|S}
  .
  .
end
```

If the user sets the -r flag, *m.ll2u* will expect the order of the coordinates to be reversed, and stated as latitude, longitude pairs, rather than as longitude, latitude pairs.

Similarly, the user can elect to send output to an output file or (by default) to standard output (the user's terminal screen). If the user sets the -w flag, output will be printed in a format suitable for input to programs like *d.points*. Example input and output are shown below (see EXAMPLE).

Program flag settings and parameters have the following meanings.

Flags:

-r	The order of coordinates is reversed in the input, and entered as: *lat lon*
-w	Do not flag invalid lon,lat input lines as errors.
-o	Flag other invalid input lines as errors.
-z	Suppress printing of the UTM zone in the output. (Note. This will produce output in a format suitable for direct input to programs like *d.points*.)

Parameters:

spheroid=*name*	Reference spheroid (ellipsoid). Options: airy, australian, bessel, clark66, everest, grs80, hayford, international, krasovsky, wgs66, wgs72, wgs84
zone=*value*	UTM zone number (results will be forced into this UTM zone). Options: 1-60
input=*name*	Name of an existing input file containing longitude, latitude coordinates to be converted. Input lines may either be input in the form of degrees:minutes:seconds, or as decimal degrees. If input as decimal degrees, *m.ll2u* recognizes negative numbers.
output=*name*	Name to be assigned to the output file containing UTM coordinates and zone designations.

AVAILABLE SPHEROIDS
(*The on-line listing includes only the spheroid names*)

Spheroid:	Semi-major axis (Equatorial Radius) (a):	Eccentricity sqrd (e), Flattening (f), or Polar Radius (b):	Commonly used for:
airy	a=6377563.396	e=.006670540	
australian	a=6378160	f=1/298.25	Australia
bessel	a=6377397.155	e=.006674372	Japan
clark66	a=6378206.4	b=6356583.8	N. America
everest	a=6377276.345	e=.0066378466	India, Burma
grs80	a=6378137	f=1/298.257	
hayford	a=6378388	f=1/297	
international	a=6378388	f=1/297	Europe
krasovsky	a=6378245	f=1/298.3	
wgs66	a=6378145	f=1/298.25	worldwide coverage
wgs72	a=6378135	f=1/298.26	worldwide coverage
wgs84	a=6378137	f=1/298.257223563	worldwide coverage

EXAMPLE

Assume the user has input the command:

m.ll2u spheroid=wgs72 input=ll.infile output=utm.outfile

where the input file *ll.infile* contains the following longitude, latitude values:

```
155:30:00W 19:35:00N
165:30:00W 19:35:00N
145:30:00W 20:00:00N
135:30:00W 21:00:00N
end
```

Output would then be sent to the output file *utm.outfile*, containing the below Easting and Northing coordinate values and UTM zone designations:

```
237740.85270818 2167292.1076231 5
447560.64349407 2165450.19058336 3
656921.61734802 2212183.40032627 6
448035.11644906 2322228.3038167 8
end
```

NOTES

Spheroids, the solids associated with an ellipse, are also known as ellipsoids. They are used as the best possible model to simulate the shape of the earth with its flattened poles and bulging equator. They are the mathematical means for establishing control points to use as a reference when determining exact locations on the earth. A cohesive set of these control points is called a **datum**.

The spheroids listed above have been used as the basis for a number of different datums. The North American Datum of 1927 (NAD 27) was based on the Clark 1866 ("clark66") spheroid. This was a recent current standard datum for North America. Be aware, however, that a new datum, NAD 83, has been developed using the Geodetic Reference System 1980 spheroid; this is now available in *m.ll2u* as the "grs80" spheroid. The "wgs66", "wgs72", and "wgs84" spheroids are for worldwide use. The "wgs72" spheroid has been used fairly widely by the Department of Defense (DOD) and is the basis for the World Geodetic System 1972 datum. This datum has also been recently replaced; the new DOD datum is WGS 84 (wgs84). Both datums are available within *m.ll2u*.

To use spheroids other than those listed here, the user can add lines to the ellipsoid parameter definition file in $GISBASE/etc/ellipse.table.

Read the marginalia of your source map to determine which spheroid was used to produce the map on which you are working.

This program recognizes negative numbers if coordinates are input in decimal degrees rather than in the form of degrees:minutes:seconds.

This program has received only limited testing. It should be used with some caution.

FILES

See ellipsoid parameter definition file in $GISBASE/etc/ellipse.table.

SEE ALSO

For australian, clark66, grs80, hayford, international, krasovsky, and wgs72 ellipsoid parameters, see:
John P. Snyder, *Map Projections - A Working Manual*, U.S. Government Printing Office, Washington DC, 1989. U.S. Geological Survey Professional Paper 1395; from Table 1, p.12.

For bessel, airy, everest, and wgs66 ellipsoid parameter values, see:
Thomas O. Seppelin, *The Department of Defense World Geodetic System 1972*, presented at the International Symposium on Problems Related to the Redefinition of North American Geodetic Networks, Fredericton, New Brunswick, Canada in May, 1974; see Table 9, p.35.

For wgs84 parameter values, see:
U.S. Naval Oceanographic Labs.

Also read GRASS User's Reference Manual entries for:
d.labels, d.points, d.sites, d.where, m.datum.shift, m.gc2ll, m.ll2gc, m.u2ll, and *parser*

AUTHOR

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

NAME

m.lulc.USGS – Creates raster map layers from a Composite Theme Grid (CTG) file created by *m.lulc.read*. *m.lulc.read* extracts the CTG data from an ASCII landuse/landcover (lulc) CTG format file supplied by the USGS.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.lulc.USGS *input_file*

DESCRIPTION

m.lulc.USGS creates the following raster map layers from the CTG file created by *m.lulc.read*, and places them into the user's current mapset:

1. Land Use & Land Cover
2. Political Units
3. Census Units
4. Hydrologic Units
5. Federal Land Ownership
6. State Land Ownership

Since the CTG file may not contain all of layers listed above, *m.lulc.USGS* will extract only the raster map layers which exist in the CTG file.

The CTG file contains all of the geographic region definition information necessary for creating the raster map layers, including:

1. Grid Zone
2. Projection Type (UTM)
3. Cell Resolution (ew_res = nw_res)
4. Multi-byte data formation
5. Geographic region coordinates

m.lulc.USGS will use the geographic region information definition supplied with the CTG file and NOT the geographic region definition currently set for the user's mapset; (note that this is different than is the case with many of the other GRASS map development commands).

When *m.lulc.USGS* is executed, it will check the file for a specified layer and then ask the user if he wishes to extract the associated raster map layer. The user is then prompted for the name of a new raster file in which output is to be placed. *m.lulc.USGS* will then create this raster file and all supporting (e.g., category, cell header, color table, etc.) files.

EXAMPLE

In the example below, raster data is extracted from a CTG file (named *lulc*) that contains only LAND USE/LAND COVER data:

m.lulc.USGS lulc

MAP TITLE: DODGE CITY, KS 1:250,000 LU, PB, CN, HU, FO
The Composite Theme Grid file contains <1> overlays
and has a map code type of <01>

Do you Wish to Create <LAND USE/LAND COVER> Raster File (y/n) [y] **y**

Enter File Name for LAND USE/LAND COVER Overlay
Enter 'list' for a list of existing raster files
Hit RETURN to cancel request
> landuse

Creating <landuse> from <LAND USE/LAND COVER> Overlay: 95%
Number of Categories: 76 (UNLABELED)
Writing Cell Header Information
Writing Color Table Information

Conversion is Complete

BUGS

m.lulc.USGS does not currently extract Census unit data. Also, only the cataloging unit is extracted from the CTG file (see the *Land Use and Land Cover Data User's Guide* supplied by the USGS).

SEE ALSO

USGS **Land Use and Land Cover Data User's Guide**

g.region, m.lulc.read

AUTHOR

Kenneth Shepardson
Spectrum Sciences & Software, Inc.
(904) 862-3031

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.lulc.read – Extracts Landuse/Landcover data in the ASCII Composite Theme Grid (CTG) data format distributed by the USGS in to a working file for *m.lulc.USGS*.
(GRASS Data Import/Processing Program)

SYNOPSIS

dd [**if**=*tapedev*] **ibs**=*input block_size* cbs=*80* **conv**=*unblock* **| m.lulc.read** *arg1*

DESCRIPTION

m.lulc.read extracts USGS Land Use/Land Cover data distributed in the ASCII CTG format. Extracted data is placed into a file specified by *arg1*. *m.lulc.read* should be used prior to using the GRASS program *m.lulc.USGS*.

m.lulc.read reads the data from standard input, allowing the user to pipe in the data from a file.

Data can also be read directly from the 1/2-inch magnetic tape distributed by USGS, by using the UNIX command *dd*.

Parameters:

if=*tapedev*
The pathname of the input tape drive (usually **/dev/rmt0**).

ibs=*input*
The physical blocking value of the data on tape, usually written on the tape spool. If this value is unknown, run *m.examine.tape* prior to *m.lulc.read*.

cbs This is the conversion blocking factor for ASCII CTG file and is by default set to **80**.

conv Set *conv* to **unblock** when extracting ASCII CTG data.

arg1 The name of the output file where the data will be stored in binary format.

EXAMPLES

READING FROM A FILE:

m.lulc.read outfile < infile

The above command will read the data from *infile* and place the results into *outfile*. Note that *infile* must be extracted from the USGS CTG 1/2 magnetic tape using the UNIX *dd* command:

dd if=/dev/rmt0 of=infile ibs=32000 cbs=80 conv=unblock

READING DIRECTLY FROM 1/2 INCH MAGNETIC TAPE:

dd if=/dev/rmt0 ibs=32000 cbs=80 conv=unblock | m.lulc.read outfile

The above command will extract data with a blocksize of 32000 from */dev/rmt0*, and put the results in the file *outfile*.

SEE ALSO

UNIX manual entry for *dd*

m.examine.tape, m.lulc.USGS

AUTHOR

Kenneth Shepardson, Spectrum Sciences & Software, Inc. (904) 862-3031

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.proj – Calculates conversion coordinates for geographic positions.
(GRASS Map Development Program)

SYNOPSIS

m.proj
m.proj help

DESCRIPTION

This program allows a user to interactively convert coordinates from one projection to another.

It allows a user to do all of the above: reading from a file and writing to the screen, or reading from the keyboard and writing to a file, or reading from a file and writing to a file.

Note: The program does not transform GRASS files; it is designed to determine coordinate values on an individual position.

The map projections currently supported are listed in etc/projections file.

Each of the above projections (with the exception of State Plane) can be computed with the spheroids listed in the etc/ellipse.table file.

INPUT FILE FORMAT

When reading from a file of LATITUDE/LONGITUDE data, the file will contain two (2) columns of information:

the first column: latitude - in degrees minutes seconds
the second column: longitude - in degrees minutes seconds

For example:

```
+40 36 31.4563     -87 2 7.8193
40n 36 31.4563     87w 2 7.8193
```

When reading from a file of PROJECTION COORDINATES data, the file will contain two (2) columns of information:

the first column: easting - ground coordinates
the second column: northing - ground coordinates

For example:

```
500000.00      4496918.64
-424489.11     1908736.13
```

Note: No column headings are required, only the numbers.

AUTHORS

Irina Kosinovsky, US Army CERL
R.L. Glenn, USDA, SCS, NHQ-CGIS

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.region.ll – Converts Universal Transverse Mercator (UTM) coordinates falling within the current geographic region from UTM coordinates to geographic (latitude/longitude) coordinates.
(GRASS Data Import/Processing)

SYNOPSIS

m.region.ll
m.region.ll help
m.region.ll spheroid=*name*

DESCRIPTION

m.region.ll takes current geographic region settings in UTM coordinates, and converts them to geographic coordinates (i.e., latitudes and longitudes). It also prints the length (in meters) of one arc-second at each of the four edges of the geographic region. The user must enter the spheroid upon which to base the geographic coordinates.

The list of spheroids available is somewhat dynamic. It may not contain exactly the ones listed below. To determine the current list of possible spheroids, simply type in:

m.region.ll help

A list of available spheroids will be printed on the screen.

AVAILABLE SPHEROIDS
(The on-line listing includes only the spheroid names.)

Spheroid:	Semi-major axis (Equatorial Radius) (a):	Eccentricity sqrd (e), Flattening (f), or Polar Radius (b):	Commonly used for:
airy	a=6377563.396	e=.006670540	
australian	a=6378160	f=1/298.25	Australia
bessel	a=6377397.155	e=.006674372	Japan
clark66	a=6378206.4	b=6356583.8	N. America
everest	a=6377276.345	e=.0066378466	India, Burma
grs80	a=6378137	f=1/298.257	
hayford	a=6378388	f=1/297	
international	a=6378388	f=1/297	Europe
krasovsky	a=6378245	f=1/298.3	
wgs66	a=6378145	f=1/298.25	worldwide coverage
wgs72	a=6378135	f=1/298.26	worldwide coverage
wgs84	a=6378137	f=1/298.257223563	worldwide coverage

EXAMPLE

m.region.ll spheroid=clark66

Results:

```
WINDOW     4928000.00N     609000.00E    ZONE 13
           4914000.00S     590000.00W

  44.30.06N     44.29.57N
103.52.04W    103.37.44W

  44.22.32N     44.22.23N
103.52.13W    103.37.55W

At northern edge 1 arc-second longitude=22.088500m
At southern edge 1 arc-second longitude=22.135998m
At western edge 1 arc-second latitude=30.860285m
At eastern edge 1 arc-second latitude=30.863082m
```

The values for the geographic coordinates are rounded to the nearest second in this example. They would be more precise in the actual output that is printed on the screen.

SEE ALSO

m.datum.shift, m.ll2u, m.u2ll, r.in.ll

AUTHOR

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

NAME

m.rot90 – Rotates elevation data extracted by either *m.dted.extract* or *m.dmaUSGSread*
(GRASS Data Import/Processing Program)

SYNOPSIS

m.rot90
m.rot90 help
m.rot90 [**-q**] **input**=*name* **output**=*name* **rows**=*value* **cols**=*value* **bpc**=*value*

DESCRIPTION

m.rot90 is used as a companion program to the DTED and DEM digital elevation data tape extraction programs *m.dted.extract* and *m.dmaUSGSread*. *m.rot90* rotates the contents of the output files generated by these tape extraction programs 90 degrees. Rotation is necessary to compensate for the orientation of the data read from tape. This program can be used in conjunction with the program *r.in.ll* to convert this rotated data into GRASS raster map layer form.

m.rot90 requires five inputs to be entered by the user. These parameters and the optional flag setting are described below.

Flags:

-q Run quietly, suppressing output of messages on program progress to standard output.

Parameters:

input=*name* The full pathname of an already-existing file containing the data to be rotated. This *input* file is usually the output file created by *m.dted.extract* or *m.dmaUSGSread*.

output=*name* Name to be assigned to the resultant, rotated output file. The full pathname of the output file in which the rotated data are to be stored.

rows=*value* The number of rows of data in the *input* file.

cols=*value* The number of columns of data in the *input* file.

bpc=*value* The number of bytes per cell in the *input* file.

EXAMPLE

The following command:

m.rot90 input=/tmp/dma.out output=/tmp/rot.out rows=301 cols=358 bpc=2

will rotate the file /tmp/dma.out, and place the rotated file in /tmp/rot.out. Here, the input file is 301 rows by 358 columns, at 2 bytes per data value.

NOTES

The user should note that since the output file is rotated 90 degrees from the original input file, the rows and columns have been interchanged. Hence, in the above example, the number of rows (301) and columns (358) stated by the user on the command line were those present in the *input* file. The output file, however, will have 358 rows and 301 columns.

SEE ALSO

Pursuance of Elevation Data, by Stuart Bradshaw, USACERL
DEM and DTED Elevation Extraction, by Stuart Bradshaw, Mary Martin, and Chester Kos, USACERL

g.region, m.dmaUSGSread, m.dted.examine, m.dted.extract, m.examine.tape, m.flip, m.region.ll, r.describe, r.in.ll, r.rescale, r.slope.aspect

AUTHOR

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

NAME

m.tiger.region – Finds geographic region information for U.S. Census Bureau TIGER input data.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.tiger.region
m.tiger.region help
m.tiger.region infile=*name [***zone**=*value*] [**spheroid**=*name*]

DESCRIPTION

m.tiger.region is a program designed to evaluate a file of raw type 1 Census (TIGER) data and determine the geographic region covered by that input file. Output is sent to standard out, and gives the east, west, north, and south boundaries for the given input data file.

If the user specifies the input file name and (optionally) the zone number or spheroid to be used on the command line, the program will run non-interactively; if no zone number or spheroid name is given, the default(s) will be used (see below). Alternately, the user may simply type *m.tiger.region* on the command line; in this case, the program will prompt the user for parameter values using the standard GRASS parser interface described in the manual entry for *parser*.

COMMAND LINE OPTIONS

Parameters:

infile=*name* Input file name in which raw TIGER data (type 1) are stored.

zone=*value* Universal Transverse Mercator (UTM) zone for this county.
Options: -60 - 60
Default: 0

spheroid=*name* Name of spheroid to be used.
Default: clark66

Available spheroids are:

australian
bessel
clark66
clark80
everest
international
wgs72
wgs84

It is recommended that the user choose the clark66 (default) spheroid when dealing with TIGER data as it is the most consistent with the original data.

EXAMPLES

If the user typed simply:

m.tiger.region infile=inputfilename

program output would look similar to this:

Number of calculated zones is: 2

INFO FOR ZONE 1:
zone number: 13

percentage of data points	
in this zone:	0.799489
regional spread of points	
within this zone:	
north:	5092049.155918
south:	5049238.983803
east:	734139.517650
west:	732514.747908

INFO FOR ZONE 2:	
zone number:	14
percentage of data points	
in this zone:	99.200508
regional spread of points	
within this zone:	
north:	5092041.463966
south:	5036134.342322
east:	398030.217441
west:	265527.656108

If the user does not input the UTM zone number, it is calculated for them. Then the zone number and region information are output, and if the program finds that the input data contains information in more than one UTM zone, then the output is given for all applicable zones.

If instead the user supplies the UTM zone number, the output would look like that shown below:

REGION FOR THIS DATA FILE:

north border:	5092049.155918
south border:	5036134.342322
east border:	398030.217441
west border:	265527.656108
(zone number:	14)

NOTES

This command must be compiled separately. It will not automatically be included in the compile of the main GRASS code. Although *m.tiger.region* does not need a FORTRAN compiler, it is used to support other TIGER data functions (like *v.db.rim*, *v.in.tiger*, and *rim*) which do require access to a FORTRAN compiler.

TIGER data are presented in latitude/longitude format, and are converted to UTM coordinates using coordinate conversion routines contained in the GRASS library. If no UTM zone number is supplied by the user, the program calculates the appropriate zone(s) based on the input data provided. The output then provides the UTM zone numbers found (if more than one), the geographic region covered within each zone, and the percentage of data points found in each zone. The user must then decide which of these UTM zones contains the major or most important portion of data values, so that the zone number can be supplied in creating the GRASS location to hold the imported data and can be provided to the importing program (*v.in.tiger*). Zone edges will be extended (reasonably) to include data values lying outside the chosen zone. If desired, *m.tiger.region* can be re-run, supplying the chosen zone number, in order to evaluate the region edges of the input data set (with the extended zone).

FILES

Source code for RIM is located under $GISBASE/../src.related/rim

Source code for *v.db.rim* is located under $GISBASE/../src.garden/grass.rim/v.db.rim

Source code for *v.in.tiger* is located under $GISBASE/../src.garden/grass.tiger/v.in.tiger

Source code for *m.tiger.region* is located under $GISBASE/../src.garden/grass.tiger/m.tiger.region

SEE ALSO

Gen.Maps, Gen.tractmap, g.region, v.db.rim, v.in.tiger, tiger.info.sh

AUTHOR

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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

m.u2ll – Converts Universal Transverse Mercator (UTM) coordinates to geographic (latitude/longitude) coordinates.
(GRASS Data Import/Processing Program)

SYNOPSIS

m.u2ll
m.u2ll help
m.u2ll [**-srwod**] **spheroid**=*name* [**zone**=*value*] [**input**=*name*] [**output**=*name*]

DESCRIPTION

m.u2ll converts Universal Transverse Mercator (UTM) northings and eastings to geographic coordinates (i.e., latitudes and longitudes). The user must specify the UTM coordinates to be converted and the spheroid on which the geographic coordinates will be based. The program also needs to know the UTM zone in which the input coordinates are located. However, if the user is running GRASS from a UTM data base LOCATION, *m.u2ll* will use this data base's UTM zone designation, if no zone is specified by the user.

The GRASS program *m.ll2u* performs the reverse operation, converting geographic coordinates to UTM coordinates.

The list of spheroids available is somewhat dynamic. At the time of this release, available spheroids included: airy, australian, bessel, clark66, everest, grs80, hayford, international, krasovsky, wgs66, wgs72, and wgs84 (see table below).

This command can be run either non-interactively or interactively. The user can run the program non-interactively by entering desired flag settings and parameter values on the command line using the following format:

m.u2ll [**-srwod**] **spheroid**=*name* [**zone**=*value*] [**input**=*name*] [**output**=*name*]

Alternately, the user can simply type:

m.u2ll

on the command line. In this case, the user will be prompted for parameter values and flag settings through the standard interface described in the manual entry for *parser*.

Input can be entered from the keyboard or from an input file. In either case, input should be entered with one UTM easting and northing pair per line, in the format shown below:

```
easting northing
easting northing
easting northing
easting northing
  .
  .
end
```

If the user sets the -r flag, *m.u2ll* will expect the order of the coordinates to be reversed, and stated as northing, easting pairs, rather than as easting, northing pairs. This is useful for passing ASCII GRASS vector (*/dig*) files, whose coordinates are stated as northing, easting pairs, directly through *m.u2ll*.

Similarly, the user can elect to send output to an output file or (by default) to standard output (the user's terminal screen). Example input and output are shown below (see EXAMPLE).

Program flag settings and parameters have the following meanings.

Flags:

-s Specified UTM zone is in the southern hemisphere.

-r The order of coordinates is reversed in the input, and entered as: *north east*. This option allows the user to pass an ascii vector file through *m.u2ll*.

-w Do not flag invalid east, north input lines as errors.

-o Flag other invalid input lines as errors.

-d Output latitude/longitude values in decimal degrees, rather than in the form *dd:mm:ss*.

Parameters:

spheroid=*name* Reference spheroid (ellipsoid).
Options: airy, australian, bessel, clark66, everest, grs80, hayford, international, krasovsky, wgs66, wgs72, wgs84

zone=*value* UTM zone in which UTM coordinates are located.
Options: 1-60

input=*name* Name of input file containing UTM values to be converted.

output=*name* Name to be assigned to output file containing longitude and latitude values.

AVAILABLE SPHEROIDS
(The on-line listing includes only the spheroid names.)

Spheroid:	Semi-major axis (Equatorial Radius) (a):	Eccentricity sqrd (e), Flattening (f), or Polar Radius (b):	Commonly used for:
airy	a=6377563.396	e=.006670540	
australian	a=6378160	f=1/298.25	Australia
bessel	a=6377397.155	e=.006674372	Japan
clark66	a=6378206.4	b=6356583.8	N. America
everest	a=6377276.345	e=.0066378466	India, Burma
grs80	a=6378137	f=1/298.257	
hayford	a=6378388	f=1/297	
international	a=6378388	f=1/297	Europe
krasovsky	a=6378245	f=1/298.3	
wgs66	a=6378145	f=1/298.25	worldwide coverage
wgs72	a=6378135	f=1/298.26	worldwide coverage
wgs84	a=6378137	f=1/298.257223563	worldwide coverage

EXAMPLE

Assume the user has input the command:

m.u2ll -s spheroid=wgs72 zone=4 input=utm.infile output=ll.outfile

where the input file *utm.infile* contains the following easting and northing UTM coordinate values and zone designations:

```
237740.85 2167292.10
238740.00 2167000.00
239000.00 2167100.00
237100.00 2166000.00
end
```

Output would then be sent to the output file *ll.outfile*, containing the below longitude and latitude coordinate values:

```
166:02:25.645137W 70:27:46.615528S
166:00:53.237056W 70:27:59.692673S
166:00:27.23258W 70:27:57.454281S
166:03:41.428895W 70:28:25.61617S
end
```

NOTES

Users can add information to the ellipsoid parameter definition file on their systems (located in $GISBASE/etc/ellipse.table) to add spheroids not now among those supported by GRASS.

See *m.ll2u* for a brief discussion of spheroids.

The UTM zone designation determines on what area of the earth a point is found. The same UTM coordinates will be found in each different UTM zone. Look at the marginalia of your source map to determine into which UTM zone your UTM coordinates fall. Although the user can permissibly omit specification of a UTM zone when running this program under a UTM data base LOCATION, it is safer to specify it (see DESCRIPTION, above).

m.u2ll converts the first pair of coordinates on each line of input and leaves anything else on the line alone. If a line begins:

```
xxxxxx.xx xxxxxxx.xx
```

then the xxxxxx.xx xxxxxxx.xx UTM coordinate pair is converted to a longitude, latitude pair. Any other information appearing on the line is left alone. If the line doesn't begin with a pair of coordinates in the above format, then the line is left as it is.

FILES

See ellipsoid parameter definition file in $GISBASE/etc/ellipse.table.

SEE ALSO

For australian, clark66, grs80, hayford, international, krasovsky, and wgs72 ellipsoid parameters, see:
John P. Snyder, *Map Projections - A Working Manual*, U.S. Government Printing Office, Washington DC, 1989. U.S. Geological Survey Professional Paper 1395; from Table 1, p.12.

For bessel, airy, everest, and wgs66 ellipsoid parameter values, see:
Thomas O. Seppelin, *The Department of Defense World Geodetic System 1972*, presented at the International Symposium on Problems Related to the Redefinition of North American Geodetic Networks, Fredericton, New Brunswick, Canada in May, 1974; see Table 9, p.35.

For wgs84 parameter values, see:
U.S. Naval Oceanographic Labs.

Also read GRASS User's Reference Manual entries for:
d.labels, d.points, d.sites, d.where, m.datum.shift, m.gc2ll, m.ll2gc, m.ll2u, and *parser*

AUTHOR

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

NAME

p.chart – Prints the color chart of the currently selected printer.
(GRASS Hardcopy Output Program)

SYNOPSIS

p.chart

DESCRIPTION

This function prints the color chart of the user's hardcopy color printer. The colors in the chart are numbered with the (device-dependent) number associated with each color. This function is useful both for the *p.colors* command as well as for color selection with *p.map*.

The user must select a printer using *p.select* before running this command.

SEE ALSO

d.colors, p.colors, p.map, p.select, r.colors

AUTHOR

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

NAME

p.colors – Allows the user to develop a color table that associates map categories with user-specified printer colors.
(GRASS Hardcopy Output Program)

SYNOPSIS

p.colors

DESCRIPTION

This function allows the user to modify a color table for a raster map layer, by assigning colors to the categories in the raster map layer based on printer color numbers (instead of red, green, blue percentages).

The user will need to have the printer color chart available to properly select the colors (see manual entry for *p.chart*).

The *p.colors* function allows the user to exercise perfect control over the colors which appear in the hardcopy image.

NOTES

If the user assigns a printer color number outside of the printer's range, *p.colors* will not complain, but will not save the out-of-range printer color number to the raster map layer's color table.

This command modifies the color table associated with a map layer, and will therefore also affect the colors in which a map layer is displayed on the user's graphics monitor. (See map color table files stored under $LOCATION/colr and $LOCATION/colr2.)

SEE ALSO

d.colors, p.chart, p.map, r.colors

AUTHOR

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

NAME

p.icons – Creates and modifies icons for map display and output.
(GRASS Hardcopy Output Program)

SYNOPSIS

p.icons

DESCRIPTION

This program allows the user to create and maintain icons which are used by the *p.map* and *d.icons* commands to depict sites.

FILES

Icon files created by the user are stored under $LOCATION/icons.

SEE ALSO

d.icons, d.points, d.sites, p.map, s.menu

AUTHOR

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

NAME

p.labels – Create labels for hardcopy maps.
(GRASS Hardcopy Output Program)

SYNOPSIS

p.labels

DESCRIPTION

This module allows the user to create or modify labels files. These labels files, which are stored in the database, define text information for printing with *p.map* and for graphics display with *d.paint.labels.* Each label has components which determine the text, the location of the text on the image, its size, and the background for the text.

The interface is a screen-oriented input/edit layout. Each label is entered with a single screen. After filling in the required information (described below), the user hits <ESC> to accept the label and start a new one. After the last label has been accepted, the user then hists the <ESC> one more time (on an empty label screen) to exit the module and save the labels.

SCREEN LAYOUT

The screen layout for the labels looks like this:

```
----------------------------------------------------------------------
PAINT LABELS:  labelfile              new labels                   [1]

TEXT: ________________________________________      SKIP: no__
      ________________________________________
      ________________________________________
      ________________________________________

LOCATION:   EAST: ________________  OFFSET: ________
           NORTH: ________________  OFFSET: ________
       PLACEMENT: center___________

FONT:               standard_________
TEXT SIZE:          500_____
TEXT COLOR:         black___________  WIDTH: 1______
HIGHLIGHT COLOR:    none____________  WIDTH: 0______

BACKGROUND COLOR:  white___________
BORDER COLOR:       black___________
OPAQUE TO VECTORS: yes_____


           AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
                       (OR <Ctrl-C> TO CANCEL)
----------------------------------------------------------------------
```

The label information that must be provided is:

TEXT: Up to four lines of text. Lines in multiple line labels will appear one above the next.

SKIP: yes| no. If *no*, label will be printed. If *yes*, the label will be retained in the file but not printed.

LOCATION: Determines where the text will be located on the image. The user specifies the easting and northing, and (optionally) specifies a vertical and horizontal offset (in printer pixels) from the specified easting/northing. (The vertical offset will shift the location to the south if positive, north if negative. The horizontal offset will shift the location east if positive, west if negative.) These offsets are provided to allow finer placement of labels.

PLACEMENT: Determines which part of the label to which the location refers. If placement is unspecified, the label is centered (*center*), by default. Label placement may be specified as:

lower left	(lower left corner of the text)
lower right	(lower right corner of the text)
lower center	(bottom center of the text)
upper left	(upper left corner of the text)
upper right	(upper right corner of the text)
upper center	(top center of the text)
center	(center of the text)

FONT: This specifies the font to use. The following fonts are available:

> cyrilc gothgbt gothgrt gothitt greekc greekcs greekp greeks italicc italiccs italict romanc romancs romand romans romant scriptc scripts

The word *standard* can be used to specify the default font (which is *romans*).

TEXT SIZE: This determines the size of the letters. The size specifies the vertical height of the letters in meters on the ground. Thus text will grow or shrink depending on the scale at which the map is drawn. The default text size, if none is specified, is *500*.

TEXT COLOR: This selects the text color. If unspecified, the label's text is drawn in *black*, by default. The text color can be specified in one of four ways:

1) By color name:
> aqua black blue brown cyan gray green grey indigo magenta orange purple red violet white yellow

2) As red, green, blue percentages. for example: .5 .4 .7
> (This form is not supported by *d.paint.labels*.)

3) By printer color number to get the exact printer color.
> (This form is not supported by *d.paint.labels*.)

4) Specify *none* to suppress the lettering.

WIDTH: This determines the line thickness of the letters. The normal text width should be set to 1. Larger numbers can be used to simulate bold face. (*d.paint.labels* ignores this value and always uses 1. *1* is also the default width to which the width is set by *paint*, if none is specified by the user.)

HIGHLIGHT COLOR: The text can be highlighted in another color so that it appears to be in two colors. The text is drawn first in this color at a wider line width, and then redrawn in the text color at the regular line width. No highlight color (*none*) is used, by default, if unspecified by the user. To specify use of no highlight color, specify *none*. (See TEXT COLOR above for a list of permissible color names.)

HIGHLIGHT WIDTH: Specifies how far from the text lines (in units of pixels) the highlight color should extend. The default highlight width is set to *0* (i.e., no highlight color).

BACKGROUND COLOR: Text may be boxed in a solid color by specifying a background color. Specify *none* for no background. The default background color setting, if unspecified by the user, is *white*. (See TEXT COLOR above for a list of permissible color names).

BORDER COLOR: Select a color for the border around the background. Specify *none* to suppress the border. The default border color used, if unspecified, is *black*. (See TEXT COLOR above for a list of permissible color names).

OPAQUE TO VECTORS: *yes*| *no*. This field only has meaning if a background color is selected. *yes* will prevent vector lines from entering the background. *no* will allow vector lines to enter the background. The default setting, if unspecified by the user, is *yes*.

SEE ALSO

p.map, p.select, p.icons, d.paint.labels

AUTHOR

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

NAME

p.map – Hardcopy color map output utility.
(GRASS Hardcopy Output Program)

SYNOPSIS

p.map
p.map help
p.map [**input**=*name*] [**scale**=*mapscale*]

DESCRIPTION

p.map produces hardcopy color map products on your system's color output device. Output can include a raster map, any number of vector overlays, site data, text labels, and other spatial data.

This program has 2 distincts modes of operation. The command-line mode requires the user to prepare a file of mapping instructions describing the various spatial and textual information to be printed prior to running *p.map*. The interactive mode (i.e., no command-line arguments) will prompt the user for items to be mapped and does not require the user to prepare a file of instructions.

The command-line parameters are:

input=*name* File containing mapping instructions. (or enter **input=-** to enter from keyboard). These instructions are described in detail below.

scale=*mapscale* Scale of the output map, e.g. 1:25000
Default: 1panel

This parameter is provided as a convenience. It is identical to the *scale* mapping instruction described below.

Note: the user must select an output device using *p.select* before running *p.map*. Also, the *preview* device can be selected to view the output from *p.map* on the graphics monitor instead of sending it to a paper printer.

MAPPING INSTRUCTIONS

The mapping instructions allow the user to specify various spatial data to be plotted. These instructions are normally prepared in a regular text file using a system editor. Some instructions are single line instructions while others are multiple line. Multiple line instructions consist of the main instruction followed by a subsection of one or more additional instructions.

colormode

Selects the appropriate method to color raster map layers and images.

USAGE: **colormode** approx| best

There are two methods: *approximate* and *best.* From a user perspective, *approximate* can be used for raster map layers with few categories, such as soils, and *best* should be used for images like LANDSAT images or NHAP photos, or maps with many categories. The *approximate* mode treats each pixel independently, giving it the printer color that best approximates the true color. The *best* mode "blends" colors from pixel to pixel using a dithering technique to simulate more colors than the printer can actually print. The default, if unspecified, is *best.*

This example would select the *approximate* colormode. The assumption is that the raster map layer being printed is has few colors or that the colors would not look good dithered.

EXAMPLE: **colormode** approx

colortable

Includes the color table for the raster map layer in the legend below the map

USAGE: **colortable** [y| n]

The color table will display the colors for each raster map layer category value and the category label. To get a color table, you must have previously requested a raster map layer. Omitting the **colortable** instruction would result in no color table. **Note**: Be careful about asking for color tables for raster map layers which have many categories, such as elevation. This could result in the printing of an extremely long color table!!!!!

This example would print a color table as part of the legend to the map.
EXAMPLE: **colortable** y

comments

Prints comments beneath the map . You may submit comment text line by line during *p.map* execution or a via a prepared comments file.

USAGE: **comments** [commentfile]
comments
end

This example prints the comment "This is a comment" in the legend below the map.
EXAMPLE: **comments**
This is a comment.
end

This example prints whatever is in the file *veg.comments* in the legend below the map.
EXAMPLE: **raster** vegetation
comments veg.comments
end

Presumably, the file *veg.comments* contain comments pertaining to the raster map layer *vegetation*, such as "This map was created by classifying a LANDSAT TM image".

defpat

Defines area fill patterns to be used in the *setpat* instruction.

USAGE: **defpat** name
pattern
color # color
end

The pattern is given a name on the *defpat* instruction line. The pattern which follows is composed of a sequence of numbers 0-9 (and blanks, which are equivalent to 0). The blanks or zeros indicate holes in the pattern where the normal category color would show thru. The other number 1-9 indicate pattern pixels and can be assigned any color. The default color for all the digits will be black unless specified with the *color* instruction. The color option will begin by entering the word *color* followed by one of the digits (1-9) in the pattern, followed by one of the NAMED COLORS. This should be repeated for each of the digits specified to avoid using black. The instruction *end* terminates the pattern definition. Of course, the user can define more patterns by entering more *defpat* instructions.

NOTE: Do NOT indent the pattern. Leading blank spaces will be interpreted as 0's.

This example creates a black horizontal line pattern.
EXAMPLE: **defpat** horiz
1
0
0
0
color 1 black
end

This example creates a green vertical line pattern.
EXAMPLE: **defpat** vert
1000
color 1 green
end

This example creates a red diagonal line pattern.
EXAMPLE: **defpat** diag
00001
0001
001
01
1
color 1 red
end

This example creates a two-toned tree pattern with orange trunks and green leaves.
EXAMPLE: **defpat** tree

```
    2
   222
  22122
 22 1 22
    1

            2
           222
          22122
         22 1 22
            1
```

color 1 orange
color 2 black
end

grid

Overlays a coordinate grid onto the output map.

USAGE: **grid** spacing
color color
numbers # [color]
end

The **spacing** of the grid is given (in the geographic coordinate system units) on the main instruction line. The subsection instructions allow the user to specify the **color** of the grid lines, whether coordinate **numbers** should appear on the grid lines, and if they should appear every grid line (1), every other grid line (2), etc., and what color the numbers should be. The defaults are black grid lines, unnumbered.

This example would overlay a green grid with a spacing of 10000 meters (for a metered database, like UTM) onto the output map. Alternate grid lines would be numbered with red numbers.

EXAMPLE: **grid** 10000
color green
numbers 2 red
end

labels

Selects a labels file for output (see manual entry for *p.labels).*

USAGE: **labels** labelfile| *list*

This example would paint labels from the labels file called *town.names*. Presumably, these labels would indicate the names of towns on the map.

EXAMPLE: **labels** town.names

line

Draws lines on the output map.

USAGE: **line** east north east north
line x% y% x% y%
color color
width #
masked [y| n]
end

The beginning and ending points of the line are entered on the main instruction. These points can be defined either by map coordinates or by using percentages of the geographic region. The user may also specify line **color, width** in pixels, and if the line is to be **masked** by the current mask. (See manual entry for *r.mask* for more information on the mask.)

This example would draw a yellow line from the point x=10% y=80% to the point x=30% y=70%. This line would be 2 pixels wide and would appear even if there is a mask.

EXAMPLE: **line** 10% 80% 30% 70%
color yellow
width 2
masked n
end

Of course, multiple lines may be drawn with multiple *line* instructions.

outline

Outlines the areas of a raster map layer with a specified color.

```
USAGE:   outline
           color  color
           end
```

Distinct areas of the raster map will be separated from each other visually by drawing a border (or outline) in the specified **color** (default: black). Note: it is important the user enter the instruction **end** even if a color is not chosen. (It is hoped that in the future the outline of a different raster map layer other than the one currently being painted may be placed on the map.)

This example would outline the category areas of the *soils* raster map layer in grey.

```
EXAMPLE:   raster soils
           outline
           color grey
           end
```

point

Places additional points or icons on the output map.

```
USAGE:   point east north
         point x% y%
           color color
           icon iconfile| list
           size #
           masked [y| n]
           end
```

The point location is entered in the main instruction line by giving either the map coordinates or by using percentages of the geographic region. The user may also specify the point **color,** the **icon** file to be used to represent the point location (see the the manual entry for *p.icons),* the **size** of the icon in integer multiples of the pattern in the icon file, and whether the point is to be **masked** by the current mask. (See manual entry for *r.mask* for more information on the mask.)

This example would place a purple diamond (from icon file *diamond*) at the point (E456000 N7890000). This diamond would be the same size it is in the diamond icon file and would not be masked by the current mask.

```
EXAMPLE:   point 456000 7890000
             color purple
             icon diamond
             size 1
             masked n
             end
```

Of course, multiple points may be drawn with multiple *point* instructions.

raster

Selects a raster map layer for output.

USAGE: **raster** mapname| *list*

For each *p.map* run, only one raster map layer can be requested. If no raster map layer is requested, a completely white map will be produced. It can be useful to select no raster map layer in order to provide a white background for vector images.

This example would paint a map of the raster map layer *soils*.
EXAMPLE: **raster** soils

read

Provides *p.map* with a previously prepared input stream.

USAGE: **read** previously prepared UNIX file

Mapping instructions can be placed into a file and read into *p.map.*

Note: *p.map* will not search for this file. The user must be in the correct directory or specify the full path on the **read** instruction. (Note to /bin/csh users: ˜ won't work with this instruction).

This example reads the UNIX file *pmap.roads* into *p.map*. This file may contain all the *p.map* instructions for placing the vector map layer *roads* onto the output map.

EXAMPLE: **read** pmap.roads

The user may have created this file because this vector map layer is particularly useful for many *p.map* outputs. By using the **read** option, the user need not enter all the input for the **vector** instruction, but simply **read** the previously prepared file with the correct instructions.

region

Places the outline of a smaller geographic region on the output.

USAGE: **region** regionfile| *list*
  color color
  width #
  end

Geographic region settings are created and saved using *g.region*. The *p.map region* option can be used to show an outline of a smaller region which was printed on a separate run of *p.map* on other user-created maps.

The user can specify the **color** and the **width** (in pixel units) of the outline. The default is a black border of one pixel width.

This example would place a white outline, 2 pixels wide, of the geographic region called *fire.zones* onto the output map. This geographic region would have been created and saved using *g.region*.

EXAMPLE: **region** fire.zones
  color white

width 2
end

scale

Selects a scale for the output map.

USAGE: **scale** *scale*

The scale can be selected either as:

a relative ratio, e.g. 1:25000;

an absolute width of the printed map, e.g. 10 inches;

the number of printed paper panels, e.g. 3 panels;

the number of miles per inch, e.g. 1 inch equals 4 miles.

This example would set the scale of the map to 1 unit = 25000 units.
EXAMPLE: **scale** 1:25000

setcolor

Overrides the color assigned to one or more categories of the raster map layer.

USAGE: **setcolor** cat(s) color

This example would set the color for categories 2,5 and 8 of the raster map layer *watersheds* to white and category 10 to green. (**NOTE**: no spaces are inserted between the category values.)

EXAMPLE: **raster** watersheds
setcolor 2,5,8 white
setcolor 10 green

Of course, *setcolor* can be requested more than once to override the default color for additional categories. More than one category can be changed for each request by listing all the category values separated by commas (but with no spaces).

setpat

Assigns a (previously defined) pattern on a raster map layer category.

USAGE: **setpat** cat name
or **setpat** *builtin*
or **setpat** *all*

The user can choose to use: the name of a specific pattern created using **defpat** (see above); the patterns built into *p.map*; or all the patterns the user may have created.

This example assigns the vertical pattern created using **defpat** (see example in **defpat** above) to category 3 of the raster map layer *vegetation* and the tree pattern (see example in **defpat** above) to category 10.
EXAMPLE: **raster** veg
setpat 3 vert
setpat 10 tree

This example reads a previously prepared UNIX file *horiz.pat* with the correct **defpat** instructions for creating a black horizontal pattern (see example in **defpat** above) and assigns that pattern to category 5 of the raster map layer *soils* via the **setpat** instruction.

EXAMPLE: **raster** soils
read horiz.pat
setpat 5 horiz

To select the builtin patterns:

EXAMPLE: **raster** soils
setpat builtin

To select individual builtin patterns:

EXAMPLE: **raster** soils
setpat 5 #1
setpat 10 #2

sites

Selects sites data to be placed on the output map (see manual entry for *s.menu*).

USAGE: **sites** sitemap| *list*
color color
icon iconfile| *list*
size #
desc [y| n]
end

The user may specify the the **color** of the sites (see section on NAMED COLORS below); the **icon** to be used to represent the presence of a site (see the manual entry for *p.icons*); the **size** of the icon (number of times larger than the size it is in the icon file); and whether or not the **desc**ription associated with each site is also to be printed.

This example would paint a sites map with blue windmills (from an icon file created by the user using the *p.icons* GRASS command) placed at all windmill locations (from a sites list). These windmills would be two times larger than the size of the icon in the icon file and have descriptions from the sites list file printed beside them.

EXAMPLE: **sites** windmills
color blue
icon windmill
size 2
desc y
end

text

Places text on the map.

USAGE: **text** east north text
text x% y% text
font fontname
color color| none
width #
hcolor color| none
hwidth #
background color| none

border color| none
size #
ref reference point
xoffset #
yoffset #
opaque [y| n]
end

The user specifies where the text will be placed by providing map coordinates or percentages of the geographic region map. The text follows these coordinates on the same instruction line. More than one line of text can be specified by notating the end of a line with **\n** (e.g. USA \nCERL).

The user can then specify various text features:

font: cyrilc gothgbt gothgrt gothitt greekc greekcs greekp greeks italicc italiccs italict romanc romancs romand romans romant scriptc scripts (The default font is romans);

color (see NAMED COLORS);

width of the lines used to draw the text (to make thicker letters);

size as the vertical height of the letters in meters on the ground (text size will grow or shrink depending on the scale at which the map is painted);

the highlight color (**hcolor**) and the width of the highlight color (**hwidth**);

the text-enclosing-box **background** color; the text box **border** color;

ref. This reference point specifies the text handle - what part of the text should be placed on the location specified by the map coordinates. Reference points can refer to: [lower| upper| center] [left| right| center] of the text to be printed;

yoffset, which provides finer placement of text by shifting the text a vertical distance in pixels from the specified north. The vertical offset will shift the location to the south if positive, north if negative;

xoffset, which shifts the text a horizontal distance in pixels from the specified east The horizontal offset will shift the location east if positive, west if negative;

whether or not the text should be **opaque** to vectors. Entering **no** to the opaque option will allow the user to see any vectors which go through the text's background box. Otherwise, they will end at the box's edge.

This example would place the text *SPEARFISH LAND COVER* at the coordinates E650000 N7365000. The text would be a total of 3 pixels wide (2 pixels of red text and 1 pixel black highlight), have a white background enclosed in a red box, and be 500 meters in size. The lower right corner of the text would be centered over the coordinates provided. All vectors on the map would stop at the border of this text.

EXAMPLE: **text** 650000 7365000 SPEARFISH LAND COVER
font romand
color red
width 2
hcolor black
hwidth 1

background white
border red
size 500
ref lower left
opaque y
end

vector

Selects a vector map layer for output.

USAGE: **vector** vectormap| *list*
color [#] color
width #
hcolor color
hwidth #
masked [y| n]
style 0-9
end

The user can specify the **color** of the vectors; the **width** of the vectors lines in pixels; the highlight color (**hcolor**) for the vector lines; the width of the highlight color (**hwidth**) in pixels; whether or not the raster map layer is to be **masked** by the current mask (see manual entry *r.mask* for more information on the mask); and the line **style**. The line style allows the vectors to be dashed in different patterns and colors. This is done by typing a series of numbers (0-9) in a desired sequence or pattern. Colors for the numbers (1-9) can be assigned using the **color** instruction. Blanks and non-digit characters are recognized as 0's. Using 0 would allow the colors of the raster map layer (or the background color if no raster map layer was selected) to show through.

This example would paint a map of the vector map layer named *streams*. These streams would be a total of 3 pixels wide (the inner two pixels blue and the outer highlight pixel white). The map would not show streams outside of the current mask.

EXAMPLE: **vector** streams
color blue
width 2
hcolor white
hwidth 1
masked y
end

This example would paint a map of the vector map layer *roads*. These roads would be 2 pixels wide and would be dashed blank-black-red (the blank areas would show what lies under the roads). This map would show roads inside and outside of the current mask.

EXAMPLE: **vector** roads
width 2
style 001122
color 1 black
color 2 red
masked n
end

verbose

Changes the amount of talking *p.map* will do.

USAGE: **verbose** [0| 1| 2]

A higher value implies more chatter. The default is 2. This example sets the amount of chatter to a minimum.

EXAMPLE: **verbose** 0

end

Terminates input and begin painting the map.

USAGE: **end**

NAMED COLORS

The following are the colors that are accepted by *p.map*:

aqua black blue brown cyan gray green grey indigo magenta orange purple red violet white yellow

ICONS VS. PATTERNS

Icons and patterns as used in *p.map* are not the same thing. Patterns are defined and are normally used to cover those extended areas covered by a raster map layer category. A pattern will repeat above, below and adjacent to itself. Icons are used to represent a single point.

Patterns are supported directly within *p.map* using the *defpat* instruction, while icons must be created using the *p.icons* program.

EXAMPLE p.map INPUT FILE

The following is an example of a *p.map* script file. The file has been name *spear.soils*. For the purposes of illustration, the file is in two columns. This script file can be entered at the command line:

p.map input=spear.soils

raster soils
vector streams
  color blue
  width 2
  hcolor white
  hwidth 1
  masked y
  end
vector roads
  width 2
  style 001122
  color 1 black
  color 2 red
  masked n
  end
labels town.names
region subregion
  color white
  width 2
  end
grid 10000
  color green
  numbers 2 red
  end
outline
  color black
  end
colortable y
comments
  This is a comment
  end
scale 1:25000 **end**
setcolor 6,8,9 white
setcolor 10 green

(cont.)
defpat diag
000001
00001
0001
001
01
1
  color 1 red
  end
setpat 4 diag
text 608000.00 3476004.73 SPEARFISH SOILS MAP
  color red
  width 2
  hcolor black
  hwidth 1
  background white
  border red
  size 500
  ref lower left
  opaque y
  end
line 606969.73 3423092.91 616969.73 3423092.91
  color yellow
  width 2
  opaque yes
  end
point 40% 60%
  color purple
  icon diamond
  size 2
  masked n
end

INTERACTIVE MODE

If the user simply enter *p.map* without arguments, then a simple prompting session occurs. Some, but not all of the non-interactive requests are available at this level.

SEE ALSO

p.chart, p.icons, p.labels, p.select

AUTHOR

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

NAME

p.map.new – Color map output utility.
(GRASS Hardcopy Output Program)

SYNOPSIS

p.map.new
p.map.new [**input=** *name*] [**scale=** *mapscale*]

DESCRIPTION

The *p.map.new* command produces color maps for output on a color hardcopy device or a graphics monitor. Output can include a raster map, any number of vector overlays, site data, text labels, and other map elements.

This command has three modes of operation. The command-line mode requires a previously prepared file of mapping instructions describing the map elements to be printed. The interactive mode (i.e., no command-line arguments) will prompt the user for items to be mapped and does not require the user to prepare a file of instructions. The keyboard mode is started by entering a hyphen (-) for the **input** parameter. The *p.map.new* instructions would then be entered via the keyboard.

The command-line parameters are:

input= File containing mapping instructions (or enter **input= -** to enter instructions from the keyboard). These instructions are described in detail below.

scale= Scale of the output map, e.g. 1:25000
Default: 1 panel
The options for this parameter are identical to the *scale* mapping instruction described below. If a *scale* instruction is present in an input file, it is superseded by the command-line **scale** parameter.

An output device can be selected using *p.select* before running *p.map.new*. Valid devices include on-line hardcopy devices, plus *preview, preview2,* and *ppm*. See manual entry for *p.select*.

The current geographic region determines the area that is mapped using *p.map.new.*

NON-INTERACTIVE MAPPING INSTRUCTIONS

Mapping instructions allow the user to specify various map elements to be plotted. These instructions are normally prepared in an ASCII text file using a system editor. All of the listed mapping instructions are usable in a prepared file in the command-line mode. Not all of them are available in the interactive and keyboard modes.

Some instructions are single line instructions while others are multiple line. Multiple-line instructions consist of the main instruction followed by a subsection of one or more additional instructions. All multiple-line instructions must be completed by the **end** terminator.

Some instructions, such as those using data layers, icons, or labels, access files via the current mapset search path.

barscale

Places a barscale on the output map.

USAGE: **barscale** east north
barscale x% y%
unit *ft*| *mi*| *m*| *km*
length #
interval #
style *dash*| *tick*
width #
color color
textsize #
textcolor color| *none*
textfont font
background color| *none*
border color| *none*
end

The location of the zero point of the scale bar is entered on the first instruction line. The location can be defined either by map coordinates or by percentages of the map area, where 0% 0% is the lower left corner of the map.

The user specifies the barscale **unit** of measurement, the total **length** using that unit, and the length of one **interval** (a smaller length evenly divisible into the total length).

The **style** of the scale bar can be specified. The *dash* style has solid lines representing each interval, separated by gaps. The *tick* style has a solid total length with vertical ticks marking each interval.

The user can also specify the **width** of the bar in pixels, its **color** (see VALID COLORS NAMES), the **textcolor**, **textsize** in geographic units, **textfont** (see VALID FONT NAMES), **background** color, and **border** color.

The **barscale** instruction set must be completed with the **end** terminator.

This example would result in a scale bar representing two kilometers. Vertical ticks would be placed at the scale origin, the mid-point, and at the end. The black bar and its accompanying black text would overlay a white box trimmed be a red border.

EXAMPLE: **barscale** 605000 4915000
unit km
length 2
interval 1
style tick
width 2
color black
textcolor black
textsize 150
background white
border red
end

colormode

Selects the method to portray the colors of the raster map layer or image.

USAGE: **colormode** *approx*| *best*

There are two options for **colormode**: *approx* and *best*. The *approx* option should be used for raster map layers with few categories, and *best* should be used for images like LANDSAT images or NHAP photos, or maps with very many categories. The *approx* mode treats each pixel independently, giving it the printer color that best approximates the true color. The *best* mode "blends" colors from pixel to pixel using a dithering technique to simulate more colors than the printer can actually print. If unspecified, the default is *best.*

This example would select the *approx* colormode. The assumption is that the raster map layer being printed has few colors or that the colors would not look good dithered.

EXAMPLE: **colormode** approx

colortable

Includes the color table for the raster map layer in the area below the map on hardcopy output.

USAGE: **colortable** [*y*| *n*]

The color table will display the colors for each raster map layer category and the category value. The **colortable** instruction can not precede the **raster** instruction in the *p.map.new* input. The color table is not shown when the output device is the color monitor.

The user should be careful about asking for color tables for raster map layers that have very many categories, such as an elevation layer. This could result in the printing of an extremely long and generally useless color table!

This example would print a color table below the data area of the map.

EXAMPLE: **colortable** y

comments

Prints comments beneath the map on hardcopy output.

USAGE: **comments** [commentfile]
comments
end

Comment text can be entered in the *p.map.new* file or from a separate, previously prepared file. Comments are not shown when the output device is the color monitor. The **comments** instruction set must be completed by the **end** terminator.

This example prints the comment "This is a comment" below the data area on the the map.

EXAMPLE: **comments**
This is a comment.
end

This example prints the text in a file called "veg.comments" in the current directory.

EXAMPLE: **raster** vegetation
comments veg.comments
end

defpat

Defines an area fill pattern to be used in **setpat** instructions.

USAGE: **defpat** name
pattern
color # color
end

An area fill pattern is given a name on the **defpat** instruction line. This name can then be used in subsequent *setpat* instructions. The **defpat** instruction can be used more than once to specify multiple patterns.

The specified pattern is composed of a sequence of numbers (0-9, and blanks, which are equivalent to 0) on one or more lines. The zeros and blanks indicate areas in the pattern where the normal category colors are visible. The other digits, 1-9, indicate pattern pixels and can be assigned any valid color.

The **color** option specifies a non-zero digit in the pattern, followed by a valid color name. It can be repeated for each of the non-zero digits in the pattern. The default color for all non-zero digits is black unless specified with the **color** option.

The **defpat** instruction set must be completed by the **end** terminator.

In the *p.map.new* input, the **defpat** instruction must precede any **setpat** instruction using the specified pattern.

Note: Indented pattern specifications will be interpreted as having leading blanks.

This example creates a black horizontal line pattern called "horiz". Each black line in the pattern would be one pixel wide and would be three pixels from neighboring lines.

EXAMPLE: **defpat** horiz
1
0
0
0
color 1 black
end

This example creates a green vertical line pattern.

EXAMPLE: **defpat** vert
1000
color 1 green
end

The following example creates a red diagonal line pattern.

EXAMPLE:
```
defpat diag
00001
0001
001
01
1
color 1 red
end
```

This example creates a two-toned tree pattern with orange "trunks" and green "leaves".

EXAMPLE:
```
defpat tree
   2
  222
 22122
22 1 22
   1

         2
        222
       22122
      22 1 22
         1
color 1 orange
color 2 black
end
```

end

Terminates input and begins the painting of the map to the output device.

USAGE: **end**

An **end** instruction completes the entire input to *p.map.new*. It is normally the last line in an input file, but it can be moved forward to eliminate any instructions following its position.

The **end** instruction for the entire input should not be confused with **end** terminators that are required with all multiple-line instruction sets.

grid

Overlays a coordinate grid on the output map.

USAGE: **grid** spacing
pattern *notick*| *tick* # #
masked [*data*| *nodata*| *all*]
style sequence
width #
color color
numbers # [color] [*in*| *out*]
numbersbg color| *none*
numbersize #

end

The spacing of the grid in geographic coordinate system units must be be specified on the first instruction line. The user can specify the overall look of the grid using the **pattern** parameter. The *notick* option is for a complete net of intersecting lines. The *tick* option is for smaller tick marks where grid intervals intersect. The horizontal and vertical lengths, in pixels, must be specified with the *tick* option.

The user can control the areas covered by the grid by using the **masked** parameter. With the *data* option of **masked**, the grid will be seen over all areas of the map's raster layer except the no-data (category 0) areas. With the *nodata* option, the grid will be seen only over the no-data areas. The entire grid is seen with the *all* option to **masked**.

The grid line **style** can be specified using a series of 1's and 0's. The 1's represent the visible dashes and the 0's represent gaps between the dashes. The default is solid lines. The **width** (in pixels) and **color** of the grid lines can also be specified.

The user can control the placement and look of grid label numbers using the **numbers**, **numbersbg**, and **numbersize** parameters. The **numbers** parameter is used to include grid labels, to specify which labels should be shown (where 1 is every grid label, 2 is every other grid label, etc.), and to specify the label color. It is also used to place the labels inside or outside the current region. The background color behind each label is specified by the **numbersbg** parameter. The user controls the grid label size using the **numbersize** parameter, in geographic units.

The **grid** instruction set must be concluded by the **end** parameter.

When used in a metric location, this example would produce grid ticks every 5000 meters. The purple ticks would have "arms" ten pixels long and would be visible over the entire map area. The purple grid numbers would be 350 meters high (to scale), inside the current region map area, and have no background color. A grid label would appear every 5000 meters.

EXAMPLE: **grid** 5000
pattern tick 10 10
masked all
width 1
color purple
numbers 1 purple in
numbersbg none
numbersize 350
end

The following example would produce black grid lines every 1000 meters. The lines would be visible only in the areas of category 0, and they would be dashed, with one long dash for every short gap. Every other grid label would be shown, each with a white background.

EXAMPLE: **grid** 1000
pattern notick
masked nodata
style 11111100
width 1
color black
numbers 2 black in
numbersbg white

numbersize 200
end

labels

Selects a labels file for output.

USAGE: **labels** labelfile| *list*

The **labels** instruction includes previously prepared label specifications. See manual entry for *p.labels* for correct format of the labels file. The labels file must be accessible via the current mapset search path. The *list* option is available in keyboard mode.

This example would paint labels from a labels file called *town.names*.

EXAMPLE: **labels** town.names

legend

Places a user-designed map legend on the output.

USAGE: **legend** east north
legend x% y%
height #
width #
vlen #
textcolor color
textsize #
textwidth #
xspace #
yspace #
background color
border color
beginrast
ramp *value*| *label vertical*| *horizontal*
catnum cat description
end
beginvect
vectname vectormap description
vecttitle vectormap
end
beginsite
sitename sitemap description
end
end

The location of the upper left corner of the legend must be entered on the first instruction line. The location can be defined either by map coordinates or by percentages of the map area, where 0% 0% is the lower left corner of the map.

The user specifies the **height** and **width** of the boxes that will show raster category colors and/or patterns. The length of line segments that will show vector line colors and patterns is specified with the **vlen** parameter. The user controls horizontal spacing between legend symbols and legend text using **xspace**.

The **yspace** parameter is used to control vertical spacing between legend symbols. All of these measurements are in pixels.

The user designs the legend text using the **textcolor**, **textsize**, and **textwidth** parameters. Colors are listed in the VALID COLOR NAMES section of this manual entry. The **textsize** is specified in geographic units. The **textwidth** is specified in pixels.

The user can specify the color for the **background** box containing the entire legend. If a color is chosen, underlying map elements are opaqued. The user can also specify a **border** color for the legend box.

The user specifies the symbols to be included in the legend using the **beginrast**, **beginvect**, and **beginsite** parameters. Each of these parameters starts a subsection of the **legend** instruction that must be completed by an **end** terminator. These should not be confused with the **end** terminator for the entire **legend** instruction set.

If the user simply uses the **beginrast** parameter followed by **end**, all categories of the map's raster layer will be shown in individual boxes, and the legend labels will be the corresponding category names in the layer's *cats* file. The user can include specific categories and optional labels by using one or more **catnum** lines, each including a category number and the accompanying legend text. If the map's raster layer portrays a continuous range of data, a **ramp** in the legend might be appropriate. The **ramp** can be vertical or horizontal, and it's accompanying text can be either the smallest and largest category values, or the *cats* labels associated with the smallest and largest categories. The **beginrast** subsection must be completed with an **end** terminator.

Symbols for vector data on the map can be included in the legend by using the **beginvect** parameter. If the user simply follows **beginvect** with **end**, all vector layers in the map will be included in the legend. The user can include specific vector layers in the legend by using the **vectname** line one or more times, each including a vector layer name and an accompanying description. The vector layer titles as written in *dig_cats* files can be included as the legend text by using the **vecttitle** line one or more times. The **beginvect** subsection must be completed with an **end** terminator.

Site symbols are included in the legend by using one or more **sitename** lines in the **beginsite** subsection. Each line includes the name of the site list and an accompanying description. The **beginsite** subsection must be completed with an **end** terminator.

The entire **legend** instruction set must be completed by an **end** terminator.

In the *p.map.new* input, the **legend** instruction can not precede the instructions for any of the map elements that are to be shown in the legend.

This example would produce a legend with five symbols: a point symbol, the colors and patterns for three raster categories, and a line representing one vector layer, in that order. The background of the legend would be white and surrounded by a red border. All text in the legend would be black.

EXAMPLE: **legend** 589000 4921200
height 10
width 20
vlen 20
xspace 10
yspace 7
textcolor black
textsize 250

textwidth 1
background white
border red
beginsite
 sitename archsites Arch. site
 end
beginrast
 catnum 4 Sandstone
 catnum 5 Limestone
 catnum 6 Shale
 end
beginvect
 vectname roads Road
 end
end

The following example would produce a legend with a vertical ramp showing all the colors in the map's raster layer. The labels of the first and last categories would be included.

EXAMPLE: **legend** 589000 4921200
height 10
width 20
xspace 10
textcolor black
textsize 250
textwidth 1
background gray
border black
beginrast
 ramp label vertical
 end
end

line

Draws a line that is independent of any vector map layer on the output map.

USAGE: **line** east north east north
line x% y% x% y%
 style sequence
 color [#] color
 width #
 hcolor color
 hwidth #
 masked [*y*| *n*]
 end

The beginning and ending points of the line are entered on the main instruction line. These points can be defined either by map coordinates or by using percentages of the geographic region, where 0% 0% is the lower left corner of the map.

The default line style is a continuous, solid line, but the user can specify a dashed line using the **style** parameter. The **style** parameter can contain a sequence of digits (0-9) that represent a colored pattern on the desired line. Colors can be assigned to each non-zero digit by using the **color** parameter multiple times. If the **color** parameter is used without a specified digit, the named color will be assigned to the entire line. Colors are listed in the VALID COLOR NAMES section in this manual entry.

The user can specify line **width** in pixels. A highlight color can be assigned with **hcolor**, and the highlight's width in pixels can be assigned with **hwidth**. The user can also specify if the line is to be **masked** by the current mask. (See manual entry for *r.mask* for more information on the mask.)

The **line** instruction set must be completed by an **end** terminator.

The **line** instruction can be used more than once to create multiple lines.

This example would draw a blue line from the point x= 10% y= 80% to the point x= 30% y= 70%. The line would be two pixels wide and would appear even if there is a mask.

EXAMPLE: **line** 10% 80% 30% 70%
color blue
width 2
masked n
end

The following example would draw a line with yellow dashes on a black background.

EXAMPLE: **line** 605000 4915000 595300 4918200
style 1111100
color 1 yellow
width 1
hcolor black
hwidth 1
end

outline

Outlines areas of a raster map layer with a specified color.

USAGE: **outline**
color color
end

The **outline** instruction can be used to place a border around all contiguous groups of same-value cells in a raster map layer. A valid color name can be specified with the optional **color** parameter. The default color is black.

The **outline** instruction set must be completed by the **end** terminator, even if the **color** parameter is not used.

The **outline** instruction can not precede a **raster** instruction in a *p.map.new* input file.

The instruction sequence in this example would outline in grey the category areas of a raster map layer called "soils".

EXAMPLE: **raster** soils

outline
color grey
end

point

Places a point symbol on the output map.

USAGE: **point** east north
point x% y%
icon iconfile| *list*
color color
size #
masked [*y*| *n*]
end

The user enters a point symbol location on the main instruction line. The location can be defined either by map coordinates or by percentages of the map area, where 0% 0% is the lower left corner of the map.

The icon to be used can be specified with the **icon** parameter. The user can use any icon in an *icons* directory within the current mapset search path. Icons can be created using *p.icons* or by simply using a system editor. The default icon is a diamond. The *list* option for **icon** is available in keyboard mode.

The user can specify the symbol **color**. Colors are listed in the VALID COLOR NAMES section of this manual entry.

The icon **size** is a positive, floating-point scaling factor of the pattern in the icon file. A size of 1 produces an icon with the same number of pixels (at the output device's resolution) as ASCII characters in the icon file.

The user can also specify whether the point symbol is to be **masked** by the current mask. (See manual entry for *r.mask* for more information on the mask.)

The **point** instruction set must be completed be an **end** terminator. Multiple points may be drawn with multiple **point** instructions.

This example would access an icon file called "box" within the current mapset search path. The red box symbol would be placed at the point E603000, N4921750. The box would have the same number of pixels as characters in the icon file. It would not be masked by the current mask.

EXAMPLE: **point** 603000 4921750
icon box
color red
size 1
masked n
end

raster

Selects a raster map layer for output.

USAGE: **raster** rastermap| *list*

Only one GRASS raster map layer can be specified in a *p.map.new* input file. If no raster map layer is requested, a white background will be produced. The *list* option is available in keyboard mode.

The raster layer must be accessible within the current mapset search path. In a *p.map.new* input file, the **raster** instruction must precede these instructions: **colortable**, **outline**, **setcolor**, and **setpat**. It also must precede any **legend** instruction set that applies to the raster map layer.

This example would paint a map of the raster map layer *soils*.

EXAMPLE: **raster** soils

read

Provides input to *p.map.new* from a previously prepared instruction file.

USAGE: **read** filename

Mapping instructions can be placed in a file and read as input to *p.map.new.* If a certain set of mapping instructions are used in many different maps, they can be placed in one separate file and efficiently accessed by each map's instructions using the **read** instruction.

Note: *p.map.new* will not search for the file to be read. The file must be in the current directory or a full path needs to be specified on the **read** instruction line. (Note to /bin/csh users: the tilde [˜] path alias will not work with this instruction).

This example reads the ASCII file "pmap.roads" into *p.map.new*.

EXAMPLE: **read** pmap.roads

region

Places the outline of a geographic region on the output map.

USAGE: **region** regionfile| *list*
style sequence
color [#] color
width #
hcolor color
hwidth #
masked [*y*| *n*]
end

The user can place the outline of a saved geographic region on the map using **region**. The named region file must be in a *windows* directory within the current mapset search path. Geographic region settings can be created and saved using *g.region*. The *list* option is available in keyboard mode.

The default region outline style is a continuous, solid line, but the user can specify a dashed line using the **style** parameter. The **style** parameter can contain a sequence of digits (0-9) that represent a colored pattern on the desired line. Colors can be assigned to each non-zero digit by using the **color** parameter multiple times. If the **color** parameter is used without a specified digit, the named color will be assigned to the entire region outline. Colors are listed in the VALID COLOR NAMES section in this manual entry.

The user can specify the region outline **width** in pixels. A highlight color can be assigned with **hcolor**, and the highlight's width can be assigned with **hwidth**. The user can also specify if the outline is to be **masked** by the current mask. (See manual entry for *r.mask* for more information on the mask.)

The **region** instruction set must be completed by an **end** terminator.

The **region** instruction can be used more than once to show multiple regions.

This example would produce a white outline, two pixels wide, showing the geographic region called "fire.zones".

EXAMPLE: **region** fire.zones
color white
width 2
end

scale

Specifies the scale of the hardcopy output map.

USAGE: **scale** scale

The scale of the output map can be specified in one of several different forms:

relative ratio e.g. 1:25000

number of geographic units per map unite.g. 1 inch equals 4 miles

absolute width of the printed mape.g. 10 inches

width in number of printed panelse.g. 3 panels

Map inches can be equated with these geographic units: miles, kilometers, and meters. Valid width units are inches, centimeters, and panels. One panel is the single-sheet maximum width available on the hardcopy medium.

The final size of the hardcopy map output is determined by the combination of the specified scale and the current geographic region.

The **scale** instruction does not affect output to the *preview* device. If used, the command-line *scale* parameter overrides the **scale** instruction.

This example would set the scale of the map to one map unit represents 25,000 geographic units.

EXAMPLE: **scale** 1:25000

The following example would specify an output map that would be fifteen centimeters wide.

EXAMPLE: **scale** 15 centimeters

setcolor

Overrides the color assigned to one or more categories of the raster map layer.

USAGE: **setcolor** cat(s) color

The user can assign a desired color to categories in a raster map layer by using **setcolor**. Categories are specified on the parameter line before a valid color name. One or more category numbers can appear on the parameter line, separated by commas (with no spaces), or in ranges using hyphens.

The **setcolor** instruction can be used more than once for assignment of multiple colors.

In the input *p.map.new* file, the **setcolor** instruction must follow the **raster** instruction.

Colors are listed in the VALID COLOR NAMES section of this manual entry.

In this example, the color for raster map categories 1 through 3, plus category 5, would be set to green, categories 4, 6, and 8 would be set to blue, and category 7 would be set to red, regardless of their assigned colors in the database.

EXAMPLE: **raster** watersheds
setcolor 1-3,5 green
setcolor 4,6,8 blue
setcolor 7 red

setpat

Assigns a previously defined pattern to one or more raster map layer categories.

USAGE: **setpat** cat(s) name
setpat cat(s) #number
setpat *all| builtin*

The user can assign a pattern to categories in a raster map layer by using **setpat**. Categories are specified on the parameter line before the name of a pattern defined earlier using **defpat**, or before the number signifying a built-in *p.map.new* pattern. One or more category numbers can appear on the parameter line, separated by commas (with no spaces), or in ranges using hyphens.

The built-in patterns are defined in etc/paint/patterns in the compiled GRASS code directory. Each built-in pattern has an assigned number. These numbers can be used following a pound sign (#) on a **setpat** instruction line. By using the *builtin* option, each category in a raster map layer can be assigned the correspondingly numbered builtin pattern.

All raster map categories can be assigned the same defined pattern if the *all* option is used. In this case, only one pattern should be defined within the *p.map.new* mapping instruction file.

The **setpat** instruction can be used more than once for assignment of multiple patterns.

In the input *p.map.new* file, the **setpat** instruction must follow the **raster** instruction, as well as the **defpat** instruction defining the pattern that is used.

This example assigns a pattern called "vert" to categories 3 and 4 of the raster map layer "vegetation" and a pattern called "tree" to category 10.

EXAMPLE: **raster** veg
setpat 3-4 vert
setpat 10 tree

This example reads a previously prepared ASCII file called *horiz.pat* containing **defpat** instructions for creating a black, horizontal pattern called "horiz", and assigns that pattern to category 5 of the raster map layer "soils".

EXAMPLE: **raster** soils
read horiz.pat
setpat 5 horiz

This example assigns built-in pattern 1 to category 1 of the "soils" raster layer, pattern 2 to category 2, and so on.

EXAMPLE: **raster** soils
setpat builtin

This example assigns built-in pattern 1 to categories 5 through 7 in the "soils" raster map layer, and built-in pattern 2 to categories 10 and 12.

EXAMPLE: **raster** soils
setpat 5-7 # 1
setpat 10,12 # 2

sites

Selects sites data to be placed on the output map.

USAGE: **sites** sitemap| *list*
icon iconfile| *list*
color color
size #
desc [*y*| *n*]
textcolor color
textsize #
end

GRASS sites data can be portrayed on the map using the **sites** instruction. The user can specify the point symbol to be used, and whether labels are to appear next to the symbols. The sites data must be accessible via the current mapset search path. The *list* option is available in keyboard mode.

An icon can be specified with the **icon** parameter. The user can use any icon in an *icons* directory within the current mapset search path. Icons can be created using *p.icons* or by simply using a system editor. The default icon is a diamond. The *list* option for **icon** is available in keyboard mode.

The user can specify the symbol **color**. Colors are listed in the VALID COLOR NAMES section of this manual entry.

The icon **size** is a positive, floating-point scaling factor of the pattern in the icon file. A size of 1 produces an icon with the same number of pixels (at the output device's resolution) as ASCII characters in the icon file.

The **desc** parameter is used to specify whether or not the description of each site in the *site_lists* file is also to be printed. These labels will appear directly to the right of each site symbol. The user controls the color of the labels using the **textcolor** parameters. Valid colors are listed in the named colors section of this manual entry. The label size is specified in geographic units using **textsize**.

The **sites** instruction set must be completed by the **end** terminator.

This instruction can be used more than once to portray multiple site lists.

This example would produce point symbols representing the data in a *site_lists* file called "windmills". An icon called "windmill" would be placed at each site location. These symbols would be two times larger than the size of the icon in the icon file (twice as many pixels as there are characters in the icon file). Descriptions from the sites list file would not be produced in this example.

EXAMPLE: **sites** windmills
 icon windmill
 color blue
 size 2
 desc n
 end

text

Places text at a user-specified location on the map.

USAGE: **text** east north text
 text x% y% text
 textfont font
 size #
 color color| *none*
 width #
 hcolor color| *none*
 hwidth #
 ref reference_point
 rotation #
 xoffset #
 yoffset #
 background color| *none*
 opaque [*y*| *n*]
 border color| *none*
 end

The user specifies where text will be placed by providing map coordinates or percentages of the map area, where 0% 0% is the lower left corner of the map. The text follows the locational information on the same instruction line. Multiple lines of text can be specified by notating the end of a line with **\n** (e.g., USA \nCERL). Leading blanks an be inserted by preceding the text string with a backslash and the blanks (e.g., text 600000 4920500 \ See \nWall Drug).

The user can control the appearance of the text, its location, and the appearance of its background box.

The user can specify **textfont** (see VALID FONT NAMES in this manual entry), **sizein geographic units, color** (see VALID COLOR NAMES), and **width** in pixels. The user can further control the text appearance by specifying a highlight color (**hcolor**) and the width of the highlight color (**hwidth**).

The text is located at the specified coordinate or percentage pair in relation to a reference point on the text string. This point, specified with the **ref** parameter, has two parts. The first part refers to a vertical location on the text string. Valid choices are *lower*, *center*, and *upper*. The second part refers to a horizontal location: *left*, *center*, and *right*.

The text string can be rotated at the reference point by using the **rotation** parameter. The value specified will be the counter-clockwise rotation in degrees from the horizontal.

The **xoffset** parameter provides finer placement of text by shifting the text a horizontal distance in pixels from the specified easting. The **xoffset** will shift the text location east if positive and west if negative. The **yoffset** parameter shifts the text a vertical distance in pixels from the specified northing. The **yoffset** will shift the location to the south if positive, north if negative.

The user can specify if a **background** box is present, and what color it should be. The user can also specify whether or not the background box is **opaque** to other map elements. The color of the **border** of this box can be specified.

This example would place the text "SPEARFISH LAND COVER" at the coordinates E650000, N7365000. The text would be a total of three pixels wide (one pixel of red text and one pixel of black on each side), have a white background enclosed in a red box, and be 500 meters in size (to scale). The lower left corner of the text would be placed at the coordinates provided. All other map elements would not be seen under the text.

EXAMPLE: **text** 650000 7365000 SPEARFISH LAND COVER
textfont romand
color red
width 1
size 500
ref lower left
hcolor black
hwidth 1
background white
border red
opaque y
end

vector

Selects a vector map layer for output.

USAGE: **vector** vectormap| *list*
style sequence
color [#] color
width #
hcolor color
hwidth #
masked [*y*| *n*]
end

GRASS vector data can be portrayed on the map using the **vector** instruction. The name of the vector layer is specified on the first instruction line. The named vector layer must be accessible via the current mapset search path. The *list* option is available in keyboard mode.

The default vector line style is a continuous, solid line, but the user can specify a dashed line using the **style** parameter. The **style** parameter can contain a sequence of digits (0-9) that represent a colored pattern on the vectors. Colors can be assigned to each non-zero digit by using the **color** parameter multiple times. If the **color** parameter is used without a specified digit, the named color will be assigned to the entire lengths of the vectors. Colors are listed in the VALID COLOR NAMES section in this manual entry.

The user can specify the vector line **width** in pixels. A highlight color can be assigned with **hcolor**, and the highlight's width in pixels can be assigned with **hwidth**. The user can also specify if the vectors are to be **masked** by the current mask. (See manual entry for *r.mask* for more information on the mask.)

The **vector** instruction set must be completed by an **end** terminator.

The **vector** instruction can be used more than once to portray multiple vector data layers.

This example would include a vector map layer named "streams" in the output map. These streams would be a total of four pixels wide (two blue pixels with a white outer highlight one pixel wide on each side). The map would not show streams outside of the current mask.

EXAMPLE: **vector** streams
color blue
width 2
hcolor white
hwidth 1
masked y
end

The following example would portray a vector map layer named "roads". These roads would be two pixels wide and would be dashed blank-black-red (the blank areas would show other map elements under the roads). The roads would be visible inside and outside of the current mask.

EXAMPLE: **vector** roads
width 2
style 001122
color 1 black
color 2 red
masked n
end

verbose

Sets the amount of feedback sent out by *p.map.new*.

USAGE: **verbose** *0| 1| 2*

A higher value set using **verbose** results in more feedback. The default is 2.

This example sets the amount of feedback to a minimum.

EXAMPLE: **verbose** 0

VALID COLOR NAMES

The following are the valid color names in *p.map.new*:

aqua	cyan	indigo	red
black	gray	magenta	violet
blue	green	orange	white
brown	grey	purple	yellow

any integer from 0 through 124, representing printer color numbers (see *p.colors* manual entry)

VALID FONT NAMES

The following are the valid font names in *p.map.new*:

cyrilc	greekcs	italict	romant
gothgbt	greekp	romanc	scriptc
gothgrt	greeks	romancs	scripts
gothitt	italicc	romand	
greekc	italiccs	romans (default)	

ICONS VS. PATTERNS

Icons and patterns as used in *p.map.new* are not the same things. Patterns can only be used to cover the extended areas of a raster map layer category. A pattern will repeat above, below and adjacent to itself. Icons are used to represent single points.

Patterns can be defined directly within *p.map.new* using the **defpat** instruction, while icons are created outside of *p.map.new* using the *p.icons* command or a system editor.

EXAMPLE p.map.new INPUT FILE

The following is an example of a *p.map.new* script file. The file has been named "spear.soils". For the purposes of illustration only, the file is shown in two columns. This script file can be entered at the command line.

p.map.new input= spear.soils

```
raster soils
vector streams
  color blue
  width 2
  hcolor white
  hwidth 1
  masked y
  end
vector roads
  width 2
  style 001122
  color 1 black
  color 2 red
  masked n
  end
labels town.names
region subregion
  color white
  width 2
  end
grid 10000
  color green
  numbers 2 red
  end
outline
  color black
  end
colortable y
comments
  This is a comment
  end
scale 1:25000
setcolor 6,8,9 white
setcolor 10 green
```

```
(cont.)
defpat diag
000001
00001
0001
001
01
1
  color 1 red
  end
setpat 4 diag
text 608000 3476004 SPEARFISH SOILS MAP
  color red
  width 2
  hcolor black
  hwidth 1
  background white
  border red
  size 500
  ref lower left
  opaque y
  end
line 606969 3423092 616969 3423092
  color yellow
  width 2
  opaque yes
  end
point 40% 60%
  color purple
  icon diamond
  size 2
  masked n
   end
end
```

INTERACTIVE MODE

If the user enters *p.map.new* on the command-line without arguments, a prompting session occurs. Some, but not all, of the non-interactive requests are available in this mode.

SEE ALSO

g.mapsets, g.region, p.chart, p.colors, p.icons, p.labels, p.ppm, p.select, r.mask

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratories
Joo Joo Chia, U.S. Army Construction Engineering Research Laboratories

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

p.ppm – Reads portable pixmap (ppm) files created by PPM utilities.
(GRASS Paint/Print Program)

SYNOPSIS

p.ppm
p.ppm help
p.ppm [**-f**] [**input**=*name*]

DESCRIPTION

This program, *p.ppm*, reads a user-specified portable pixmap (ppm) file and outputs it to the currently selected printer (see *p.select*). The *input* ppm file should be one that has been created using the PPM utilities developed by Jeff Poskanzer. These utilities can import various image formats (including SUN raster, X Windows pixmaps, and others) into the PPM formats *ppm* (color pixmaps), *pgm* (grey scale maps), and *pbm* (black and white bit-maps).

If the image doesn't fit the output device, it won't get printed. If you want the image printed (but clipped), use the **p.ppm -f** option, or scale the input using *ppmscale*, or rotate the image using *ppmrotate* (if it will fit that way -- otherwise you might have to scale it as well). If the image doesn't fit, *p.ppm* will tell you what scaling value to enter to *ppmscale* that will make it fit.

EXAMPLES

If the user is running GRASS on a SUN machine, the following command could be used to send a monitor screen image to the printer:

screendump | rasttoppm | p.ppm

If you are running suntools, the user might type:

sleep 10; screendump | rasttoppm | ppmrotate 90 | p.ppm

The UNIX *sleep* command allows you time to arrange the frames before the screen dump starts. The *ppmrotate* is usually needed because the SUN screens are wider than they are long (and wider than 1024 pixels - which is the width of most of our printers).

If you are running X, the user might type:

xwd | xwdtoppm | ppmrotate 90 | p.ppm

NOTES

This program only supports the ppm binary format (P6).

Maximum color level is 255. If the ppm file has more color levels, use *ppmcscale* to reduce the number of colors.

No scaling is done. Use *ppmscale* to change image size.

No rotation is done. Use *ppmrotate* to rotate the image.

SEE ALSO

See also:

The PPM utilities *ppmrotate (rotates ppm images), ppmscale (scales ppm images for printing), rasttoppm (converts a SUN raster file* and *xwdtoppm (converts an X Windows dump*

The SUN program *screendump* (dumps the image on the color graphics monitor into a file in SUN raster file format).

The X program *xwd* (dumps the image in an X window into a file in X window dump [xwd] format).

The GRASS programs *d.save, d.savescreen, p.map, p.select,* and *parser*

AUTHOR

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

This program uses the PPM utilities developed by Jeff Poskanzer.

NAME

p.select – Selects a device (printer) for GRASS hardcopy output.
(GRASS Hardcopy Output Program)

SYNOPSIS

p.select
p.select help
p.select [**–lpq**] [**painter**=*name*]

DESCRIPTION

p.select allows the user to select the device for GRASS hardcopy output. The user must select a device *before* running the other GRASS hardcopy output functions (e.g., *p.map*, *p.labels*, *p.chart*).

COMMAND LINE OPTIONS

Flags:

-l List all available painters.

-p Print name of currently selected painter.

-q Quietly select painter.

Parameter:

painter=*name* Name of painter to select.
Options: (this will be a list of available hardcopy output devices, and also *preview*, the user's graphics monitor)

INTERACTIVE MODE

If the user runs *p.select* without specifying program arguments on the command line, the program will prompt the user for the name of a hardcopy output device to select.

NOTE

The options available with *p.select* vary from system to system and depend upon which drivers have been compiled.

SEE ALSO

p.chart, p.labels, p.map

AUTHOR

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

NAME

GRASS parser interface – A set of GRASS routines that standardizes GRASS commands.

DESCRIPTION

A set of routines in GRASS provides a mechanism for command-line parsing. The routines standardize commands that expect command-line arguments. When a user is familiar with the general form of command-line input as defined by the parser, it simplifies the necessity of remembering (or at least guessing) required command-line arguments for any GRASS command.

The parser behaves in one of three ways:

> If no command-line arguments are entered by the user, the parser searches for an interactive version of the command. If it is found, control is passed over to this interactive version. If it is not found, the parser prompts the user for programmer-defined options and flags. This prompting conforms to the same standard for every GRASS command that uses the parser routines.
>
> If command-line arguments are entered but do not include all the options and flags that the programmer has defined as required arguments, three things happen: 1) the parser passes an error message to the user indicating which required options and/or flags were missing from the command-line, 2) then the parser displays a complete usage message for that command, and finally 3) the parser cancels execution of the command.
>
> If all necessary options and flags are entered on the command line by the user, the parser executes the command with the given options and flags.

This page was derived from the GRASS 4.1 Programmer's Manual, Command-Line Parsing, *which offers detailed information about the GRASS parsing routines.*

This is not a command in itself, but an option of the command i.ortho.photo.

NAME

photo.2image - An imagery function that enables you to mark known reference points (fiducial marks, reseau marks, etc.) on imagery group files, and compute the image-to-photo coordinate transformation parameters. The transformation parameters are required for the GRASS Image Processing Program *i.ortho.photo*.
(GRASS Image Processing Program)

SYNOPSIS

Selected from i.ortho.photo

DESCRIPTION

photo.2image is an imagery function that enables you to mark fiducial or reseau points on an image to be ortho-rectified and then computes the image-to-photo coordinate transformation parameters. The coordinates of the fiducials or reseau marks can be choosen for a camera reference file.

During the process of marking reference points with known photo coordinates, you may compute the RMS (root mean square) error for each reference point entered. *photo.2image* does this by calculating the transformation equation (the same one that is calculated in the GRASS program *i.points*), and then plugging these results into an equation for RMS error.

photo.2image offers a zoom option to locate precisely the fiducial or reseau point to be marked on an image.

To run *photo.2image*, a graphics monitor is required.

The procedure for marking fiducial or reseau points, entering known photo-coordinates, and analyzing the RMS error is described below.

The terminal screen displays this message:

use mouse now...

The graphics monitor displays the following screen:

imagery	

A pop-down menu like that shown below will be superimposed on the left half of the screen:

Double click on cell file to be plotted **Double click here to cancel**

Mapset *imagery*	
gs13.1	**gs21.1**
gs14.1	**gs22.2**

Any single file in the imagery group may be used on which to mark points, and you may mark points on more than one file in the imagery group to accumulate the suggested minimum number of 4-8 fiducial or ressue points. The imagery file selected is displayed in the upper left quadrant of the screen.

CAMERA FILE

The camera reference file may be viewed by placing the mouse cross hairs on the words CAMERA. The camera reference file will be displayed in the lower left quadrant of the display:

```
---------------------------------------------------------------|
|Camera Reference File                                         |
---------------------------------------------------------------|
| CAMERA NAME :       camera name                              |
| CAMERA ID   :       L1234                                    |
| CAMERA CFL  :       153.021 mm.                              |
| CAMERA XP   :       0.050 mm.                               |
| CAMERA XP   :       0.056 mm.                               |
| number of fid.      4                                        |
```

ID	(X PHOTO) mm.	(Y PHOTO) mm.
1	-105.023	110.123
2	107.987	109.834
3	110.965	-104.329
4	-103.932	-110.352

The following menu is displayed at the bottom of the graphics display:

Quit	Zoom	Plot Cell	Camera	Analyze	Input->KEYBOARD	CAMERA FILE

ZOOM

To magnify the displayed file, you must place the mouse cross hairs on the word ZOOM. The following menu will then be displayed at the bottom of the screen:

Cancel	Box	Point	Select type of ZOOM

You have the option of identifying the zoom window either by using the mouse to make a box, or by using the mouse to mark the center of the desired window, and adjusting the magnification factor. The terminal screen will display a mouse button menu to guide you in identifying the window. The section of the image within the zoom window will be displayed in the upper right hand quadrant.

MARKING REFERENCE POINTS

To mark the known reference points (fiducial marks, reseau marks, etc.) on the image, you must place the mouse cross hairs on the corresponding location on the image to be marked and press the left hand button on the mouse. A diamond shaped symbol will be marked on the image.

If you wish to use the camera reference file only as a comparative reference, then the keyboard can be chosen as the means to input photo coordinates corresponding to the marked reference points on the image. This is done by placing the mouse cross hairs on the word KEYBOARD and pressing the left button on the mouse.

The following menu is displayed on the graphics terminal:

```
Point 1 marked at IMAGE COORDINATES
east:  1023.77
north:  506.56

Enter PHOTO COORDINATES as x and y:
```

You then enter the known (x,y) photo coordinates, relative to the perspective center, for the reference point marked on the image. If you do not wish to enter a coordinate, simply hit RETURN to continue; the marked reference point will disappear.

If you select the CAMERA FILE option, then reference points marked on the image will be associated with selected photo coordinates from the displayed camera reference file. In this option, when you mark a point on the image, the following menu is displayed on the graphics terminal:

```
------------------------------------------------------------
CANCEL   Double click on the fiducial mark to be referenced
------------------------------------------------------------
```

If you would like to select the photo coordinates of a displayed reference mark, this can be accomplished by placing the mouse cross hairs on the reference point to be selected and pressing the left button on the mouse twice. After a reference point is selected from the display, you are prompted with "Look ok? (Y/N)". If you respond with no, the reference point is ignored. If you respond with yes, the following is displayed on the terminal:

```
Point 1 marked at IMAGE COORDINATES
east:  1023.77
north: 1065.41


Point 1 referenced at PHOTO COORDINATES
X: -105.023
Y:  110.122

use mouse now...
```

The photo coordinates and the corresponding image coordinates are automatically saved in the photo_points file associated with the imagery group.

ANALYZE

After a number of points have been marked (4 to 8), you can check the RMS error of the points marked on the image. This is done by placing the mouse cross hairs on the word ANALYZE at the bottom of the monitor. An error report resembling the one shown below is superimposed on the monitor:

		error		**image**		**photo**	
#	**row**	**col**	**photo**	**east**	**north**	**x**	**y**
1	**0.0**	**-0.9**	**1.0**	**1048.5**	**-144.8**	**-105.023**	**110.122**
2	**0.4**	**1.0**	**1.3**	**2153.1**	**-567.2**	**107.987**	**109.834**
3	**-1.2**	**-0.5**	**.6**	**1452.8**	**-476.5**	**110.965**	**-104.329**
4	**1.1**	**0.5**	**1.3**	**1034.0**	**-109.2**	**-103.932**	**-110.352**
	overall	rms	error:	4.46			

The following menu then appears at the bottom of the monitor:

DONE	**PRINT**	**FILE**	**Double click on point to be included/excluded**

The RMS error for the image is given under the column titled "error" and subtitled "row" and "col". In the above report, point number 1 is 0.0 rows and -0.9 columns from the predicted location calculated from the transformation equation. The RMS error for the photo coordinates is listed under the heading "photo". This is the RMS error for the x and y coordinates of the photo coordinates but it is presented in the table using one general value. The overall RMS error is displayed at the bottom of the screen in milimeters. Points that create high RMS error are displayed in red on the monitor (represented here in italics).

The location of the point marked on the imagery group file is given under the heading "image" and the subheadings "east" and "north". The location of the point in the photo coordinates is given under the heading "photo" and the subheadings "x" and "y". If you would like to exclude or include a point, this can be accomplished by placing the mouse cross hairs on the point number to be included (if the point is absent) or excluded (if the point is displayed) and pressing the left button on the mouse twice. When a point is excluded, it is not afterwards included in the calculation of the RMS error, or included in the final transformation matrix. However, it can be retrieved within *photo.2image* at any time by double clicking with the mouse as described above.

QUIT

To end the *photo.2image* program place the mouse cross hairs on the word QUIT; the marked reference points (including coordinates) will be saved.

NOTES

A good rule of thumb is to mark at least 4 to 8 points which are evenly distributed over the perimeter of the imagery group file in order to obtain an accurate transformation equation for the rectification process. The RMS error may increase with more points added, but the transformation equation will be more accurate.

An RMS error of less than or equal to approximately one resolution unit (pixel) for the image being rectified is generally considered acceptable.

SEE ALSO

i.ortho.photo[2], photo.camera[2], photo.2target[2], photo.init[2], photo.rectify[2]

AUTHOR

Mike Baba DBA Systems, Inc.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

This is not a command in itself, but an option to the command i.ortho.photo.

NAME

photo.2target - An imagery function that enables you to mark control points on an image to be ortho-rectified, and then input the coordinates of each control point for determination of ortho-rectification parameters. The ortho-rectification parameters are required as input for the GRASS program *photo.rectify*. *(GRASS Imagery Processing Tool)*

SYNOPSIS

Selected from i.ortho.photo

DESCRIPTION

photo.2target is an imagery function that enables you to mark control points on an image to be ortho-rectified and then input the coordinates of each point for calculation of rectification parameters.

Rectification is the mapping of an image from one coordinate system to another. The geometry of an image extracted into a GRASS LOCATION having an x,y coordinate system is not planimetric. To create a planimetric image, that is, to convert the x,y coordinate system into a standard coordinate system (for example, the UTM coordinate system or the State Plane coordinate system), points from a map having the standard coordinates must be associated with the same points on the image to be rectified.

The ortho-rectification parameters are computed in two phases. The first phase computes a transformation matrix between image (row,col) coordinates and photo (x,y) coordinates relative to the perspective center. The transformation matrix is computed explicitly in the option *photo.2image*.

The second phase, *photo.2target*, enables you to mark control points on the image and then input the standard coordinates (Easting,Northing, and elevation) to determine the parameters for a three dimensional projective transformation.

photo.init may be run before running *photo.2target* to modify the initial camera exposure station parameters, and/or modify the standard deviation of these parameters. *photo.init* is not required but generally is helpful.

During the process of marking control points and entering standard coordinates, you may compute the RMS (root mean square) error for each control point entered. photo.2target does this by calculating the transformation equation, and then plugging these results into an equation for RMS error.

photo.2target offers a zoom option to locate precisely the point to be marked on an image. This program also offers you the option of acquiring standard coordinates for a marked point from a map layer in the target database, or from a digitizer.

To run *photo.2target*, a graphics monitor is required.

The procedure for marking points, entering coordinates, and calculating RMS error is described below.

The terminal screen displays the following message:

use mouse now...

The graphics monitor displays the following screen:

imagery *filename* **(mag)**	**target** *filename* **(mag)**
QUIT ZOOM PLOTCELL ANALYZE	

A pop-down menu like that shown below will be superimposed on the left half of the screen:

Double click on cell file to be plotted **Double click here to cancel**

Mapset *demo*	
gs13.1	**gs14.1**
gs21.1	**gs22.2**

Any single file in the imagery group may be used to mark points, and you can mark points on more than one file in the imagery group to accumulate the 12 points suggested minimum. Any file in the imagery group can be rectified (using *photo.rectify*) based on the rectification parameters computed from these control points.

The imagery file you select is displayed in the upper left quadrant of the screen.

ZOOM

To magnify the displayed file, you must place the mouse cross hairs on the word ZOOM. The following menu will then be displayed at the bottom of the screen:

Cancel	**Box**	**Point**	**Select type of ZOOM**

You may identifying the zoom window either by using the mouse to make a box, or by using the mouse to mark the center of the window and entering a magnification factor. The terminal screen will display a mouse button menu to guide you in identifying the window.

MARKING POINTS

To mark the points on the image that correspond to the points on a standard coordinate system map, you must place the mouse cross hairs on the corresponding location on the image to be marked and press the left hand button on the mouse. A diamond shaped symbol will be marked on the image. The terminal will display the following menu:

Point 1 marked at IMAGE COORDINATES **IMAGE X: 1023.77** **IMAGE Y: -164.41**	
Enter CONTROL COORDINATES as east,north,elevation:	

You then enter the easting, northing, and elevation for the point marked on the image. If you wish not to enter a coordinate, simply hit RETURN to return control to the mouse; the marked point then disappears.

PLOT CELL

In addition to acquiring control points from a standard map, you have the option of acquiring the points from a cell-map in the target database. The database map is displayed by placing the mouse cross hairs on the words PLOT CELL. The following line is then displayed at the bottom of the monitor:

Cancel	**Indicate which side should be plotted**

Which side of the monitor is to be plotted is indicated by placing the mouse cross hairs on the half of the monitor screen that you would like to use, and pressing the left mouse button. The following pop-down menu will be superimposed on the half of the screen that was chosen:

Double click on cell file to be plotted **Double click here to cancel**

Mapset *demo*	
tm.rectified	
tm.classified	
Mapset PERMANENT	
elevation	**geology**
slope	**soils**
aspect	
roads	
streams	
airfields	

After the map is displayed the following message appears at the bottom of the monitor:

input method -->	**keyboard**	**screen**

If you wish to use the plotted map only as a comparative reference, the keyboard can be chosen as the means to input coordinates corresponding to the marked control points. This is done by placing the mouse cross hairs on the word KEYBOARD and pressing the left button on the mouse.

If you select the SCREEN option, points marked on the image will automatically be associated with the coordinates from the corresponding points on the target database map, and a corresponding elevation from the cell-file selected for elevation data. In this option, when you mark a point on the image, the following menu is displayed on the terminal:

Point 5 marked at IMAGE COORDINATES **IMAGE X: 1023.77** **IMAGE Y: -164.41** **Control Point location** **East: 679132.57** **North: 4351080.67** **Elevation: 1010.00**	
use mouse now...	

The coordinates for the target database map are automatically saved as the coordinates corresponding to the marked control point on the image.

ANALYZE

After a number of points have been marked (4 to 7), you can check the RMS error of the points marked on the image. This is done by placing the mouse cross hairs on the word ANALYZE at the bottom of the monitor. An error report resembling that shown below is superimposed on the monitor:

				error				
#	**east**	**north**	**target**	**east**	**north**	**east**	**north**	**elev.**
1	**0.0**	**-0.9**	**1.0**	**48.5**	**4.8**	**79132.5**	**351080.6**	**010.0**
2	**0.4**	**1.0**	**1.3**	**53.1**	**7.2**	**84314.7**	**399001.4**	**239.3**
3	**-1.2**	**-0.5**	**.6**	**52.8**	**6.5**	**67841.4**	**457682.8**	**209.5**
4	**1.1**	**0.5**	**1.3**	**34.0**	**9.2**	**77573.8**	**352626.4**	**432.5**
5	*-2.7*	*14.0*	*14.2*	*48.6*	*-144.9*	*79132.6*	*351080.7*	*985.0*
		overall	**rms**	**error:**	**4.46**			

The following menu then appears at the bottom of the monitor:

DONE	**PRINT FILE**	**Double click on point to be included/excluded**

The RMS error for the image is given under the column titled "error" and subtitled "east" and "north". In the above report, point number 1 is 0.0 and -0.9 meters (east and north) from the predicted location calculated from the transformation equation. The RMS error for the target map is listed under the heading "target". This is the RMS error for the east and north coordinates of the target map but it is presented in the table using one general value. The overall RMS error is displayed at the bottom of the screen in meters. Points that create high RMS error are displayed in red on the monitor (represented here in italics).

The image coordinates of the point marked on the imagery group file is given under the heading "image" and the subheadings "east" and "north". The location of the control point in the target database is given under the heading "control" and the subheadings "east","north", and "elev". If you would like to exclude or include a control point, this can be accomplished by placing the mouse cross hairs on the control point number to be included (if the point is absent) or excluded (if the point is displayed) and pressing the left button on the mouse twice. When a point is excluded, it is not afterwards included in the calculation of the RMS error, or included in the final rectification parameters. However, it can be retrieved within *photo.2target* at any time by double clicking with the mouse as described above.

QUIT

To end the *photo.2target* program place the mouse cross hairs on the word QUIT; the marked control points (including coordinates) will be saved.

NOTES

During the course of marking control points and computing the ortho-rectification parameters, a matrix inversion error may occur. This is caused by trying to invert a non-singular normal equation matrix. When this situation arises, the status of all previously selected control points are modified, the control points are excluded. Running *photo.init* for the selected imagery group with accurate camera exposure station parameters should remedy the situation. The excluded control points may again be included as described in the section ANALYZE.

A good rule of thumb is to mark at least 12 to 15 points which are evenly distributed over the entire imagery group file in order to obtain an accurate transformation parameters for the rectification process. The RMS error may increase with more points added, but the transformation parameters will be more accurate over the entire image.

An RMS error of less than or equal to approximately one resolution unit (pixel) for the image being rectified is generally considered acceptable.

SEE ALSO

i.ortho.photo[2], photo.camera[2], photo.2image[2], photo.init[2], photo.rectify[2]

AUTHOR

Mike Baba DBA Systems, Inc.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

This is not a command in itself, but an option to the command i.ortho.photo.

NAME

photo.camera - An imagery function that creates or modifies entries in a camera reference file.
(GRASS Image Processing Program)

SYNOPSIS

photo.camera option must be selected from GRASS Image Processing Program *i.ortho.photo*.

DESCRIPTION

photo.camera creates or modifies entries in a camera reference file. For ortho-photo rectification, a camera reference file is required for computation of scanned image to photo-coordinate transformation parameters.

The first prompt in the program will ask you for the name of the camera reference file to be created or modified. You may create a new camera reference file by entering a new name, or modify an existing camera reference file by entering the name of an existing camera file.

After entering the camera file name, following menu is displayed:

Please provide the following information

CAMERA NAME: *camera name*______
CAMERA IDENTIFICATION: *identification*___
CALIBRATED FOCAL LENGTH mm.:________________
POINT OF SYMMETRY (X) mm.:________________
POINT OF SYMMETRY (Y) mm.:________________
MAXIMUM NUMBER OF FIDUCIALS:________________

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The camera name and identification describe the camera reference file. The calibrated focal length and the point of symmetry are used in computing the photo-to-target transformation parameters. These values should be entered from the camera calibration report (usually available from the photograph supplier). The maximum number of fiducials will determine the number of fiducial or reasuae coordinate pairs to be entered below.

You are then ask to enter the X and Y photo coordinates of each fiducial as follows:

Please provide the following information

Fid#	**FID ID**	**X**	**Y**
1__	**_____**	**0.0___**	**0.0___**
2__	**_____**	**0.0___**	**0.0___**
3__	**_____**	**0.0___**	**0.0___**
4__	**_____**	**0.0___**	**0.0___**
5__	**_____**	**0.0___**	**0.0___**
6__	**_____**	**0.0___**	**0.0___**
7__	**_____**	**0.0___**	**0.0___**
8__	**_____**	**0.0___**	**0.0___**
9__	**_____**	**0.0___**	**0.0___**
10_	**_____**	**0.0___**	**0.0___**

next: end__

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The input display is repeated until the number of MAXIMUM FIDUCIALS is reached. On this screen you should enter the fiducial or ressuae photo-coordinates as given in the camera calibration report. The X, and Y coordinates are in milli-meters from the principle point.

SEE ALSO

i.ortho.photo[2], photo.2image[2], photo.2target[2], photo.init[2],

AUTHOR

Mike Baba DBA Systems, Inc.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

This is not a command itself, but an option of the command i.ortho.photo.

NAME

photo.init - An imagery function that creates or modifies the optional entries in a camera initial exposure station file for an imagery group.
(GRASS Image Processing Program)

SYNOPSIS

Selected from i.ortho.photo

DESCRIPTION

photo.init creates or modifies entries in a camera initial exposure station file for imagery group referenced by a sub-block. These entries include: the (XC,YC,ZC) UTM approximate coordinates of the camera exposure station; initial roll, pitch, and yaw angles (in degrees) of the cameras attitude; and the *a priori* standard deviations for these parameters. During the imagery program, hoto.rectify, the initial camera exposure station file is used for computation of the ortho-rectification parameters. If no initial camera exposure station file exist, the default values are computed from the control points file created in *photo.2target*.

The following menu is displayed:

Please provide the following information

INITIAL XC: Meters __________
INITIAL YC: Meters __________
INITIAL ZC: Meters __________
INITIAL omega (roll) degrees: __________
INITIAL phi (pitch) degrees: __________
INITIAL kappa (yaw) degrees: __________

Standard Deviation XC: Meters __________
Standard Deviation YC: Meters __________
Standard Deviation ZC: Meters __________
Std. Dev. omega (roll) degrees: __________
Std. Dev. phi (pitch) degrees: __________
Std. Dev. kappa (yaw) degrees: __________

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

The INITIAL values for (XC,YC,ZC) are expressed in UTM coordinates, and represent an approximation for the location of the camera at the time of exposure.

The INITIAL values for (omega,phi,kappa) are expressed in degrees, and represent an approximation for the cameras attitude at the time of exposure.

The standard deviations for (XC,YC,ZC) are expressed in meters, and are used as *a priori* values for the standard deviations used in computation of the ortho rectification parameters.

The standard deviations for (omega,phi,kappa) are expressed in degrees, and are used as *a priori* values for the standard deviations used in computation of the ortho rectification parameters.

SEE ALSO

i.ortho.photo[2], photo.camera[2], *photo.2image[2]*, *photo.2target[2]*, *photo.rectify[2]*

AUTHOR

Mike Baba DBA Systems, Inc.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

This is not a command in itself, but an option of the command i.ortho.photo.

NAME

photo.rectify - An imagery function that rectifies an image by computing a coordinate transformation for each cell in the image using parameters computed by the GRASS programs photo.2image and photo.2target.
(GRASS Image Processing Program)

SYNOPSIS

Selected from i.ortho.photo

DESCRIPTION

photo.rectify rectifies an image by using the image to photo coordinate transformation matrix created by photo.2image and the rectification parameters created by photo.2target. Rectification is the process by which the geometry of an image is made planimetric. This is accomplished by mapping an image from one coordinate system to another. In photo.rectify the parameters computed by photo.2image and photo.2target are used in equations to convert x,y image coordinates to standard map coordinates for each pixel in the image. The result is an image with a standard map coordinate system, compensated for relief distortions and photographic tilt. Upon completion of the program the rectified image is deposited in a previously targeted GRASS LOCATION.

You are asked to select the file(s) within the imagery group to be rectified:

Please select the file(s) to rectify by naming an output file

gs13.1 in demo *gs13.orect*...
gs14.1 in demo ....
gs21.1 in demo ....
gs22.1 in demo ....

(enter list by any name to get a list of existing cell files)

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO CANCEL)

More than one file may be rectified at a time. Each file should have a unique output file name.

The next prompt asks you to select one of two windows:

Please select one of the following options
1. Use the current window in the target location
2. Determine the smallest window which covers the image
>

photo.rectify will only rectify that portion of the image that occurs within the chosen window. Only that portion will be relocated in the target database. It is therefore important to check the current window in the target LOCATION if choice number one is selected.

photo.rectify will run in the background and notify you by mail when it is finished. The process may take an hour or more depending on the size of the image, the number files, and the size and resolution of the selected window.

The rectified image will be located in the target LOCATION when the program is completed. The original unrectified files are not modified or removed.

SEE ALSO

i.ortho.photo[2], photo.camera[2], photo.2image[2], photo.2target[2], photo.init[2]

AUTHOR

Mike Baba DBA Systems, Inc.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

ps.icon – Creates and modifies icons for use with *ps.map*.
(GRASS Hardcopy PostScript Output Program)

SYNOPSIS

ps.icon

DESCRIPTION

This program allows the user to create and maintain icons which are used by *ps.map* to depict sites.

FILES

Icon files created by the user are stored under $LOCATION/ps_icon.

SEE ALSO

ps.map

AUTHOR

Paul Carlson, USDA, SCS, NHQ-CGIS

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

ps.map – Hardcopy PostScript map output utility.
(GRASS Hardcopy PostScript Output Program)

SYNOPSIS

ps.map
ps.map help
ps.map [**r**] [**input**=*name*] [**scale**=*mapscale*] [**copies**=*n*] **output**=*name*

DESCRIPTION

ps.map produces an output file containing a PostScript program to produce hardcopy map products on your system's PostScript output device. Output can include a raster map, any number of vector overlays, site data, text labels, and other spatial data.

This program has 2 distincts modes of operation. The command-line mode requires the user to prepare a file of mapping instructions describing the various spatial and textual information to be printed prior to running *ps.map*. The interactive mode (i.e., no command-line arguments) will prompt the user for items to be mapped and does not require the user to prepare a file of instructions.

The command line flag is:

r Rotate plot 90 degrees.

The command-line parameters are:

input=*name* File containing mapping instructions. (or enter **input=-** to enter from keyboard). These instructions are described in detail below.

scale=*mapscale* Scale of the output map, e.g. 1:25000
Default: 1 panel

This parameter is provided as a convenience. It is identical to the *scale* mapping instruction described below.

output=*name* Name of output the file to contain the PostScript program.

Note: the user must select a PostScript output device using *ps.select* before running *ps.map*.

MAPPING INSTRUCTIONS

The mapping instructions allow the user to specify various spatial data to be plotted. These instructions are normally prepared in a regular text file using a system editor. Some instructions are single line instructions while others are multiple line. Multiple line instructions consist of the main instruction followed by a subsection of one or more additional instructions.

Instructions that may be included in the subsection under several different main instructions are:

where *x y* The top left corner of the bounding box of the item to be plotted is located *x* inches from the left edge of the paper and *y* inches from the top edge of the paper. If *x* is less than or equal to zero, the default horizontal location is used. If *y* is less than or equal to zero, the default vertical location is used.

font *font name* The name of the PostScript font. The default is *Helvetica*.

fontsize *font size* The size of the PostScript font in 1/72 inch. The default is 10.

colortable

Prints the color table for the raster map layer anywhere on the page.

USAGE: **colortable** [y| n]
  where x y
  width table width
  cols table columns
  font font name
  fontsize font size
  color text color
  end

The color table will display the colors for each raster map layer category value and the category label. To get a color table, you must have previously requested a raster map layer. The default location for the colortable is immediately below any other map legend information, starting at the left margin. The default text color is black. Omitting the **colortable** instruction would result in no color table. **Note**: Be careful about asking for color tables for raster map layers which have many categories, such as elevation. This could result in the printing of an extremely long color table!!!!!

This example would print a color table immediately below any other map legend information, starting at the left margin.
EXAMPLE: **colortable** y

comments

Prints comments anywhere on the page.

USAGE: **comments** commentfile
  where x y
  font font name
  fontsize font size
  color text color
  end

The default location is immediately below the last item item printed, starting at the left margin. The default text color is black.

This example prints in blue whatever is in the file *veg.comments* starting at 1.5 inches from the left edge of the page and 7.25 inches from the top of the page, using a 15/72 inch Helvetica Bold font.
EXAMPLE: **raster** vegetation
 comments veg.comments
  where 1.5 7.25
  font Helvetica Bold
  fontsize 15
  color blue
  end

Presumably, the file *veg.comments* contain comments pertaining to the raster map layer *vegetation*, such as "This map was created by classifying a LANDSAT TM image".

copies

Specifies the number of copies to be printed.

USAGE: **copies** n

Each page will be printed n times.

greyrast

Selects a raster map layer for output in shades of grey.

USAGE: **greyrast** mapname| *list*

For each *ps.map* run, only one raster map layer can be requested (using either the *greyrast* or the *raster* instruction).

grid

Overlays a coordinate grid onto the output map.

USAGE: **grid** spacing
color color
numbers # [color]
font font name
fontsize font size
end

The **spacing** of the grid is given (in the geographic coordinate system units) on the main instruction line. The subsection instructions allow the user to specify the **color** of the grid lines, whether coordinate **numbers** should appear on the grid lines, and if they should appear every grid line (1), every other grid line (2), etc., and what color the numbers should be. The defaults are black grid lines, unnumbered.

This example would overlay a green grid with a spacing of 10000 meters (for a metered database, like UTM) onto the output map. Alternate grid lines would be numbered with red numbers.

EXAMPLE: **grid** 10000
color green
numbers 2 red
end

header

Prints the map header above the map.

USAGE: **header**
file header file
font font name
fontsize font size
color text color
end

If the *header* instruction or the *filesub-instruction is absent, the header will consist of the map title and* *location, each centered on the page above the map. The default text color is black.*

This example prints (in red) whatever is in the file soils.hdr above the map, using a 20/72 inch Courier font.

EXAMPLE: **header**
file soils.hdr
font Courier
fontsize 20
color red
end

labels

Selects a labels file for output (see manual entry for *p.labels).*

USAGE: **labels** labelfile| *list*
font font name
end

This example would paint labels from the labels file called *town.names*. Presumably, these labels would indicate the names of towns on the map.

EXAMPLE: **labels** town.names

line

Draws lines on the output map.

USAGE: **line** east north east north
line x% y% x% y%
color color
width #
masked [y| n]
end

The beginning and ending points of the line are entered on the main instruction. These points can be defined either by map coordinates or by using percentages of the geographic region. The user may also specify line **color, width** in pixels, and if the line is to be **masked** by the current mask. (See manual entry for *r.mask* for more information on the mask.)

This example would draw a yellow line from the point x=10% y=80% to the point x=30% y=70%. This line would be 2 pixels wide and would appear even if there is a mask.

EXAMPLE: **line** 10% 80% 30% 70%
color yellow
width 2
masked n
end

Of course, multiple lines may be drawn with multiple *line* instructions.

mapinfo

Prints the portion of the map legend containing the scale, grid and region information, on or below the map.

USAGE: **mapinfo**
where x y
font font name
fontsize font size
color text color
end

The default location is immediately below the map, starting at the left edge of the map. The default text color is black.

This example prints (in brown) the scale, grid and region information immediately below the map and starting 1.5 inches from the left edge of the page, using a 12/72 inch Courier font.

EXAMPLE: **mapinfo**
where 1.5 0
font Courier
fontsize 12
color brown
end

maploc

Positions the map on the page.

USAGE: **maploc** x y [width height]

The upper left corner of the map will be positioned *x* inches from the left edge of the page and *y* inches from the top of the page. If *widthand height* (in inches) are present, the map will be rescaled, if necessary, to fit.

This example positions the upper left corner of the map 2.0 inches from the left edge and 3.5 inches from the top edge of the map.

EXAMPLE: **maploc** 2.0 3.5

outline

Outlines the areas of a raster map layer with a specified color.

USAGE: **outline**
color color
end

Distinct areas of the raster map will be separated from each other visually by drawing a border (or outline) in the specified **color** (default: black). Note: it is important the user enter the instruction **end** even if a color is not chosen. (It is hoped that in the future the outline of a different raster map layer other than the one currently being painted may be placed on the map.)

This example would outline the category areas of the *soils* raster map layer in grey.

EXAMPLE: **raster** soils
outline
color grey
end

point

Places additional points or icons on the output map.

USAGE: **point** east north
point x% y%
color color
icon iconfile| *list*
size #
masked [y| n]
end

The point location is entered in the main instruction line by giving either the map coordinates or by using percentages of the geographic region. The user may also specify the point **color,** the **icon** file to be used to represent the point location (see the the manual entry for *ps.icons),* the **size** of the icon as a multiple of the size of the pattern in the icon file (default is 1.0, aprroximately equivalent to a 10 point character), and whether the point is to be **masked** by the current mask. (See manual entry for *r.mask* for more information on the mask.)

This example would place a purple diamond (from icon file *diamond*) at the point (E456000 N7890000). This diamond would be the approximately the size of a 15 point character and would not be masked by the current mask.

EXAMPLE: **point** 456000 7890000
color purple
icon diamond
size 1.5
masked n
end

Of course, multiple points may be drawn with multiple *point* instructions.

psfile

Copies a file containing PostScript commands into the output file.

Note: *ps.map* will not search for this file. The user must be in the correct directory or specify the full path on the **psfile** instruction. (Note to /bin/csh users: ˜ won't work with this instruction).

USAGE: **psfile** filename

This example copies the file "logo.ps" into the output file.

EXAMPLE: **psfile** logo.ps

raster

Selects a raster map layer for output.

USAGE: **raster** mapname| *list*

For each *ps.map* run, only one raster map layer can be requested. If no raster map layer is requested, a completely white map will be produced. It can be useful to select no raster map layer in order to provide a white background for vector images.

This example would paint a map of the raster map layer *soils*.
EXAMPLE: **raster** soils

read

Provides *ps.map* with a previously prepared input stream.

USAGE: **read** previously prepared UNIX file

Mapping instructions can be placed into a file and read into *ps.map.*

Note: *ps.map* will not search for this file. The user must be in the correct directory or specify the full path on the **read** instruction. (Note to /bin/csh users: ˜ won't work with this instruction).

This example reads the UNIX file *pmap.roads* into *ps.map*. This file may contain all the *ps.map* instructions for placing the vector map layer *roads* onto the output map.

EXAMPLE: **read** pmap.roads

The user may have created this file because this vector map layer is particularly useful for many *ps.map* outputs. By using the **read** option, the user need not enter all the input for the **vector** instruction, but simply **read** the previously prepared file with the correct instructions.

region

Places the outline of a smaller geographic region on the output.

USAGE: **region** regionfile| *list*
  color color
  width #
  end

Geographic region settings are created and saved using *g.region*. The *ps.map region* option can be used to show an outline of a smaller region which was printed on a separate run of *ps.map* on other user-created maps.

The user can specify the **color** and the **width** (in pixel units) of the outline. The default is a black border of one pixel width.

This example would place a white outline, 2 pixels wide, of the geographic region called *fire.zones* onto the output map. This geographic region would have been created and saved using *g.region*.

EXAMPLE: **region** fire.zones
  color white
  width 2
  end

scale

Selects a scale for the output map.

USAGE: **scale** *scale*

The scale can be selected either as:

a relative ratio, e.g. 1:25000;

an absolute width of the printed map, e.g. 10 inches;

the number of printed paper panels, e.g. 3 panels *(at the present time, only 1 panel is supported);*

the number of miles per inch, e.g. 1 inch equals 4 miles.

This example would set the scale of the map to 1 unit = 25000 units.
EXAMPLE: **scale** 1:25000

setcolor

Overrides the color assigned to one or more categories of the raster map layer.

USAGE: **setcolor** cat(s) color

This example would set the color for categories 2,5 and 8 of the raster map layer *watersheds* to white and category 10 to green. (**NOTE**: no spaces are inserted between the category values.)

EXAMPLE: **raster** watersheds
setcolor 2,5,8 white
setcolor 10 green

Of course, *setcolor* can be requested more than once to override the default color for additional categories. More than one category can be changed for each request by listing all the category values separated by commas (but with no spaces).

sites

Selects sites data to be placed on the output map (see manual entry for *s.menu*).

USAGE: **sites** sitemap| *list*
color color
icon iconfile| *list*
size #
desc [y| n]
font font name
end

The user may specify the the **color** of the sites (see section on NAMED COLORS below); the **icon** to be used to represent the presence of a site (see the manual entry for *p.icons*); the **size** of the icon (number of times larger than the size it is in the icon file); and whether or not the **desc**ription associated with each site is also to be printed. If the **desc**ription is to be printed, the **font name may be specified. The size of the font is proportional to the icon size**. This example would paint a sites map with blue windmills (from an icon file created by the user using the *p.icons* GRASS command) placed at all windmill locations (from a sites list). These windmills would be two times larger than the size of the icon in the icon file and have descriptions from the sites list file printed beside them.

EXAMPLE: **sites** windmills
color blue
icon windmill
size 2
desc y
end

text

Places text on the map.

USAGE: **text** east north text
text x% y% text
font fontname
color color| none
width #
hcolor color| none
hwidth #
background color| none
border color| none
size #
ref reference point
xoffset #
yoffset #
opaque [y| n]
end

The user specifies where the text will be placed by providing map coordinates or percentages of the geographic region map. The text follows these coordinates on the same instruction line. More than one line of text can be specified by notating the end of a line with **\n** (e.g. USA \nCERL).

The user can then specify various text features:

font: cyrilc gothgbt gothgrt gothitt greekc greekcs greekp greeks italicc italiccs italict romanc romancs romand romans romant scriptc scripts (The default font is romans);

color (see NAMED COLORS);

width of the lines used to draw the text (to make thicker letters);

size as the vertical height of the letters in meters on the ground (text size will grow or shrink depending on the scale at which the map is painted);

the highlight color (**hcolor**) and the width of the highlight color (**hwidth**);

the text-enclosing-box **background** color; the text box **border** color;

ref. This reference point specifies the text handle - what part of the text should be placed on the location specified by the map coordinates. Reference points can refer to: [lower| upper| center] [left| right| center] of the text to be printed;

yoffset, which provides finer placement of text by shifting the text a vertical distance in pixels from the specified north. The vertical offset will shift the location to the south if positive, north if negative;

xoffset, which shifts the text a horizontal distance in pixels from the specified east The horizontal offset will shift the location east if positive, west if negative;

whether or not the text should be **opaque** to vectors. Entering **no** to the opaque option will allow the user to see any vectors which go through the text's background box. Otherwise, they will end at the box's edge.

This example would place the text *SPEARFISH LAND COVER* at the coordinates E650000 N7365000. The text would be a total of 3 pixels wide (2 pixels of red text and 1 pixel black highlight), have a white background enclosed in a red box, and be 500 meters in size. The lower right corner of the text would be centered over the coordinates provided. All vectors on the map would stop at the border of this text.

EXAMPLE: **text** 650000 7365000 SPEARFISH LAND COVER
font romand
color red
width 2
hcolor black
hwidth 1
background white
border red
size 500
ref lower left
opaque y
end

vector

Selects a vector map layer for output.

USAGE: **vector** vectormap| *list*
color color
width #
hcolor color
hwidth #
masked [y| n]
style 0-9
end

The user can specify the **color** of the vectors; the **width** of the vectors lines in pixels; the highlight color (**hcolor**) for the vector lines; the width of the highlight color (**hwidth**) in pixels; whether or not the raster map layer is to be **masked** by the current mask (see manual entry *r.mask* for more information on the mask); and the line **style**. The line style allows the vectors to be dashed in different patterns. This is done by typing a series of numbers (0's and 1's) in a desired sequence or pattern. Blanks and non-digit characters are recognized as 0's. Using 0 would allow the colors of the raster map layer (or the background color if no raster map layer was selected) to show through.

This example would paint a map of the vector map layer named *streams*. These streams would be a total of 3 pixels wide (the inner two pixels blue and the outer highlight pixel white). The map would not show streams outside of the current mask.

EXAMPLE: **vector** streams
color blue
width 2
hcolor white

hwidth 1
masked y
end

verbose

Changes the amount of talking *ps.map* will do.

USAGE: **verbose** [0| 1| 2]

A higher value implies more chatter. The default is 2. This example sets the amount of chatter to a minimum.

EXAMPLE: **verbose** 0

vlegend

Prints the portion of the map legend containing the vector information, on or below the map.

USAGE: **vlegend**
where x y
font font name
fontsize font size
end

The default location is immediately below the legend containing the scale, grid and region information, starting at the left edge of the map. If the *where* instruction is present and *y* is less than or equal to zero, the vector legend will be positioned immediately below the map, starting *x* inches from the left edge of the page.

This example prints the vector legend immediately below the map and starting 4.5 inches from the left edge of the page, using a 12/72 inch Helvetica font.
EXAMPLE: **vlegend**
where 4.5 0
font Courier
fontsize 12
end

end

Terminates input and begin painting the map.

USAGE: **end**

NAMED COLORS

The following are the colors that are accepted by *ps.map*:

aqua black blue brown cyan gray green grey indigo magenta orange purple red violet white yellow

EXAMPLE ps.map INPUT FILE

The following is an example of a *ps.map* script file. The file has been named *spear.soils*. For the purposes of illustration, the file is in two columns. This script file can be entered at the command line:

ps.map input=spear.soils output=soils.ps

```
raster soils
vector roads
  width 2
  style 0011
  color 1 black
  color 2 red
  masked n
  end
labels town.names
region subregion
  color white
  width 2
  end
grid 10000
  color green
  numbers 2 red
  end
outline
  color black
  end
colortable y
comments soil.cmt
  font Helvetica
  end
scale 1:25000    end
setcolor 6,8,9 white
setcolor 10 green
```

```
(cont.)
vlegend
 where 4.5 0
 end
text 608000.00 3476004.73 SPEARFISH SOILS MAP
  color red
  width 2
  hcolor black
  hwidth 1
  background white
  border red
  size 500
  ref lower left
  opaque y
  end
line 606969.73 3423092.91 616969.73 3423092.91
  color yellow
  width 2
  opaque yes
  end
point 40% 60%
  color purple
  icon diamond
  size 2
  masked n

end
```

INTERACTIVE MODE

If the user simply enters *ps.map* without arguments, then a simple prompting session occurs. Some, but not all of the non-interactive requests are available at this level.

SEE ALSO

ps.icons, ps.select

AUTHOR

Paul Carlson, USDA, SCS, NHQ-CGIS

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

ps.select – Selects a PostScript device for GRASS hardcopy output.
(GRASS Hardcopy PostScript Output Program)

SYNOPSIS

ps.select
ps.select help
ps.select [**–lpq**] [**painter**=*name*]

DESCRIPTION

ps.select allows the user to select the PostScript device for GRASS hardcopy output. If the user has not selected a device *before* running *ps.map*, the default device configuration (shown under **NOTES** below, will be used.

COMMAND LINE OPTIONS

Flags:

-l	List all available PostScript painters.
-p	Print name of currently selected PostScript painter.
-q	Quietly select PostScript painter.

Parameter:

painter=*name*	Name of PostScript painter to select. Options: (this will be a list of available hardcopy output PostScript devices)

INTERACTIVE MODE

If the user runs *ps.select* without specifying program arguments on the command line, the program will prompt the user for the name of a hardcopy output PostScript device to select.

NOTES

Each PostScript device must have a device description file in the $GISBASE/etc/paint/psdevices directory. The name of the device description file is used as the name of the PostScript painter. A device description file contains the following information (default values are shown):

```
level: 2
page width: 8.5
page height: 11.0
top margin: 0.5
bottom margin: 0.5
left margin: 0.25
right margin: 0.25
resolution: 75
```

where *level* is the PostScript level supported by the device (1 or 2); *page width* and *page height* are the paper dimensions (in inches); *top margin*, *bottom margin*, *left margin* and *right margin* are the unprintable edges of the page (in inches); and *resolution* is the effective resolution (in pixels per inch).

SEE ALSO

ps.map

AUTHOR

Paul Carlson, USDA, SCS, NHQ-CGIS

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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 be averaged.

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 value	cat label
0	no 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):

cat value	cat 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 Research Laboratory

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

See Appendix A of the **GRASS Tutorial: r.watershed** for further details on the combined use of *r.basins.fill* and *r.watershed*.

d.digit, d.display, r.watershed, v.digit, and *parser*

AUTHORS

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

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.

COMMAND LINE 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 (forthcoming). 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 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 (forthcoming).

AUTHOR

Kurt Buehler, Purdue University

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.

0000000000000000000000000	1111222222222222222333333
1110000000000000000000000	0001111111111112222222222
0001111111111100000000000	1110000000000011112222222
0000000010000011100000000	2211111101111100011111111
0000000010000000011111111	2222222101221111100000000
0000000010000000000000000	3222222101222222111111111
0000000010000000000000000	3333322101222222222222222
0000000010000000000000000	3333322101223322222222222
0000000010000000000000000	3333322101223333333333333
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

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

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, and *parser*

AUTHOR

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

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 map layers' 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 |
|-----------------------------------------------------|
|r     0 |    50.63 |  37.49 |    88.12 |     88.12 |
|o     1 |     1.53 |   0.68 |     2.21 |      2.21 |
|a     2 |     2.92 |   0.72 |     3.64 |      3.64 |
|d     3 |     3.97 |   2.57 |     6.54 |      6.54 |
|s     4 |     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 |
|-------------------------------|
|r     0 |    88.12 |     88.12 |
|o     1 |     2.21 |      2.21 |
|a     2 |     3.64 |      3.64 |
|d     3 |     6.54 |      6.54 |
|s     4 |     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

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

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:*

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

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 re-entered 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 0          1 1 1  results
                     OR -->   1 1 0
raster map 2   1 0 1          1 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 layerr *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
oldmap               2 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 CATS raster map

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

[1]:CATS owner

EXP EXP 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**

WINDOW WINDOW 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 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

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

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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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:
  COST SURFACE
. . . . . . . . . . . . . . .
. 2 . 2 . 1 . 1 . 5 . 5 . 5 .
. . . . . . . . . . . . . . .
. 2 . 2 . 8 . 8 . 5 . 2 . 1 .
. . . . . . . . . . . . . . .
. 7 . 1 . 1 . 8 . 2 . 2 . 2 .
. . . . . . . . . . . . . . .
. 8 . 7 . 8 . 8 . 8 . 8 . 5 .
. . . . . . . . . . _____ . .
. 8 . 8 . 1 . 1 . 5 | 3 | 9 .
. . . . . . . . . . |___| . .
. 8 . 1 . 1 . 2 . 5 . 3 . 9 .
. . . . . . . . . . . . . . .
```

```
Output (using -k):                  Output (not using -k):
  COST SURFACE                        CUMULATIVE COST SURFACE
. . . . . . . . . . . . . . .       . . . . * * * * * . . . . . .
. 21. 21. 20. 19. 17. 15. 14.       . 22. 21* 21* 20* 17. 15. 14.
. . . . . . . . . . . . . . .       . . . . * * * * * . . . . . .
. 20. 19. 22. 19. 15. 12. 11.       . 20. 19. 22* 20* 15. 12. 11.
. . . . . . . . . . . . . . .       . . . . . . * * * * * . . . .
. 22. 18. 17. 17. 12. 11.  9.       . 22. 18. 17* 18* 13* 11.  9.
. . . . . . . . . . . . . . .       . . . . . . * * * * * . . . .
. 21. 14. 13. 12.  8.  6.  6.       . 21. 14. 13. 12.  8.  6.  6.
. . . . . . . . . . _____ . .       . . . . . . . . . . . . . . .
. 16. 13.  8.  7.  4|  0|  6.       . 16. 13.  8. 7 .  4.  0.  6.
. . . . . . . . . . |___| . .       . . . . . . . . . . . . . . .
. 14.  9.  8.  9.  6.  3.  8.       . 14.  9.  8. 9 .  6.  3.  8.
. . . . . . . . . . . . . . .       . . . . . . . . . . . . . . .
```

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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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

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*.

OPTIONS

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 value	category 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

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

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 Lab

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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*.

OPTIONS

Program parameters are as follows.

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.

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, r.poly

AUTHOR

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

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.

COMMAND LINE OPTIONS

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

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:

COMMAND LINE 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

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.

COMMAND LINE 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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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.

OPTIONS

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.mapcalc, r.weight, and *parser*

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

Special recognition goes to:
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

Mr. Hageman, in the spring of 1986, submitted an inference engine to the UNIX network. Mr. Cox reworked the code submitting an new version shortly thereafter. It is this code that forms the guts of *r.infer*.

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:  raster          Number 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

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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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 *coordinate*s 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 the whole area within a certain distance of the viewing point inside the geographic region boundaries.
Default: assign all cells that are within the *max_dist* and within 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.

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 Algebra for GIS and Image Processing," by Michael Shapiro and Jim Westervelt, U.S. Army Construction Engineering Research Laboratory (March/1991).

"GRASS Tutorial: *r.mapcalc*," by Marji Larson, U.S. Army Construction Engineering Research Laboratory.

Grey scale conversion is based on the C.I.E. x,y,z system where y represents luminance. See "Fundamentals of Digital Image Processing," by Anil K. Jain (Prentice Hall, NJ, 1989; p 67).

blend.sh, g.region, r.colors, r.combine, r.infer, r.mask, r.weight

AUTHOR

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

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

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.

COMMAND LINE 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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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
```

TITLE — A 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

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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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.

```
   Raw Data      Operation     New Data
----------------             ----------------
| 7  | 7  |  5 |             |    |    |    |
|----|----|----|  average    |----|----|----|
| 4  | 7  |  4 |------------>|    |  6 |    |
|----|----|----|             |----|----|----|
| 7  | 6  |  4 |             |    |    |    |
|----|----|----|             |----|----|----|
```

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,

```
                        _ _ _
                       |_|_|_|
3 x 3 neighborhood --> |_|_|_|
                       |_|_|_|
```

-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 *average*d 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

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 Research Laboratory

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 of 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

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
Andrew Heekin, U.S. Army Construction Engineering Research Laboratory

Modified program for smoothed lines:
David Satnik, Central Washington University, WA

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).

COMMAND LINE 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 4.0:

Single-cell profile:

r.profile map=elevation.dem line=593655,4917280,593726,4917351

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

(Output appears as multiple lines here, but is really one line)

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

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

NAME

r.reclass – Creates a new map layer whose category values are 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      NEW
                           NUM      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      NEW
                           NUM      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 NUM	**NEW CATEGORY NAME**
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.

A *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

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 print-out 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	4.06	
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

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

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.mapcalc, r.reclass, r.report, r.resample, r.stats, and *parser*

AUTHOR

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

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.coin, r.describe, r.mask, r.report, and *parser*

AUTHOR

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

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

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.

r.surf.contour

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

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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 minimize the 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 the number 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 the interpolated 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 map of 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 in choosing 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) Global Climate Research Project U.S. EPA Environmental Research Laboratory 200 S.W. 35th Street, JSB Corvallis, OR 97333

koerper@cs.orst.edu

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

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*.

NOTE

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.

NOTE

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).

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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

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).

COMMAND LINE 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

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*.

COMMAND LINE 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 Number	Average in clump	Data Total	# Cells in clump	Centroid Easting	Centroid Northing	Total 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

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.

depression
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 Models from Digital Elevation Data*, in **Proceedings of International Geographic Information Systems (IGIS) 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

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 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.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 the A^T Search Algorithm to Develop Hydrologic Models from Digital Elevation Data*, in **Proceedings of International Geographic Information Systems (IGIS) 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.grass.armsed r.mask

AUTHOR

Charles Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory

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

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

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

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

NAME

s.in.ascii – Converts an ASCII listing of site locations and their descriptions into a GRASS site list file. *(GRASS Sites Program)*

SYNOPSIS

s.in.ascii
s.in.ascii help
s.in.ascii sites=*name* [**input**=*name*] [**fs**=character|space|tab]

DESCRIPTION

s.in.ascii converts an ASCII listing of site locations and category labels into a file in GRASS site list file format.

Input can be entered via standard input or from the file input=*name*. Each line of input should contain the easting, northing, and (optionally) the category label associated with a site. The fs=*name* option (where *name* is either a character, a space, or a tab) can be used to specify the use of a particular field separator between these three input fields. This is useful when input is obtained from other programs (see NOTES, below). Output is stored in the file sites=*name* and placed in the site_lists directory under the user's current mapset.

The GRASS program *s.out.ascii* can be used to perform the reverse function, converting a file in GRASS site list format into an ASCII listing of eastings, northings, and category labels associated with site locations.

Parameters:

sites=*name* Name of the new GRASS site list file to be output.

input=*name* Name of an existing ASCII file containing site locations and labels.

fs=character|space|tab
The field separator separating the easting, northing, and category label in each line of the *input* file. The field separator can be a character, a space, or a tab.
Default: space

s.in.ascii can be run either non-interactively or interactively. The program will be run non-interactively if the user specifies a name to be assigned to the *sites* file output, the name of an existing ASCII file containing *input*, and (optionally) a field separator *fs* appearing in the *input* file, using the form:

s.in.ascii sites=*name* [**input**=*name*] [**fs**=character|space|tab]

Alternately, the user can simply type **s.in.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*. If the user does not specify the name of an *input* file containing site locations and (optionally) category labels, these should be entered to the program via standard input.

NOTES

Other GRASS programs can be used to produce output in a format suitable for input to *s.in.ascii*. For example, the user might pipe output produced by *d.where* into *s.in.ascii* to create a site list file called *my.sites* containing site locations pointed to with the mouse, as illustrated below. In this example it was unnecessary to specify the field separator used in the input, since *d.where* output separates the easting and northing values with spaces, and spaces are the default field separator assumed by *s.in.ascii*.

d.where | s.in.ascii sites=my.sites

SEE ALSO

d.points, d.sites, d.what.rast, d.where, s.out.ascii, and *parser*

AUTHOR

Michael Shapiro, U.S. Construction Engineering Research Laboratory

NAME

s.menu – Accesses and manipulates GRASS site location data.
(GRASS Sites Program)

SYNOPSIS

s.menu

OVERVIEW

The *s.menu* program provides the user with the capability of interfacing site location data with the geographic data in raster map layers. Two types of spatial analysis reports on sites can be generated, and an interface to the "S" statistical package is provided.

DESCRIPTION

The *s.menu* program is an interface to functions that allow the user to manipulate GRASS site "lists". A site list is a list of eastings and northings describing the location of some point feature. It can also contain a category value and category label for each site location. The program is interactive. After typing **s.menu** on the command line, the user selects site functions from a menu.

The main menu is shown below:

```
SITES MAIN MENU    (current list: no sites)

LOCATION: spearfish    REGION 4928000.00(N)  4914000.00(S)  100.00 (RES)
MAPSET:   PERMANENT           609000.00(E)    590000.00(W)  100.00 (RES)
MASK:  none

Please select one of the following

   1  Read an existing site list
   2  Mask current site list
   3  Save the current site list in your mapset

   4  Check site list for duplicate sites
   5  Edit site list using a UNIX editor

   6  Convert site list to raster file (0/1)
   7  Convert site list to raster file (frequency of occurrence)

   8  Run reports on the current site list


stop  Leave the s.menu program
```

At the top of the menu is general information about the user's current MAPSET, LOCATION, etc. Note the above message in parentheses "current list: no sites." This message will vary depending on the status of the list. For example, after the user reads the existing site list file *arch_sites*, the message would read (given the geographic region indicated): "current list: 25 sites, 24 in current region."

1 Read an existing site list

This option will copy an existing site list file into the current site list within the sites server. Existing site lists are stored under a GRASS data base and are pulled into the *s.menu* server via this option. Other sites menu functions operate only on the current site list file in the server -- you must therefore "read an existing site list" file BEFORE performing any of the other sites menu functions.

Note: Site lists can be created and placed into a GRASS data base using *s.menu* option 5 (edit) followed by option 3 (save). However, the user can also create site lists using other methods or programs. One simple way to do this is to create a site list file in the appropriate format using any text editor (e.g., "vi"), and to put this site list file under the *site_lists* directory under the user's current GRASS mapset (i.e., under $LOCATION/site_lists). The user can do this either inside of GRASS or outside of GRASS. Alternately, the user can run other GRASS programs which format their output as a GRASS site list file (*r.random, s.db.rim, v.mkquads, v.to.sites*), or the user can use UNIX programs like *awk* and *sed* to format other GRASS programs' output in the form of a site list file (*d.what.rast, d.what.vect*).

2 Mask current site list

The site list can be reduced to a subset that includes only sites which fall in specific categories within a specified raster map layer. The user will be asked to specify the name of a raster map layer to form the basis for the mask, and will then be allowed to specify categories from this raster map that will limit the site list. As with *r.mask,* the category values selected designate the areas of the map in which information will remain. Areas assigned category values *not* selected will be re-assigned to category value "0" ("no data").

Note: This masking operation is performed only against the site list itself and not against other raster map layers. If the user wishes to analyze masked raster map layers, a mask should be create using the *r.mask* program.

3 Save the current site list in your mapset

The current site list can be stored permanently in your current mapset with this option. You will be asked to name the saved site list and to provide a short description of it. Saved site lists can be retrieved (option 1) during later runs of *s.menu.* Once saved, these site list files can be used with other GRASS programs, like *d.sites, d.points, d.icons, p.icons, s.surf.idw*, and *s.db.rim*.

Note: Saved lists will be removed if the GRASS mapset under which they are stored is removed.

4 Check site list for duplicate sites

It is not desirable that a site list contain multiple references to the same site. This option attempts to recognize duplicate sites. Duplicates are displayed to the user and can be removed automatically if the user desires. Duplicates can also be removed by hand using option 5 (edit).

5 Edit site list using a UNIX editor

The user can modify the current site list or create a new site list by hand using a UNIX editor. You will be asked to specify the text editor you prefer to use. You should exercise some care if you select this option. Lines in the site list which have invalid formats will be (silently) ignored by *s.menu*. See the GRASS manual entry *sites.format* for a description of the site list format.

Note: This option will only modify the site list copied into the server. It does not modify the original site list stored under a GRASS mapset. If you wish to modify a stored site list file, you will have to combine options 1 (read), 5 (edit) and 3 (save).

6 Convert site list file to raster file (0/1)

You can create a raster data representation of the site list in your GRASS mapset.

Once created, this raster map layer can be used with other raster map layers in further analyses. Allowing the user to create a raster map layer of sites opens up the full analysis capabilities for site data that are available for raster map layers within GRASS.

You have the option of specifying the number of cells to represent a site. The minimum is one cell per site. The alternatives are squares around the site: 3x3, 5x5, 7x7, etc.

The number of categories present in the new raster map layers will depend on the format of your site list file (see *sites.format*). You can create a non-binary raster map layer representation of your site list by creating the site list in the format:

E|N| *#n label*

where **E** is the Easting, **N** is the Northing, and *#n label* is the description field. The description field consists of a pound sign # followed by the category value *n* to be associated with the site's cell location, and the category label *label* for *n*.

If the user does not include a description field starting with

#n

beside the Easting and Northing on *every line* in the sites list file, a binary raster map layer will be created instead. In the binary raster file, each site will be represented as the category value 1. Non-site cells will be coded as category value 0.

Note that only sites within the current geographic region will be considered. However, if the size of the sites is more than one cell (3x3, 5x5, etc.) and a site lies near an edge of the geographic region, some of the cells for the site may fall outside the geographic region. These cells will not appear in the raster map layer, and the site will no longer be 3x3 or 5x5 but will be clipped to fit the geographic region.

7 Convert site list file to raster file (frequency of occurrence)

You can also create a frequency of occurrence raster map layer representation of the site list file.

The raster category values will be coded as the number of sites that fall within the cell.

In this function, you do NOT have the option of specifying the number of cells to represent a site.

8 Run reports on the current site list

The current list of sites is passed to the report server after removing sites which do not fall within the geographic region of the user's current GRASS mapset (see *g.region*). The user then selects the names of one or more raster map layers for analysis. Data at (or near) the sites extracted from these raster map layers form the basis for all reports.

The user specifies the 'size' of a site in cells. The 'size' may be specified as a single cell, or as a 3x3 square around the site, or 5x5, or 7x7, etc (where the size is an odd integer).

The following menu of reports is then presented:

```
SITES REPORT MENU


Please select one of the following

   1  Site characteristics report
   2  Site occurrence report

   3  Convert data to S input format
   4  Produce machine-readable data file

stop  return to SITES MAIN MENU
```

1 Site characteristics report

This report provides geographic information about each site.

Each site is identified by description and locational information. The 'description' is an identification of the site. The site location is an easting and a northing. (The location does not denote the extent of the site, since, for example, an archeologic site which takes up two hectares would be represented as a single point).

The information reported for each site is displayed by raster map layer, and, within each map layer, gives the categories (i.e., characteristics) that occurred at the site (as well as a count of the number of cells in each category).

This can easily generate a massive amount of information, which is difficult to handle or digest quickly. Therefore, option 2 produces a synopsis of the information.

2 Site occurrence report

This report provides aggregate site characteristic information organized by raster map layer. The report produces chi-square statistics for each raster map layer, measuring number of expected sites (assuming a random distribution of sites) against actual site occurrence. The site characteristic is the most frequently occurring cell category in the site (i.e., the statistical mode). See the GRASS manual entry for *sites.occur* for details on this report.

3 Convert data to S input format

This function converts the GRASS data extracted for the sites into a form usable by the S statistical package. The user provides a file to contain the information. Once the file is written, the user must exit *s.menu,* run S on an S data base, and issue the S command

source *file*

to bring the data into the S data base. (Of course, *file* would be the name of the actual file supplied by the user.) See manual entry *sites.S* for an explanation of the S data structures created by this interface.

4 Produce machine-readable data file

This option provides a mechanism for the user to write his/her own reports. The data is written into a user-specified file as a text file, which has a format readable by UNIX utilities (e.g., *awk*) or user-written programs. See GRASS manual entry *sites.report* for details on this format.

FILES

$LOCATION/site_list/<file >

SEE ALSO

d.icons, d.graph, d.points, d.sites, p.icons, r.random, d.what.rast, d.what.vect, r.what, s.in.ascii, s.out.ascii, s.db.rim, s.surf.idw, s.surf.tps, v.db.rim, v.mkquads, v.to.sites,
and
sites.format, sites.report, sites.occur, sites.S

AUTHORS

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
James Farley, Arkansas Archeological Survey, University of Arkansas
contributed the frequency of occurrence sites to cell function

NAME

s.out.ascii – Converts a GRASS site list file into an ASCII listing of site locations and their descriptions. *(GRASS Sites Program)*

SYNOPSIS

s.out.ascii
s.out.ascii help
s.out.ascii [**-ad**] **sites**=*name* [**fs**=character|space|tab]

DESCRIPTION

s.out.ascii converts an existing site list file (sites=*name*) into an ASCII listing of site locations and (optionally) their category labels, in a format suitable for input to other programs (e.g., *d.points*, *m.u2ll*, etc.).

Each line of output consists of the easting, northing, and category label for a site listed in the named *sites* file. The fs=*name* option (where *name* is either a character, a space, or a tab) can be used to place a particular field separator between these three output fields. This is useful when output is to be manipulated by other programs, like *awk* or *sed*.

The GRASS program *s.in.ascii* can be used to perform the reverse function, converting a UNIX file containing eastings, northings, and category labels associated with site locations into GRASS site list file format.

Flags:

-a Output all sites found in the named *sites* file, rather than limiting output to sites falling within the current geographic region.

-d Include site descriptions (category labels) in the output.

Parameters:

sites=*name* Name of an existing site list file.

fs=character|space|tab
The field separator to be placed between the easting, northing, and (optionally) category label on each line of output. The field separator can be a character, a space, or a tab.
Default: space

s.out.ascii can be run either non-interactively or interactively. The program will be run non-interactively if the user specifies the name of an existing site list file and (optionally) a value for *fs*, using the form

s.out.ascii [**-ad**] **sites**=*name* [**fs**=character|space|tab]

where *name* is the name of an existing site list file to be converted to a brief ASCII listing, and *fs* is the field separator to be placed between output fields. The user can also the -a and -d options to use all sites in the named *sites* file and to include site descriptions in the output.

Alternately, the user can simply type **s.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 *s.out.ascii* may be placed into a file by using the UNIX redirection mechanism; e.g.:

s.out.ascii sites=archsites > out.file

s.out.ascii output may also be redirected into other programs; e.g.:

s.out.ascii sites=archsites | d.points color=red size=10 type=diamond

SEE ALSO

d.points, d.sites, m.ll2u, m.u2ll, s.in.ascii, and *parser*

AUTHOR

Michael Shapiro, U.S. Construction Engineering Research Laboratory

NAME

s.surf.idw **–** Surface generation from sites data program.
(GRASS Raster Program)

SYNOPSIS

s.surf.idw input=*name* **output**=*name* [**npoints**=*count*]

DESCRIPTION

s.surf.idw 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.

This program allows the user to use a GRASS site list file, rather than a raster map layer, as input.

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:

s.surf.idw input=*name* **output**=*name* [**npoints**=*count*]

Alternately, the user can simply type **s.surf.idw** 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*.

Parameters:

input=*name* Name of an input site list file 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 site list file.

npoints=*count* The number of points to use for interpolation. By default, the 12 nearest points are used for interpolation.
Default: 12

NOTES

The amount of memory used by this program is related to the number of non-zero data values in the *input* sites list file. If the *input* site list 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* map are used even if they fall outside the mask.

SEE ALSO

d.sites, g.region, r.mask, r.surf.contour, r.surf.idw, r.surf.idw2, s.db.rim, s.menu, and *parser*

AUTHOR

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

NAME

s.surf.tps - Interpolates and computes topographic analysis from given site data to GRASS raster format using spline with tension.
(GRASS Raster Program)

SYNOPSIS

s.surf.tps
s.surf.tps help
s.surf.tps input = name elev = name
[**slope = name**] [**aspect = name**]
[**pcurv = name**] [**tcurv = name**] [**mcurv = name**]
[**maskmap= name**] [**dmin1 = val**] [**zmult = val**]
[**tension = val**] [**smooth = val**]
[**segmax = val**] [**npmin = val**]

DESCRIPTION

s.surf.tps
This program interpolates the values to grid cells from point data (digitized contours, climatic stations, drill holes, etc.) given in a sites file named *input*. The output raster file is *elev*. As an option, simultaneously with interpolation, topographic parameters slope, aspect, profile curvature (measured in the direction of steepest slope), tangential curvature (measured in the direction of a tangent to contour line) or mean curvature are computed and saved as raster files as specified by the options *slope, aspect, pcurv, tcurv, mcurv* respectively.

User can define a raster file named *maskmap*, which will be used as a mask. The interpolation is skipped for cells which have zero value in mask. Zero values will be assigned to these cells in all output raster files.

Data points are checked for identical points and points that are closer to each other, then the given *dmin1* are removed (this is necessary especially for digitized contours). Parameter *zmult* allows the user to rescale the z-values for sites (useful, e.g., for transformation of elevations given in feet to meters, so that the proper values of slopes and curvatures can be computed).

Regularized spline with tension is used for the interpolation. The *tension* parameter tunes the character of the resulting surface from thin plate to membrane. Higher values of tension parameter reduce the overshoots that can appear in surfaces with rapid change of gradient (see suggested values for different types of surfaces given in notes). For noisy data, it is possible to define a smoothing parameter, *smooth*. With the smoothing parameter set to zero (*smooth=0*), the resulting surface passes exactly through the data points.

If the number of given points is greater than 400, segmented processing is used. The region is split into rectangular segments, each having less than *segmax* points and interpolation is performed on each segment of the region. To ensure the smooth connection of segments the interpolation function for each segment is computed using the points in a given segment and the points in its neighborhood. The minimum number of points taken for interpolation is controlled by *npmin*, the value of which must be larger than *segmax* and less than 400. This limit of 400 was selected to ensure the numerical stability and efficiency of the algorithm. The program writes important values related to the computation to the history file of raster map *elev*.

OPTIONS

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

s.surf.tps input = name elev = name
[**slope = name**] [**aspect = name**]
[**pcurv = name**] [**tcurv = name**] [**mcurv = name**]
[**maskmap= name**] [**dmin1 = val**] [**zmult = val**]
[**tension = val**] [**smooth = val**]

[**segmax = val**] [**npmin = val**]

Alternatively, the user can simply type **s.surf.tps** 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*.

Parameters:

input=*name*	Use the existing site file *name* as input.
elev=*name*	Output elevation values to raster file named *name*.
slope=*name*	Output slope values to raster file named *name*.
aspect=*name*	Output aspect values to raster file named *name*.
pcurv=*name*	Output profile curvature values to raster file named *name*.
tcurv=*name*	Output tangential curvature values to raster file named *name*.
mcurv=*name*	Output mean curvature values to raster file named *name*.
maskmap=*name*	Use the existing raster file *name* as a mask.
dmin1=*val*	Set min distance between points to *val*. Default value is set to 0.5 grid cell size.
zmult=*val*	Convert z-values using conversion factor *val*. Default value is 1.
tension=*val*	Set tension to *val* . Default value is 40, appropriate for smooth surfaces.
smooth=*val*	Set smoothing parameter to *val* . Default value is 0, no smoothing is performed.
segmax=*val*	Set max number of points per segment to *val*. Default value is 40.
npmin=*val*	Set min number of points for interpolation to *val*. Default value is 150, for data with heterogeneous spatial distribution higher value is suggested (see Notes).

NOTES

s.surf.tps uses regularized spline with tension for interpolation from point data (as described in Mitasova and Mitas, in press). The implementation has an improved segmentation procedure based on quadtrees which enhances the efficiency for large data sets. Special color tables are created by the program for output raster files.

Topographic parameters are computed directly from the interpolation function so that the important relationships between these parameters are preserved. The equations for computation of these parameters and their interpretation is described in Mitasova and Hofierka, in press. Slopes and aspect are computed in degrees (0-90 and 1-360 respectively). The aspect raster file has value 0 assigned to flat areas (with slope less than 0.1%) and to singular points with undefined aspect. Aspect points downslope and is 90 to the North, 180 to the West, 270 to the South and 360 to the East, the values increase counterclockwise. Curvatures are positive for convex and negative for concave areas. Original values of curvatures are multiplied by 100000, to conform with GRASS integer raster files. Therefore any curvature lower than 0.00001 will be zero. Flat areas have zero curvatures and singular points have codes 1000000, 2000000, 3000000 for peak, pit and saddle respectively. We suggest to use these codes only to distinguish areas (grid cells) with undefined curvature because the codes are assigned using the theorems from differential geometry but have never been tested.

The program gives warning when significant overshoots appear and higher tension should be used. However, with tension too high the resulting surface changes its behavior to membrane (rubber sheet stretched over the data points resulting in a peak or pit in each given point and everywhere else the surface goes rapidly to trend). Smoothing can also be used to reduce the overshoots if the resulting surface should be smooth.

For data with values changing over several magnitudes (sometimes the concentration or density data) it is suggested to interpolate the log of the values rather than the original ones.

The program checks the numerical stability of the algorithm by computation of values in given points, and prints the maximum difference found into the history file of raster map *elev*. Significant increase in tension is suggested if the difference is unacceptable. For computation with smoothing set to 0 this difference should be 0. With smoothing parameter greater than zero the surface will not pass through the data points and the higher the parameter the closer the surface will be to the trend. The maximum difference between the given and approximated value in this case reflects the smoothing effect on data.
For theory on smoothing with splines and their statistical interpretation see Talmi and Gilat 1977, Wahba 1990, and Hutchinson 1992, where you can find also a comparison of smoothing splines with kriging.

The program writes the values of parameters used in computation into the comment part of the history file *elev* as well as the following values which help to evaluate the results and choose the suitable parameters: minimum and maximum z values in the data file (zmin_data, zmax_data) and in the interpolated raster map (zmin_int, zmax_int), maximum difference between the given and interpolated z value in a given point (errtotal), rescaling parameter used for normalization (dnorm), which influences the tension (see Mitasova, 1992; Mitasova and Mitas, in press).

If a visible connection of segments appears, the program should be rerun with higher *npmin* to get more points from the neighborhood of the given segment.

If the number of points in a site file is less then 400, *segmax* should be set to 400 so that segmentation is not performed when it is not necessary.

The program gives a warning when the user wants to interpolate outside the rectangle given by the minimum and maximum coordinates in the site file, zooming into the area where the points are is suggested in this case.

When a mask is used, the program takes all points in the given region for interpolation, including those in the area which is masked out, to ensure proper interpolation along the border of the mask. It therefore does not mask out the data points; if this is desirable, it must be done outside *s.surf.tps*.

The program was used for various applications with the following parameters :

interpolation of DEM from digitized contours

tension	20. - 80.
smoothing 0.01 - 1.0	
segmax	40
npmin	200 - 300

(low tension was used for relatively flat terrain, high tension was necessary for terrain with sharp changes in slope, low value of smoothing is usually sufficient for dense and accurately digitized contours, for less dense and not very carefully digitized contours, higher smoothing is suggested)

interpolation of precipitation from climatic stations

tension	40. - 150.
smoothing	0. - 2.
segmax	40
npmin	200

interpolation of concentration of chemicals

tension	20. - 60.
smoothing	0.5 - 5.0

The user must run *g.region* before the program to set the region for interpolation.

SEE ALSO

v.to.sites, g.region, r.surf.contour, r.surf.idw, r.surf.idw2, r.surf.sor, s.surf.idw

AUTHOR

Original version of program (in FORTRAN):
Helena Mitasova, Illinois Natural History Survey and US Army CERL, Champaign, Illinois
Comenius University, Bratislava, Czechoslovakia,
Lubos Mitas, Department of Physics, University of Illinois at Urbana Champaign, Illinois Institute of Physics, Bratislava, Czechoslovakia

Modified program (translated to C, adapted for GRASS , segmentation procedure):
Irina Kosinovsky, US Army CERL
Dave Gerdes, US Army CERL

REFERENCES

Hutchinson, M. K. and Gessler, P. E., 1992. Splines: More than just a smooth interpolator, *Geoderma*.

Mitasova, H. and Mitas, L., in press. Interpolation by regularized spline with tension: I. Theory and implementation, *Mathematical Geology*.

Mitasova, H. and Hofierka, L., in press. Interpolation by regularized spline with tension: II. Application to terrain modeling and surface geometry analysis, *Mathematical Geology*.

Mitasova, H., 1992. New capabilities for interpolation and topographic analysis in GRASS, *GRASSClippings*, v.6, No.2 (summer), p 13.

Mitasova, H., 1992. Surfaces and modeling, *GRASSclippings*, v.6, No.3 (winter), pp 16-18.

Talmi, A. and Gilat, G., 1977. Method for smooth approximation of data, *Journal of Computational Physics*, 23, pp 93-123.

Wahba, G., 1990. Spline models for observational data, CNMS-NSF Regional Conference series in applied mathematics, 59, SIAM, Philadelphia, Pennsylvania.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.alabel – Bulk-labels unlabeled area features in a binary GRASS vector file.
(GRASS Vector Program)

SYNOPSIS

v.alabel
v.alabel help
v.alabel [**-i**] **map**=*name* [**value**=*value*]

DESCRIPTION

v.alabel allows the user to bulk-label (area features) in a binary GRASS vector file (i.e., a *dig* file). The user must run *v.support* on the vector file before running *v.alabel* if any modifications have been made to the file since the last time *v.support* was run on it, to ensure that all area features are properly identified in the file topology.

The user must also run *v.support* on the vector file after *v.alabel* is run for labeling changes to be made evident.

v.support builds GRASS support files for (binary) GRASS vector data files. These support files supply topology (*dig_plus*) and category (*dig_att*) information that are needed by other GRASS programs (e.g., by *v.digit*, *v.to.rast*, etc.).

OPTIONS

Program parameters and flags have the following meanings.

Flag:

-i	Label areas incrementally. For each unique, unlabeled polygon in the vector file, increment the category value by one, starting from the initial user-assigned *value*.

Parameters:

map=*name*	Name of binary GRASS vector data file whose unlabeled areas are to be labelled. This map layer must exist in the user's current GRASS mapset.
value=*value*	The category value to be assigned to all unlabeled areas in the vector map layer. If the *-i* flag is set, *v.alabel* will increment the initial *value* by one for each unique unlabeled polygon in the vector map. Default: 1

The user can run this program non-interactively by specifying parameter values (and optionally, the flag setting) on the command line.

Alternately, the user can simply type the command **v.alabel** on the command line. In this event, the program will prompt the user to enter parameter values and flag settings.

NOTES

A *dig_plus* file must be created for each imported vector map before it can be used in post-GRASS 3.0 versions of *digit* (now referred to as *v.digit*).

Topological information for GRASS vector files can also be built using *option 4* of the *v.import* program.

The user can bulk-label unlabeled line features in a binary vector file using *v.digit*.

SEE ALSO

v.digit, v.import, v.in.ascii, v.prune, v.support, v.to.rast

AUTHORS

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.apply.census – Calculate/Import Demographics from Census STF1 Files
(GRASS Vector Program)

SYNOPSIS

v.apply.census
v.apply.census help
v.apply.census [-d] [-p] [-l] [-u]
v.apply.census input_stf1=name [out=name] [ef=name] [formula=mapname=expression] [name_field=name] [zone=value] [spheroid=name]

Flags:
-d Output IDENTIFICATION SECTION to stdout (20 pages)
-p Output STF1 MATRIX TABLE to stdout (30 pages)
-l Output PL94-171 MATRIX TABLE to stdout (1 page)
-u Output STF1 SUMLEV TABLE to stdout (4 pages)

Note: Only the first flag given will be executed; the program exits after sending one table to stdout. Other parameters are ignored.

Parameters:
input_stf1 Path/name of STF1 or PL94 input file
out Type of output: site | atts | table:Lxxx | - (stdout)
default: -
ef Path/name of text file with formula expression(s)
formula map=expression
name_field field name for parsing stdin lines (-a to ignore)
default: -a
zone UTM zone number; default is location zone
options: 1-60
default: 10
spheroid Spheroid for LL to UTM conversion; see m.gc.ll
default: clark66

NOTE: Only one of the "ef" or "formula" input fields may be used.

DESCRIPTION

This program reads a previously selected subset of STF1 or PL94-171 U.S. Census Bureau demographic records (see *m.in.stf1.tpe* and *m.in.stf1.db3*), and writes one of the following:

GRASS site list (coordinates and a value).
GRASS vector polygon attribute labels file (coordinates and a label value).
Text report to stdout in one of two formats.

Any arithmetic expression combining constants with any of the more than 900 hundred demographic (numeric) fields can be defined and will be evaluated (by the Unix *bc -l* utility) to compute the value for each input record.

The location coordinates for the polygon label point or the site location is obtained from certain columns (269-287) in the input records, making this program quite specific for the specified types of STF1 and PL94-171 input file records. Use with other types of input data is not advised.

SOME PARAMETERS EXPLAINED BRIEFLY

out=
  att   for results to vector map attribute file
  site for results to site list
  -     for results to stdout; this is the default
  table: for results in table form to stdout with
        the column value(s) represented by the ':Lxxx:Lxxx' string
        preceeding the expression value instead of easting and northing.
        Lxxx is in same choice of representation as column designation
        in formula (see below).

formula=
  mapname=Computation with STF1A columns, constants and *bc* operators
        (see below). mapname of vector or site map to make is required
        in all cases (but ignored for '-' or 'table:' output modes).

name_field=
  name is Keyword in stdin stream preceeding the number of a STF1 record
        to read from input_stf1 file. The number is a physical sequence
        number, not a record identification number. This parameter is
        appropriate only in special uses of this program. If '-a' is
        given, or this parameter is omitted, all records in STF.file will
        be processed.

THE formula PARAMETER

The string after the '=' in this parameter almost always must be enclosed in quotes to protect it from Shell interpretation of characters such as * / ( ) and spaces (which may be used to increase readability, but are not necessary). This string is always of the form *mapname=expression*.

The *mapname* string can be any legal GRASS map name (up to 14 characters). The second '=' is required and may be preceeded and followed by a space.

In the *expression*, great flexibility is provided for the computation of interesting combinations of the data fields contained in the demographic records.

The usual operators used in the *expression* include: +, -, *, /, and %. The user should review the Unix manual entry for the *bc* calculator for the complete list of available functions and other operators.

Parentheses are used in the normal algebraic manner.

The operands in the *expression* may consist of any mixture of:

  integer constants
  real constants
  functions allowed by *bc*
  numeric fields from the demographic records

Numeric fields from both the IDENTIFICATION and MATRIX SECTIONS of the demographic records may be used. Review the User and Technical Documentation for these demographic files; or use the *-p* option of this program to print the MATRIX SECTION document (the demographic data fields), or *m.in.stf1.db3* to print a simplified list of IDENTIFICATION SECTION field names. When substituting values from the numeric fields into the expression, <SPACE> characters are replaced by zero. (Spaces, which are rare in the demographic data, are usually the result of missing values or restricted information).

The numeric fields from the demographic records may be designated in one of three ways in the *expression*.

1. 'Lcccc', where 'L' is an upper case alphabetic letter which indicates the length of the numeric field in the data records: A=1, B=2, ... , I=9, J=10, O=15, etc; and 'cccc' is the starting column number for the data field of interest: 301 for 100% population count, 7221 for total number of four-room housing units, 58 for 101st Congressional District, etc. The proper specification of the Congressional District number would be 'B58' because it is a two-column field. 'I301' would indicate that the 100% poplation should be used in the calculation; it's a nine-column field.

2. 'Pnna0nnn' or 'Hnna0nnn', where 'n' is a digit and 'a' is a digit or (rarely) upper case letter. These forms represent the MATRIX field naming schemes used in the CD-ROM "dBase 3" files. They can be used in processing STF1 records extracted from the CD-ROM or TAPE distribution media. All eight characters of the field name must be used. (Note: this form cannot be used in processing the PL94-171 records.)

3. 'ID_NAME'. The STF1 and PL94-171 files use the same set of IDENTIFICATION SECTION field names (67 fields) for the "locational" information contained in the first 300 characters of each record. The field name, in all upper case letters, may be used if the numeric value of the field is needed in the *expression*. Two of the most useful fields are AREALAND and AREAWAT which allow the direct computation of population density values.

The *bc* calculator usually returns a value with a very large number of decimal places. Vector attributes are automatically truncated(!) to integers by the v.support program when the topology file is built. Site list descriptions are likewise truncated to integers by GRASS programs which use the descriptions as numeric values (e.g., sites to cell in *s.menu*).

THE 'out=table:' PARAMETER

The default operation of *v.apply.census* when tabular reports are produced to *stdout* (when not making a sites list or vector attribute file) is to print the easting and northing coordinates and then the value resulting from the *expression*. Often the user wishes to have an identifier different from the coordinates. The construction *out=table:field* will replace the coordinates with the character string indicated by *field*, which may have any of the three forms used for numeric fields in the formula *expression*, see above. Note: a special case exists for the 66 character description field which begins in column 192; the entire field will be printed if designated by either of these two forms: *out=table:ANPSADPI* or *out=table:Z192*.

Complex tables may be produced by making multiple runs of *v.apply.census*, redirecting the tabular output to files, and using the Unix tools *cut, paste,* and *colrm* to combine the resulting files.

EXAMPLES

[These examples assume that STF1 records for the census tracts (SUMLEV=140) in a particular county (CNTY=037) have been extracted from the distribution source (with *m.in.stf1.db3* or *m.in.stf1.tpe* and saved in a file named 037.140 in the current directory.]

1. Create a site list where the label values are the percentage of females in each input record:

```
v.apply.census in=037.140 out=site formula='pct.female = 100 * (I373 / I301)'
```

2. Label an existing vector map (named tract.pop) of tract boundaries with the total population of each tract (run *v.support* and build topology after creating the attributes file):

```
v.apply.census in=037.140 out=atts formula=tract.pop=P0010001  (or)
v.apply.census in=037.140 out=atts formula=tract.pop=I301
```

3. Produce a tabular report of the census tract numbers and the number of Hispanics per square kilometer:

```
v.apply.census in=037.140 out=table:TRACTBNA formula="m=(P0080001/AREALAND)*1000"
```

4. Produce a tabular report of the number of people per housing unit for each tract and the coordinates of the internal (label) point:

```
v.apply.census in=037.140 formula="map=P0010001/H0010001"
```

FORMAT OF THE FORMULA TEXT FILE

Running this program with the *ef=file* command line parameter causes the named file to be read and each formula contained therein to be processed as if it were entered on the command line. The *out=* parameter may optionally be respecified in this file; each *out=* selection remains in effect until explicitly changed. The program exits after the last formula is processed.

The following is a sample formula file. Note the use of lines beginning with '#' as manditory formula separators and comment lines. Also note that expressions may be continued on successive lines; the lines are concatenated to a maximum of 500 characters for a single formula. Blank lines are ignored.

```
popden.sqkm = 1000 * P0010001 / (AREALAND+AREAWAT)
# that computes population density in people per sq. km.
#
# next do people per sq. mile as a vector attribute file
out=att
popden.sqmi = 2.59*1000 *
P0010001 / (AREALAND+
AREAWAT)
#
# next do total population as a vector attribute file
total.pop = I301
#
# output the county identification numbers as the descriptions
# in a site list
out=sites county = CNTY
#
# output the 66 char. description and FIPS Place Code as a table
out=table:ANPSADPI
map=PLACEFP
# optional ending comment line
```

BUGS/FEATURES

Computational errors in *bc* are not handled too gracefully: a warning is output and a zero result is used.

bc tends to output lots of decimal places. The user must clean this up for output sent to stdout.

The GRASS site list output format used includes the '#' before the label value to facilitate the production of raster maps with cell values representing the results of the formula execution.

If using the "name_field" and "ef" parameters, the formula file may contain only one formula.

SEE ALSO

m.in.stf1.tpe is used as a preprocessor to select subsets of STF1 or PL-171 tape format records for input to this program. *m.in.stf1.db3* is used as a preprocessor to select subsets of STF1 records from the CD-ROM in "dBase 3" format. Unix utility programs such at *grep* or *awk* can also be used to select subsets of lines from the PL94-171 files, but not from the STF1 tape files due to their very long record lengths.

AUTHOR

Dr. James Hinthorne, GIS Laboratory, Central Washington University. July, 1992.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.area – Display GRASS area and perimeter information for GRASS vector map.
(GRASS Vector Program)

SYNOPSIS

v.area
v.area help
v.area [**–f**] **map**=*name* [**color**=*name*]

DESCRIPTION

The GRASS program *v.area* first displays the selected vector file. Then user can select area on map by clicking with mouse within the desired area. Selected area will be highlighted in selected color on graphics display. On regular screen area information will be displayed, in square meters, hectares, acres, and square miles, Perimeter measurements, in meters, feet, and miles, are also displayed.

User can repeatedly select areas for analysis, one at a time.

Flag:

-f	Fill selected area with selected color on graphics display, rather than simply outlining it in the *highlight* color.

Parameters:

map=*name*	Name of the vector map layer to be displayed.
color=*name*	Color in which perimeter will be highlighted. Default: red

If the user simply types **v.area** without specifying program arguments on the command line, the program will prompt the user for the name of a map. Then the user is given the option of specifying a highlight color, and then, whether or not the display of selected area(s) is to be filled, rather than merely outlined, with the highlight color.

SEE ALSO

d.vect

AUTHOR

Bruce Powell, National Park Service, Denver CO.

NAME

v.cadlabel - Attaches labels to (binary) vector contour lines that have been imported to GRASS from DXF format.
(GRASS Vector Program)

SYNOPSIS

v.cadlabel
v.cadlabel help
v.cadlabel lines=*name* **labels**=*name*

DESCRIPTION

v.cadlabel attaches labels to index contour lines by using the index contour lines and labels files that have been converted from DXF format to GRASS vector format by the *v.in.dxf* program. Users have the option of creating either binary or ASCII output with *v.in.dxf*. Since *v.cadlabel* works only on binary vector files, the ASCII GRASS vector (*dig_ascii*) files that are generated by *v.in.dxf* must be converted to binary GRASS vector (*dig*) files using the *v.in.ascii* program before *v.cadlabel* can be executed.

v.cadlabel searches a binary GRASS vector file of contour lines to find the contour lines that are closest to each box (contour label) in the binary GRASS vector file containing contour labels. The two contour lines that are closest to a box are tagged with the same label (an elevation value) as that of the box.

OPTIONS

Program parameters are described below.

Parameters:

lines=*name* Name of the binary GRASS vector (*dig*) file, imported from DXF format, that contains index contour lines.

labels=*name* Name of the binary GRASS vector (*dig_att*) file, imported from DXF format, that contains index contour line labels.

NOTES

Because line data that are created in CAD format may have unsnapped nodes or gaps, *v.cadlabel* will not always be able to label every index contour line. Also, intermediate contour lines that may be contained in the index contour vector file (because they resided on the same DXF level as the index contour lines in the DXF design file) will not be labeled. Any lines that are not labeled with *v.cadlabel* can be labeled with the contour labeling program in *v.digit*.

If the intermediate contour lines are in a separate GRASS *dig* file, they can be patched to an labeled index contour file with the GRASS program *v.patch*, and then labeled in *v.digit*.

The user does not need to *v.patch* the labels vector file (boxes) to the lines vector file (contour lines). The purpose of the labels vector file is to determine which labels should be assigned to which contour lines (in the lines vector file) during the execution of *v.cadlabel*.

SEE ALSO

v.digit, v.in.ascii, v.in.dxf, v.out.dxf, v.patch and *dxfout* (Microstation tool)

AUTHOR

David Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.clean – Cleans out *dead* lines in GRASS vector files.
(GRASS Vector Program)

SYNOPSIS

v.clean
v.clean help
v.clean map=*name*

DESCRIPTION

When programs like digit delete lines, they do not really go away, but are simply marked as deleted. This is done primarily for speed. But a side effect is that vector files can end up carrying a lot of dead weight with them.

v.clean will go through a vector file and remove all dead lines in the file thus potentially reducing the size of the file.

You do *not* have to run **v.support** afterwards.

COMMAND LINE OPTIONS

Parameter:

map	Name of the GRASS vector file to be cleaned.

BUGS

None known.

SEE ALSO

v.digit, v.support

AUTHOR

David Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.cutter – Polygon Cookie Cutter (Boolean *AND* Overlay)
(GRASS Vector Program)

SYNOPSIS

v.cutter
v.cutter help
v.cutter [-u] [-q] cutter=*name* **data**=*name* **out**=*name*

DESCRIPTION

This program provides a way to generate new maps based on an intersection of two existing maps. It in effect provides a way to create masked versions of vector maps. Both lines, sites, and polygons are clipped correctly.

COMMAND LINE OPTIONS

Parameters:

-q	Run quietly. Don't print percent information.
-u	Intersect all areas even those that are unlabeled. Not generally useful and also not fully tested.
cutter=mapname	Name of the binary vector file to use for the cookie cutter.
data=mapname	Name of the binary vector file which is to be cut.
out=mapname	Name of new vector file to be created.

NOTES

The user must run *v.spag -i* after running *v.cutter* to remove identical lines.

The polygons of the cutter map must be labeled to be effective unless the -u flag is specified.

The attributes of created polygons will be generated from the attributes of the *data* map.

BUGS

There are a few rare situations that are not currently handled correctly. These mostly involve nodes or vertices intersecting exactly with lines of the opposite map. You will know you have hit one of these cases because a lot of strange text will start spitting out.

Currently the program generates many duplicate lines. So *v.spag -i* must be run on the new file to clean them up. The -i flag specifies to only do the identical line removal phase and is a new option to *v.spag* in release 4.1. Don't forget to use this flag, as it will run much faster, and will not make any other changes to your data.

Borders between areas with the same attributes are not disolved.

There is no *fuzzy* data handling code, so expect to see slivers.

There are many places where the code could be optimized greatly.

SEE ALSO

v.spag, v.digit, v.support, v.clean

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.digit − A menu-driven, highly interactive map development program used for vector digitizing, editing, labeling and converting vector data to raster format.
(GRASS Vector Program)

SYNOPSIS

v.digit

DESCRIPTION

The GRASS program *v.digit* is a menu-driven, highly interactive map development program used for inputting analog map data into a GRASS vector format. *v.digit* contains programs for vector digitizing, editing, labeling, windowing, and converting vector data to GRASS raster format.

v.digit consists of two parts: an initialization procedure, and a multiple menu-driven environment.
1) The initialization procedure involves choosing a digitizer, choosing the name of a vector map layer to digitize, and, if needed, registering a map to the digitizing surface.
2) The second part of *v.digit* is a multiple-menu environment in which digitizing, editing, labeling, and other options are available. It is within this second portion of *v.digit* that all vector creation occurs.

NOTES

The GRASS 4.1 version of v.digit has been upgraded so that it now no longer requires drivers to be written for each digitizer supported. Instead we are using device independent code written originally by John Dabritz formerly of the Forest Service, which uses ascii description files to define how to communicate with the digitizer. The result is that we should be able to support just about any digitizer out there now with a minimum of work. Refer to the "digitizer definition manual" for information on the format of these files.

v.digit was written to optimize digitizing speed and performance. It has a convenient graphic display format and very convenient windowing capabilities. Features are color-coded for ease of identification and verification. Different color schemes may be user-chosen to customize a digitizing session.

Area, line, and point features may be digitized in both "stream" and "point" modes. Both a mouse and a digitizer may be used to perform windowing functions. Labeling and editing are done from within *v.digit*, rather than through a separate set of programs.

v.digit has capabilities that allow users to convert vector map layers to GRASS raster format, to overlay already existing vector map layers onto the feature being digitized, and set a multitude of parameters to customize a digitizing session.

v.digit may be used with or without a digitizer. Many options are available if no digitizer is used.

All options available within *v.digit* are contained within ten menus. Movement from menu to menu is done by chosing the movement options specified at the bottom of each menu.

SEE ALSO

GRASS Tutorial - digit: Its Use and Its Features

v.import, v.in.ascii, v.out.ascii, v.out.rast, v.support

AUTHORS

David Gerdes, U.S. Army Construction Engineering Research Laboratory
Mike Higgins, U.S. Army Construction Engineering Research Laboratory

NAME

v.digit2 − A menu-driven, highly interactive map development program used for vector digitizing, editing, labeling and converting vector data to raster format.
(GRASS Vector Program)

SYNOPSIS

v.digit2

DESCRIPTION

The GRASS program *v.digit2* is a menu-driven, highly interactive map development program used for inputting analog map data into a GRASS vector format. *v.digit2* contains programs for vector digitizing, editing, labeling, windowing, and converting vector data to GRASS raster format.

v.digit2 consists of two parts: an initialization procedure, and a multiple menu-driven environment.
1) The initialization procedure involves choosing a digitizer, choosing the name of a vector map layer to digitize, and, if needed, registering a map to the digitizing surface.
2) The second part of *v.digit2* is a multiple-menu environment in which digitizing, editing, labeling, and other options are available. It is within this second portion of *v.digit2* that all vector creation occurs.

NOTES

v.digit2 was written to optimize digitizing speed and performance. It has a convenient graphic display format and very convenient windowing capabilities. Features are color-coded for ease of identification and verification. Different color schemes may be user-chosen to customize a digitizing session.

Area, line, and point features may be digitized in both "stream" and "point" modes. Both a mouse and a digitizer may be used to perform windowing functions. Labeling and editing are done from within *v.digit2*, rather than through a separate set of programs.

v.digit2 has capabilities that allow users to convert vector map layers to GRASS raster format, to overlay already existing vector map layers onto the feature being digitized, and set a multitude of parameters to customize a digitizing session.

v.digit2 may be used with or without a digitizer. Many options are available if no digitizer is used.

All options available within *v.digit2* are contained within ten menus. Movement from menu to menu is done by chosing the movement options specified at the bottom of each menu.

SEE ALSO

GRASS Tutorial - digit: Its Use and Its Features

v.import, v.in.ascii, v.out.ascii, v.out.rast, v.support

AUTHORS

David Gerdes, U.S. Army Construction Engineering Research Laboratory
Mike Higgins, U.S. Army Construction Engineering Research Laboratory

NAME

v.import – Converts ASCII Digital Line Graph (DLG) files, binary DLG files, and ASCII vector files into binary vector files and creates the needed vector support files.
(GRASS Vector Program)

SYNOPSIS

v.import

DESCRIPTION

v.import is an interactive program which runs other vector import programs. It allows you to import DLG or ASCII vector files into GRASS vector format. It also runs *v.support*. No arguments are required on the command line. However, before running *v.import*, the user should move the ASCII DLG files to be imported into GRASS in a directory named **dlg** under the user's current GRASS mapset (i.e., under the directory known to GRASS as $LOCATION). If a **dlg** directory does not already exist under the user's current GRASS mapset, then the user will have to make one using the UNIX *mkdir* command, and then move the ASCII DLG files to be imported under that new **dlg** directory.

This program performs all of the processes that are needed to convert ASCII Digital Line Graph (DLG) files, binary DLG files, and ASCII vector files into binary vector files. It also creates two support files, the **dig_plus** file and the **dig_att** file (only created when importing DLG files). The **dig_plus** file contains topological information obtained by analyzing the vector file. The **dig_att** file contains category (attribute) information files by the labeling function of the GRASS *v.digit* program. All of the above files must be run through *v.import* before they can be used in the *v.digit* program of GRASS 4.0.

IMPORT FILES

After entering the command **v.import**, the user will be asked for the type of file that is to be imported and for which support files will be created:

Import to GRASS Vector Format and Create Needed Support Files

1 - ASCII DLG file to GRASS Vector Format
2 - Binary DLG file to GRASS Vector Format
3 - ASCII VECTOR file to GRASS Vector Format
4 - Binary VECTOR file to GRASS Vector Format

If numbers 1-3 are chosen, *v.import* will respond with the current data base units (in feet or meters), and ask if the new vector file is in the correct units for the data base location. If the new vector file is not in the correct units, *v.import* will not allow it to be placed in the current data base location. For each data base location, all map layers should have the same units. If, for some reason, a map layer uses different units than the rest of the map layers in the same data base location, a new data base location will have to be created for it.

ASCII DLG File to GRASS Vector Format:

Convert an ASCII DLG file (such as those created in GRASS 2.0) to a GRASS vector file and creates the **dig_plus** and **dig_att** support files. The user is asked two questions:

1. Determine if this map is composed of Area or Line information.
Do you want to give precedence to Areas (opposed to Lines)? (y/n) [y]

NOTE: Some machine-processed DLG files do not make the distinction between lines and area edges. For example, in a roads map, where the desired information is line data, a downtown block surrounded by roads may be processed as an area. Because of this, the user is asked to choose whether to give precedence to areas or lines. If precedence is given to lines, the user should be aware that any lines that bound unlabeled areas in the DLG file will be stored as line data. Any unlabeled areas would therefore be lost (this is only a concern when areas are unlabeled, labeled area information will be retained). If precedence is given to areas, lines will be stored as boundaries to areas that are unlabeled.

2. During the building of the Vector format:
Do you want to snap nodes to other nodes within a threshold (y/n) [n]

NOTE: BE CAREFUL! This threshold is calculated using the scale of the original DLG or *dig* file. If the threshold is too high, excessive snapping may occur, destroying the file. In general, users seldom need to snap nodes. If snapping of nodes is desired, the user may want to run *v.support* separately. *v.support* allows the user to set the snapping threshold.

This process is done in three phases:
1. The ASCII DLG file is converted to a binary DLG file.
2. The binary DLG file is converted to a binary Vector file, and the **dig_att** support file containing attribute (category) information is created.
3. The **dig_plus** support file is created by analyzing the vector file for topological information.

Binary DLG File to GRASS Vector Format:

Converts binary DLG files (such as those created in GRASS 2.0) to a vector file and creates the **dig_plus** and **dig_att** support files. The user is asked whether precedence should be given to Areas or Lines and if nodes should be snapped to other nodes within a calculated threshold.

This process is done in two phases:
1. The binary dlg file is converted to a binary vector file, and the **dig_att** vector support file containing attribute information is created.
2. The **dig_plus** vector support file is created by analyzing the vector file for topological information.

ASCII Vector file to GRASS Vector Format:

Converts ASCII vector files (such as those created in GRASS 2.0) into GRASS binary vector files, and creates the **dig_plus** vector support file. Since a vector maintains the distinction between lines and area edges, the user is not asked to give precedence to either. However, the user will be asked if snapping from nodes to other nodes within a calculated threshold is desired.

This process is done in two phases:
1. The ASCII vector file is converted to a binary vector file, and the **dig_plus** vector support file is created.
2. The **dig_plus** vector support file is created by analyzing the vector file for topological information.

Binary Vector file to GRASS Vector Format:

Creates the **dig_plus** vector support file.

This process is done in one phase:
1. The **dig_plus** vector support file is created by analyzing the vector file for topological information.

NOTES

Before running *v.import*, the user should move the ASCII DLG files to be imported into GRASS in a directory named **dlg** under the user's current GRASS mapset (i.e., under the directory known to GRASS as $LOCATION). If a **dlg** directory does not already exist under the user's current GRASS mapset, then the user will have to make one using the UNIX *mkdir* command, and then move the ASCII DLG files to be imported under that new **dlg** directory.

SEE ALSO

v.digit, v.in.ascii, v.support

AUTHORS

Michael Higgins, U.S. Army Construction Engineering Research Laboratory
Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.in.arc **–** Converts data in ARC/INFO format to GRASS's vector format, and stores output in the user's current GRASS mapset.
(GRASS Vector Data Import Program)

SYNOPSIS

v.in.arc
v.in.arc help
v.in.arc [**-n**] **type**=*name* **lines_in**=*name* [**points_in**=*name*] [**text_in**=*name*]
vector_out=*name* [**idcol**=*value*] [**catcol**=*value*] [**attcol**=*value*]

DESCRIPTION

The user may wish to use GRASS programs on data files that were created by other GISs. To do this, the user must first convert data files in these systems' formats to GRASS's file format. Bringing data from other systems into GRASS is termed file *import*. Sending GRASS data files out into other systems' formats is termed file *export*.

A variety of GRASS programs exist to import and export data to and from GRASS. The *v.in.arc* program will convert vector data in ARC/INFO's "Generate" format to GRASS's vector file format, and bring it into the user's current GRASS mapset. The files to be imported to GRASS must first have been exported from ARC/INFO using the ARC/INFO *Ungenerate* command, and may represent either linear features ("line coverage") or areal features ("polygon coverage"). The *ARC/INFO User's Guide* describes how files containing linear and polygonal features can be exported from ARC/INFO, in a section detailing the use of the *Ungenerate* command.

Note: The ARC coverage must be set to single precision before running *Ungenerate*. If it is not, first copy it to another coverage that is set to single precision, then run *Ungenerate*.

Once converted with the ARC/INFO *Ungenerate* command, the files to be imported into GRASS must be placed in a directory named *arc* in the user's current mapset. If the *arc* directory does not exist, it must be created (e.g., with the command *mkdir $LOCATION/arc*) before copying the ARC-INFO files to be converted into it.

(*v.in.arc* can be used to convert ARC-INFO data from other mapsets as well, since the program searches for the specified input file names in the *arc* directories, if any exist, in the mapsets in the user's current mapset search path.)

OPTIONS

Program parameters and the flag have the following meanings.

Flag:

-n Neatline. Vectors representing a box (neatline) around the input vector data will be inserted into the output GRASS vector file.

Parameters:

type=*name* Coverage type. Either polygon, or line.
Options: polygon, line

lines_in=*name* ARC/INFO ungenerate lines file; ungenerate format input file containing line or polygon coordinates.

points_in=*name* ARC/INFO ungenerate label-points file; ungenerate format input file containing label-point coordinates; only applies to 'polygon' type data.

text_in=*name* ARC/INFO ungenerate label-text file; ungenerate format input file containing category numbers and (optionally) attribute text.

vector_out=*name*	Resultant GRASS vector output file.
idcol=*value*	ID Number column in label-text file. Number of label-text column containing line-ID numbers.
catcol=*value*	GRASS category column in label-text file. Number of label-text column containing category values.
attcol=*value*	GRASS attribute column in label-text file. Number of label-text column containing attribute text.

This program can be run either non-interactively or interactively. The program will run non-interactively if the user specifies the (optional) flag setting and needed parameter values on the command line, using the form:

> **v.in.arc** [**-n**] **type**=*name* **lines_in**=*name* [**points_in**=*name*] [**text_in**=*name*]
> **vector_out**=*name* [**idcol**=*value*] [**catcol**=*value*] [**attcol**=*value*]

Alternately, the user can type:

> **v.in.arc**

on the command line without program arguments; in this case, the program will prompt the user for the flag setting and parameter values in the manner shown below.

In ARC/INFO, three files are used to store polygon data:
1) a *lines file*, which contains coordinates of all the area edge lines;
2) a *label-point file*, which contains coordinates of label-points (each of which has associated with it a unique label-point ID number). One label-point is associated with each polygon defined in the *lines file*;
3) a *label-text file*, which associates each label-point ID number with a category number and category ("attribute") text.

Linear feature data are stored in two files:
1) a *lines file*, which contains geographic coordinates defining lines, each with a line-ID number; and
2) a *label-text file*, which associates each line-ID number with a category number and attribute text.

These data files are described in further detail below, under the DATA FILE FORMATS section.

INTERACTIVE MODE

The program will prompt the user for the flag setting and parameter values if the user does not specify these on the command line. First, the user will be asked to assign a name to the vector file to store program output:

```
 VECTOR (DIGIT) FILENAME
Enter 'list' for a list of existing binary vector files
Hit RETURN to cancel request
>
```

Next, the user is asked to specify the COVERAGE (feature) type to be imported into GRASS. Valid coverage types are **polygon** and **line**.

```
 COVERAGE TYPE
Enter "polygon" or "line"
Hit RETURN to cancel request
>
```

IMPORTING A POLYGON COVERAGE

If the user chooses POLYGON coverage, he is asked if he wishes a neatline placed around his data. (The existence of neatlines in the output file can facilitate subsequent patching of data files.)

```
 NEATLINE
Do you want a neatline ?
Enter "yes" or "no"
>
```

If the user types **yes**, vectors that box the data will be inserted into the GRASS vector output file (*vector_out*); otherwise, no neatline will be inserted into the output file.

Next, the user is prompted for the name of an existing lines-file containing the geographic coordinates of the arcs forming polygon perimeters. The lines-file is created with the ARC/INFO *Ungenerate LINES* option, and is in the same format at the *prefix.pol* file created by the *v.out.arc* program. The user sees the following prompt:

```
 LINES FILENAME
Enter name of the file created with the LINES
option of the ARC/INFO Ungenerate command.
Hit RETURN to cancel request
>
```

The next prompt for coverage type "polygon" asks for the name of an existing label-points file. The label-points file is created with the *Ungenerate POINTS* option, and is in the same format as the *prefix.lab* file created by the *v.out.arc* program. The user sees the following prompt:

```
 LABEL-POINTS FILENAME
Enter name of file created with the POINTS
option of the ARC/INFO Ungenerate command.
Hit RETURN if there is no such file
>
```

Finally, the program asks the user for the name of an existing label-text file. This file associates each label-point ID number with a text string. It is in the same format as the *prefix.txt* file created by the *v.out.arc* program.

```
 LABEL-TEXT FILENAME
Enter the name of a file that associates
label-point ID numbers with text label strings
Hit RETURN if there is no such file
>
```

v.in.arc then scans the label-text file to find the numbers of lines and columns, the column headers (if any), and the first three lines of actual data in the file. It displays this information to standard output to help the user determine which columns will hold the ID, Category value, and Attribute text data in the new vector output file. A sample of the program's output is shown below:

```
The LABEL-TEXT file has been scanned. There are 132
 lines in the file and 8 columns in the file

Column headers of the LABEL-TEXT file:
   rec# AREA PERIMETER SOILS# SOILS-ID SOIL-CODE DRAIN_CODE TEXTUR-CODE
```

Here are the first three lines :

1	-2.30228E+07	19,399.848	1	0	0	0	0
2	81,079.875	1,678.826	2	1	15	3	3
3	955,952.500	10,229.637	3	2	19	8	8

The column of category values must contain only integer values. The attribute text column can contain a floating point number, an integer, or a word (text string).

Finally, the user is prompted to enter line ID, category value, and attribute text column numbers.

```
Enter the number of the column that should be used
for line IDs (probably the column with -ID) :

Enter the number of the column that is to be used
for GRASS category values:


Enter the number of the column that should be used
for GRASS attribute text:
```

Once these column numbers have been entered, *v.in.arc* will begin converting the ARC/INFO "Generate" format files into GRASS vector file format.

IMPORTING A LINE COVERAGE

The user will also be prompted for input when importing ARC/INFO files containing linear features ("line coverage") data. Like polygon data, linear features are constructed by the series of arcs (aka, vectors) defining their perimeters. If the user selects LINE coverage, the prompts seen by the user will be different in two respects from those for POLYGON coverage. First, the user will not be asked whether or not a neatline is desired; and second, no label-points file name will be requested. In other respects, the treatment of LINE coverage is identical to that for POLYGON coverage.

The user is prompted for the name of the lines-file containing the geographic coordinates of these arcs. The lines-file must first have been created with the ARC/INFO *Ungenerate LINES* option, and is in the same format as the *prefix.lin* file created by the GRASS *v.out.arc* program.

DATA FILE FORMATS

Following are examples of the data files discussed above.

LINES FILE, also known as *prefix.lin* or *prefix.pol* file.
This type of file can be created in ARC/INFO by using the *lines* subcommand of the *Ungenerate* command. Each line (aka, arc) is defined by a line-ID number, followed by a list of at least two easting and northing coordinate pairs, followed by a line with the word "END". The file is terminated with the word "END".

The line-ID number is important only for line coverage data. For a line coverage, the line-ID number is the number that associates each line with its attribute data.

```
        3
711916.000000    4651803.000000
711351.875000    4651786.000000
END
```

```
        3
709562.500000    4651731.000000
709617.250000    4651624.000000
709617.250000    4651567.000000
709585.000000    4651503.000000
709601.125000    4651470.000000
709696.875000    4651503.000000
709720.500000    4651574.000000
709823.750000    4651575.000000
709893.125000    4651741.000000
END
        3
710296.875000    4651491.000000
710295.125000    4651470.000000
710223.000000    4651454.000000
710154.500000    4651463.000000
END
END
```

LABEL-POINTS FILE, also known as *prefix.lab* file.
This type of file can be created in ARC/INFO using the *Points* option of the *Ungenerate* command.
The first number on each line is a label-point ID number, and the following two numbers are (respectively) the easting and northing coordinate pair representing the geographic location of the label-point.

```
1     711539.875000    4651743.000000
2     711429.000000    4650632.000000
3     711027.625000    4651736.000000
4     711022.625000    4651519.000000
5     710482.750000    4651494.000000
6     710474.500000    4651667.000000
7     709269.750000    4651018.000000
8     709726.500000    4651604.000000
9     708926.375000    4651195.000000
10    708567.500000    4651644.000000
11    708272.750000    4651407.000000
END
```

LABEL-TEXT FILE, also known as *prefix.txt* file.
The ARC/INFO *Display* command can be used to create this type of file.

```
1    -2.30228E+07   19,399.848    1   0    0   0
2      81,079.875    1,678.826    2   1   15   3
3     955,952.500   10,229.637    3   2   19   8
4      41,530.875      926.887    4   3   17   3
5      87,900.188    1,900.909    5   4   13   3
6     166,125.125    3,512.950    6   5   15   3
7      29,460.563      824.968    7   6   17   3
8     1022769.875    9,105.707    8   7   20   9
9      51,385.500    1,075.638    9   8   17   3
10    376,834.875    4,470.027   10   9    9   2
```

11	65,802.688	1,575.088	11	10	16	3

NOTES

ARC/INFO data can be imported even if a label-points and/or a label-text file are missing; however, the lines and/or areas imported will not be labeled.

v.in.arc can handle label-text files both with and without header lines.

SEE ALSO

v.out.arc, v.support

AUTHOR

Dave Johnson
DBA Systems, Inc.
10560 Arrowhead Drive
Fairfax, Virginia 22030

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.in.ascii – Converts ASCII vector map layers into binary vector map layers.
(GRASS Vector Data Import Program)

SYNOPSIS

v.in.ascii
v.in.ascii help
v.in.ascii input=*name* **output**=*name*

DESCRIPTION

v.in.ascii converts a vector map in ASCII format to a vector map in binary format. The user can run this program non-interactively by specifying all program options on the command line, in the form:

v.in.ascii input=*name* **output**=*name*

Parameters:

input=*name*	Name of an ASCII vector file to be converted to binary vector file.
output=*name*	Name given to binary vector output file.

If the user runs *v.in.ascii* without giving program arguments on the command line, the program will prompt the user for *input* and *output* file names.

NOTES

After running this program, GRASS support files must be built for the binary *output* file before the user can use the file in *v.digit*. The user can run *v.support* to create GRASS support files for the *output* file.

The GRASS program *v.out.ascii* performs the function of *v.in.ascii* in reverse; i.e., it converts vector files in binary format to ASCII format. These two companion programs are useful both for importing and exporting vector files between GRASS and other software, and for transferring data between machines.

The output from *v.in.ascii* will be placed into *$LOCATION/dig*.

SEE ALSO

v.digit, v.import, v.out.ascii, v.support

AUTHORS

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

NAME

v.in.dlg – Converts an ASCII USGS DLG-3 Optional file to a binary GRASS vector (dig) file.
(GRASS Vector Data Import Program)

SYNOPSIS

v.in.dlg
v.in.dlg help
v.in.dlg [**-l**] **input**=*name* **output**=*name* [**matt**=*name*] [**base**=*val*]

DESCRIPTION

This program converts an ASCII USGS DLG-3 (*dlg*) file into a binary GRASS vector (*dig*) file.
Warning: The program reads DLG-3 Optional format only.

v.in.dlg also creates a *dig_att* file containing the label information 'stripped' from the DLG-3 file (the first minor attribute for each record unless *matt* is specified).

If the *matt* is specified, *v.in.dlg* creates an additional attribute file containing identifiers for every record with correspon ding multiple attributes. In this case *matt* file contains identifiers starting with base *base* for the attributes stored in *matt* file (as opposed to the first minor attributes with no *matt* file). The example of *matt* with *base* = 34 would be:

34	0	0
35	180	201
36	180	208
	170	240
	190	201
37	160	220

With the corresponding *dig_att* looking like this:

```
A 648467.190000 4456367.320000    34
L 667989.290000 4458393.520000    35
L 651002.470000 4473793.390000    36
L 663816.680000 4471412.080000    37
```

However, the user must run *v.support* (or *v.import* option 4) on the *output* file created by *v.in.dlg* to create a *dig_plus* file containing the file topology, before using the *output* file in *v.digit*.

The user can avoid this two-step process by converting the ASCII DLG-3 file to binary GRASS vector format using option 1 of the GRASS program *v.import*.

–l Give precedence to line information (default is area).

Parameters:

input=*name* Name of USGS DLG-3 Optional format input file.

output=*name* Name to be assigned to the binary GRASS vector files created.

matt=*name* Name of file with multiple attributes (optional).

base=*val* Identifier base for multiple attributes (default is 1).

If the user simply types **v.in.dlg** without specifying parameter values on the command line, the program will prompt the user to enter these.

NOTES

Area vs Line Precedence:

Some machine-processed DLG-3 files do not make the distinction between line edges and area edges. For example, in a roads map, where the desired information is line edge data, a downtown block surrounded by roads may be processed as an area. Because of this, the user is asked to choose whether to give precedence to areas or lines. If precedence is given to lines, the user should be aware that any lines that bound unlabeled areas in the DLG-3 file will be stored as line data. Any unlabeled areas would therefore be lost (this is only a concern when areas are unlabeled; labeled area information will be retained). If precedence is given to areas, lines will be stored as boundaries to areas that are unlabeled.

Building support files with *v.support*:

When you run *v.support* you will have the option of snapping the nodes in your vector file that fall within a certain threshold of one another. WARNING: the default threshold is calculated using the scale of the original DLG-3 file. If the threshold is too high, excessive snapping may occur, destroying the file!! With *v.support*, the user has the option of snapping or not snapping nodes, and further, of setting a particular snapping threshold.

SEE ALSO

v.digit, v.import, v.support

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory
Irina Kosinovsky, U.S. Army Construction Engeneering Research Laboratory

NAME

v.in.dlg2 - Converts an ASCII or binary USGS DLG-3 (bdlg) file to a binary GRASS vector (dig) file.
(GRASS Vector Data Import Program)

SYNOPSIS

v.in.dlg2
v.in.dlg2 help
v.in.dlg2 [**-bl**] **input**=*name* **output**=*name*

DESCRIPTION

This program converts an ASCII or binary USGS DLG-3 (*dlg.old* or *bdlg.old*) file into a binary GRASS vector (*dig*) file.

v.in.dlg2 also creates a *dig_att* file containing the label information 'stripped' from the DLG-3 file. However, the user must run *v.support* (or *v.import* option 4) on the *output* file created by *v.in.dlg2* to create a *dig_plus* file containing the file topology, before using the *output* file in *v.digit*.

The user can avoid this two-step process by converting the ASCII or binary DLG-3 file to binary GRASS vector format using option 1 or 2 of the GRASS program *v.import*.

Flags:

−b	Input is a binary DLG-3 file (default is ASCII).
−l	Give precedence to line information (default is area).

Parameters:

input=*name*	Name of USGS DLG-3 Optional format input file.
output=*name*	Name to be assigned to the binary GRASS vector files created.

If the user simply types **v.in.dlg2** without specifying parameter values on the command line, the program will prompt the user to enter these.

NOTES

Area vs Line Precedence:

Some machine-processed DLG-3 files do not make the distinction between line edges and area edges. For example, in a roads map, where the desired information is line edge data, a downtown block surrounded by roads may be processed as an area. Because of this, the user is asked to choose whether to give precedence to areas or lines. If precedence is given to lines, the user should be aware that any lines that bound unlabeled areas in the DLG-3 file will be stored as line data. Any unlabeled areas would therefore be lost (this is only a concern when areas are unlabeled; labeled area information will be retained). If precedence is given to areas, lines will be stored as boundaries to areas that are unlabeled.

Building support files with *v.support*:

When you run *v.support* you will have the option of snapping the nodes in your vector file that fall within a certain threshold of one another. WARNING: the default threshold is calculated using the scale of the original DLG-3 file. If the threshold is too high, excessive snapping may occur, destroying the file!! With *v.support*, the user has the option of snapping or not snapping nodes, and further, of setting a particular snapping threshold.

SEE ALSO

v.digit, v.import, v.support

AUTHOR

Michael Higgins, U.S. Army Construction Engineering Research Laboratory
Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.in.dxf – Converts files in DXF format to ASCII or binary GRASS vector file format.
(GRASS Vector Data Import Program)

SYNOPSIS

v.in.dxf
v.in.dxf help
v.in.dxf [**−a**] **dxf**=*name* [**lines**=*name*[,*name*,...]] [**labels**=*name*[,*name*,...]] [**prefix**=*name*]

DESCRIPTION

The *v.in.dxf* data conversion program generates GRASS *dig*, *dig_ascii*, and *dig_att* files from a file in DXF format. Each layer in the DXF input file is converted to a separate *dig* (or *dig_ascii*) layer. For each DXF layer containing text, a *dig_att* file is also created. These output files are placed in the *dig*, *dig_ascii*, and *dig_att* directories under the user's current GRASS mapset.

Output from this program is designed to be used as input to the program *v.cadlabel*.

The *v.in.dxf* program will only recognize points, lines, polylines, and text in the DXF format, and will translate these to GRASS vector format; other types of data are ignored.

Flag:

-a Output an ASCII GRASS vector (*dig_ascii*) file rather than a binary GRASS vector (*dig*) file.

Parameters:

dxf=*name* Name of the DXF input design file to be converted to GRASS vector format.

lines=*name*[,*name,...*] or **lines**=*in_name:out_name*[*,in_name:out_name,...*]
Name(s) of layer(s) in the DXF input file containing line data, and (optionally) the name(s) to be assigned to the GRASS vector data (*dig* or *dig_ascii*) files output.
Default: Convert each layer containing data in the Idxf file to a GRASS vector data (*dig* or *dig_ascii*) file.

labels=*name*[*,name,...*] or **labels**=*in_name:out_name*[*,in_name:out_name,...*]
Name(s) of layer(s) in the DXF input file containing text labels, and (optionally) the name(s) to be assigned to the GRASS vector attribute (*dig_att*) files output.
Default: Convert each layer containing text labels in the *dxf* map to a GRASS vector attribute (*dig_att*) file.

prefix=*name* Prefix assigned to the *dig* or *dig_ascii* and *dig_att* output file names.

The names of the GRASS vector (*dig*, *dig_ascii*, and *dig_att*) files output are constructed as *prefix.extension*, where *prefix* is the *prefix* name specified by the user and *extension* is the number of the DXF layer from which the data were obtained. If the user does not specify a *prefix* name, the output files take their prefix from the prefix of the input DXF map layer. For example, for the DXF file named *streams.dxf* containing line data on layer 15, the GRASS vector map layer output would be named *streams.15*.

EXAMPLES

lines=15 Outputs line data in DXF layer 15.

lines=15,16 Outputs line data in DXF layers 15 and 16.

lines=ROADS,WATER
Converts line data in DXF layers ROADS and WATER.

lines=15:16 Outputs line data in DXF layer 15, and places it in the *dig* (or *dig_ascii*) file for DXF layer 16.

The below examples are given for a DXF design file named *cont.dxf* containing contour lines and contour line labels, in which:

> index contour lines are in DXF layer 9,
> intermediate contour lines are in DXF layer 11, and
> index labels and some intermediate contour lines are in DXF layer 12.

v.in.dxf can be run with default values, as shown below:

> **v.in.dxf dxf=cont.dxf**

Here, this is equivalent to running the following command:

> **v.in.dxf dxf=cont.dxf lines=9,11,12 labels=12**

Either of the above commands will produce three GRASS *dig* files (named *cont.9, cont.11,* and *cont.12*) and one *dig_att* file (named *cont.12*).

In our example, however, the *cont.12* file contains intermediate contour lines that the user would like to add to the *dig* file *cont.11*. Our user also wishes to use a different file prefix than the default prefix *cont*. The user therefore types the following command:

> **v.in.dxf dxf=cont.dxf lines=9,11,12:11 labels=12 prefix=contour**

The above command will generate three *dig* files (named *contour.9, contour.11, contour.12*), and will create one *dig_att* file containing text labels (called *contour.12*). No contour lines will appear in the *dig_att* file.

NOTES

Output Filenames:
The output filename, *prefix.extension*, conforms with the GRASS limit of 14 characters. The entire prefix name is used, a '.' inserted, and as much of the extension name is used as the 14 character limit will permit. Excess characters are truncated. To minimize the possibility of creating output files with the same names (resulting in loss of data from the DXF file), use the prefix option to abbreviate the DXF file name. This will leave the majority of characters available for differentiating between layer names.

Translation:
This data translation program does not contain any of the quality control functions available in *v.digit* that will prevent data in an improper format from being input to a GRASS data base. If present, DXF entities are placed in output file(s) corresponding to the layers on which they occurred in the DXF design file input.

Editing:
If the user asks *v.in.dxf* to output ASCII vector (*dig_ascii*) files, they must be converted to binary vector format before they are usable by most GRASS vector commands. The user can convert GRASS vector files from ASCII to binary format by running such programs as *v.support* or *v.in.ascii*. After conversion to binary format the vector files can be displayed (e.g., with *d.vect*); however, the user must run *v.support* on the binary vector files before they can be edited in *v.digit*. The files output by *v.in.dxf* will preserve the data in whatever form they exist in the DXF file. This means that output files may contain unsnapped nodes, overshoots, gaps, and replicated lines. The data, and the file header information (including the owner's name, map's name, date, and scale, and UTM zone) for the GRASS vector files output may require editing by the user in *v.digit*.

Attributes:
The *v.in.dxf* program attaches attributes only to DXF text data that are converted to GRASS vector data (such as contour line labels). Attributes are not attached to converted DXF line data. For each layer of text data in the DXF design file, *v.in.dxf* generates a vector file consisting of rectangular boxes (lines) that are drawn around the DXF text data, and uses the text values to create a GRASS attribute file for the boxes. The vector and attribute files can then be used to label contour lines with the *v.cadlabel* program.

SEE ALSO

v.cadlabel, v.digit, v.in.ascii, v.out.dxf, v.support

AUTHORS

Original *dxf2dig* program written by Jan Moorman, U.S. Army Construction Engineering Research Laboratory (6/89)

Revised by Dave Gerdes, U.S. Army Construction Engineering Reseach Laboratory (12/89)

Revised and appended by Jan Moorman, U.S. Army Construction Engineering Research Laboratory (7/90)

Code for arcs and circles from National Park Service GIS Division written by Tom Howard.

NAME

v.in.tig.basic – Create GRASS vector map from TIGER files
(GRASS Vector Program)

SYNOPSIS

v.in.tig.basic
v.in.tig.basic help
v.in.tig.basic [-pqt] t1=TIGER.1 t2=TIGER.2 out=name [zone=utm_zone] [spheroid=spheroid] [tlid=file]

PARAMETERS

t1 TIGER Type 1 path/file name
t2 TIGER Type 2 path/file name
out Name of vector map to create
zone UTM zone number; default is location zone
options: 1-60
default: (current zone)
spheroid Spheroid for LL to UTM conversion; see m.gc.ll
default: clark66
tlid Path/file for list of TLID numbers to process from TIGER.1 File
default: none

Flags:
-p Create new disk file for Type 2 pointers every time
-q Perform functions quietly
-t Build topology (dig_plus) file when done (can't be quiet)

DESCRIPTION

This program creates a GRASS vector map in the current mapset (UTM or Lat-Long locations only) with labelled line segments constructed from the end points (nodes) from the Type 1 TIGER file records and shape points from the Type 2 TIGER file. The nodes and shape points are matched by the TIGER record number (TLID field). In the attribute (label) file which is built (in the *dig_att* directory), the lines are given the "area boundary" type and line labels are the record numbers (9 digits!).

The Type 1 file may contain the records for a complete county, a subset for a county, or those from more than one county concatenated. The Type 2 file must contain all those records corresponding to the Type 1 records, but may contain extra records which will not be used. Typically, a subset of Type 1 records is used and a full county Type 2 file is used.

The program should function well independent of the termination character(s) at the end of each line in the input files. <LF>, <CR>, neither, one, or both are acceptable. Different distribution media apparently have different record delimiting characters.

The -p flag should not normally be used. It causes the program to build a disk file pointer table for all of the Type 2 records each time the program runs, rather than using the table built previously. Detection, naming and verification of the pointer file are automatic. It is created in the "tmp" space in the user's location.

THE tlid PARAMETER

The file specified by the *tlid* parameter may be of any size. Any number at the beginning of a line will be interpreted to be the TLID of a Type 1 record which is to be included in the output map. Other information on these lines is ignored, as are lines which do not begin with a number. If fewer than 5000 numbers are in this file, it is only read once and execution speed is greatly enhanced. Normally, a full Type 1 file will be specified by the *t1* parameter when a *tlid* file is used.

BUGS/RESTRICTIONS

Caution: This program will overwrite an existing vector map of the same name without warning.

The Type 2 input file must be on a device capable of "seeking," i.e., not a tape drive. CD-ROM is OK (but program will run much faster from disk files).

Input lines in a *tlid* file are limited to 300 characters in length. If the system can allocate space to store all the TLID numbers read from this file, it is only read once and execution speed is greatly enhanced; on a typical system, space for more than 10,000 numbers can be allocated.

SEE ALSO

v.apply.census will generate polygon labels for Census tracts, block groups, etc., from STF1 or PL94-171 files.
m.in.stf1.tape and/or unix text tools can be used to extract/concatenate useful subsets of TIGER Type 1 records prior to running **v.in.tig.basic**.

AUTHOR

Dr. James Hinthorne, GIS Laboratory, Central Washington University

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.in.tig.lndmk – Create GRASS vector map from TIGER files
(GRASS Vector Program)

SYNOPSIS

v.in.tig.lndmk
v.in.tig.lndmk help
v.in.tig.lndmk t1=TIGER.1
v.in.tig.lndmk [-is] t1=TIGER_1 [t2=TIGER_2] [t7=TIGER_7] [t8=TIGER_8] [tI=TIGER_I] [tP=TIGER_P] [input=file] [vect=name] [site=name] [zone=value] [spheroid=name]

PARAMETERS

t1 Path/name of Type 1 Tiger File
t2 Path/name of Type 2 Tiger File
t7 Path/name of Type 7 Tiger File
t8 Path/name of Type 8 Tiger File
tI Path/name of Type I Tiger File
tP Path/name of Type P Tiger File
input file path/name for input commands
vect Name of vector map to create
site Name of site map to create
zone UTM zone number; default is location zone
options: 1-60
default: 12
spheroid Spheroid for LL to UTM conversion; see m.gc.ll
default: clark66

Flags:
-i Force interactive mode
-s Input commands from stdin

Notes: **-i** flag implied if neither flag set and no **input** file specified. Omitted Tiger file names will be derived from the TIGER.1 name given by suitably changing the last character.

DESCRIPTION

This program creates a GRASS site or vector map of Census "Landmark" features in the current mapset (UTM or Lat-Long locations only) with labelled points, areas or lines constructed from the TIGER file records. Each point or area Landmark is a record in the Type 7 TIGER/Line file for a county.

The user may select the Landmark Features to include in the new map by specifying CFCC's (Census Feature Classification Code), or by specifying text string(s) to match with the Landmark Feature Name field. [Many Landmark records do not have an entry in the Feature Name field, so use the string matching selection with care.]

Point Landmarks are generated only from information in the Type 7 file. Area Landmarks, which often comprise several individually labelled polygons, are generated from data in TIGER file Types 7, 8, I, P, 1 and 2 (in order of use).

When making "line" maps, this program functions as a selection front end to the *v.in.tig.basic* program; see that programs documentation for the information on how these line maps will be labelled.

TIGER/Line files from multiple counties should not be mixed or concatenated when using this program.

The program should function well independent of the termination character(s) at the end of each line in the input files. <LF>, <CR>, neither, one, or both are acceptable. Different distribution media apparently have different record delimiting characters.

THE COMMAND MODE OF OPERATION

Starting *v.in.tig.lndmk* with the *-s* flag invokes the command mode of operation. Commands are expected from *stdin*, that is, the a terminal or a file redirected with '|' or '<'. When using a terminal for input, the prompt lists the available commands thus:
.p(oints) .a(reas) .l(ines) .c(ode)list
.s(trings) .m(atch)both .e(nd) .ex(it) >>

Syntax for the dot commands

.points [site_map]
  lines containing requested CFCC Codes
  (CFCC>> prompt will be given if input is from a terminal)
.end

.areas [vect_map]
  lines containing requested CFCC Codes
  (CFCC>> prompt will be given if input is from a terminal)
.end

.lines [vect_map]
  lines containing requested CFCC Codes
  (CFCC>> prompt will be given if input is from a terminal)
.end

.strings [output_file]
  1 to 10 lines of 30 chars to match with Landmark names;
  use '!' as first character to require exact case match;
  (*String #n:* prompt will be given if input is from a terminal)
.end

.matchboth [no]

.codes [P|A|L[=output_file]]

.exit

The details of the dot commands

Note: Always use the **.strings**, **.matchboth** and/or **.codes** commands prior to creating a map with **.points**, **.areas** or **.lines**.

.points mapname begins the process of making a GRASS4.1 site map of point locations. This site map name takes precedence over the site=name parameter that may be given on the command line.

Specification of the CFCC's, if any, for the features you want included in the map should be done on the line(s) following **.points** and should conclude with a line containing **.end**, which begins the map generation process. The CFCC's should be of the form Dnn (*e.g.,* D00; D10 D85 | D23, H12); several may be given on one line, separated by any common separator character. Requested CFCC's may be abbreviated to D1 (meaning D10 through D19) or just D (meaning D00 through D99).

If matching text strings are desired (see *.strings ƒR), they must be specified prior to the* **.points** command.

.areas mapname begins the process of making a GRASS4.1 vector map of polygons (areas). This vector map name takes precedence over the vect=name parameter that may be given on the command line. Specification of the CFCC's, if any, for the features you want included in the map should be done on the line(s) following **.areas** and should conclude with a line containing **.end**, which begins the map generation process.

If matching text strings are desired (see *.strings ƒR), they must be specified prior to the* **.areas** command.

.lines mapname begins the process of making a GRASS4.1 vector map of lines. This option uses only the Type 1 and 2 TIGER files. It searches the Type 1 records for the specified CFCC's and matches any specified text strings to the Feature Name (FENAME) field in the Type 1 records. Specification of the CFCC's, if any, for the features you want included in the map should be done on the line(s) following **.areas** and should conclude with a line containing **.end**, which begins the map generation process.

If matching text strings are desired (see *.strings ƒR), they must be specified prior to the* **.lines** command.

.codes searches the Type 7 Landmark file and sends to stdout, or to a file if one specified, the CFCC and the Classification description of each CFCC present. If a 'P' or 'A' is specified, both the point and area codes present are listed. 'L' lists the CFCC codes contained in the Type 1 file. The default type, if no type is specified, is 'P'. This command is provided as a useful aid to the user.

.strings records up to 10 following lines (until ".end") as text strings to match to the Landmark Feature Name Field in the Type 7 records (Type 1 for making ".line" maps). A match is made if any of the specfied strings is a *substring* of the Feature Name. If exact case matching is required, '!' should be the first character of the entered text strings. Example: The Feature Name "Walter Reed Army Hospital" would be matched by any of the following strings.

```
!Walt
hosp
Reed
army hosp
```

If the optional filename parameter is given, Type 7 records which match the specified strings are reported to that file, otherwise they are reported to stdout.

.matchboth is used to specify that a Landmark record (or Type 1 record for ".lines") must match both one of the specified CFCC's *and* one of the specified text strings to be selected. A match to *either* condition is the default mode, and can be explicitly set or reset by the optional "no" parameter; *i.e.,* **.matchboth no**.

.exit is used to exit from the program.

THE INTERACTIVE MODE OF OPERATION

The following "VASK" menus will be encountered in using the interactive mode of this program. Comments following each menu will help you decide how to answer the questions and create products.

```
GRASS IMPORT FROM CENSUS LANDMARK FEATURE RECORDS

STEP 1: Select Type(s) of features to be extracted

There are Point and Area (polygon) Landmark feature types.
```

```
  in the Landmark (Type 7) Records

 There may also be some Line (vector) Landmark features
  in the Basic Data (Type 1) Records

 Mark one feature type you wish extracted:

          Point   x
          Area    _
          Line    _

 View CFCC Codes for selected type  x

 or  Exit program  _

    AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
            (OR <Ctrl-C> TO CANCEL)
```

Comments: This is the "Main Menu." You must select (by entering 'x') **one** of the Point, Area or Line choices. The viewing of the CFCC codes is optional, but is usually done initially.

The following screen results from a request to View CFCC Codes after selecting Point features:

```
TIGER Type 7 file <tgr53037.f47>
examined.  15 unique CFCC codes found in 123 records.

Mark 'x' to see list on screen  _

Enter file name (home directory) to save list to file  _______________


    AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
            (OR <Ctrl-C> TO CANCEL)
```

Comments: You may request to see the list on the screen, or have the list sent to a file in your home directory, or both. If you request neither, the program returns to the Main Menu. Both point and area landmarks are reported if either "Point" or "Area" was selected on the Main Menu. If "Line" was requested on the Main Menu, all the CFCC codes in the Type 1 (Basic Data Record) file will be tabulated (this can be a useful product for other purposes); remember that the Type 1 file is large and this processing may take a few minutes or more.

The following is typical output of CFCC Codes:

```
TIGER Type 7 file <tgr53037.f47>
Number of Each Area Landmark CFCC
   6 D00 Landmark Feature, Class Unknown or Not Elsewhere Classified
   3 D10 Military installation
   1 D28 Campground
   1 D31 Hospital
   5 D43 Educational institution
   1 D81 Golf course
   7 D82 Cemetery
  14 D85 State or local park or forest
   7 H00 Water Feature, Class Unknown or Not Elsewhere Classified
```

```
    8 H11 Perennial stream
    9 H30 Lake or pond
   32 H31 Perennial lake or pond
    1 H32 Intermittent lake or pond
Total Area Records: 95

Number of Each Point Landmark CFCC
    1 D00 Landmark Feature, Class Unknown or Not Elsewhere Classified
    5 D28 Campground
    1 D31 Hospital
   10 D43 Educational institution
    4 D44 Religious institution
    5 D71 Lookout tower
    1 D81 Golf course
    1 D82 Cemetery
Total Point Records: 28
```

After reviewing the list of available CFCC Codes, select "Point, Area or Line" again on the Main Menu and something like the following two menus will appear. On the first one, select any number of CFCC Codes (categories) that you desire to be included in a new map.

```
 LANDMARK FEATURE CATEGORY SELECTION MENU: POINT LANDMARKS

D00 _ Landmark Feature, Class Unknown or Not Elsewhere Classified
D28 x Campground
D31 _ Hospital
D43 _ Educational institution
D44 _ Religious institution
D71 _ Lookout tower
D81 _ Golf course
D82 _ Cemetery

Done: _  Start over: _

     AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
                (OR <Ctrl-C> TO CANCEL)
```

Comments: In this case "Point" was selected on the Main Menu, and the user has indicated that a GRASS site map of Campgrounds is to be made.

Now you have the opportunity to enter character strings which will be matched against the Feature Name field (in the Type 7 record in the case of "Point" or "Area" maps, and in the Type 1 record for "Line" maps).

```
  Step 1C  Enter text(s) to match in finding Landmark Features.
        (Use ! as first char to require exact case match.)

             ______________________________
             ______________________________
             ______________________________
             ______________________________
             ______________________________
             ______________________________
             ______________________________
```

____
____
____

Check for any Landmark Records that match your
text strings before proceeding to next step? _

File to save matches to (omit for screen display):
____

AFTER COMPLETING ALL ANSWERS, HIT <ESC> TO CONTINUE
(OR <Ctrl-C> TO ERASE AND START OVER)

Comments: See *.strings* discussion in the command line section of this manual page for examples of what can be entered on the lines. If no entries are made then no string matching to the Feature Name field will be done.

BUGS/RESTRICTIONS

Some area features are included in the Type 7 (Landmark) file which do not fit the definition given of "Landmark" (CFCC's beginning with 'D'), as a mechanism to have them included in the Tiger/Line data. Many Hydrography area features fall in this category for some counties. Also, there may be records in both the Type 1 and Type 7 files for some Landmark features, such as airports, and to make a complete airport map for a county both sets must be extracted.

In the first trials of this program is was discovered that the TIGER/Line files have several defects in topological completeness when area landmark polygons are created. These defects express themselves when v.support is run (automatically) to attach labels to the polygons and is unable to attach one or more label points. Some editing with v.digit should be able to resolve these problems.

Caution: This program will overwrite an existing vector map of the same name without warning. Site_list maps will not be overwritten.

Since this program opens many TIGER files at once, the set for a county must reside on a disk or CD-ROM, not tape. Only file types 1, 2, 7, 8, I and P are used so only those would have to be copied from tape to disk.

SEE ALSO

v.in.tig.basic is called automatically to produce the area and line maps.
v.apply.census will generate polygon labels for Census tracts, block groups, etc., from STF1 or PL94-171 files.

AUTHOR

Dr. James Hinthorne, GIS Laboratory, Central Washington University, 1992.

APPENDIX

List of Census CFCC Landmark 'D' Codes

[Extracted from file CODESTIG.ASC on TIGER distribution CD-ROM.]

CFCC CLASSIFICATION D = LANDMARK FEATURES

D00 Landmark Feature, Classification Unknown or Not Elsewhere Classified

D10 Military installation

D20 Multihousehold and transient quarters
D21 Apartment building or complex
D22 Rooming or boarding house
D23 Trailer court or mobile home park
D24 Marina
D25 Crew of vessel
D26 Housing facility for workers
D27 Hotel, motel, resort, spa, YMCA, or YWCA
D28 Campground
D29 Shelter or mission

D30 Custodial facility
D31 Hospital
D32 Halfway house
D33 Nursing home, retirement home, or home for the aged
D34 County home or poor farm
D35 Orphanage
D36 Jail or detention center
D37 Federal penitentiary, state prison, or prison farm

D40 Educational or religious institution
D41 Sorority or fraternity
D42 Convent or monastery
D43 Educational institution
D44 Religious institution

D50 Transportation terminal
D51 Airport or airfield
D52 Train station
D53 Bus terminal
D54 Marine terminal
D55 Seaplane anchorage

D60 Employment center
D61 Shopping center or major retail center
D62 Industrial building or industrial park
D63 Office building or office park
D64 Amusement center
D65 Government center
D66 Other employment center

D70 Tower

D71	Lookout tower
D80	Open space
D81	Golf course
D82	Cemetery
D83	National park or forest
D84	Other federal land
D85	State or local park or forest
D90	Special purpose landmark
D91	Post office box ZIP code

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.in.tig.rim – Imports Census Bureau line data (TIGER files) to GRASS vector format.
(GRASS Vector Data Import Program)

SYNOPSIS

v.in.tig.rim
v.in.tig.rim help
v.in.tig.rim dbname=*name* **in1file**=*name* **in2file**=*name* **zone**=*value*

DESCRIPTION

v.in.tig.rim imports Census line data (called TIGER) and creates a "master" binary vector file containing a large amount of data. Various map layers can then be created by querying information from the master vector file using *v.db.rim* or one of the *Gen.* shell scripts listed in the SEE ALSO section, below. The database name (*dbname*) given on the command line will be the name of the rim data base, and the master vector file in GRASS will be named "dbname.Master". The master vector file will include all information from the type1 and type2 TIGER files given on the command line as *in1file* and *in2file*. If the user simply types *v.in.tig.rim* on the command line, all parameters will be queried using the standard GRASS parser described in the manual entry for *parser*.

COMMAND LINE OPTIONS

Parameters:

dbname=*name*	Vector/rim data name (with a maximum of 7 characters).
in1file=*name*	TIGER type1 input file name.
in2file=*name*	TIGER type2 input file name.
zone=*value*	Universal Transverse Mercator (UTM) zone in which these data are located. Options: -60 - 60

NOTES

TIGER data are presented in latitude/longitude format, and are converted to UTM (Universal Transverse Mercator) coordinates as part of this importing routine. The spheroid used in the conversion is clark 66, as it is the most consistent with the original data.

This command must be compiled separately, and requires the use of *rim* and *v.db.rim* which contain FORTRAN code. The user must have access to a FORTRAN compiler in order to compile and use this command, since it calls both *rim* and *v.db.rim*.

If the user does not know the UTM zone for this data input file, the command *m.tiger.region* should be run first to determine the zone.

v.support must be run separately on the output file if needed.

FILES

Source code for RIM is located under $GISBASE/../src.related/rim

Source code for *v.db.rim* is located under $GISBASE/../src.garden/grass.rim/v.db.rim

Source code for *v.in.tig.rim* is located under $GISBASE/../src.garden/grass.tig.rim/v.in.tig.rim

Source code for *m.tiger.region* is located under $GISBASE/../src.alpha/misc/m.tiger.region

SEE ALSO

tig.rim.sh, m.tiger.region, v.db.rim, tiger.info.sh

AUTHOR

Jim Hinthorne and David Satnik, GIS Lab, Central Washington University, Ellensburg, WA.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of

GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.in.transects – import transect data to a GRASS vector map
(GRASS Vector Program)

SYNOPSIS

v.in.transects
v.in.transects help
v.in.transects [**-a**] **input**=*name* [**fs**=*character*"] **length**=*value* [**units**=*name*] [**decl**=*name*] **vect**=*name* [**title**=*name*"]

DESCRIPTION

v.in.transects imports "transects" into a GRASS vector map. A transect is a line (or an area) which is described by a starting coordinate, a direction or azimuth, (or backward, forward, rightside, and leftside distances from the line transect), and a length (in meters or feet). The information describing the transects must be in a file prepared by the user before running **v.in.transects.** The format of this file is described below in the section **TRANSECT FILE FORMAT .**

COMMAND LINE OPTIONS

Flag:

-a Run for area transects. Default is line transects, i.e., this flag is not used.

Parameters:

input transect information file

This is the file containing the transect information to be imported. The format of this file is described below in the section **TRANSECT FILE FORMAT**

fs input data field separator

The transect file is organized one transect per line with at least 4 fields. The *fs* option specifies the character that is used in the transect information file to separate the fields. If not specified, fields are assumed to be separated by blanks (white space). (Quote the delimeter whenever you are not sure whether it has special meaning in UNIX or not, to avoid the misunderstanding by GRASS.)

length transect length

This is the length of the transects (default is in grid units, e.g., meters). It is assumed that all transects have the same length. If they do not, then more than one import process must be run to create two or more vector files and the results patched together (using v.patch).

units units of the length
default: meter

This is the unit of the transect. It can be meter or foot.

decl declination - angle (in degrees) to be added to input azimuth angles
default: 0

Each transect has a direction or azimuth angle associated with it. The map projection may have a declination associated with it and if the azimuth angles embedded in the transect input file do not account for this declination, it may be specified here.

vect Vector map to be created

title Title for resultant vector map
default: Transect map

If title is more than one word, it should always be quoted.

TRANSECT FILE FORMAT

The format for the transect file consists of one record or line per transect with 4 mandatory fields and a 5th optional field for line transects. The first field is the GRASS category number to be assigned to the transect. The second and third fields are the easting and northing (respectively) of the starting coordinate for the transect. The fourth field is the azimuth in degrees clockwise from north of the transect. Following the fourth field is an optional fifth field that is the category label for the transect. The following is a simple example with 3 transects:

```
1   709818   5453991   246.0   P CLGR 4 73 1 21 0 0 0 0 1
2   698350   5464162   128.0   P CLGR 0 55 0 36 0 0 0 0 1
3   704615   5461874   190.5   P DEGR 0 34 4 15 0 0 0 0 0
```

Note that the fifth field (i.e., the label) is really everything after the azimuth, not just the P.

This file could be formatted as follows:

```
1:709818:5453991:246.0:P:CLGR:4:73:1:21:0:0:0:0:1
2:698350:5464162:128.0:P:CLGR:0:55:0:36:0:0:0:0:1
3:704615:5461874:190.5:P:DEGR:0:34:4:15:0:0:0:0:0
```

In this case the fields are separated by a colon so *fs=:* must be specified on the command line. The labels (starting with the P) would retain the colons (i.e., they are not removed from the label even though they act to define the first 4 fields).

When area transects are required, the transects file should include four (4) more fields for backward, forward, rightside, and leftside distance from the corresponding line transect. The format will be as the following:

```
1   709818   5453991   246.0   10.0   15.0   20.0    5.0   P CLGR 4 73 1 21 0 0 0 0 1
2   698350   5464162   128.0    5.0   15.0   10.0   25.0   P CLGR 0 55 0 36 0 0 0 0 1
3   704615   5461874   190.5   15.0   20.0   10.0    5.0   P DEGR 0 34 4 15 0 0 0 0 0
```

Note that delimiter could be other than white space in transect file. Vector files of line transects can be generated from the above area transects file by not using *-a* flag. However, the label will include four (4) more items.

NOTES

The resulting vector map is a complete GRASS vector map: it will have a category file with the labels from the input file and it will have the topology file already built. (v.support is run automatically by **v.in.transects** as the final step in creating the GRASS vector map.)

AUTHORS

Tao Wen, University of Illinois at Urbana-Champaign, Illinois.
Michael Shapiro, U.S. Army Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.mkgrid – Creates a (binary) GRASS vector map of a user-defined grid.
(GRASS Vector Program)

SYNOPSIS

v.mkgrid
v.mkgrid help
v.mkgrid map=*name* **grid**=*rows,columns* **coordinate**=*x,y* **box**=*length,width*

DESCRIPTION

v.mkgrid will create a binary format, vector map representation of a regular coordinate grid.

Parameters:

map=*name*	Name to be assigned to binary vector map layer output.
grid=*rows,columns*	Number of ROWS and COLUMNS to appear in grid.
coordinate=*x,y*	Lower left EASTING and NORTHING coordinates of vector map layer output.
box=*length,width*	LENGTH and WIDTH of boxes in grid.

If the user simply types **v.mkgrid** on the command line without specifying parameter values, the program will prompt the user for inputs using the standard interface described in the manual entry for *parser*.

NOTES

The new binary vector map output is placed under the *dig* directory in the user's current mapset, and can be used like any vector map layer. Run *v.support* to build the topology information for the vector map before using *v.mkgrid* map layer outputs in the *v.digit* program.

Since the grid is computer-generated, the corner coordinates will be exact and can be used when patching together several *v.mkgrid* grids.

This is NOT to be used to generate a vector map of USGS quadrangles, because USGS quads are not exact rectangles. To generate a vector map of USGS quads, use the program *v.mkquads*.

The program ignores the current GRASS geographic region settings. It will create the complete grid even if part of this grid falls outside the current geographic region.

SEE ALSO

v.digit, v.mkquads, v.patch, v.support and *parser*

AUTHOR

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

NAME

v.mkquads − Creates a GRASS vector map layer and/or sites list and/or geographic region definition file for a USGS 7.5-minute quadrangle.
(GRASS Vector Program)

SYNOPSIS

v.mkquads
v.mkquads help
v.mkquads [**−esrvx**] **map**=*name*

DESCRIPTION

There are three types of output available from the GRASS program *v.mkquads* :

(1) a vector map of all the full USGS quadrangles that will fit within the boundaries of the current geographic region.

(2) a GRASS sites list containing the corner coordinates of each of these quads.

(3) GRASS geographic region definition files associated with each of the quads created.

A *quad* is defined as the area covered by a USGS 7.5-minute (1:24,000) map. This program is useful for managing a GRASS data base LOCATION which contains a number of quads which are to be patched together.

Flags:

−e Encompass current geographic region with quads (rather than only creating those quads that lie inside of the geographic region). Use of this option will affect all output options.

−s Create a GRASS sites list file. The sites list will contain all the corner coordinates of all the full quads that can be built in the current geographic region. The sites list file can then be displayed using the *d.sites* program.

−r Create region file(s): quad.1 quad.2 ... The program will generate a separate geographic region definition file for each quad; each of the geographic region files created will have the prefix *quad.* with some number attached to it. For example, if only one quad were created, the geographic region file *quad.1* would also be created in the *windows* directory under the user's current mapset. To make the program-generated geographic region definition file *quad.1* your *current* geographic region setting, run the GRASS *g.region* program.

−v Create vector file (default). Only full quads will be created. The binary vector map layer output is placed under the user's *dig* directory and can be used like any other vector map layer. Run *v.support* to build the topology information for the vector map before using *v.mkquads* map layer outputs in the *v.digit* program. Since the quads are computer-generated, the corner coordinates will be exact. This simplifies digitizing if one or more quad sheets will have to be brought together for a data base, because all of the quad corner points to be joined will be guaranteed to match.

−x Create a GRASS registration (*reg*) file.

Parameter:

map=*name* The name of a file to contain program output.

If the user runs *v.mkquads* without including program parameter value and desired flags on the command line, the program will prompt the user for the above information using the standard GRASS interface described in the manual entry for *parser*.

NOTES

All output options can be used on the command line at the same time. A listing of all the quad points in latitude/longitude and UTM coordinates will be displayed each time the program is executed. The spheroid being used for the lat/lon to UTM conversions is *clark66*.

BUGS

Currently, this program only works for GRASS locations in a Universal Transverse Mercator (UTM) coordinate system (in meters). There are no guarantees that *v.mkquads* will function properly if a quadrangle crosses UTM zones. This program has not been tested outside the northwest UTM quadrant.

SEE ALSO

d.sites, g.region, v.digit, v.mkgrid, v.support and *parser*

AUTHORS

Michael Higgins, U.S. Army Construction Engineering Research Laboratory
Marilyn Ruiz, U.S. Army Construction Engineering Research Laboratory

NAME

v.out.arc **–** Converts GRASS vector files to ARC/INFO's "Generate" file format.
(GRASS Vector Data Export Program)

SYNOPSIS

v.out.arc
v.out.arc help
v.out.arc type=*name* **vect**=*name* **arc_prefix**=*name*

DESCRIPTION

v.out.arc is a GRASS data export program that converts files in GRASS vector format to ARC/INFO's "Generate" file format. The companion program *v.in.arc* imports data in ARC/INFO's "Generate" format and converts them to GRASS vector format.

This program can be run either non-interactively or interactively. The program will be run non-interactively if the user specifies parameter values on the command line using the following format:

v.out.arc type=*name* **vect**=*name* **arc_prefix**=*name*

Alternately, the user can simply type:

v.out.arc

on the command line; in this case, the program will prompt the user for parameter values.

Parameters:

type=*name*	Coverage (feature) type. Options: polygon, line
vect=*name*	The name of a GRASS vector file to be converted to ARC/INFO format.
arc_prefix=*name*	A prefix to be assigned to the ARC/INFO-format files output by *v.out.arc*.

INTERACTIVE MODE: USER PROMPTS

v.out.arc will prompt the user to enter the name of a GRASS vector file to be exported to ARC/INFO and for a filename prefix to be used in naming the files created by the program.

A GRASS vector file to be exported to ARC/INFO must either contain only linear features (i.e., have only line coverage) or contain only area edge features (i.e., have only polygon coverage). *v.out.arc* will begin by asking the user which type of coverage (line or polygon) is to be imported:

```
 COVERAGE TYPE
Enter "polygon" or "line"
Hit RETURN to cancel request
>
```

The program then prompts the user for the name of the GRASS vector file to be converted to ARC/INFO format:

```
 VECTOR (DIGIT) FILENAME
Enter 'list' for a list of existing binary vector files
Hit RETURN to cancel request
>
```

Next, the user is asked for a file-name prefix to be used in naming the output ARC/INFO Generate format files:

```
 ARC/INFO (GENERATE) FILENAME PREFIX
Hit RETURN to cancel request
>
```

The filename prefix will be used to name the various files that will be created for export to ARC/INFO. When labeled polygon coverage data are exported, three such files will be created: a *lines file* with the suffix .lin, a *label-points file* with the suffix .lab, and a *label-text file* with the suffix .txt. When line coverage data are exported, two such files will be created: a *lines file* with the suffix .lin, and a *label-text file* with the suffix .txt. Export of unlabeled polygon or line coverage data will result in creation of a *lines file* (.lin suffix) only. See the DATA FILE FORMATS section for more information on these files.

EXAMPLE

Linear features and polygon data are made up of the series of arcs (aka, vectors) outlining their perimeters. The *ARC/INFO Users' Guide*, in its discussion of the *Ungenerate* command, explains how line and polygon coverage data can be created from files (like *prefix.lin* and *prefix.pol*) containing the geographic coordinates of these arcs, and from files (like *prefix.lab*) containing the geographic coordinates of label-points. Below is an example which illustrates the creation, within ARC/INFO, of a polygon coverage data file (named *soils*) from the files *soils.pol* and *soils.lab*.

```
Arc: GENERATE SOILS
Generate: INPUT soils.pol
Generate: LINES
Generating lines ...
Generate: INPUT soils.lab
Generate: POINTS
Generating points ...
Generate: QUIT
Arc: _
```

The above example would create a polygon coverage data file named *soils* with label-points. The label-points would have ID numbers that correspond to the GRASS category values for the polygons in the coverage. The INFO portion of ARC/INFO can be used to associate these label-point ID numbers with descriptive text from the *soils.txt* file.

DATA FILE FORMATS

LINES FILE, also know as *prefix.lin* or *prefix.pol* file:
This text file is a "Generate" format lines file. The *lines* option of the ARC/INFO *Generate* command can be used to read this file into ARC/INFO. Each line in the file has a unique line-ID number.

```
101
223343.62 218923.15
223343.62 222271.06
259565.31 222271.06
259565.31 195577.37
END
102
237862.53 203392.37
244970.75 203744.28
253137.66 195577.37
259565.31 195577.37
END
103
```

```
237862.53 203392.37
237862.53 203744.28
223343.62 218392.37
END
104
239072.44 186200.56
237862.53 187410.50
237862.53 203392.37
END
END
```

LABEL-POINTS FILE, also known as *prefix.lab* file:
This text file will be created by *v.out.arc* if the vector file being exported represents a polygon coverage. *prefix.lab* consists of a list of label-point (x,y) coordinates, each with a unique label-point ID number.

```
1 242777.81 211533.09
2 243458.37 199282.28
3 243458.37 195199.28
```

LABEL-TEXT FILE, also known as *prefix.txt* file:
In the case of polygon coverage data, this file associates an integer category value and a category label ("attribute") text string (containing no spaces) with each label-point ID number. In the case of line coverage data, this file associates an integer category value and an attribute text string with each line-ID number.

The first column is the row number (which is arbitrary), the second column contains the category value, the third column holds the line or label-point ID number, and the fourth column contains the attribute text string.

```
1 4 1 Coniferous
2 5 2 Deciduous
3 2 3 Rangeland
```

SEE ALSO

v.in.arc, v.support

AUTHOR

Dave Johnson
DBA Systems, Inc.
10560 Arrowhead Drive
Fairfax, Virginia 22030

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.out.ascii – Converts a binary GRASS vector map layer into an ASCII GRASS vector map layer.
(GRASS Vector Data Export Program)

SYNOPSIS

v.out.ascii
v.out.ascii help
v.out.ascii input=*name* **output**=*name*

DESCRIPTION

v.out.ascii converts a GRASS vector map in binary format to a GRASS vector map in ASCII format.

The program can be run non-interactively if the user specifies all needed program arguments on the command line, in the form

v.out.ascii input=*name* **output**=*name*

Parameters:

input=*name*	Name of the binary GRASS vector input file to be converted to ASCII format.
output=*name*	Name of the ASCII GRASS vector output file.

If the user runs *v.out.ascii* without giving the names of an input and output file on the command line, the program will prompt the user for these names.

NOTES

The GRASS program *v.in.ascii* performs the function of *v.out.ascii* in reverse; i.e., it converts vector files in ASCII format to their binary format. These two companion programs are useful both for importing and exporting vector files between GRASS and other software, and for transferring data between machines.

The output from *v.out.ascii* will be placed in the user's current mapset under the *$LOCATION/dig_ascii* directory.

v.out.ascii does not copy the *dig_cats* file associated with the binary vector *input* map to the new *output* file name. The user must copy the *dig_cats* file to the new *output* name if this is desired (e.g., using the UNIX *cp* command).

SEE ALSO

v.digit, v.import, v.in.ascii, v.support

AUTHORS

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

NAME

v.out.dlg – Converts binary GRASS vector data to DLG-3 Optional vector data format.
(GRASS Vector Data Export Program)

SYNOPSIS

v.out.dlg
v.out.dlg help
v.out.dlg input=*name* **output**=*name*

DESCRIPTION

The GRASS program *v.out.dlg* allows the user to convert GRASS vector data to DLG-3 Optional format, for export to other systems.

The user can run the program non-interactively by specifying all program arguments on the command line, in the form:

v.out.dlg input=*name* **output**=*name*

Parameters:

input=*name* Name of the binary GRASS vector data file to be converted to DLG-3 format.

output=*name* Name to be assigned to the DLG-3 Optional format output file created.

If the user does not give the names of an input and output file on the command line, the program will prompt the user to enter these names.

NOTES

The *v.out.dlg* program requires that the *input* vector map layer have full topological information associated with it. This means that the GRASS program *v.support* should have been the last program to have effected any changes upon the vector map layer before it is run through *v.out.dlg*. If this is not the case, *v.out.dlg* will terminate with a message that *v.support* needs to be run.

The output from *v.out.dlg* will be placed in *$LOCATION/dlg*.

SEE ALSO

v.import, v.in.ascii, v.in.dlg, v.support

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.out.dxf - GRASS vector format to DXF format conversion program.
(GRASS Vector Data Export Program)

SYNOPSIS

v.out.dxf
v.out.dxf help
v.out.dxf input=*name* **output**=*name*

DESCRIPTION

The GRASS program *v.out.dxf* conversion program generates an ASCII DXF (AutoCad) file from a GRASS vector ASCII file. The output file is placed in the user's current working directory unless the user specifies a full pathname for the *output*.

Parameters:

input=*name* The name of an existing GRASS vector ASCII file.

output=*name* Name to be assigned to the DXF output file.
NOTE: DXF files output by AutoCad have the suffix .dxf

NOTES

This program does not currently read in binary files.

SEE ALSO

v.cadlabel, v.digit, v.in.ascii, v.in.dxf, v.support

AUTHOR

Chuck Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory, wrote original *v.out.dxf* program in 4/89.

NAME

v.out.moss – Converts GRASS site, line, or area data into MOSS import format.
(GRASS Vector Data Export Program)

SYNOPSIS

v.out.moss

DESCRIPTION

This program produces a MOSS import file containing GRASS site, line, or area features that have been converted into MOSS point, line, or polygon features, respectively. Only one type of data (site, line, or area) can be converted at a time. Site data can be extracted from GRASS *site lists* files or *vector* files, at the user's discretion. The resultant MOSS files will be created in the *moss* subdirectory of the current mapset.

The user will be prompted for the GRASS data type, the name of the *site lists* or *vector* file to be converted, and the name of the the MOSS import file to be created. If line data is being converted, the user will be asked whether area edges should be processed as lines. If so, the line features produced from the area edges will not have attributes associated with them.

Upon successful completion of the data export, the number of items converted will be printed, and the user will be given the option of processing another GRASS file.

NOTES

Vector data cannot be exported until the necessary GRASS support files have been built.

This program is interactive and requires no command line arguments.

SEE ALSO

v.support

AUTHOR

Chris Emmerich, Autometric, Inc.

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.patch **–** Creates a new binary vector map layer by combining other binary vector map layers.
(GRASS Vector Program)

SYNOPSIS

v.patch
v.patch help
v.patch input=*name*[,*name*,...] **output**=*name*

DESCRIPTION

v.patch allows the user to combine any number of vector map layers together to create one composite vector map layer.

Parameters:

input=*name,name, ...*	Name(s) of input vector map(s) to be patched together.
output=*name*	Name assigned to composite "patched" vector output map.

The program will be run non-interactively if the user specifies the names of the vector map(s) to be patched and the name of an output file to store the resulting composite patched vector map on the command line, in the form:

v.patch input=*name*[,*name*,...] **output**=*name*

Alternately, if the user runs *v.patch* without specifying input and output file names on the command line (by typing simply **v.patch**), the program will prompt the user for inputs using the standard GRASS interface described in the manual entry for *parser*.

NOTES

The vector map layers to be patched together must exist in the user's current mapset search path, and the composite vector map layer name given must not already exist in the user's current mapset.

After running *v.patch*, the header file will contain only information taken from the first *input* file name given in the string **input**=*name,name, ...*, with the exception of the geographic region's edge information, and the scale and threshold information. (The user's current geographic region settings are ignored; this information is instead extracted from the vector file headers.) In the new composite vector map layer, the boundaries of the geographic region will be expanded to encompass all of the geographic area included in the map layers being patched, and the scale will be set equal to the smallest (i.e., most gross) scale used by any of the patched map layers (this will affect default node-snapping thresholds). The map threshold is calculated automatically from the map scales given in the file headers, and (currently) is not used directly. The composite vector map layer's header will probably need to be edited; this can be done from within the GRASS program *v.digit*.

The GRASS programs *v.mkgrid* and *v.mkquads* can be used to ensure that the borders of the maps to be patched together align neatly.

Any vectors that are duplicated among the maps being patched together (e.g., border lines) will have to be edited or removed after *v.patch* is run. Such editing can be done using *v.digit*.

After running *v.patch* the user must run *v.support* on the composite vector map layer in order to create a *dig_plus* (topology) file for it. At this time, you can request that a *very* small snapping threshold be used, to cause the nodes that match up across vector map layers to snap together without affecting the integrity of the remainder of the vector map layer.

BUGS

The *dig_cats* and *reg* file information for the maps being patched together is not copied to the composite, patched map layer. The user should therefore run *v.support* on the output file produced by this program.

SEE ALSO

v.digit, v.in.ascii, v.mkgrid, v.mkquads, v.support, and *parser*

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.proj – Allows projection conversion of vector files.

SYNOPSIS

v.proj
v.proj help
v.proj map=*name [***out**=*name*] **inloc**=*name* [**dbase**=*name*] [**set**=*name*]

DESCRIPTION

allows a user to convert a vector map in a specified mapset of a specified location (different from current) with projection of input location to the vector map in a current mapset of current location with projection of current location (both projections are defined by corresponding PROJ_INFO files).

v.proj will create *dig*, *dig_att*, and *dig_cats* directories in the output mapset, if they do NOT exist. Map files for *dig*, *dig_att*, and *dig_cats* are also created for the new map layer.

Parameters:

map=*name*	Name of the input vector file.
out=*name*	Name of the output vector file.
inloc=*name*	Name of the location containing input vector file
dbase=*name*	Name of the database containing input location
set=*name*	Name of the mapset containing input vector file

If the user simply types **v.proj** without specifying parameter values on the command line, the program will prompt the user to enter these.

NOTES

If **out** is not specified it is set to be the same as input map name.
If **dbase** is not specified it is assumed to be the current database.
If **set** is not specified, its name is assumed to be the same as the current
mapset's name.

SEE ALSO

g.setproj

AUTHOR

Irina Kosinovsky, US ARMY CERL
M.L. Holko, USDA, SCS, NHQ-CGIS
R.L. Glenn, USDA, SCS, NHQ-CGIS

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.prune – Prunes points from binary GRASS vector data files.
(GRASS Vector Program)

SYNOPSIS

v.prune
v.prune help
v.prune [**-i**] **input**=*name* **output**=*name* **thresh**=*value*

DESCRIPTION

The GRASS program *v.prune* allows the user to remove extra points from a vector file. This allows users to reduce disk space required by a vector file and still have data accuracy within a given tolerance.

Flag:

-i The pruning threshold value is specified in map inches, rather than in data base units on the ground.

Parameters:

input=*name* Name of binary GRASS vector file containing data to be pruned.

output=*name* Name to be assigned to new, pruned vector output file.

thresh=*value* Threshold value used for pruning.

The program will be run non-interactively if the user specifies all parameters and (optionally) the *-i* flag on the command line, using the form:

v.prune [**-i**] **input**=*name* **output**=*name* **thresh**=*value*

If the user simply types **v.prune** without specifying program arguments on the command line, the program will prompt the user to enter parameter values.

NOTES

The threshold value is the same as the *v.digit* pruning threshold. This is specified in data base units on the ground (e.g., in ground meters for UTM data bases). The threshold can also be specified in inches on the map, and the program will convert these to data base ground units using the *scale* in the vector file. If you specify the scale in map inches rather than in ground units, you must specify that inches are used by setting the *-i* flag. The input vector data layer will be read and the resultant pruned vector data layer will be placed into a newly created output file whose name is specified by the user, leaving the original vector map unchanged.

The pruning algorithm throws away redundant points within the specified threshold. It works on each vector separately, working from node to node. It does not change the position or number of nodes.

SEE ALSO

v.digit, v.in.ascii

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.reclass – Creates a new map layer whose category values are based upon the user's reclassification of categories in an existing vector map layer.
(SCS GRASS Vector Program)

SYNOPSIS

v.reclass
v.reclass help
v.reclass [-d] input=*name* **output=***name* **type=***area, line,* or *site* [**title=***name*]

DESCRIPTION

v.reclass creates an *output* map layer based on an *input* vector map layer. The output map layer will be a reclassification of the input map layer based on reclass rules input to *v.reclass*. 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).

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:

v.reclass [-d] input=*name* **output=***name* **type=***area, line,* or *site* [**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.

Parameters:

-d	Dissolve common boundaries between areas
input=*name*	Vector input map *name*.
output=*name*	Vector output map *name*.
type=*name*	Select type of vector input file. Options: area, line, site

Alternately, the user can simply type **v.reclass** on the command line, without program arguments. In this case, the user will be prompted for all needed inputs.

Before using **v.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 vector 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* vector 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 vector 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 vector 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 NUM	NEW CATEGORY NAME
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 and supporting category files are created.

NON-INTERACTIVE PROGRAM USE: RECLASS RULES

In non-interactive program use, the names of an input map, output map, type of input 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 vector map layer, output map layer name, type of input map layer, and (optionally) output map layer title by typing

v.reclass [-d] input=*name* **output**=*name* **type**=*area, line,* or *site* [**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 vector 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 vector map layer to category 1 with category label "poor quality" in the output map layer, and reclassifies input vector 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 vector 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 vector map layer categories 1 thru 19 and 51 thru 100 to category 1 in the output map layer, input vector map layer categories 20 thru 24 and 26 thru 50 to the output map layer category 2, and input vector 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

The *v.reclass* program generates a new vector map layer. The *v.support* program will have to be run on the newly created vector map layer to build topology.
The dissolve *[-d]* option will work on only those areas which are of the same conversion category value. If a map area is inside (island) a converted area and is NOT converted to the same value, its boundaries are output to the resultant map.

WARNING

Category values which are not explicitly reclassified to a new value by the user will be reclassified to 0.

SEE ALSO

v.support

AUTHORS

Paul H. Fukuhara, USDA SCS, NCG-GSS
R. L. Glenn, USDA SCS, NHQ-CGIS

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.spag – Process spaghetti-digitized binary vector file.
(GRASS Vector Program)

SYNOPSIS

v.spag
v.spag help
v.spag [-i] map=*name* [**threshold**=*value*]

DESCRIPTION

This program will fix vector data that were not digitized in correct GRASS vector format. It will create a node at every line crossing, and will delete all hanging lines of length less than the specified threshold.

COMMAND LINE OPTIONS

Parameters:

-i	Only do Identical line removal pass.
map=name	Name of the binary vector file to be fixed.
threshold=value	Node-snapping threshold value.

NOTES

The user must run *v.support* after running *v.spag* to correct the topology (*dig_plus*) file.

v.spag generally deletes many lines from the input vector map layer. Because deleted lines are not eliminated from a vector data (*dig*) file, but are only marked as deleted, the user will probably wish to run *v.clean* on the vector file after running *v.spag* to eliminate any deleted lines from the *dig* file and decrease its size.

BUGS

This program contains several known bugs. It will sometimes get stuck in a loop in which it creates the same line over and over again. It will occasionally create an incorrect line. Users are strongly urged to back-up their vector data files before running this alpha version of *v.spag*.

Update Note 1/93: For GRASS Release 4.1, the -i flag has been added and bugs fixed. It works better than the 4.0 release.

SEE ALSO

v.clean, v.digit, v.support

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.stats − Prints information about a binary GRASS vector map layer.
(GRASS Vector Program)

SYNOPSIS

v.stats
v.stats help
v.stats [**−h**] **map**=*name*

DESCRIPTION

The program *v.stats* will print to standard output information about a user-specified binary GRASS vector map layer.

Flag:

−h Display header information from the vector file.

Parameter:

map=*name* Name of the binary GRASS vector file to be queried.

The program will be run non-interactively if the user specifies the parameter value and (optionally) sets the -h flag on the command line, using the form:

v.stats [**−h**] **map**=*name*

If the user instead simply types **v.stats** without specifying program arguments on the command line, the program will prompt the user to enter inputs through the standard user interface described in the manual entry for *parser*.

NOTES

Sample output follows:

```
Format: Version 4.0  (Level 2 access)  (Portable)

Number of Lines: 3
Number of Nodes: 2
Number of Areas: 2   (complete)
Number of Isles: 1   (complete)
Number of Atts : 2
```

The GRASS version number is given. Parenthetical text following this describes the read access level available and notes whether or not the file is in GRASS version 4.0 portable data format. The access level indicates the types of data available for the named vector map layer. "Level 1" denotes a binary vector file (without any accompanying *dig_plus* file), while "Level 2" denotes the existence of a *dig_plus* (vector topology) file for the named map layer (generally created by *v.support*). If only Level 1 information is available for a vector map layer, only the number of lines and (optionally) the file header will be printed to standard output.

If all areas and islands (isles) in the vector file have been identified (usually by *v.support*), their counts will be followed by "*complete*." If not, they will be followed by "*incomplete*". The *dig_plus* file, which is created and updated by *v.support*, must exist for this information to be output.

SEE ALSO

v.digit, v.import, v.support, and *parser*

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

v.support – Creates GRASS support files for (binary) GRASS vector data.
(GRASS Vector Program)

SYNOPSIS

v.support
v.support help
v.support [**-spr**] **map**=*name* [**option**=*name*] [**threshold**=*value*]

DESCRIPTION

v.support builds GRASS support files for (binary) GRASS vector data files. These support files supply topology (*dig_plus*) and category (*dig_att*) information that are needed by other GRASS programs (e.g., by *v.digit*, *v.to.rast*, etc.).

The program gives the user the options of: (1) building topological information (the *dig_plus* file) needed by *v.digit* for the vector data file, and/or (2) editing the category label (*dig_cats*) file associated with the vector data file.

OPTIONS

Program parameters and flags have the following meanings.

Flags:

-s	Snap nodes. Valid only with *build* option.
-p	Prompt user for threshold value. Valid only with *build* option. This flag is designed to be used only by the *v.import* program, and can usually be ignore.
-r	Reset corners of map region from range of data values.

Parameters:

map=*name*	Name of binary GRASS vector data file for which support files are to be created or modified. This map layer must exist in the user's current GRASS mapset.
option=*name*	Build topology information (*dig_plus* file), or edit categories (*dig_att* file) for *map*. Options: *build*, *edit* Default: *edit*
threshold=*value*	Snapping threshold value to be used when building topology. Valid only with *build* option and -s flag.

The user can run this program non-interactively by specifying parameter values (and optionally, flag settings) on the command line.

Alternately, the user can simply type the command **v.support** on the command line. In this event, the program will prompt the user to enter parameter values and flag settings.

If the *build* option is chosen, the user may further specify the *-s* flag, to snap nodes in the vector file. If nodes are to be snapped, the user can either: (1) use the calculated default threshold (based on the scale of vector data); (2) enter the -p flag, causing the program to prompt the user for a snapping threshold value; or (3) enter a specific threshold value on the command line.

The spatial assignment of category values (found in the *dig_att* file) is also performed during building of file topology.

The *edit* option allows the user to edit the category labels to be associated with the category values attached to the vector data during topology building. These labels, if created, are then used for raster map layers derived from their vector counterparts. The labels are placed in the *dig_cats* directory.

NOTES

A *dig_plus* file must be created for each imported vector map before it can be used in post-GRASS 3.0 versions of *digit* (now referred to as *v.digit*).

Topological information for GRASS vector files can also be built using *option 4* of the *v.import* program.

This program will convert pre-4.0 version GRASS vector files to 4.0 format.

v.support creates support files only for binary vector files located in the user's current mapset.

SEE ALSO

v.digit, v.import, v.in.ascii, v.prune, v.to.rast

AUTHORS

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory
Michael Higgins, U.S. Army Construction Engineering Research Laboratory

NAME

v.to.rast – Converts a binary GRASS vector map layer into a GRASS raster map layer.
(GRASS Vector Program)

SYNOPSIS

v.to.rast
v.to.rast help
v.to.rast input=*name output*=*name*

DESCRIPTION

v.to.rast transforms (binary) GRASS vector map layers into GRASS raster map layer format. Most GRASS analysis programs operate on raster data.

Parameters:

input=*name*	Name of the binary vector map layer to be converted.
output=*name*	Name to be assigned to the raster map layer output.

The user can run the program non-interactively by specifying the names of a vector input file and raster output file on the command line, using the form:

v.to.rast input=*name* **output**=*name*

If the user instead types simply **v.to.rast** on the command line, the program will prompt the user to enter these names.

NOTES

v.to.rast will only affect data in areas lying inside the boundaries of the current geographic region. Before running *v.to.rast*, the user should therefore ensure that the current geographic region is correctly set and that the region resolution is at the desired level; the program may otherwise create an empty raster map layer. An empty raster map layer will also be created if the vector map layer has not been assigned category/attribute labels (e.g., through use of the *v.digit* program).

The *v.to.rast* program creates two files: a raster map layer, and a history file. The GRASS program *r.support* must be run to create additional support files for the raster map.

Additional problems sometimes lead to the creation of empty raster map layers. Unfortunately, error messages explaining these phenomena do not yet exist.

SEE ALSO

g.region, r.support, v.digit, v.import, v.support

AUTHORS

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

NAME

v.to.sites **–** Converts point data in a binary GRASS vector map layer into a GRASS *site_lists* file.
(GRASS Vector Program)

SYNOPSIS

v.to.sites
v.to.sites help
v.to.sites input=*name* **output**=*name [***dmax**=*value* **-a -c -i]**

DESCRIPTION

The *v.to.sites* program extracts data from a GRASS vector map layer and stores output in a new GRASS *site_lists* file. If *-a* flag is selected, *v.to.sites* extracts all vertices from a vector file, if not selected, it extracts site (point) features only, ignoring lines and areas. If *-i* flag is selected then, for each line, if the distance between any two vertices on this line is greater that *dmax*, additional points are added to keep the distance withing *dmax* range. The resulting sites map layer can be used with such programs as *d.sites*. The user can run the program non-interactively by specifying the names of an existing vector *input* map layer and a new site list file to be *output* on the command line. The program will be run interactively if the user types **v.to.sites** without arguments on the command line. In this case, the user will be prompted to enter parameter values through the standard user interface described in the manual entry for *parser*.

Parameters:

input=*name*	Name of an existing binary vector map layer from which site data are to be extracted.
output=*name*	Name to be assigned to the resultant *site_lists* file.
dmax=*value*	Maximum distance between points (valid only when -a and -i are used)

Flags:

-a	Outputs all vertices (instead of site data only) from vector file to site file
-c	The Category data is used instead of attribute as a site description (valid only when -a is used)
-i	Additional sites are added between each 2 points on a line if the distance between them is greater than specified *dmax*. (valid only when -a is used)

If any of the sites have been labeled in *v.digit,* then the resultant site list will contain category information (or attribute in case **-a** was used but **-c** was not). If none of the sites are labeled, a binary (0/1) site list file will be produced.

SEE ALSO

d.sites, s.db.rim, s.menu, v.db.rim, v.digit and *parser*

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

Irina Kosinovsky, U.S. Army Construction Engineering Research Laboratory

NOTICE

This program is part of the *alpha* section of the GRASS distribution. Unlike the code in the *main* section of GRASS, the *alpha* code has not yet been fully tested for one release cycle.

NAME

v.transform – Transforms an ASCII vector map layer from one coordinate system into another coordinate system.
(GRASS Vector Program)

SYNOPSIS

v.transform
v.transform help
v.transform [**–y**] **input**=*name* *output*=*name* [**pointsfile**=*name*]

DESCRIPTION

This program has been used to import vector files that were in scanner or digitizer (x,y) coordinates and to transform these into UTM coordinates.

Flag:

-y Suppress the printing of residuals or other information to standard output.

Parameters:

input=*name* Name of the ASCII vector map layer to be transformed.

output=*name* Name to be assigned to the resultant (transformed) ASCII vector map layer.

pointsfile=*name* Name of a file containing transformation points, whose format is given below. Give a full path name for this file or it will be assumed to be located in the user's current directory.

The user can run this program non-interactively by specifying parameter values (and optionally, the flag setting) on the command line.

If the user runs **v.transform** without specifying program arguments on the command line, the program will prompt the user for inputs. When the program prompts the user for two sets of transformation points, the first set of points entered by the user should be in the coordinate system of the input map, and the second set of points should represent the corresponding geographic points in the coordinate system into which the map will transformed. A user must enter 4 to 10 of each set of points for the transformation to work correctly.

After the user has entered both sets of points, the program will show the amount of error associated with the transformation of the given points as the *residual mean average* (RMS). (An acceptable RMS for a 1:24,000 UTM map would be 1.2 to 2.4 (meters).) It will then ask if the transformation RMS value is acceptable. After an RMS is accepted by the user, *v.transform* will transform the ASCII (*dig_ascii*) vector map and its associated attribute (*dig_att*) file into the requested coordinate system.

Remember to run *v.support* or *v.import* on the output map.

NOTES

When rectifying a map to another coordinate system using *v.transform*, the user should specify the coordinates of between 4 to 10 points, and state these both in the coordinate systems of the input and output maps. The two sets of coordinates can be input to *v.transform* interactively, or from a file specified on the command line with the *pointsfile* option. The *pointsfile* option is especially useful when several maps in the same geographic area require transformation, as it eliminates the necessity for the user to repeatedly type in the same transformation coordinates.

A *pointsfile* file will contain between 4 to 10 lines; each line will contain the set of coordinate transformation points for the input map and the corresponding set of coordinates for the output map. The minimum number of lines for the transformation to take place is four.

The format of the *pointsfile* file is shown below:

Input Map		Output Map	
x	y	x	y
x	y	x	y
x	y	x	y
x	y	x	y

In the format shown above the *x*'s and *y*'s can be thought of as eastings and northings, depending on what coordinate systems you are transforming to and from.

An example of the *pointsfile* file:

1	1	589000	4913000
1	17000	589000	4930000
17000	17000	610000	4930000
17000	1	610000	4913000

Within the *pointsfile* file, numbers on a line must be spaced apart with tabs or blanks. The example shown above was used to convert a map in digitizer coordinates (range of 1-18000) to UTM coordinates within the UTM zone for the Spearfish sample data base location.

Because this *pointsfile* file is not your usual GRASS data file, the user will have to keep track of where it is on the system. When the *pointsfile* option is used on the command line to input the transformation points, the program does not ask whether or not everything is acceptable before converting the vector file and the attribute file.

The user is advised to run this program interactively with a specific set of transformation coordinates and to examine the resulting residuals, to determine how accurate the transformation will be (i.e., pick points with known values in both coordinate systems). After the residuals are acceptable, those transformation coordinates can be used with the program run non-interactively to transform other maps in the same geographic area.

WARNING

This is a general purpose program and can be fooled into giving low residuals. It is strongly suggested that any transformed map be checked for accuracy. The program assumes that the coordinate systems will be planimetric and has never been tested with negative values.

If this program is being used to transform maps from State Plane to UTM coordinates, and vice versa, users should be aware of the following points. This program will work better with State Plane zones that use the Transverse Mercator projection. Those are states that have their state zones splitting the state vertically, like Illinois. This program will not work as well with states that use the Lambert Conformal Conic projection. Those are states that have their state zones splitting the state horizontally, like Wisconsin. It is also best to keep the area being transformed relatively small.

SEE ALSO

v.digit, v.import, v.support

AUTHOR

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

NAME

v.trim **–** Trims small spurs, and removes excessive nodes from a binary GRASS vector (*dig*) file.
(GRASS Raster Program)

SYNOPSIS

v.trim
v.trim help
v.trim input=*name* **output**=*name* [**trim**=*value*]

DESCRIPTION

v.trim scans the user-specified GRASS vector *input* file and removes from it all lines having a length less than a user-specified trimming factor. Excess nodes (those unnecessary to a line's definition) between line junctions are also removed. The resulting vector output is sent to a user-named *output* file; the original vector *input* file is not modified by *v.trim*.

The trimming factor parameter (trim=*value*) gives the user control over the size of small spurs or "dangling lines" to be removed. The trimming factor is expressed in the same units (map coordinates) as the vector (*dig*) data within the user's current GRASS data base LOCATION (e.g.: in meters for UTM locations; in pixels or cells for locations in an x,y coordinate system; etc.).

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:

v.trim input=*name* **output**=*name* [**trim**=*value*]

If vector map *input* and *output* names are given 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 **v.trim** on the command line, without program arguments. In this case, the user will be prompted for needed parameter values using the standard GRASS user interface described in the manual entry for *parser*.

Parameters:

input=*name* Name of an existing vector map layer in the user's current mapset search path containing lines to be "trimmed".

output=*name* Name of a new vector file to contained the "trimmed" output.

trim=*value* A user-specified trimming factor, denoting the length of trimmed lines in map units. All lines having a length less than this trimming factor will be "trimmed" (i.e., removed) from the named vector input file.
Default: 10 (in units of meters or cells)

NOTES

v.support must be run on the vector *input* file prior to running *v.trim*.

v.support must also be run on the resultant vector *output* file to build the needed topology information stored under the user's *dig_plus* directory.

r.line maintains the same format (binary or ASCII) and attribute type (linear or area edge) as those of the original vector (*dig*) *input* file.

A trimming factor of zero (0) will not remove any small spurs, but will remove all excess nodes.

SEE ALSO

r.line, r.thin, v.digit, v.import, v.support, and *parser*

AUTHOR

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

3 UNIX SHELL SCRIPT PROGRAMS

Like Main Programs, UNIX shell script programs have received both alpha- and beta-testing. However, while other GRASS programs are written in the "C" programming language, shell scripts are UNIX Bourne and C-shell scripts readily developable by nonprogrammers and others unfamiliar with "C." Shell scripts are separately described for the convenience of potential system developers at GRASS user sites, who are encouraged to examine the shell scripts located on-line in the directories indicated in this manual, and to develop their own scripts.

NAME

3d.view.sh **–** Displays several 3-dimensional views of a landscape on the user's graphics monitor.
(GRASS Shell Script)

SYNOPSIS

3d.view.sh
3d.view.sh help
3d.view.sh **file**=*mapname* **ef**=*mapname* **vh**=*viewing_height* **sv**=*sink_value* **exag**=*exag* **lf**=*line_frequency* **back**=*background_color*

DESCRIPTION

3d.view.sh is a Bourne shell (sh(1)) script that displays several 3-dimensional views of a landscape on the user's graphics monitor. It erases the graphics monitor and then prepares it for the display of nine equally-sized frames. The user-specified raster map layer (given by file=*name*) is displayed using *d.rast* in the middle frame. The remaining frames are then used to display 3-d perspective views. The top middle panel is a view from the north, the top right from the north-east, the right from the east, and so on. Each is drawn with a call to the *d.3d* program. The viewing angles are calculated automatically. If options are not stated on the command line, default values will be used. These values are listed under Parameters, below.

Parameters:

file=*mapname* Name of raster map layer to be displayed.
Default: *elevation*

ef=*mapname* Name of raster map layer whose category values will supply the elevation values used to generate 3-d perspective views.
Default: *elevation*

vh=*viewing_height* Height (in meters) of the location from which scenes will be viewed.
Default: 30000

sv=*sink_value* Sink factor value, causing the image to be displayed lower, or higher, on the graphics screen.
Default: 0

exag=*vertical_exaggeration*
Vertical exaggeration factor of the values in the elevation file.
Default: 3

lf=*line_frequency* Contour intervals at which vector grid lines will be drawn, in meters.
Default: 20

back=*background_color*
Color of the background of the display frames.
Options: red, orange, yellow, green, blue, indigo, violet, magenta, brown, gray, white, and black
Default: black

NOTES

In the *spearfish* sample data base, the user must specify a viewing height when running *3d.view.sh*. Note also that the raster elevation map layers in the PERMANENT mapset under spearfish are named *elevation.dem* and *elevation.dma*.

This program will not prompt the user for inputs; if the user types **3d.view.sh** without program arguments on the command line, default values will be used.

FILES

This program is simply a shell script. Users are encouraged to make their own shell scripts using similar techniques. See $GISBASE/scripts/3d.view.sh.

SEE ALSO

d.3d, d.rast

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

blend.sh **–** Combines the red, green, and blue color components of two raster map layers.
(GRASS Shell Script)

SYNOPSIS

blend.sh file1 file2 perc outbase

DESCRIPTION

blend.sh is a Bourne shell (sh(1)) script that extracts the red (R), green (G), and blue (B) color components from each of two raster map layers, and creates three new raster map layers whose category values respectively represent the combined red, combined blue, and combined green color values from the two input layers. Category values in each of the output map layers will fall within the range of 0 - 255.

The R,G,B values from the two input map layers (*file1* and *file2*) are not simply added together, but are instead combined by a user-named percentage (*perc*) of the R,G,B values in *file1*. Specifically, *blend.sh* executes three *r.mapcalc* statements that: (1) convert the R,G,B values in *file1* and *file2* to the range 0 - 255; (2) multiply the R, G, and B values in *file1* by a user-named percentage (*perc*); (3) multiply the R, G, and B values in *file2* by (100 - *perc*); (4) create three new raster map layers, whose category values represent the summed R, summed G, or summed B values resulting from (2) and (3). Resulting R, G, and B values will respectively be stored in three new raster map layers named *outbase*.r, *outbase*.g and *outbase*.b.

This program runs non-interactively; the user must state all parameter values on the command line.

Parameters:

file1	Name of a first raster map layer, whose R, G, and B color components will be combined with those of the second raster map layer (*file2*) named. The percent value (*perc*) given will apply to *file1*.
file2	Name of a second raster map layer, whose color components will be combined with those of *file1*. The percent value (*perc*) given will apply to the R,G,B values in *file1*. The R, G, and B values in *file2* will be multiplied by (100 - *perc*) %.
perc	Percentage or amount of the color contribution in terms of color intensity. This value is multiplied by the R,G,B values in *file1*.
outbase	The root name assigned to each of the three output files created. A suffix is added to each file name, indicating which hold the red, green, and blue color values.

blend.sh executes three *r.mapcalc* statements:

r.mapcalc "*outbase*.r = r#*file1* ∗ .*perc* + (1.0 - .*perc*) ∗ r#*file2*"
r.mapcalc "*outbase*.g = g#*file1* ∗ .*perc* + (1.0 - .*perc*) ∗ g#*file2*"
r.mapcalc "*outbase*.b = b#*file1* ∗ .*perc* + (1.0 - .*perc*) ∗ b#*file2*"

It uses the # operator to separately extract the red, green, and blue components in the named raster map layers, essentially allowing color separates to be made.

EXAMPLE

Typing the following at the command line:

blend.sh aspect elevation 40 elev.asp

will create three new raster map layers named *elev.asp.r*, *elev.asp.g*, and *elev.asp.b*, that, respectively, contain 40 of the red, green, and blue components of the *elevation* map layer and contain 60 of the red, green, and blue components of the *aspect* map layer.

FILES

This program is simply a shell script. Users are encouraged to make their own shell scripts using similar techniques. See $GISBASE/scripts/blend.sh.

SEE ALSO

r.colors, r.mapcalc

AUTHOR

Dave Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

bug.report.sh − A mechanism for writing, storing, and e-mailing to USACERL users' bug reports on GRASS 4.0 commands.
(GRASS Shell Script)

SYNOPSIS

bug.report.sh *4.0_program.name* [*program_arguments*]

DESCRIPTION

bug.report.sh is a Bourne shell (sh(1)) script which, when given a GRASS 4.0 program name, allows the user to complete a bug report for that program. Completed bug reports can be e-mailed to the GRASS development group at USACERL, saved to a file in the user's home directory, or discarded.

This program is not interactive; the user must specify the name of a 4.0 program on the command line. Program arguments can optionally be entered by the user.

Parameters:

4.0_program.name The name of an existing GRASS version 4.0 program tested by the user.

program_argument(s)
Program arguments (parameters and/or flags) for the *4.0_program.name* tested by the user. The user should enter whatever command arguments were actually run when the program bug occurred.

EXAMPLE

For example, if the user wished to complete a bug report on *v.to.rast* after running the program, the user might then type:

bug.report.sh v.to.rast

A standard bug report form would then be displayed on the user's text terminal, containing the user's current GRASS region settings, and the machine name, GRASS data base, location, and mapset, on which the user is currently running GRASS. Program parameters and flag settings, and the command entered by the user on the command line, are also automatically entered on the bug report form. The user is put into a text editor and expected to enter additional information describing the nature of the bug found or the results of program testing.

The user is then asked whether the completed bug report is to be:
1 - mailed to westerve@zorro.cecer.army.mil
2 - added to the file "grass.bugs" in the user's home directory
3 - both 1 and 2
4 - thrown away

NOTES

This program prints whatever region settings, GRASS data base, location, and mapset are current when the user runs *bug.report.sh*. The user is therefore advised to run *bug.report.sh* immediately after experiencing a program bug, to ensure that the settings current when the bug occurred are reported on the bug report form.

FILES

This shell script is stored under the $GISBASE/scripts directory on the user's system. The user is encouraged to examine the shell script commands stored in this directory and to produce similar scripts for their own use. Users might modify this shell script to e-mail reports of program bugs to a local systems' administrator in addition to someone at USACERL.

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

dcorrelate.sh − Graphically displays the correlation raster map layers in the active frame on the graphics monitor.
(GRASS Shell Script)

SYNOPSIS

dcorrelate.sh *layer1 layer2* [*layer3* [*layer4*]]

DESCRIPTION

dcorrelate.sh is a C-shell (csh(1)) script that graphically displays the results of an *r.stats* run on two raster map layers. This shell script is useful for highlighting the correlation (or lack of it) among data layers.

The results are displayed in the active display frame on the user's graphics monitor. *dcorrelate.sh* erases the active frame before displaying results.

Parameters:

layer1 layer2 [*layer3* [*layer4*]]
The names of from two to four existing raster map layers to be included in the correlation.

NOTES

This is a shell script that uses *r.stats* and the UNIX *awk* command to calculate the correlation among data layers, and uses *d.text* and *d.graph* to display the results.

If three or four map layers are specified, the correlation among each combination of two data layers is displayed.

This command is written for /bin/csh. If your system doesn't support this shell, don't install this script.

FILES

This program is simply a shell script. Users are encouraged to make their own shell script programs using similar techniques. See $GISBASE/scripts/dcorrelate.sh.

SEE ALSO

The UNIX *awk* command

d.text, d.graph, r.coin, r.stats

AUTHOR

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

NAME

grass.logo.sh – Displays a GRASS/Army Corps of Engineers logo in the active display frame on the graphics monitor.
(GRASS Shell Script)

SYNOPSIS

grass.logo.sh

DESCRIPTION

grass.logo.sh is primarily a demonstration of the GRASS *d.graph* program in a UNIX Bourne shell macro that generates the U.S. Army Corps of Engineers logo. Users are encouraged to generate their own unique logos by writing similar macro shell scripts. Examine the contents of the input file called *grass.logo.sh* located in the GRASS shell script command directory (*$GISBASE/scripts*) to see how the GRASS logo was generated using *d.graph* graphics commands. The coordinates for this logo were taken from a drawing done on graph paper.

To view the graphics described by this file use *d.graph* or *d.mapgraph*, making sure that a copy of this file is either in your current directory or is given by its full path name.

NOTES

grass.logo.sh, like *d.rast*, will overwrite (not overlay) whatever display appears in the active graphics frame.

This program requires no command line arguments.

SEE ALSO

d.font, d.graph, d.mapgraph, d.rast

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

hsv.rgb.sh – Converts HSV (hue, saturation, and value) cell values to RGB (red, green, and blue) values. *(GRASS Shell Script)*

SYNOPSIS

hsv.rgb.sh *file1 file2 file3*

DESCRIPTION

hsv.rgb.sh is a Bourne shell (sh(1)) program that converts HSV to RGB using *r.mapcalc*. The Foley and Van Dam algorithm is the basis for this program. Input must be three raster files - each file supplying the HSV values. Three new raster files are created representing RGB.

NOTES

Do not use the same names for input and output.

FILES

This program is simply a shell script stored in the file *hsv.rgb.sh* under the $GISBASE/scripts directory. Users are encouraged to make their own shell script programs using similar techniques.

SEE ALSO

rgb.hsv.sh

AUTHOR

James Westervelt, U.S. Army Coonstruction Engineering Research Laboratory

NAME

old.cmd.sh − Provides the new GRASS version 4.0 program name for any program name in GRASS version 3.2.
(GRASS Shell Script)

SYNOPSIS

old.cmd.sh *3.2_program.name*

DESCRIPTION

old.cmd.sh is a Bourne shell (sh(1)) script which, when given a GRASS 3.2 program name, returns the name of the new GRASS version 4.0 program performing its functions. This program is useful as a quick on−line cross-reference between GRASS versions 3 and 4. This program is not interactive; the user must specify the name of a 3.2 program on the command line.

Parameter:

3.2_program.name The name of an old GRASS version 3.2 program.

Output will be in the form:

old 3.2 program name replaced with: *new 4.0 program name*

EXAMPLE

For example, to learn which GRASS 4.0 command performs the function of the GRASS 3.2 *list* command, the user might type:

old.cmd.sh list

The user would then see the following message displayed to standard output:

list replaced with: *g.list*

FILES

This shell script is stored under the $GISBASE/scripts directory on the user's system. The user is encouraged to examine the shell script commands stored in this directory and to produce similar scripts for their own use.

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

rgb.hsv.sh – Converts RGB (red, green, and blue) cell values to RGB (red, green, and blue) values. *(GRASS Shell Script)*

SYNOPSIS

rgb.hsv.sh *file1 file2 file3*

DESCRIPTION

rgb.hsv.sh is a Bourne shell (sh(1)) program which converts RGB values to HSV using *r.mapcalc*. The Foley and Van Dam algorithm is the basis for the program. Input must be three raster files - each file supplying the RGB values. Three new raster files are created representing HSV.

NOTES

Do not use the same names for input and output.

FILES

This program is simply a shell script stored under the $GISBASE/scripts directory. The user is encouraged to examine the shell script programs stored here and to produce other such programs.

SEE ALSO

hsv.rgb.sh

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

shade.rel.sh − Creates a shaded relief map based on current resolution settings and sun altitude and azimuth values entered by the user.
(GRASS Shell Script)

SYNOPSIS

shade.rel.sh

DESCRIPTION

shade.rel.sh is a Bourne shell (sh(1)) script that creates a raster shaded relief map based on current resolution settings and on sun altitude and azimuth values entered by the user. The new shaded relief map map is named *shade* and stored in the user's current mapset. The map is assigned a grey-scale color table.

This program is interactive; the user enters the command:

shade.rel.sh

The program then prompts the user to enter values for:

(1) the altitude of the sun in degrees above the horizon (a value between 0 and 90 degrees), and
(2) the azimuth of the sun in degrees to the east of north (a value between -1 and 360 degrees).
(3) the name of a raster map layer whose cell category values are to provide elevation values for the shaded relief map. Typically, this would be a map layer of elevation; however, any raster map layer can be named.

Specifically, *shade.rel.sh* executes the following *r.mapcalc* statement:

```
r.mapcalc << EOF
shade = eval( \
 x=($elev[-1,-1] + 2*$elev[0,-1] + $elev[1,-1] \
   -$elev[-1,1] - 2*$elev[0,1] - $elev[1,1])/(8.*$ewres) , \
 y=($elev[-1,-1] + 2*$elev[-1,0] + $elev[-1,1] \
   -$elev[1,-1] - 2*$elev[1,0] - $elev[1,1])/(8.*$nsres) , \
 slope=90.-atan(sqrt(x*x + y*y)), \
 a=round(atan(x,y)), \
 aspect=if(x||y,if(a,a,360)), \
 cang = sin($alt)*sin(slope) + cos($alt)*cos(slope) * cos($az-aspect), \
 if(cang < 0,0,100*cang))
EOF
```

Refer to the manual entry for *r.mapcalc* for an explanation of the filtering syntax shown in the above expression. See, for example, the section on "The Neighborhood Modifier".

shade.rel.sh then runs *r.colors* to assign a grey-scale color table to the new shaded relief map *shade*, by executing the command:

r.colors shade color=grey

FILES

This program is simply a shell script. Users are encouraged to make their own shell scripts using similar techniques. See $GISBASE/scripts/shade.rel.sh.

SEE ALSO

"*r.mapcalc*: An Algebra for GIS and Image Processing," by Michael Shapiro and Jim Westervelt, U.S. Army Construction Engineering Research Laboratory (March/1991).

"GRASS Tutorial: *r.mapcalc*," by Marji Larson, U.S. Army Construction Engineering Research Laboratory.

blend.sh, g.ask, g.region, r.colors, and *r.mapcalc*

AUTHOR

Jim Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

show.color.sh **–** Displays and names available primary colors used by GRASS programs, in frames on the graphics monitor.
(GRASS Shell Script)

SYNOPSIS

show.color.sh

DESCRIPTION

show.color.sh is a UNIX Bourne shell macro that displays and names available primary colors used by GRASS programs in frames on the graphics monitor. Available colors are: *red, orange, yellow, green, blue, indigo, violet, white, black, gray, brown, magenta*, and *aqua*.

No program arguments are required to run this program.

NOTES

The full monitor screen is made the active frame after this program ends.

This macro is located in *$GISBASE/scripts/show.color.sh*.

SEE ALSO

d.colormode, d.colors, d.colortable, d.display, d.frame, grass.logo.sh, show.fonts.sh

AUTHOR

David Gerdes, U.S. Army Construction Engineering Research Laboratory

NAME

show.fonts.sh – Displays and names available font types in the active display frame on the graphics monitor.
(GRASS Shell Script)

SYNOPSIS

show.fonts.sh

DESCRIPTION

show.fonts.sh is a UNIX Bourne shell macro which runs the *d.erase -a* command, then names and displays the font types that can be selected using *d.font*. This macro also runs the GRASS commands *d.font* and *d.text*. See the manual entry for *d.font* for instructions on choosing a font type.

No program arguments are required to run this program.

BUGS

The font is set to *romans* (Roman simplex) after running *show.fonts.sh.* There is no mechanism to query the current font, so there is no way to automatically restore the font. The user will have to reset the font type using *d.font* if *romans* is not desired.

FILES

This program is simply a shell script stored under the $GISBASE/scripts directory. Users are encouraged to examine the shell script programs stored here and to produce others for their own use.

SEE ALSO

d.display, d.erase, d.font, grass.logo.sh, d.label, d.legend, d.paint.labels, d.text, d.title

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

slide.show.sh – Displays a series of raster map layers existing in the user's current mapset search path on the graphics monitor.
(GRASS Shell Script)

SYNOPSIS

slide.show.sh [**across**=*value*] [**down**=*value*] [**mapsets**=*list*]

DESCRIPTION

slide.show.sh is a UNIX Bourne shell macro which clears the entire screen, creates a series of display frames on the graphics monitor, and displays in slideshow format each of the raster map layers listed in the user-specified *mapsets*. This is a shell script example which makes extensive use of GRASS and UNIX commands. Users are encouraged to examine this macro and develop similar on-line demos using their own data files.

Parameters:

across=*value* The number of display frames across the graphics monitor screen to be used for map display.
Default: 4

down=*value* The number of display frames down the graphics monitor screen to be used for map display.
Default: 3

mapsets=*list* The names of the mapsets under the user's current location whose raster map layers are to be displayed, separated by commas.
Default: All mapsets listed in the user's current mapset search path.

FILES

See the file *slide.show.sh* under $GISBASE/scripts.

SEE ALSO

d.display, d.erase, d.text, g.mapsets, 3d.view.sh, demo.sh, grass.logo.sh, show.fonts.sh

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

split.sh – Divides the graphics monitor into two frames and then displays two maps in these frames.
(GRASS Display Program)

SYNOPSIS

split.sh mapname mapname [**cmd**=*GRASS_command*] [**cmd2**=*GRASS_command*] [**view**=**horiz**]

DESCRIPTION

split.sh is a Bourne shell (sh(1)) script that clears the entire graphics screen and divides it into two display frames. Map layers are then displayed in each of the two frames. This command is very useful for visually comparing maps (raster, vector, and 3-d views) and can be used by other GRASS shell macros. It is also useful for creating demos. Program parameters are given below.

Parameters:

view=**horiz**
The graphics screen can be split either horizontally or vertically. The default view splits the screen into two frames, one on the left and one on the right (a vertical split). Some maps (3-d views) are better represented with more width then height (horizontal split). The first map name listed on the command line will be displayed in the top or left window (depending on whether the screen was split horizontally or vertically), and The second map will be displayed in the bottom or right window.

cmd=*GRASS_command*
The GRASS command used to display the named *mapname*s. If no command is specified by the user, *d.rast* is used by default. However, any GRASS display command (e.g., *d.3d, d.vect*, etc...) can be entered.

cmd2=*GRASS_command*
This command will be used to display map data in the second frame only.

If the user fails to specify the values of both *cmd* and *cmd2fR, split.sh* will use the default command (*d.rast*) to display user-specified map layer names in both frames. If the user specifies only the value of *cmd* on the command line, then that command will be executed for both frames. If the user specifies the values of both *cmd* and *cmd2* on the command line, the *cmd* command will be executed in frame 1 and the *cmd2* command will be executed in frame 2.

EXAMPLES

split.sh soils vegcover

split.sh soils cmd2=d.legend "soils red"

split.sh elevation vegcover cmd=d.3d view=horiz

NOTES

split.sh leaves the frame that the last map was drawn in as the active frame. The order in which the options (*cmd, cmd2, view*) are placed on the command line doesn't matter, but the order is important for the map names.

FILES

This program is simply a shell script. Users are encouraged to make their own shell scripts using similar techniques. See $GISBASE/scripts/split.sh.

SEE ALSO

d.3d, d.frame, d.rast, d.sites, d.vect

AUTHOR

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

NAME

start.man.sh – Creates the template for a manual entry in standard User's Reference Manual format for a user-specified GRASS 4.0 command.
(GRASS Shell Script)

SYNOPSIS

start.man.sh *4.0_program.name*

DESCRIPTION

start.man.sh is a Bourne shell (sh(1)) script which, when given a GRASS 4.0 program name, creates a basic manual entry for that program in the same standard format as that used by the GRASS User's Reference Manual. The named program must already exist under a directory for GRASS main, alpha, or contributed source code.

This program is not interactive; the user must specify the name of a 4.0 program on the command line.

By default, program output will be sent to standard output (i.e., displayed to the user's text terminal). If the user wishes to save the manual entry created by *start.man.sh*, program output can be redirected into a file. For example, the below command will create a manual entry for the program *new.program*, and save output to the file *new.program.man* in the user's current directory.

start.man.sh new.program > new.program.man

Parameter:

4.0_program.name The name of an existing GRASS program located in a source code directory for main, alpha, or contributed software.

FILES

This shell script is stored under the $GISBASE/scripts directory on the user's system. The user is encouraged to examine the shell script commands stored in this directory and to produce similar scripts for their own use.

SEE ALSO

GRASS 4.0 User's Reference Manual, by Jim Westervelt, Michael Shapiro, et al (USACERL).

bug.report.sh, g.help, g.manual

AUTHOR

James Westervelt, U.S. Army Construction Engineering Research Laboratory

NAME

tig.rim.sh – Generates various vector maps from a rim/TIGER data base.
(GRASS Shell Script)

SYNOPSIS

tig.rim.sh help
tig.rim.sh *dbname tractN1 tractN2 ...*

DESCRIPTION

tig.rim.sh is a shell script which queries information from a rim data base using the GRASS command *v.db.rim*, which is an interface between GRASS and RIM. The *dbname* given on the command line should be the name of a rim data base created using *v.in.tiger*. *tig.rim.sh* will create several new vector files. Three of these are: a county outline map, a map of tract boundaries within the county, and a map showing block group boundaries. For every tract number given on the input line, additional vector maps will be created showing: the tract outline, a block group boundary map for each block group within the tract, and a map showing block boundaries within each block group. Output files will be named *dbname.county*, *dbname.tract* and *dbname.bg* for the map layers showing the county outline, the tract outlines within the county, and the block group boundaries within the county. The vector file showing the boundary for an individual tract will be named *TtractN*, where *tractN* is the tract number given on the command line. The vector files created to show an individual block group boundary and block boundaries within that block group will be named using the appropriate tract number and block group number as part of the name, with suffixes of .bg and .bk respectively.

Parameters:

dbname Name of an existing rim data base.

tractN Number of a tract located within this county.

NOTES

This command must be installed separately as part of the package of routines dealing with the import of Census (TIGER) data. It requires the use of *rim* and *v.db.rim*, which must be compiled first.

You must include at least one tract number on the command line for this command to function. Use *tiger.info.sh* to obtain all tract numbers for a given TIGER type1 data file.

If the master binary vector file created using *v.in.tiger* is modified after it is in GRASS, this program will probably not work. In that situation, the processes from this shell script may simply be run by hand, using *v.db.rim* directly, but searching the old vector file to find lines for the new vector files without using the binary offset field (vectoff).

Vector files showing the block boundaries within a block group may contain hydrology lines, which in fact define the edge of a census block.

SEE ALSO

v.in.tig.rim, v.db.rim, tiger.info.sh

AUTHOR

Jim Hinthorne and David Satnik, GIS Lab, Central Washington University, Ellensburg, WA.

NAME

tiger.info.sh – Provides tract number(s) and classification codes found within a given U.S. Census Bureau TIGER type1 data file.
(GRASS Shell Script)

SYNOPSIS

tiger.info.sh help
tiger.info.sh *infile*

DESCRIPTION

tiger.info.sh is a shell script which outputs tract number(s) and classification codes found within the TIGER type1 data file *infile*. Output is written to standard out, and can be captured in a file by redirecting output. This information is useful when querying (using *v.db.rim*) from the master binary vector file created by *v.in.tig.rim*. It also provides tract number(s) which can be used as input to the command *Gen.Maps*.

Parameters:

infile Name of a TIGER type1 data file.

NOTES

This command must be installed separately as part of the package of routines dealing with the import of Census (TIGER) data.

SEE ALSO

m.tiger.region, v.in.tig.rim, v.db.rim, Gen.Maps, Gen.tractmap

AUTHOR

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

4 INDEX of CONTRIBUTED PROGRAMS

This chapter lists code contributed primarily by other user sites. The code may have undergone limited internal testing and has not yet been beta-tested. These programs are in the *src.contrib* directory and are not compiled during general GRASS compilation; users must compile these programs separately to run them.

Be advised that these programs may not work as expected.

Index of Contributed Code

The following programs were contributed for the 4.1 GRASS release and are accompanied by on-line documentation.

d.linegraph -	Generates and displays simple line graphs in the active graphics monitor display frame. (GRASS Display Program)
f.acap -	Calculates area-capacity for select areas of interest and shows relationships between water surface elevation, water surface area, and volume of water.
f.detail -	Functions like f.econ but outputs two detailed reports: economic damage and depth of flooding.
f.econ -	Takes as input the results of f.wsurf along with a user-supplied vector map of building sites and two ASCII files of economic data and generates a vector map of total damage.
f.input -	Reads the results of the HEC-2 Water Surface Profile model and generates a vector map of water surface evaluations at hydraulic sections.
f.reach -	Provides floodwater statistics, including area extent of flooding, average flood depth, and volume of water, calculated on a reach-by-reach basis.
f.wsurf -	Takes the results of f.input along with a raster terrain elevation map and generates two raster maps describing the floodplain.
f.xsection -	Provides an interactive means for digitizing new cross-sections to be used as input to HEC-2 modeling.
i.texture -	Calculates textural features on a raster file. (GRASS Image Processing Program)
r.traj -	Ballistic trajectory modeling program.
s.medp -	Median polish for a GRASS sites list. (GRASS sites)
s.to.rast -	Converts a site file to a cell file. (CCRS Contrib Program)
s.voronoi -	Uses a GRASS sites list to produce a Voronoi diagram.
v.autocorr -	Calculates spatial autocorrelation statistics for GRASS vector files. (GRASS vector)

v.in.poly - Creates a vector file of polygons centered on given locations. (GRASS Vector Program)

v.mkgrid - Creates a (binary) GRASS vector map of a user-defined grid. (GRASS vector)

v.reg - Computes the latitude and longitude of the exterior and interior corner tic marks for USGS 7.5' quad maps.

wrat - WRAT is an interactive module for evaluating water resources.

The following programs were incorporated into contributed code in pre-4.1 GRASS releases and may/may not be accompanied by on-line documentation.

DOS.delete

DOS.list

DOS.save

DOS.show

Darea

Dtrue

ISMann2dig

ISMcell2grd

ISMcell2scat

ISMcont2dig

ISMdig2ann

ISMgrd2cell

Mlulc.USGS

Mlulc.read

SG3d

d.6386.delete

d.6386.save

d.6386.show

d.distance

d.fix.ortho

d.text

d.vect.cats

grass.to.ipw

i.rvi.prediction

i.shape

m.6386

m.bsplit

m.dos

m.eigensystem

m.futil

m.geo

m.get.fips

m.get.stp

m.qcalc

m.regression

m.setproj

m.stp.proj

m.strip99s

p.hpx1300

r.gif

r.in.elas

r.in.erdas

r.in.miads

r.in.ppm

r.in.tang

r.in.utm

r.linear.regression

r.ndvi.model

r.out.ascii

r.out.ppm

r.out.ppm3

r.out.sites

r.rational.regression

r.reclass.scs

r.rvi

r.support

r.tiff

r.to.rlc

r.traj.data

s.makedots

s.to.vect

s.voronoi.shar

v.dump

v.export

v.extract

v.import

v.in.dlg.scs

v.in.gef

v.in.tiger.scs

v.make.subj

v.merge

v.mk_stats

v.mkquads

5 DESCRIPTIONS of PROGRAM FORMATS

This chapter contains descriptions of file formats and drivers used by some GRASS programs.

NAME

imagery - Description of GRASS image processing functions.

IMAGE PROCESSING IN GRASS

The following discussion is intended to provide a quick overview of image processing in GRASS. Some concepts and some hints are provided. For a more complete discussion and description of image processing in GRASS see *GRASS Tutorial: Image Processing*.

EXTRACTING IMAGERY DATA INTO A GRASS DATABASE

Remotely sensed images are captured for computer processing by filtering radiation emanating from the image into various electromagnetic wavelength bands, converting the overall intensity for each band to digital format, and storing the values on computer compatible media such as magnetic tape.

The GRASS programs which extract image data from magnetic tape can read LANDSAT multi-spectral scanner (MSS) data (*i.tape.mss*), LANDSAT thematic mapper (TM) data, (*i.tape.tm*), and other formats, such as scanned aerial photography or SPOT satellite data (*i.tape.other*). They extract the band data into raster files in a GRASS database. Each band becomes a separate raster file, with standard GRASS map layer support, and can be displayed and analyzed just like any other raster file.

UNREGISTERED DATA

The band data extracted from tapes are assumed to be unregistered data. This means that the GRASS software does not know the earth coordinates for pixels in the image. The only coordinates known at the time of extraction are the columns and the rows relative to the way the data was stored on the tape.

Data can only be extracted into a database which has an x,y coordinate system, and not into a projected database (e.g., a UTM database). This is to prevent users from mixing the unregistered data with registered data in the same database. The GRASS system comes with the database *imagery* which is an x,y database. New databases can be created by users during GRASS startup. See the *g.help* section on "Setting Up a GRASS Database" for instructions on creating a new database.

CELL HEADERS

The cell headers for the band files in these x,y databases are set to reflect the rows and columns of the extracted data. The north-south values represent the rows, and the east-west values represent the columns. The resolution of the unregistered data is set to 1.

Note, however, that while the row numbers increase from 1 to n from north to south, GRASS requires that the values of the user's current geographic region decrease from north to south. The solution adopted was to represent the rows with negative values (i.e., -1 to -n). This allows them to decrease from north to south and, if the minus sign is ignored, to reflect the row numbers.

The cell headers for the layers in x,y databases are set so that the coordinates at the center of each pixel exactly reflect the row and column for that pixel. The northern edge is set to 0.5 less than the first row, the southern edge 0.5 larger than the last row, the west to 0.5 less than the first column, and the east to 0.5 larger than the last column. When the image is displayed on the graphics monitor, the *d.where* command can be used to report row and column values.

For example, suppose rows 100-500 and columns 200-800 are extracted. Then the cell headers for the extracted data will be given the following values:

north:	-99.5
south:	-500.5
west:	199.5
east:	800.5
ns res:	1.0
ew res:	1.0

REGION AND MASK

Since the data layers are given essentially contrived cell headers, users must exercise extra care when analyzing or displaying unregistered images. It is very easy for the user's GRASS region to have absolutely no relationship to the data he is trying to display. This could happen when the region is set for data extracted from one tape, but the analysis is attempted on data extracted from another tape. A good habit to develop is to set the region to exactly match one of the band files. This can be done using the GRASS *g.region* command.

Another pitfall is to have a mask set to a band file from one data set while trying to read another. Even if the region is set properly, the data will appear to be all no-data since the mask will effectively knock out any data. Be sure that the mask is either set to a related data layer or not set at all. See *r.mask* for information on setting and unsetting the mask.

Please note that the tape extraction routines set your database region to match the rows and columns of the data that is extracted.

GROUPS

Since the band files are individual raster files, it is necessary to have a mechanism to maintain a relationship between band files from the same image as well as raster files derived from the band files. The GRASS *group* data structure accomplishes this goal. The group is essentially a list of names of raster files that belong in the group. For user convenience, groups are also created (and updated) by the tape extraction routines. The tape extraction programs ask the user to supply a group name as well as to specify the bands to be extracted. Suppose that the user extracts bands 1, 2, and 3 into a group called *nhap*. Then the band files will become the raster files *nhap.1, nhap.2,* and *nhap.3* and the group *nhap* will list these 3 raster files as members of the group.

Groups can also be created and modified by the user using the GRASS command *i.group.*

IMAGE REGISTRATION AND RECTIFICATION

Image registration and image rectification is the process of associating earth coordinates with pixels on the image and then converting the unregistered raster files to raster files in a projected database.

Image registration (*i.points*) is applied to a group, rather than to individual raster files. The control points are stored in the group, allowing all related band files to be registered in one step rather than individually.

Image rectification (*i.rectify*) is applied to individual raster files, with the control points for the group used to control the rectification and the group target (*i.target*) used to specify the database where the rectified layer will live.

IMAGE CLASSIFICATION

Image classification methods process all or a subset of the band files as a unit. Spectral signatures for the image are generated (*i.cluster*) and then used to produce a landcover map (*i.maxlik*).

The signatures must be associated only with the raster files actually used in the analysis. This means that with a group *subgroups* must be created (*i.group*) which list the band files to be grouped for classification purposes. The signatures are stored with the subgroup.

Note that multiple subroups can be created within a group. This allows different classifications to be run with different combinations of band files.

Also note that raster files produced by the classification process (*i.maxlik*) are automatically listed as part of the group.

RECTIFIED VS. UNRECTIFIED ANALYSIS

There are two possible routes for processing image data. The first is to register the group (*i.points*), perform the analyses on the unrectified band data (*i.maxlik*), and then rectify the results (*i.rectify*). The second is to register the group (*i.points*), rectify the band data (*i.rectify*), then run analyses on the rectified band data in the target location (*i.rectify*). Both routes are permissible in GRASS. Users will most likely prefer the first. The second route requires leaving GRASS and re-running GRASS under the target location. It also will require that a group be created to hold the rectified band files since *i.rectify* does not create or modify groups. Also, spatial filtering may not be as effective on rectified data since the rectification of the data requires resampling the original data.

SEE ALSO

GRASS Tutorial: Image Processing

d.where, g.region, i.cluster, i.group, i.maxlik, i.points, i.rectify, i.tape.other, i.tape.mss, i.tape.tm, i.target, r.mask

AUTHOR

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

NAME

monitorcap – Description of device driver database file format.

DESCRIPTION

The *d.mon* command and all graphics commands access a device driver database stored in the file $GISBASE/etc/monitorcap. This file contains a line for each graphics monitor on the system. Each line contains 6 fields separated by colons:

name the name by which the user identifies the driver

program the full pathname of the executable driver or a pathname relative to $GISBASE

description
A short 4 or 5 word description of the driver.

fifos full pathnames to the input and output fifos over which the driver communicates with the libraries

tty the name of the tty from which the driver must be started, or empty if there is no such restriction

msg a message to print to the user if an attempt is made to start the driver from the wrong tty.

EXAMPLE

The following is an example of a single monitor system. The driver may be started from any terminal and uses /dev/fifo.a and /dev/fifo.b for communication with application programs:

x:driver/X:graphics monitor:/dev/fifo.a /dev/fifo.b::any terminal

This example has 2 monitors on the system, using the same device driver. Note that each has a different *name* field and uses a different pair of fifos. Also, each must be started on a specific terminal:

1:driver/X:monitor 1:/dev/fifo.1a /dev/fifo.1b:/dev/tty8:tty8
2:driver/X:monitor 2:/dev/fifo.2a /dev/fifo.2b:/dev/tty10:tty10

NOTES

Each line in this file must refer to a valid driver. No comment lines are supported. Also, no blank lines are allowed.

SEE ALSO

d.mon

NAME

paint – Description of hardcopy color output system for GRASS.

INTRODUCTION

The **paint** system allows the user to produce color hardcopy maps of vector, raster, and sites file data at any scale. For a discussion of the GRASS **paint** functions, see the manual entries for *p.chart, p.colors, p.icons, p.labels, p.map, p.screen*, and *p.select*.

PAINT DEVICES

The GRASS paint system supports multiple color printers using a *device* driver concept. The paint (p.) functions listed above send graphics requests to device-dependent paint drivers. These drivers translate the application requests into device-dependent requests to produce hardcopy maps.

INSTALLING A PAINT DRIVER

A number of paint drivers have been distributed with GRASS. The installation of a driver is a 2 step process. The first involves identifying the driver(s) which correspond to printer(s) connected to your system and compiling those drivers. The second involves telling each driver which i/o port it is to use.

1 The source code for the drivers lives in $GISBASE/../src/paint/Drivers (The variable GISBASE refers to the directory in which GRASS is installed on your system.)

  The selection and compilation of the drivers is done when GRASS is compiled as a whole.

2 The port configuration is handled using the UNIX *ln* command.

  Each driver expects to send its output to /dev/*driver*. For example, the **tek4695** driver expects to find a tektronix 4695 (or 4696) printer on /dev/tek4695.

  Suppose the printer is actually on /dev/tty10. Then, a link named /dev/tek4695 is made to /dev/tty10:

```
ln /dev/tty10 /dev/tek4695
```

NOTES

There are 2 drivers which do not use i/o ports. One is the **preview** driver, which sends its output to the graphics screen instead of a hardcopy printer. This driver is very handy and should definitely be compiled on your system.

The other is the **null** driver, which is used for debugging purposes and probably should not be compiled on your system.

If you compile either of these drivers, you shouldn't create a /dev file for them.

SEE ALSO

p.chart, p.colors, p.icons, p.labels, p.map, p.screen, and *p.select*

AUTHOR

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

NAME

sites.S – Description of the GRASS *s.menu* "S" interface option.

DESCRIPTION

This describes the GRASS *s.menu* "S" statistical package interface option. See also the manual entry for *s.menu*.

SPECIAL ADJUSTMENT

Due to the fact that "S" references all subscripts beginning with 1, and the GIS data begins with 0, it was necessary to add 1 to the category numbers. Therefore, category value 0 ("no data") becomes category value 1, category 1 becomes category 2, etc.

S DATA STRUCTURES

Below are descriptions of the data structures created by the interface. As a general comment, within "S", simply typing the data structure name will display the values contained in the structure. You may also find it interesting to display the data structures using the "S" function dput().

cat.histo, (cat.#.histo)

This structure contains the histogram for the categories in each map layer. The histogram contains the number of cells of each category which occur in the user's geographic region (see wind.n, et al., below). It is a 2-dimensional integer array (matrix). The first subscript references the map layer. The second subscript references the category value.

examples:

cat.histo[2,5] is the cell count for layer 2, category 5.
cat.histo[2,] is the full histogram for layer 2.

Note: since more than one layer may occur in the data, and the number of categories in each layer varies, it was necessary to create this structure with sufficient dimension to hold the largest category value for all layers. Histogram data for categories which do not occur are set to NA ("S" notation for "no data"). However, there is also an individual histogram structure for each layer: cat.1.histo, cat.2.histo, etc. These are simple vectors.

cat.name

This structure contains the names for the categories in each map layer. It is a 2-dimensional character array (matrix). The first subscript references the map layer. The second subscript references the category.

Examples:

cat.name[2,5] is the name of category 5 for layer 2.
cat.name[2,] are all category names for layer 2.

Note: since more than one layer may occur in the data, and the number of categories in each layer varies, it was necessary to create this structure with sufficient dimension to hold the largest category value for all layers. Names for categories which do not occur are set to ''. However, there is also an individual category name structure for each layer: cat.1.name, cat.2.name, etc. These are simple vectors.

site.data

This structure contains the data for each site. It is a 3-dimensional integer array. The first subscript references the categories which occurred at the site. The second subscript references the site. The third subscript references the layer.

Examples:

site.data[,5,3] is the data for site 5 in layer 3.

site.data[,2,] is the data for site 2 in all layers.
site.data[,,1] is all site data for layer 1.

Note: the size of the first dimension will be the number of cells in a site, the size of the second dimension will be the number of sites, the size of the third dimension will be the number of layers.

site.mode

Since "S" does not provide a statistical mode function, this structure contains the most frequently occurring category for each site in each layer. It is a 2-dimensional integer array (matrix). The first subscript references the site. The second subscript references the layer.

Examples:

site.mode[5,3] is the mode for site 5 in layer 3.
site.mode[2,] are the modes for site 2 in all layers.
site.mode[,1] are all site modes for layer 1.

Note: this 'mode' is not the strict definition of the mode. Since category 0 (which in "S" is category 1) represents "no data" in the GIS databases, it was excluded from the mode calculations (essentially as if it had been NA). For example, if the data for a site are 1 1 1 1 2 2 3 2 3 1 1, the mode will be 2.

layer.name

This structure contains the raster map layer names. It is a 2-dimensional array (i.e., a matrix). The first subscript references the map layer. The second subscript selects either the map layer name, or the map layer title.

Examples:

layer.name[3,1] is the name of layer 3.
layer.name[3]   is also the name of layer 3.
layer.name[2]   is the name of layer 2.
layer.name[3,2] is the title for layer 3.

location

This is a simple character vector giving the GRASS location from which the data was extracted.

mapset

This is a simple character vector giving the GRASS mapset.

nlayers

This is a simple integer giving the number of map layers.

nsites

This is a simple integer giving the number of sites.

site.e

This is a simple integer vector giving the geographic easting for each site.

site.n

This is a simple integer vector giving the geographic northing for each site.

site.name

This is a simple character vector giving the description for each site.

sitelist
This is a simple character vector giving the name and description of the site lists file from which the sites were taken.

wind.n, wind.s, wind.w, wind.e
These are simple real numbers giving the north, south, west, east of the mapset's current geographic region.

wind.res
This is a simple real number giving the GRASS database resolution (in meters).

S MACROS

You may find the following "S" macros helpful when referencing the 'site.data' and 'site.mode' structures, since they allow you to specify parameters:

```
MACRO site.data(site, layer)
({
        site.data[ , site, layer]
})
END

MACRO site.mode(site, layer)
({
        site.mode[site, layer]
})
END
```

MACRO USAGE

To select the site.data for all sites for layer 2:

```
?site.data(layer=2)
```

to select the site.data for site 4 in all layers:

```
?site.data(site=4)
```

to select the site.mode for site 10 in layer 1:

```
?site.mode(layer=1,site=10)
```

SORRY, BUT ...

These macros are not provided by the interface. To use these macros, you will have to type them into a text file and then, from "S", issue the command:

```
define("<file>")
```

SEE ALSO

s.db.rim, s.in.ascii, s.menu, s.out.ascii, s.surf.idw, sites.format, sites.occur, sites.report

NAME

sites.format – Description of GRASS site list file format.

DESCRIPTION

This is a description of the site list files found in the GRASS data base. The files are kept in the site_lists directory under the current mapset.

SAMPLE SITES LIST

Here is a sample site list:

```
name|sample
desc|sample site list
728220|5182440|27
727060|5181710|28
725500|5184000|29
719800|5187200|30
```

EXPLANATION

Each record of data begins with a named field describing the record type, followed by fields of information. The fields are separated by the pipe character |.

name

This field contains the name of the site list file, and is printed on all the reports.

It is permissible for this field to be missing, since the sites software will force this field to agree with the name of the site list file itself.

desc

This field contains a description of the site list file, and is printed on all the reports.

It is permissible for this field to be missing, in which case the site list will have no description.

point

Each site is described by a 'point' record. The fields for this record are:

<east> <north> <description>

<east> easting for the site
<north> northing for the site
<description> site description

The east and north fields, which are integers, must be present and valid or the record itself will be skipped by the sites software. The description field is optional, but recommended.

BUGS?

Invalid records will be (silently) ignored by the sites software. Other record types may be added in the future.

SEE ALSO

s.db.rim, s.in.ascii, s.menu, s.out.ascii, s.surf.idw, sites.S, sites.occur, sites.report

NAME

sites.occur – Description of the site occurrence report produced by *s.menu*.

DESCRIPTION

This is a discussion of the site occurrence report produced by *s.menu.*

SAMPLE OUTPUT

```
                    SITE OCCURRENCE REPORT
                 Yakima Firing Center, Washington


Location:  yakima
Mapset:    sample
Site List: arch_erosion - archeology (34 sites)

Analysis Region:
             north: 5199975.00
  west:  692975.00               east:  737975.00
             south: 5154975.00


==============================================================================

Layer: arch_survey        Archeologic Survey Areas (arch_survey)

cells in analysis region:    810000
sites in analysis region:        34

                                                              degrees
                         cells     %    expected actual   chi      of
Site Characteristics           cover   cover   sites    sites  square  freedom
--------------------       ------- ------- -------- ------ -------- -------
(  0) No data              764310                  11

(  1) Hartmann,phase 1       25821    56.5     19.2      6    9.088     1
(  2) Hartmann,phase 2       11032    24.1      8.2     14    4.084     1
(  3) Wapora,phase 3          8837    19.3      6.6      3    1.945     1
                           ------- ------- -------- ------ -------- -------
     Totals                  45690   100.0     34.0     23   15.117     2

==============================================================================
```

EXPLANATION

The "Site Occurrence Report" provides aggregate site information organized by raster map layers.

The report header contains the following information:

Location
: GRASS database location (e.g., yakima),

Mapset
: GRASS mapset (e.g., sample)

Analysis Region
: the rectangular area in which the analysis was run, reflecting the current geographic settings of the user's mapset.

Site List
: name and description of the site list used for the report and the number of sites in the list. See manual entry for *s.menu* for more discussion about these site lists.

The rest of the report is divided into reports for each raster map layers specified by the user. In this SAMPLE OUTPUT, only one map layer (arch_survey) was specified.

Certain information heads up each of these reports:

layer
: identifies the raster map layer along with the map layer title.

cells in analysis region
: the size of the analysis region (in cells).

sites in analysis region
: the number of sites in the analysis region.

For each map layer category which occurs in a cell falling within the boundaries of the analysis region, as well as totals for the map layer as a whole, various statistics are presented:

Site Characteristics
: These are the numbers (i.e., values) and names (i.e., descriptions) for each category (i.e., characteristic) which occurs in the map layer.

cells cover
: the number of cells in the analysis region having the category value (i.e., site characteristic).

% cover
: the predominance of the category within the analysis region.

expected sites
: the number of sites in the analysis region which would be expected to be observed having the category value if all sites were evenly distributed by in the region.

actual sites
: the number of sites in the analysis region which were occurred in cells having the category value.

chi square
: gives the user a statistical test of the hypothesis that ´site distribution is dependent on environmental factors'. For this hypothesis to be true, the chi-square test must fail since we are calculating expected sites according to the hypothesis that ´sites are randomly distributed'.

degrees of freedom
: Number of degrees of freedom for the chi-square test. Each category value will have 1 degree of freedom, and can be tested alone. The entire map layer will have 1 less degree of freedom than the number of categories.

NOTE

Category zero (0) throughout GRASS represents "no data". Therefore, it is reported, but not included in the totals or in the computation of the chi-square statistics.

SEE ALSO

s.db.rim, s.in.ascii, s.menu, s.out.ascii, s.surf.idw, sites.S, sites.format, sites.report

NAME

sites.report – Description of the machine readable report output by *s.menu*.

DESCRIPTION

This is a description of the machine readable report produced by the *s.menu* program.

SAMPLE REPORT

```
location|yakima|Yakima Firing Center, Washington
mapset|sample
region north|5199975.000000
region south|5154975.000000
region west|692975.000000
region east|737975.000000
resolution|50.000000
site list|sample|sample site list

point|1|728220|5182440|27
point|2|727060|5181710|28
point|3|725500|5184000|29
point|4|719800|5187200|30

layer|1|arch_survey|Archeologic Survey Areas (arch_survey)
layer|2|vegetation|Vegetation Study Plots

cat|1|0|764310|No data
cat|1|1|25821|Hartmann,phase 1
cat|1|2|11032|Hartmann,phase 2
cat|1|3|8837|Wapora,phase 3
cat|2|0|774333|no data
cat|2|1|9299|stiff sagebrush/sandberg bluegrass type (S-1)
cat|2|2|732|big sagebrush/sandberg bluegrass type (S-2)
cat|2|3|397|buckwheat/sandberg bluegrass type (S-3)
cat|2|4|2526|bluebunch wheatgrass/sandberg wheatgrass type (S-4)
cat|2|5|20184|big sagebrush/bluebunch wheatgrass type (L-1)
cat|2|6|82|big sagebrush/needle-and-thread type (L-2)

matrix size|9

matrix|-1|-1|0|-1|1|-1
matrix|-1|0|0|0|1|0
matrix|-1|1|0|1|1|1

data|1|1|0|0|0|0|0|0|0|0|0
data|1|2|1|1|1|1|1|1|1|1|1
data|1|3|1|1|1|1|1|1|1|1|1
data|1|4|1|1|1|1|1|1|1|1|1
data|2|1|0|0|0|0|0|0|0|0|0
data|2|2|0|0|0|0|0|0|0|0|0
data|2|3|4|4|4|0|0|0|0|0|0
data|2|4|1|5|5|5|5|5|5|5|5
```

EXPLANATION

The machine readable report file consist of lines of text, with fields that are separated by the pipe character |. The first field on a line defines the record type, and the remaining fields are defined according to the record type.

location

This record gives the GRASS database location.

location <name> <full name>

<name> database location
<full name> long name for the location

mapset

This record gives the GRASS mapset.

mapset <mapset>

<mapset> mapset under which the report was run

region

The region records give the mapset region coordinates.

region north <N>
region south <S>
region west <W>
region east <E>

<N> north for current region (float)
<S> south for current region (float)
<W> west for current region (float)
<E> east for current region (float)

resolution

This record gives the GRASS database resolution in meters.

resolution <res>

<res> resolution of the database, in meters (float)

point

These records describe the sites that were used in the report. There is one 'point' record for each site.

point <site_ref> <E> <N> <desc>

<site_ref> a reference number, used by other lines
in the report to reference this site
<E> site easting
<N> site northing
<desc> a description of the site (usually an site number)

layer

These records describe the map layers that were used in the report. There is one 'layer' record for each layer.

layer <layer_ref> <name> <full_name>

<layer_ref> a reference number, used by other lines
in the report to reference this layer
<name> name of the map layer (raster file name)
<full name> full name (title of raster file)

cat

These records give information about each category in each layer.

cat <layer_ref> <cat> <count> <name>

<layer_ref> reference number of the layer
<cat> category number
<count> number of times this category occurred
<name> full name of the category

matrix size

This record gives the number of cells in each site.

matrix size <matrix size>

<matrix size> number of cells per site.

matrix

These records describe the locations of each cell in the site relative to the center of the site.

matrix <ew_disp> <ns_disp> ...

<ew_disp> east-west displacement from the site
<ns_disp> north-south displacement from the site

The displacements are in units of 1 cell, which is equal to the resolution of the database. They are paired, and there will be <matrix size> pairs. To compute the coordinates for the cell:

north = site_north - <ns_disp> * resolution
east = site_east + <ew_disp> * resolution

data

This is the site data. There is one 'data' record for each site for each layer. The <cat> field is repeated <matrix size> times.

data <layer_ref> <site_ref> <cat> ...

<layer_ref> reference number of the layer
<site_ref> reference number of the site
<cat> category which occurred

SEE ALSO

s.db.rim, s.in.ascii, s.menu, s.out.ascii, s.surf.idw, sites.S, sites.format, sites.occur

6 INDEX

The index lists main, alpha, and shell commands, and format descriptions.