

GRASS Raster Commands

(a * indicates that further updating to 5.x is required)

r.agnps50.input	r.in.tang	r.recode
r.agnps50.run	r.in.tiff	r.random.cells
r.agnps50.view	r.in.utm	r.random.surface
r.answers	r.infer	r.rational.regression
r.average	r.info	r.random
r.basins.fill	r.kappa	r.reclass
r.bilinear	r.kineros	r.region
r.binfer	r.lags	r.report
r.buffer	r.le.dist	r.resample
r.cats	r.le.null	r.rescale
r.circle	r.le.patch	r.rescale.eq
r.clump	r.le.pixel	r.rescale.inf
r.cn	r.le.rename	r.ros
r.coin	r.le.setup	r.slope.aspect
r.colors	r.le.trace	r.spread
r.colors.paint	r.line	r.spreadpath
r.combine	r.linear.regression	r.statistics
r.composite	r.los	r.stats
r.compress	r.mapcalc (*)	r.sum
r.contour	r.mask	r.sun
r.cost	r.mask.points	r.sunmask
r.covar	r.median	r.support (*)
r.cross	r.mfilter	r.surf.area
r.describe	r.mode	r.surf.contour
r.digit	r.moran	r.surf.fractal
r.direct	r.neighbors	r.surf.gauss
r.distance	r.nntool	r.surf.idw
r.drain	r.null	r.surf.idw2
r.featt.thin	r.out.agnps	r.surf.random
r.fill.dir	r.out.arc	r.texture
r.fillnulls	r.out.ascii	r.thin
r.flow	r.out.bin	r.timestamp
r.flowmd	r.out.elas	r.to.gnuplot
r.grow	r.out.gridatb	r.to.sites
r.his	r.out.hdf	r.topidx
r.hydro.CASC2D (*)	r.out.mpeg	r.topmodel
r.in.arc (*)	r.out.pov	r.transect
r.in.ascii	r.out.ppm	r.tribs
r.in.bin	r.out.rlc	r.univar
r.in.dem	r.out.tga	r.volume
r.in.doq	r.out.tiff	r.water.fea
r.in.dted	r.out.xyz	r.water.outlet
r.in.elas	r.param.scale	r.watershed
r.in.gdal	r.patch	r.weight
r.in.gridatb	r.plane	r.weight2
r.in.hdf	r.poly	r.weighted.cn

GRASS Raster Commands

[r.in.ll](#)

[r.in.miads](#)

[r.in.pbm](#)

[r.in.poly](#)

[r.in.pbm](#)

[r.in.pgm](#)

[r.in.ppm](#)

[r.in.shape](#)

[r.in.sunrast](#)

[r.profile](#)

[r.proj](#)

[r.quant](#)

[r.what](#)

[r.wrast](#)

Other Commands

[help](#), [home](#), [database](#), [display](#), [drivers](#), [general](#), [grid3d](#), [imagery](#), [import](#), [misc](#), [models](#), [paint](#), [photo](#), [postscript](#), [raster](#), [scripts](#), [sites](#), [vector](#)





NAME

r3.in.ascii – Imports files in 3D ASCII format into G3D.
(GRASS 3D Program)

SYNOPSIS

r3.in.ascii [*type* = *default* | *double* | *float*] [*precision* = *default* | *max* | *0 – 52*] [*compression* = *default* | *rle* | *none*] [*tiledimension* = *XxYxZ*] [*nv* = *none* | *double*] **input** = *ascii-file* **output** = *g3d-map*

DESCRIPTION

Parameters:

type

Data type used in the output file
Options: default, double, float

precision

Precision used in the output file
Options: default, max, 0–52

compression

Note that the *none* option only specifies that neither LZW nor RLE is used for compression. It does not turn off the compression all together. G3D does not support non-compressed files.
Options: default, rle, none

tiledimension

The dimension of the tiles used in the output file. The format is: XxYxZ

nv

Specifies which value to convert to NULL-value. If the specified value is *none*, no conversion is performed. Default is *none*.

input

Path and name of ASCII file to be imported

output

Name of the G3D output map

NOTES

The format for the ASCII file:

```
north: floating point
south: floating point
east: floating point
west: floating point
top: floating point
```

GRASS Raster Commands

bottom: *floating point*
rows: *integer*
cols: *integer*
levels: *integer*

This header is followed by the cell values in *floating point* format organized in rows with constant *col* and *level* coordinate. The rows are organized by constant *level* coordinate. Individual cell values are separated by *space* or *CR*.

NOTE: Currently, after the file has been imported, the stored values are compared with the original data. This feature is used to find bugs in the library at an early stage and will be turned off as soon as confidence has built up.

AUTHORS

Roman Waupotitsch, Michael Shapiro, Helena Mitsova, Bill Brown, Lubos Mitas, Jaro Hofierka

SEE ALSO

[*r3.out.ascii*, *s.to.rast3*, *GRASS ASCII formats*](#)

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.info – Displays information about *map*.
(GRASS 3D Program)

SYNOPSIS

r3.info
r3.info [*grid3* = *g3d-map*]

DESCRIPTION

Command line and interactive versions of *r3.info* are supported.

Parameter:

grid3
Name of the G3D map that the user seeks information on

SEE ALSO

[*r.info*](#)

AUTHORS

Roman Waupotitsch, Michael Shapiro, Helena Mitsova, Bill Brown, Lubos Mitas, Jaro Hofierka

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.in.grid3 – Imports files in Bill Brown's *grid3* format into G3D format.
(GRASS 3D Program)

SYNOPSIS

r3.in.grid3 [*type* = *default* / *double* / *float*] [*precision* = *default* / *max* / *0 – 52*] [*compression* = *default* / *rle* / *none*] [*tiledimension* = *XxYxZ*] [*nv* = *yes* / *no*] **input** = *grid3-file* **output** = *g3d-map*

DESCRIPTION

Parameters:

type

The default cell-type is float.
Options: default, double, float

precision

Options: default, max, 0–52

compression

Note that the *none* option only specifies that neither LZW nor RLE is used for compression. It does not turn off the compression all together. G3D does not support non-compressed files.
Options: default, rle, none

tiledimension

Format: XxYxZ

nv

Specifies whether zeros in the grid3 file are converted to NULL-values. Conversion is performed if the value is *yes*. Default is *no*.

input

Path and name of grid3 file to be imported

output

Name of the G3D output map

NOTES

r3.in.grid3 uses *float* as default cell-type.

Currently, after the file has been imported, the stored values are compared with the original data. This feature is used to find bugs in the library at an early stage and will be turned off as soon as confidence has built up.

AUTHORS

Roman Waupotitsch, Michael Shapiro, Helena Mitsova, Bill Brown, Lubos Mitas, Jaro Hofierka

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.in.v5d – Imports files in v5d format to G3D format.
(GRASS G3D Program)

SYNOPSIS

r3.in.v5d *input=name* *output=name* [*nv=name*] [*type=name*] [*precision=name*] [*compression=name*] [*tilerdimension=name*]

DESCRIPTION

[Vis5D](#) is a system for interactive visualization of large 5-D gridded data sets such as those produced by numerical weather models. One can make isosurfaces, contour line slices, colored slices, volume renderings, etc of data in a 3-D grid, then rotate and animate the images in real time. There's also a feature for wind trajectory tracing, a way to make text annotations for publications, support for interactive data analysis, etc. **r3.in.v5d** imports 3-dimensional files (i.e. the v5d file with 1 variable and 1 time step). Otherwise, only first variable and timestep from 4/5D v5d file will be imported.

Parameters:

input

Path and name of v5d file to be imported

output

Name of the G3D output raster map

nv

String representing NULL value data cell (use 'none' if no such value)
Default: none

type

Data type used in the output file
Options: default, double, float
Default: default

precision

Precision used in the output file
Options: default, max, 0-52
Default: default

compression

The compression method used in the output file
Options: default, rle, none
Default: default

tilerdimension

The dimension of the tiles used in the output file
Default: default

SEE ALSO

[r3.out.v5d](#)

AUTHOR

Jaro Hofierka, GeoModel s.r.o., Slovakia

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.mapcalc – G3D grid volume data calculator.
(GRASS 3D Program)

SYNOPSIS

r3.mapcalc [*result=expression*]

DESCRIPTION

r3.mapcalc performs arithmetic on 3D grid volume data. New 3D grids can be created which are arithmetic expressions involving existing 3D grids, floating point constants, and functions.

PROGRAM USE

If used without command line arguments, *r3.mapcalc* will read its input, one line at a time, from standard input (which is the keyboard, unless directed from a file or across a pipe). Otherwise, the expression on the command line is evaluated. *r3.mapcalc* expects its input to have the form:

result=expression

where *result* is the name of a 3D grid to contain the result of the calculation and *expression* is any legal arithmetic expression involving existing 3D grid, 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 *r3.mapcalc* by the user is recorded both in the *result* grid 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

GRASS Raster Commands

-	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.

3D GRID NAMES

Anything in the expression which is not a number, operator, or function name is taken to be a 3D grid name. Examples:

```
volume
x3
3d.his
```

Most GRASS raster map layers and 3D grids meet this naming convention. However, if a 3D grid 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 3D grid named *a-b*

Also

```
x = 3107
```

would create *x* filled with the number 3107, while

```
x = "3107"
```

would copy the 3D grid *3107* to the 3D grid *x*.

Quotes are not required unless the 3D grid 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.

r3.mapcalc will look for the 3D grids 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 3D grid. This is done by specifying the 3D grid 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 *r3.mapcalc*.)

THE NEIGHBORHOOD MODIFIER

3D grids are data base files stored in voxel format, i.e., three-dimensional matrices of float/double values. In *r3.mapcalc*, 3D grids may be followed by a *neighborhood* modifier that specifies a relative offset from the current cell being evaluated. The format is *map[r,c,d]*, where *r* is the row offset, *c* is the column offset and *d* is the depth offset. For example, *map[1,2,3]* refers to the cell one row below, two columns to the right and 3 levels below of the current cell, *map[-3,-2,-1]* refers to the cell three rows above, two columns to the left and one level below of the current cell, and *map[0,1,0]* 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 3D grid or across multiple 3D grids.

FUNCTIONS

The functions currently supported are listed in the table below.

function	description
abs(x)	return absolute value of x
atan(x)	inverse tangent of x (result is in degrees)
cos(x)	cosine of x (x is in degrees)
col()	return current column
depth()	return current depth
eval([x,y,...],z)	evaluate values of listed expr, pass results to z
exp(x)	exponential function of x
exp(x,y)	x to the power y
ewres()	east-west resolution from WIND3D
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
isnull(x)	1 if x not zero, 0 otherwise
log(x)	natural log of x
log(x,b)	log of x base b
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
mode(x,y[,z...])	most frequently value of those listed
null()	return 0
nsres()	north-south resolution from WIND3D
rand(x,y)	random value between x and y
round(x)	round x
row()	return current row
sin(x)	sine of x (x is in degrees)

GRASS Raster Commands

<code>sqrt(x)</code>	square root of x
<code>tan(x)</code>	tangent of x (x is in degrees)
<code>tbres()</code>	top-bottom resolution from WIND3D
<code>x()</code>	return current x value
<code>y()</code>	return current y value
<code>z()</code>	return current z value

Note, that the `row()`, `col()` and `depth()` indexing starts with 1.

EXAMPLES

To compute the average of two 3D grids *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 3D grid *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 3D grid *b* by 3D grid *a*:

```
result = if(a,b)
```

REGION/MASK

The user must be aware of the current geographic region and current mask settings when using *r3.mapcalc*. All 3D grids are read into the current geographic region masked by the current mask. If it is desired to modify an existing 3D grid without involving other 3D grids, the geographic region should be set to agree with the cell header for the 3D grid. For example, suppose it is determined that the *volume* 3D grid must have each category value increased by 10 meters. The following expression is legal and will do the job:

```
new_volume = volume + 10
```

Since a category value of 0 is used in GRASS for locations which do not exist in 3D grid, the new 3D grid will contain the category value 10 in the locations that did not exist in the original volume. 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 3D grid is masked when it is written; i.e., 0 category values in the mask force zero values in the output.

NOTES

GRASS Raster Commands

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 `r3.mapcalc` would see something other than the `*`.

If the input comes directly from the keyboard and the `result` 3D grid exists, the user will be asked if it can be overwritten. Otherwise, the `result` 3D grid 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 3D grid 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 `\e` to the end of an input line, `r3.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 `r3.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 `r3.mapcalc` non-interactively on the command line, the program will not warn the user not to overwrite existing 3D grids. Users should therefore take care to assign program outputs 3D grid 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 `r3.mapcalc` will be meaningless and the equation will not work as the user intended.

Error messages produced by `r3.mapcalc` are almost useless. In future, `r3.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 `r3.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*](#)

AUTHOR

Tomas Paudits & Jaro Hofierka, funded by GeoModel s.r.o., Slovakia
tpaudits@mailbox.sk, hofierka@geomodel.sk

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.mask – Creates a new 3D–mask file.
(GRASS 3D Program)

SYNOPSIS

r3.mask
r3.mask [**maskvalues**=*val*[-*val*][,*val*[-*val*],...]] **grid3d** = *g3d–map*

DESCRIPTION

File *map* is used as reference file. Cells in the mask are marked as "mask out" if the corresponding cell in *map* contains a value in the range specified with *maskvalues*.

Before a new 3d–mask can be created the existing mask has to be removed with *g.remove*.

Parameters:

maskvalues
 Values specified to be masked out

grid3d
 Name of G3D–map that is used as the mask reference

SEE ALSO

[*g.remove*, *r.mask*](#)

AUTHORS

Roman Waupotitsch, Michael Shapiro, Helena Mitsova, Bill Brown, Lubos Mitas, Jaro Hofierka

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.mkdspf – Creates a display file from an existing grid3 file according to specified threshold levels.
(GRASS 3D Program)

SYNOPSIS

```
r3.mkdspf [-qf] grid3=name dspf=name [levels=value[,value,...]] [min=value] [max=value] [step=value]
[tnum=value]
```

DESCRIPTION

Creates a display file from an existing grid3 file according to specified threshold levels. The display file is a display list of polygons that represent isosurfaces of the data volume. If specific *levels* are given, additional optional parameters are ignored. *Min* or *max* may be used alone or together to specify a sub-range of the data. The *step* parameter is given precedence over *tnum*.

Flags:

- q** Suppress progress report & min/max information
- f** Use flat shading rather than gradient

Parameters:

- grid3** Name of an existing 3dcell map
- dspf** Name of output display file
- levels** List of thresholds for isosurfaces
- min** Minimum isosurface level
- max** Maximum isosurface level
- step** Positive increment between isosurface levels
- tnum** Number of isosurface threshold levels
Default: 7

Example:

With grid3 data (*phdata*) in the range 3–7, we only want to see isosurface values for the range 4–6. Any of these commands will produce the same results:

```
r3.mkdspf phdata dspf=iso min=4.0 max=6.0 tnum=5
r3.mkdspf phdata dspf=iso levels=4.0,4.5,5.0,5.5,6.0
r3.mkdspf phdata dspf=iso min=4.0 max=6.0 step=0.5
```

NOTE

Currently the grid3 file must be in the user's mapset since the display files being created are specific to particular grid3 files and are contained in directories under them. We should create a mechanism where users may make display files from others' grid3 files without having to copy them to their mapset.

AUTHOR

Bill Brown, bbrown@gis.uiuc.edu

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.null – Modifies the NULL values of *map*.
(GRASS 3D Program)

SYNOPSIS

r3.null [**setnull=***val[-val][,val[-val],...]*] [**null=***double*] **grid3** = *g3d-map*

Parameters:

setnull

The cell values specified in the range are to be set to NULL values. A range can be a single value (e.g., 2.5) or multiple values (e.g., 3.21–10.95).

null

Value that the existing NULL values in *map* are converted to. This applies only to existing NULL values, and not to the NULLs created by the *setnull* argument.

grid3

Name of the G3D map for which to modify null values

SEE ALSO

[*r.null*](#)

AUTHORS

Roman Waupotitsch, Michael Shapiro, Helena Mitasova, Bill Brown, Lubos Mitas, Jaro Hofierka

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.out.ascii – Outputs G3D maps in ASCII format.
(GRASS 3D Program)

SYNOPSIS

r3.out.ascii [-h] *grid3=name* [**output=name**] [**dp=value**] [**null=name**]

DESCRIPTION

Outputs *G3D* maps in *ascii* format. *map* is a valid G3D map in the current mapset. *output* is the name of an *ascii* file which will be written in the current working directory. If *output* is not specified then **stdout** is used. The *-h* flag may be used to suppress header information. The module is sensitive to region settings (set with *g3.region*).

Flag:

-h
Suppress printing of header information

Parameters:

grid3
G3d raster map to be converted to *ascii*

output
Name for *ascii* output file

dp
Number of decimal places for floats (options: 0–20)
Default: 8

null
Char string to represent no data cell
Default: *

NOTES

The default format for the *ascii* file is equivalent to that required by *r3.in.ascii*. In particular, files output by *r3.out.ascii* with header information may be converted back to G3D maps with *r3.in.ascii*.

The format for the *ascii* file is:

The header is followed by cell values in *floating point* format. Cell values are output as a series of horizontal slices in row-major order. That is, One level maps can be imported with r.in.ascii (Raster 2D) after removing the header lines "top", "bottom" and "levels".

SEE ALSO

[r3.in.ascii](#)

[g3.region](#)

AUTHORS

Roman Waupotitsch, Michael Shapiro, Helena Mitasova, Bill Brown, Lubos Mitas, Jaro Hofierka

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.out.v5d – Exports G3D grids to v5d format (VIS5D).
(GRASS 3D Program)

SYNOPSIS

r3.out.v5d [-m] *grid3=name* [*output=name*]

DESCRIPTION

Exports *G3D* grids to *v5d* format. *map* is a valid G3D grid in the current mapset. *output* is the name of a v5d file which will be written in the current working directory. [VIS5D](#) is a system for interactive visualization of large 5-D gridded data sets such as those produced by numerical weather models. One can make isosurfaces, contour line slices, colored slices, volume renderings, etc of data in a 3-D grid, then rotate and animate the images in real time. There's also a feature for wind trajectory tracing, a way to make text anotations for publications, support for interactive data analysis, etc.

Flags:

-m
Use map coordinates instead of xyz coordinates

Parameters:

grid3
G3d grid map to be converted to v5d

output
Name for v5d output file

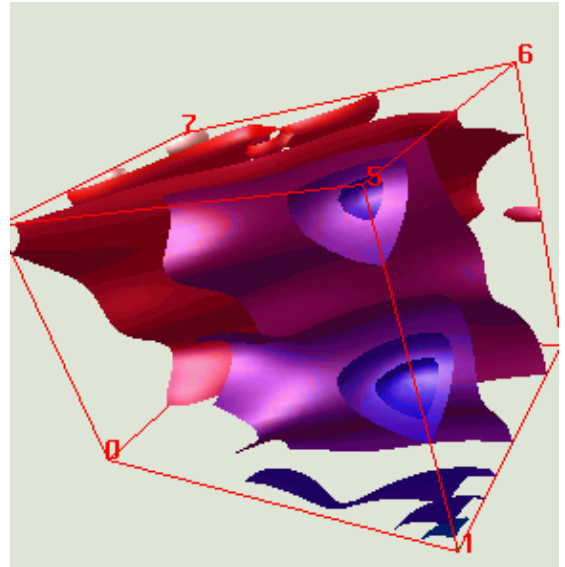
SEE ALSO

[r3.in.v5d](#)

AUTHOR

Jaro Hofierka, GeoModel s.r.o., Slovakia

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.showdspf – Visualization program which loads the isosurfaces previously calculated using *r3.mkdspf* and displays them according to commands given at the prompt.
(GRASS 3D Program)

SYNOPSIS

```
r3.showdspf grid3=name dspf=name [color=name]
```

Parameters:

grid3
Name of an existing 3dcell map

dspf
Name of an existing display file

color
Name of existing color table

DESCRIPTION

Visualization program which loads the isosurfaces previously calculated using *r3.mkdspf* and displays them according to commands given at the prompt. *r3.mkdspf* creates a *dspf* file from the 3D raster and *r3.showdspf* uses this *dspf* file to draw isosurfaces and *g3d* file to draw planes and everything related (boxes, etc). Upon initialization of the program, two graphics windows are opened, one for the color table and the other for

GRASS Raster Commands

data display. The display window initially contains a red bounding box. Command options are then printed to the terminal and user is prompted for drawing instructions:

THE INTERACTIVE OPTIONS ARE:

```
?, (l #), L, (t #), (T # #), I, +, -
(x #) (y #) (z #) r (X #) (Y #) (Z #)
(B(x,y,z)#), (E(x,y,z)#), S, R, F, C, c, s, b, g, n, p[#], d, D, w, Q, h
```

USAGE AND MEANING:

```
?          lists available thresholds
l index# [index#...] add threshold to display list
L          Draw current display list
t index#  reset so only this threshold is displayed
T index# index#  show thresholds between hi & lo
I          toggle thresholds INSIDE hi/lo or OUTSIDE hi/lo
+(+++)  
-(- -)    display thresholds with consecutively increasing index#
          display thresholds with consecutively decreasing index#

x int#   absolute rotation around x-axis in degrees(int)
y int#   absolute rotation around y-axis in degrees(int)
z int#   absolute rotation around z-axis in degrees(int)
r         rotate_model
X int#   scale model in x
Y int#   scale model in y
Z int#   scale model in z

B(x,y,z)int# begin display along (x,y,z) axis at #
E(x,y,z)int# end display along (x,y,z)axis #
S int#   specular highlight control
R        resets display along axis to show all data
F grid3name colortablename load new color file

C        toggles the clear flag
c        clears the display (no thresholds)
s        swap buffers
b        toggles draw a box
g        toggles grid
n        toggle surface normal direction

p        draw all walls
p#       draw a wall: 1-top, 2-bottom, 3-east, 4-west, 5-north, 6-south

d        draw (implement the option)
D        draw a solid defined by T(isosurface + parts of walls)

w        dump image to a file
Q        QUIT
h        help
```

```
enter desired manipulations then press return
>>
```

Hints:

- To navigate around the data, use the **r** command, then place the mouse pointer in the graphics window and drag with the left mouse to rotate the bounding box. To zoom in and out, drag right or left with the middle mouse. When satisfied with the new viewing position, click with the right mouse.

GRASS Raster Commands

- To quickly view a series of isosurfaces, enter a series of + or – characters, i.e. +++++++
- Scripts using above commands on separate lines may be directed to `r3.showdspf` as standard input. Use the # sign as the first character on a line to indicate a comment.

EXAMPLE

After generating a "dspf" control file with `r3.mkdspf` start `r3.showdspf`. Display/add the layers using "+". List available thresholds with "?". Use "1" to select isosurfaces (available number can be adjusted with `r3.mkdspf`) and "L" to display:

```
l 1 2 3 4 5
L
```

To select and display a single threshold (here: 2), use:

```
t 2
```

To select and display a range of thresholds (here: 3–5), use:

```
T 3 5
D
```

To draw a box, enter

```
p
the p# to plot a selected wall (here top wall):
p1
```

tp draw a cut-off box, define it's position

```
Ex20
p
```

Here Ex20 defines the x coordinate of the end of the box.

In general – p draws a side of a box, E, B, define where that box starts or ends, so to make a fence diagram, the user draws sides of a series of boxes which have their starting (or ending) side shifting by a given interval. (this way the user can draw even more complex fence diagrams which have perpendicular fences, by using Ey or By). It is sufficient to use only E or B depending whether fence are drawn by using the end side or front side of a box).

To draw a fence, a sequence like this would be needed

```
Ex10
p5
Ex15
p5
Ex20
p5
Ex25
p5
or the same would be
Bx10
p6
Bx15
p6
Bx20
p6
```

The *p* is needed for the fence diagram, solids and boxes.

SEE ALSO

[*r3.mkdspf*](#)

AUTHORS

Bill Brown, brown@gis.uiuc.edu

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.timestamp print/add/remove a timestamp for a grid3d map
(GRASS 3D Program)

SYNOPSIS

r3.timestamp
r3.timestamp help
r3.timestamp grid3d=name [date=timestamp],timestamp]

DESCRIPTION

This command has 2 modes of operation. If no date argument is supplied, then the current timestamp for the grid3d map is printed. If a date argument is specified, then the timestamp for the grid3d map is set to the specified date(s). See EXAMPLES below.

EXAMPLES

r3.timestamp grid3d=soils

Prints the timestamp for the "soils" grid3d map. If there is no timestamp for soils, nothing is printed. If there is a timestamp, one or two lines are printed, depending on if the timestamp for the map consists of a single date or two dates (ie start and end dates).

r3.timestamp grid3d=soils date='15 sep 1987'

Sets the timestamp for "soils" to the single date
"15 sep 1987"

r3.timestamp grid3d=soils date='15 sep 1987,20 feb 1988'

Sets the timestamp for "soils" to have the start date
"15 sep 1987" and the end date "20 feb 1988"

r3.timestamp grid3d=soils date=none

Removes the timestamp for the "soils" grid3d map

COMMAND LINE OPTIONS

Parameters *grid3d*: grid3d map name *date*: date/time stamp or date1,date2 range

TIMESTAMP FORMAT

The timestamp values must use the format as described in the GRASS datetime library. The source tree for this library should have a description of the format. For convenience, the formats as of Feb, 1996 are reproduced here:

There are two types of datetime values: absolute and relative. Absolute values specify exact dates and/or times. Relative values specify a span of time. Some examples will help clarify:

Absolute

The general format for absolute values is

```
day month year [bc] hour:minute:seconds timezone
```

day is 1–31

month is jan,feb,...,dec

year is 4 digit year

[bc] if present, indicates dates is BC

hour is 0–23 (24 hour clock)

minute is 0–59

second is 0–59.9999 (fractions of second allowed)

timezone is +hhmm or –hhmm (eg, –0600)

parts can be missing

```
1994 [bc]
```

```
Jan 1994 [bc]
```

```
15 jan 1000 [bc]
```

```
15 jan 1994 [bc] 10 [+0000]
```

```
15 jan 1994 [bc] 10:00 [+0100]
```

```
15 jan 1994 [bc] 10:00:23.34 [-0500]
```

Relative There are two types of relative datetime values, year– month and day–second. The formats are:

```
[–] # years # months
```

```
[–] # days # hours # minutes # seconds
```

The words years, months, days, hours, minutes, seconds are literal words, and the # are the numeric values.

Examples:

```
2 years
```

```
5 months
```

```
2 years 5 months
```

```
100 days
```

```
15 hours 25 minutes 35.34 seconds
```

```
100 days 25 minutes
```

```
1000 hours 35.34 seconds
```

The following are *illegal* because it mixes year–month and day–second (because the number of days in a month or in a year vary):

3 months 15 days

3 years 10 days

BUGS

Spaces in the timestamp value are required.

AUTHOR

Michael Pelizzari

Lockheed Martin Space Systems

based on r.timestamp by Michael Shapiro,

U.S.Army Construction Engineering Research Laboratory

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

r3.to.sites – Converts 3dcell values in a GRASS G3D volume into a GRASS site_lists file.
(GRASS Raster Program)

SYNOPSIS

```
r3.to.sites
r3.to.sites help
r3.to.sites grid3=name sites=name
```

DESCRIPTION

The *r3.to.sites* program extracts data from a GRASS 3dcell volume and stores output in a new GRASS *site_lists* file. 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 raster input 3dcell and a new site list file to be output on the command line. The program will be run interactively if the user types *r3.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*.

OPTIONS:

Parameters:

grid3
Name of an existing 3dcell volume from which site data are to be extracted

sites
Name of new sites file

SEE ALSO

[r3.out.ascii](#), [r3.out.v5d](#), [s.to.rast3](#), [parser](#)

AUTHOR

Jaro Hofierka

Last changed: \$Date: 2002/01/25 05:45:35 \$



NAME

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

SYNOPSIS

r.answers

DESCRIPTION

r.answers integrates ANSWERS (1991) with GRASS. ANSWERS (Areal Nonpoint Source Watershed Environmental Response Simulation) is an event oriented, distributed parameter model that was developed to simulate the behavior of watersheds having agriculture as their primary land use. Its primary applications are watershed planning for erosion and sediment control on complex watersheds, and water quality analysis associated with sediment associated chemicals.

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

```
ANSWERS on GRASS Project Manager Main Menu Project Name:
<sample>
```

```
Status Option Description
```

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

```
Option: 0__
```

Steps 1–11 record and display their status to the left of the step number. If a step has not been run, no status is displayed (as seen above). If the step has been successfully completed, the status will be listed as "done". In

some cases, a change in one step will cause the need to run another step again, in which case the status will read "rerun". If a step has a status of "done" or "rerun", if it is run again it will attempt to offer previous inputs as defaults.

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

- 1) Text input that can be completed by hitting the RETURN key. In most cases, if no text was entered, the given question or operation is canceled. Often times text input will consist of the name of a new or existing map or project name, in which case entering the word "list" will provide a list of currently used names.
- 2) Text or menu options that can be completed by hitting the ESC (escape) key. This type of interface is used for menus or for entering tables of parameters. All menus have a default answer of Exit (0), so that by simply hitting ESC one may leave the program's menus. The following keystroke guide is helpful to know when using the parameter entry worksheets that use this interface:
<RETURN> moves the cursor to next prompt field
<CTRL-K> moves the cursor to previous prompt field
<CTRL-H> moves the cursor backward non-destructively within the field
<CTRL-L> moves the cursor forward non-destructively within the field
<CTRL-A> writes a copy of the screen to a file named "visual_ask" in your home directory
<CTRL-C> where indicated (on bottom line of screen) can be used to cancel operation

Description of Main Menu Steps

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

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

Step 1 Set mask, region, and resolution

Map input: Watershed mask

Other inputs: Project resolution, project region (optional)

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

Step 2 Catalogue soils parameters

GRASS Raster Commands

Map input: Soils

Other input: Soils parameters, tile drainage coefficient, groundwater release fraction.

Description: This step prompts for the name of a soil map, then reads the map and lists all soil categories found in the watershed mask. For each soil found in the watershed, ANSWERS requires values for the parameters listed below. The Project Manager facilitates preparation parameters by input into a table.

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

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

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

Step 3 Catalogue land use and surface parameters

Map input: Land cover/use.

Other input: Land cover parameters

Description: For each category in the land use map found in the watershed, ANSWERS requires values for the parameters listed below. The Project Manager facilitates preparation parameters by input into a table.

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

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

Step 4 Identify elevation-based input layers

Map input: Slope and aspect.

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

Step 5 Prepare rain gauge data

Map input: Rain gauge areas (for multiple gauges)

Other input: Rain gauge data

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

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

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

Example rain gauge data input files:

one gauge		two gauges		
0	0	0	0	
10	3	11	1	data for gauge 1
20	10	25	7	
35	22	-1		delimiter
55	9	0	0	
67	4	15	6	data for gauge 2
100	0	10	4	

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

Step 6 Identify outlet cell

Map input: none

Other input: row and column number of watershed outlet element

Description: ANSWERS needs to know the row and column number of the element at the watershed outlet. To facilitate your finding this information, The raster map `<project name>.ELEMENT` has been created. The category values of this map are the sequentially numbered cells of the watershed. The category descriptions are the cell's row and column number. Using a tool such as **d.what.rast**, the row and column number of the outlet cell can be queried from the displayed element map.

Step 7 Specify areas with subsurface drainage

Map input: Areas with subsurface drainage (optional)

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

Step 8 Catalogue channel parameters

Map input: Channels

Other input: Channel width and roughness coefficient for each category of channel

Description: Watershed cells with a well-defined channel should be defined to ANSWERS. ANSWERS assumes the channel is rectangular in cross-section and is sufficiently deep to handle runoff.

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

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

Step 9 Define channel slopes

Map input: Channel slope

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

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

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

Step 10 Specify BMP's in watershed

Map input: Tile Outlet Terrace, Sedimentation Pond, Grassed Waterway, and/or Field Borders.

Other input: Grassed waterway or field border width (meters)

Description: This step provides a menu to prepare any or none of the four structural Best Management Practices (BMPs) that ANSWERS recognizes. Many BMPs can be described to ANSWERS by changing variables describing the surface condition of the soil. Practices which are tillage-oriented, for example, are described in the soils and land use sections. Gully structures such as a drop spillway may be simulated by reducing channel slope. On the other hand, BMPs which are structural in nature require a change in land use (row crop to grass for waterways, for example). ANSWERS recognizes four optional BMP structures. Even though the use of BMP structures is optional, this step still must be run to verify this. NOTE: Since ANSWERS will recognize one BMP for a given watershed element, the most effective BMP should be used. The following is a brief discussion of the BMPs:

1. ANSWERS Tile Outlet Terrace Assumptions:

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

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

2. ANSWERS Sedimentation Pond Assumptions:

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

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

3. ANSWERS Grassed Waterway Assumptions:

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

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

4. ANSWERS Field Border Assumptions

- The vegetated area within the affected element is no longer subject to any sediment detachment.

For each category found in the layer, you will be prompted for width of the field border.

Step 11 Prepare ANSWERS input and run simulation

Description: Steps 1–10 must have a status of "done" before this step can be run. (Even steps for optional inputs must be run before an ANSWERS input file can be completed.) Each of the prior steps will have prepared their part of the ANSWERS input. The first function of this step is to compile all the parts together. Once the input file is complete, the simulation can be run. (NOTE: r.answers will call the ANSWERS

program, which must be compiled as a part of the `r.answers` installation. The source code for ANSWERS should be part of the software distributed with `r.answers`.) The error messages that ANSWERS may send to "standard output" are captured to a file by `r.answers` and displayed. If none, a message to that effect will be printed to the screen (although this doesn't mean that the simulation ran entirely error-free). The primary output of the simulation is captured to another file, then processed to separate it into component parts of

- 1) *text* – the verbose reiteration of the inputs and summary of watershed characteristics. This is useful for checking to insure that inputs were read in by ANSWERS properly;
- 2) *outlet hydrograph data* of rainfall, runoff and sediment yield and concentration. If these data are in order, they will be processed into a format readable by the `d.linegraph` program for display;
- 3) *individual element net sedimentation* showing sediment loss or deposition, if any, for each raster element in the watershed.

Also, *sediment deposition in channel elements*. This step will prompt for names to use for new watershed maps it will create by extracting these data from the output. If the simulation event did not create sediment loss or deposition, or channel deposition for the scenario, the given map will not be created. To find out how to access the output files, see the description of step 12, below.

Step 12 Miscellaneous Command Menu

This step calls a menu that

- 1) allows access to the project files in the project database and to
- 2) a function that prepares a summary of the project's current status.

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

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

FILES

Index of ANSWERS on GRASS database files

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

General project data

```
reclass      reclass rules to create project MASK
region      project region coordinates
```

ANSWERS Input file

answers_input file created to use as input to ANSWERS

ANSWERS Output

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

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

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

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

ANSWERS Element data

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

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

ANSWERS Predata

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

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

Parameter data

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

chnl_data	channel parameters
cover_data	cover parameters
soil_data	soil parameters

SEE ALSO

Answers Manual
Answers Exercises
Demonstration scripts in the answer source tree

d.INTRO
[d.rast.edit](#)
[d.rast.num](#)
[d.what.rast](#)
[r.slope](#)
[r.fill.dir](#)
[r.direct](#)
[parser](#)

AUTHOR

Chris Rewerts, Agricultural Engineering, Purdue University

Last changed: \$Date: 2002/01/25 05:45:34 \$



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. 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

GRASS Raster Commands

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

output=name

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

EXAMPLE

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

cat	cat
value	label
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	cat
value	label
1	.1023
2	.1532
3	.172
4	.3872
5	.003
6	.28
7	.2345

NOTES

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

category	value
label	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](#)

[parser](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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](#).

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.

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.display](#)

[r.watershed](#)

[v.digit](#)

[parser](#)

AUTHORS

Dale White, Dept. of Geography, Pennsylvania State University
Larry Band, Dept. of Geography, University of Toronto, Canada

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.bilinear – bilinear interpolation utility for raster map layers.

SYNOPSIS

r.bilinear [-q] **input=name** **output=name** [**north=value**] [**south=value**]

DESCRIPTION

r.bilinear fills a grid cell (raster) matrix with interpolated values generated from a set of input layer data points. It uses the bilinear interpolation method, a simple algorithm usually applied only to completely defined raster areas (input data void of null data values). Here the values of 4 input cells are used to define an interpolation function of constant gradient within each rectangular area defined by the cell centers. User should be aware that the gradient of the interpolation functions changes discontinuously across lines intersecting the cell centers of the input raster.

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 procedure for selection of input data will consider all input data relevant to interpolating values at the cell centers of the current geographic region, ignoring the current mask. Note that cells of the output raster that cannot be bounded by 4 input cell centers are set to null.

The command line input is as follows:

Flags:

-q
specifies that *r.bilinear* run quietly (suppressing the printing of program messages to standard output)

Parameters:

input=name

Name of an input raster map layer containing data values to apply in the interpolation.

output=name

Name to be assigned to new output raster map that represents the surface generated from the data values in the input layer.

north=value

Input raster value for the north pole (90N). Valid for longitude–latitude grids only.

south=value

Input raster value for the south pole (90S). Valid for longitude–latitude grids only.

NOTES

The **north** and **south** parameters have been included to allow for specific input values to be assigned to the north and/or south poles for longitude–latitude grids. These data, if included, are used to interpolate values for cells that are north or south of a line intersecting the cell centers of the first or last row of input, respectively. When utilized, the interpolation procedure will be continuous from the north and/or south boundary of the current geographic region. This option is necessary, since the data structure defining a raster will not allow for data to be assigned to a cell centered at 90N or 90S. By including the option, the user can create output surfaces that are continuous between the poles. The interpolation will be "wrap-around" from west to east (across latitude) only if the input raster has an east edge identical to its west edge.

For longitude–latitude databases, the interpolation algorithm is based on degree fractions, not on the absolute distances between cell centers. Any attempt to implement the latter would violate the integrity of the interpolation method.

r.bilinear may be used in some instances as an alternative to the nearest neighbor approach inherent to *r.resample*. Note, however, that the extent of non–null data area of the output raster must be less than that of the input raster. The only exception to this occurs in the case where the **north** and **south** parameters are utilized for longitude–latitude rasters.

SEE ALSO

[*r.surf.idw*](#)

[*r.surf.idw2*](#)

[*g.region*](#)

[*r.resample*](#)

AUTHOR

Greg Koerper
ManTech Environmental Technology, Inc.
Global Climate Research Project
U.S. EPA Environmental Research Laboratory

Last changed: \$Date: 2002/02/25 13:03:54 \$



NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

OPTIONS

Flag:

-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

GRASS Raster Commands

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 greater 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.
```

GRASS Raster Commands

```
#
#
# 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
#
```

GRASS Raster Commands

```
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),
```

GRASS Raster Commands

```
# 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; # slope conditionals
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".
```

GRASS Raster Commands

```
# 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.
```

AUTHOR

Kurt Buehler, Civil Engineering, Purdue University

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.buffer – Creates a raster map layer showing buffer zones surrounding cells that contain non-NULL 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 (a.k.a. "distance" or "proximity") zones around all cells that contain non-NULL 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.

```

000000000000000000000000000000  22223333333333333333444444
111000000000000000000000000000  11122222222222223333333333
0001111111111100000000000000  22211111111111222233333333
0000000010000111000000000000  332222221222211122222222
000000001000000011111111  333333321233222211111111
0000000010000000000000000000  433333321233333222222222
0000000010000000000000000000  444443321233333333333333
0000000010000000000000000000  444443321233443333333333
0000000010000000000000000000  444443321233444444444444
Category 0: No roads
Category 1: Road location
Category 2: Buffer Zone 1 around roads
Category 3: Buffer Zone 2 around roads
Category 4: 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-NULL 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.]

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,...]
```

Flags:

-q
Run quietly

Parameters:

input=name

The name of an existing raster map layer whose non-NULL 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-NULL category value cell in the *input* map.

distances=value[,value,...]

The distance of each buffer zone from cells having non-NULL 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): 0-100, 101-200, 201-300, 301-400 and 401-500.

Format:

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

Example:

```
r.buffer input=roads output=roads.buf distances=100,200,300,400,500 units=meters
```

Result:

r.cats input=roads.buf

1	distances calculated from these locations
2	0-100 meters
3	100-200 meters
4	200-300 meters
5	300-400 meters
6	400-500 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-NULL 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 NULL.

SEE ALSO

[g.region](#)

[r.mapcalc](#)

[r.reclass](#)

AUTHORS

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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] [vals=value]

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.

Parameter:

vals=value

Comma separated value list: e.g. 1.4, 3.8, 13

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*](#)

[*d.what.rast*](#)

[*r.support*](#)

[*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.circle – Creates a raster map containing concentric rings around a given point.

SYNOPSIS

r.circle [-b] **output**=*name* **coordinate**=*x,y* [**min**=*value*] [**max**=*value*] [**mult**=*value*]

DESCRIPTION

This module creates a raster map containing concentric rings around a given point. The cell values are increasing linear from the center point to outer rings. Minimum and maximum radius can be selected. Optionally a "donut" ring can be created to use as (binary) filter for *i.iff* inverse Fourier transform (apply filter with *r.mask*).

OPTIONS

Flag:

-b
Generate binary raster map

Parameters:

*output=**name*
Name for new raster file

coordinate=*x,y*
The coordinate of the center (easting, northing)

*min=**value*
Minimum radius for ring/circle map (in cell units)

*max=**value*
Maximum radius for ring/circle map (in cell units)

*mult=**value*
Multiplier

EXAMPLES

Generate a circle at current map center with a radius of 300m:

```
EASTCENTER=`g.region -c | awk ' /region center easting:/ { print $4 }`
NORTHCENTER=`g.region -c | awk ' /region center northing:/ { print $4 }`
r.circle -b out=circle coordinate=$EASTCENTER,$NORTHCENTER max=300
```

SEE ALSO

[g.remove](#), [g.rename](#), [i.fft](#), [i.ifft](#), [r.mask](#)

AUTHOR

Bill Brown, U.S. Army Construction Engineering Research Laboratory
Additional flag/min/max parameter by Markus Neteler, University of Hannover

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.clump – Recategorizes data in a raster map layer by grouping cells that form physically discrete areas into unique categories.

(GRASS Raster Program)

SYNOPSIS

r.clump

r.clump help

r.clump [-q] **input**=name **output**=name [TITLE="string"]

DESCRIPTION

r.clump finds all areas of contiguous cell category values in the input raster map layer *name*. It assigns a unique category value to each such area ("clump") in the resulting output raster map layer *name*. If the user does not provide input and output map layer names on the command line, the program will prompt the user for these names, using the standard [parser](#) interface (see manual entry for *parser*). Category distinctions in the input raster map layer are preserved. This means that if distinct category values are adjacent, they will NOT be clumped together. (The user can run [r.reclass](#) prior to *r.clump* to recategorize cells and reassign cell category values.)

Flag:

-q

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

Parameters:

input=name

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

output=name

Name of new raster map layer to contain program output.

TITLE="string"

Optional TITLE for output raster map layer, in quotes. If the user fails to assign a TITLE for the *output* map layer, none will be assigned it.

ALGORITHM

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

NOTES

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

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

SEE ALSO

[*r.average*](#)

[*r.buffer*](#)

[*r.combine*](#)

[*r.grow*](#)

[*r.infer*](#)

[*r.mapcalc*](#)

[*r.mfilter*](#)

[*r.neighbors*](#)

[*r.poly*](#)

[*r.reclass*](#)

[*r.support*](#)

[*r.weight*](#)

[*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.cn – Generates a SCS curve number map layer
(GRASS Raster Program)

SYNOPSIS

r.cn

r.cn *sg=hy_soil_group lu=landuse_map pr=practice_map hc=hy_condition_map cn=cn_map
amc=AMC_number*

DESCRIPTION

r.cn is a command–line interface for generating an SCS CN (Curve Number) map. The command format is:

r.cn sg=hysoilgroup lu=landusemap pr=practicemap hc=hyconditionmap cn=cnmap amc=AMCnumber

Parameters:

sg=map

Hydrologic soil group map name

lu=map

Landuse map name

pr=map

Practice or management map name

hc=map

Hydrologic condition map name

cn=map

Output CN map name

amc=number

Equivalent AMC condition number for the CN output

The command–line ordering can be in any form but all key words must be there to run the program.

EXAMPLE

r.cn sg=hy.soil.group lu=landuse hc=hy.cond pr=veg.cover cn=cname amc=2
will create a CN map *cname* for equivalent AMC condition II

ATTENTION

r.cn uses information from the four map layers by reading the cell layer and its category files. So it is

GRASS Raster Commands

important to update the category files (text attributes) for all four layers and it expects the categories of the map layer exactly as shown below:

For the **hydrologic soil group** map the text categories may be either "A", "B", "C" or "D". You will reclassify an existing soil type map using **r.reclass** or **v.reclass**.

For the **landuse map** layer the text categories may be either "fallow", "row crops", "small grain", "close-seeded legumes", "rotation meadow", "pasture", "range", "meadow", "woods", "farmsteads", "roads (dirt)", "hard surface" or "water".

For the **practice** or management map layer the categories may be either "straight row", "contoured" or "contoured and terraced".

For the **hydrological condition** map the categories may be "poor", "fair" or "good". The rating (good, fair, poor) is based on a combination of factors that affect infiltration and runoff, which includes:

- density and canopy of vegetated areas
- amount of year-round ground cover
- amount of grass or close-seeded legumes in rotations
- percent of residue cover on the land surface (good is more than 20%)
- degree of surface roughness

AMC is the antecedent moisture condition. AMC I: dry, AMC II: normal, AMC III (wet). Specify this number during module run.

The category text labels can be applied to above maps using **r.support** ("Edit category file").

If the combination of the four layers categories does not exist in the SCS CN table, an error message is printed and the program quits.

NOTE

The *r.cn* program is sensitive to the current window setting. Thus the program can be used to generate a CN map of any sub-area within the full map layer. Also, *r.cn* is sensitive to any mask in effect.

REFERENCE

SCS Curve number method: [TR-55, Urban Hydrology for Small Watersheds](#)

SEE ALSO

[r.weighted.cn](#) [r.support](#) [r.reclass](#) [v.reclass](#)

AUTHORS

Raghavan Srinivasan and Dr. Bernie A. Engel, Agricultural Engineering Department, Purdue University
Extensions by Markus Neteler, University of Hannover, Germany (fixes for [AGNPS 5](#))

Last changed: \$Date: 2002/03/01 00:08:31 \$



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-iteratively 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 suppress 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

GRASS Raster Commands

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:

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

GRASS Raster Commands

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 |
| 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

```

+-----+
| cat# | owner |      |      | Panel Row Total |
|      |       | 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  |
+-----+

```

```

+-----+
| cat# | Table Row Total |
|      | 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 reclassified 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

[*g.region*](#)

[*m.ipf*](#)

[*r.cats*](#)

[*r.describe*](#)

[*r.mask*](#)

[*r.reclass*](#)

[*r.report*](#)

[*r.stats*](#)

AUTHORS

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

Last changed: \$Date: 2003/03/16 12:25:19 \$



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**] [**rast=name**]

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)
grey.log	(histogram logarithmic transformed grey scale)
byg	(blue through yellow to green colors)
byr	(blue through yellow to red colors)
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), *byg* (blue-yellow-green), *byr* (blue-yellow-red), *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.

GRASS Raster Commands

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
```

To set the NULL (no data) color, use the "nv" parameter:

```
0 black
10% yellow
nv white
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 be 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
```

 Option **rast** allows user to specify a raster map *name* from which to copy the color map.

SEE ALSO

[d.colormode](#)

[d.colors](#)

[d.colortable](#)

[d.display](#)

[d.legend](#)

[p.colors](#)

[r.support](#)

AUTHORS

Michael Shapiro and David Johnson

Last changed: \$Date: 2003/04/18 08:58:21 \$



NAME

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

SYNOPSIS

r.combine
r.combine < *inputfile*

DESCRIPTION

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

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

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

GRASS Raster Commands

COVER	existing raster map	sandstone
[cover]		

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.

GRASS Raster Commands

Resultant categories:

```
0 - black : no data occurred any 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
              OR --> 1 1 1 results
raster map 2  1 0 1
              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

GRASS Raster Commands

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.

GRASS Raster Commands

example:

(NAME *park.or.priv* (OVER red (GROUP 1 (owner))))

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

Resultant categories:

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

example:

(NAME *roads.or.not* (OVER *park.or.priv* yellow (GROUP 0 (roads))))

Category 0 in the map layer *roads* is overlain in yellow on top of the *park.or.priv* map layer created above.

The output is placed in a new map layer named *roads.or.not*.

Resultant categories in *roads.or.not* are:

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

example:

(NAME *low.elev* (OVER *park.or.priv* blue (GROUP 1-19 (elevation.255))))

The elevation categories of 1123 meters or less from the map layer *elevation.255* are assigned the color blue and then overlain on *park.or.priv* (generated in the previous example).

Resultant categories in the new map layer *low.elev* are:

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

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

COVER

(COVER existing_map (Expression))

Performs an *opaque* overlay operation. This means that where the top map layer contains "holes" (cell category values of 0), the bottom map layer will show through. Where the top map layer contains information on a feature, it will cover (substitute its category value for) whatever is below it. The top map layer is that which is defined by *Expression*. The bottom map layer is *existing_map*; this map layer must already exist. The user does not specify colors with COVER. COVER uses the default color table that is listed above with OVER. Colors are assigned starting with the lower map layer. The category values are assigned the color from the table that corresponds with that value. For example, 1 would be red; 2, yellow; 3, orange, etc. Moving to the upper map layer COVER starts wherever it left off after the lower one. If the highest value of the lower map layer was 5, then all non-zero (i.e., places where a feature exists) cells of the upper map layer would be assigned the value of 6 (green). Note that if, in this case, the upper map layer did not have any cells of value zero, then the entire resulting new map layer would be green. The upper map layer would have been assigned the value 6 and would have completely covered that which was below it.

This is what happens:

```
Expression          1 1 1 0
(top raster map)    1 1 0 0
```

GRASS Raster Commands

```
          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
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

GRASS Raster Commands

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

[*r.infer*](#)

[*r.mapcalc*](#)

[*r.weight*](#)

AUTHORS

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.composite – Converts three user-specified raster map layers into a composite RGB layer.
(GRASS Raster Program)

SYNOPSIS

r.composite

r.composite help

r.composite [-o] [-d] [-c] **r_map=name** **g_map=name** **b_map=name** [**levels=value**] [**lev_r=value**]
[**lev_g=value**] [**lev_b=value**] **output=name**

DESCRIPTION

This program combines three raster map layers to form a composite RGB layer. For each layer, the corresponding component from the layer's color table is used (e.g. for the red layer, the red component is used, and so on). In general, the layers should use a gray-scale color table.

OPTIONS

Flags:

- o** Overwrite output maps.
- d** Dither.
- c** Use closest color.

Parameters:

r_map=name

Name of raster map layer to be used for RED component.

g_map=name

Name of raster map layer to be used for GREEN component.

b_map=name

Name of raster map layer to be used for BLUE component.

levels=value

Number of levels to use for each component (default 32), except where overridden by the appropriate **lev_[rgb]** parameter.

lev_r=value

Number of levels to use for the RED component (overrides the **levels** parameter).

lev_g=value

Number of levels to use for the GREEN component (overrides the **levels** parameter).

lev_b=value

Number of levels to use for the BLUE component (overrides the **levels** parameter).

output=name

Name of raster map to contain results

NOTES

The default number of intensity levels for each component is 32, resulting in a total of 32768 possible colors (equivalent to 15 bits per pixel). If significantly more levels than this are used, not only will *r.composite* take longer to run, but displaying the resulting layer with [d.rast](#) will also be significantly slower.

Floyd–Steinberg dithering is optionally used.

SEE ALSO

[d.rast](#)

[d.rgb](#)

[r.colors](#)

[i.composite](#)

AUTHOR

Glynn Clements

Last changed: \$Date: 2003/08/25 07:49:30 \$



NAME

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

SYNOPSIS

```
r.compress
r.compress help
r.compress [-qu] 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.

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).

-q

Run quietly. Suppress printing of percent complete messages to standard output.

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](#)
[parser](#)

AUTHORS

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.contour – Produces a GRASS binary vector map of specified contours from GRASS raster map layer.
(GRASS Raster Program)

SYNOPSIS

r.contour

r.contour help

r.contour [-qn] **input**=name **output**=name [**levels**=value,value,...,value] [**minlevel**=value] [**maxlevel**=value]
[**step**=value] [**cut**=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.

OPTIONS

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 [-qn] input=name output=name [levels=value,value,...,value] [minlevel=value]  
[maxlevel=value] [step=value] [cut=value]
```

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

Flags:

- q Suppress progress report & min/max information
- n Suppress single crossing error messages

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.

cut=value

Minimum number of points for a contour line (0 -> no limit, default: 0)

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.

Zero 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.

The optional *cut* parameter allows to specify a minimum number of point for a contour line to be put into output vector map, so that small spurs, single points and so on won't be present and the output will be more clear.

AUTHORS

Terry Baker, U.S. Army Construction Engineering Research Laboratory
3/2001: cut parameter and fixes by Andrea Aime (aaime@libero.it)

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 [-v] [-k] [-n] **input=name** **output=name** [**start_sites=name**] [**stop_sites=name**] [**start_rast=name**]
[**coordinate=x,y[,x,y,...]**] [**stop_coordinate=x,y[,x,y,...]**] [**max_cost=cost**] [**null_value=value**]

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. The output map will be of the same data format than the input map, integer or floating point.

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.

Flags:

- v Run verbosely.
- k Use the 'Knight's move'; slower, but more accurate.
- n Keep null values in output map

Parameters:

input=name

Name of raster map containing grid cell cost information.

output=name

- Name of raster map to contain results.
- start_sites=name**
Starting points site file.
- stop_sites=name**
Stop points site file.
- start_rast=name**
Starting points raster file.
- coordinate=x,y[,x,y,...]**
The map E and N grid coordinates of a starting point (E,N).
- stop_coordinate=x,y[,x,y,...]**
The map E and N grid coordinates of a stopping point (E,N).
- max_cost=cost**
An optional maximum cumulative cost (default: 0).
- null_value=value**
Cost assigned to null cells. By default, null cells are excluded.

The **inputname** 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. These starting points could be read from a sites file through the **start_sites=name** option or from a raster map through the **start_rast=name** option. *r.cost* will stop cumulating costs when either *maxcost* is reached, or one of the stop points given with the **stop_coordinates=x,y** is reached. Alternatively, the stop points can be read from a site file through the **stop_sites=name** option. Both points read from a site file and those given on the command line will be processed. The null cells in the input map can be assigned a (positive floating point) cost with the *null_value* option.

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.

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, sites file or raster map. All non-NULL cells are considered to be starting points.

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

```

```

. . . . .
. . K . X . X . X . K . .
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .

```

-n

When input map null cells are given a cost with the **null_value** option, the corresponding cells in the output map are no longer null cells. With this option, the null cells of the input map are retained as null cells in the output map.

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.

start_sites=*name*

name is the name of a site file that holds the coordinates of starting points from which the transportation cost should be figured.

stop_sites=*name*

name is the name of a site file that hold the coordinates of stopping points. During execution, once the cumulative cost to all stopping points has been determined, processing stops.

coordinate=*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,...]*

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.

max_cost=*maxcost*

This is the cost limit that forces **r.cost** to restart the current point to be considered a stop point.

null_value=*value*

The optional value that will be assigned to the null cells in the input map. This is a positive floating point

value.

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):

```

  COST SURFACE
. . . . .
. 21. 21. 20. 19. 17. 15. 14.
. . . . .
. 20. 19. 22. 19. 15. 12. 11.
. . . . .
. 22. 18. 17. 17. 12. 11. 9.
. . . . .
. 21. 14. 13. 12. 8. 6. 6.
. . . . .
. 16. 13. 8. 7. 4| 0| 6.
. . . . . |_____| .
. 14. 9. 8. 9. 6. 3. 8.
. . . . .
  
```

Output (not using -k):

```

  CUMULATIVE COST SURFACE
. . . . . * * * * * . . . . .
. 22. 21* 21* 20* 17. 15. 14.
. . . . . * * * * * . . . . .
. 20. 19. 22* 20* 15. 12. 11.
. . . . . * * * * * . . . . .
. 22. 18. 17* 18* 13* 11. 9.
. . . . . * * * * * . . . . .
. 21. 14. 13. 12. 8. 6. 6.
. . . . .
. 16. 13. 8. 7. 4. 0. 6.
. . . . .
. 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).

NULL CELLS

By defaults, null cells in the input raster map are excluded from the algorithm, and thus retained on output.

If one wants **r.cost** to transparently cross the null cells, one must use the option **null_value=0.0**. Then, null cells just propagate adjacent costs. These cells could then be retained as null cells into the output map through the **-n** flag.

NOTES

Sometimes, when the differences among **integer** cell category values 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](#). This problem doesn't arise with floating point maps.

SEE ALSO

[g.copy](#)

[g.region](#)

[g.rename](#)

[r.drain](#)

[r.in.ascii](#)

[r.mapcalc](#)

[r.out.ascii](#)

[parser](#)

AUTHOR

Antony Awaida,
Intelligent Engineering
Systems Laboratory,
M.I.T.

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

Updated for Grass 5
Pierre de Mouveaux (pmx@audiovu.com)

Last changed: \$Date: 2003/07/04 08:35:34 \$



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):

```
1.000000  0.914922  0.889581
0.914922  1.000000  0.939452
0.889581  0.939452  1.000000
```

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.

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](#)

[parser](#)

AUTHOR

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

Last changed: \$Date: 2002/10/14 15:41:07 \$



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-actively 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.

Flags:

- q** Run quietly. Suppresses 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.covar*](#)

[*r.stats*](#)

[*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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[nv=string][nsteps=value]
```

DESCRIPTION

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

OPTIONS

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](#).

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

Flags:

-l

Print the output one value per line, instead of the default short form. In the above example, the **-l** 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 **-l** 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).

Parameters:

map=name

Name of raster map.

nv=string

String representing no data cell value.

nsteps=value

Number of quantization steps.

NOTES

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

SEE ALSO

[*g.region*](#), [*r.mask*](#), [*r.reclass*](#), [*r.rescale*](#), [*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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:

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.
2. If you choose to make an area or line you must identify the category number for that area or line.
3. Using the mouse trace the line or circumscribe the area; or, finish (go to Step 1).

SEE ALSO

[*v.digit*](#) Highly interactive tool for digitizing, editing, and labeling vector data

[*d.display*](#) Tool for displaying and producing maps

[*d.mapgraph*](#) Draws simple graphics on a map

[*r.in.poly*](#) Tool for importing "polygon" data to raster format

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.direct – Generates a flow direction map from a given elevation layer
(*GRASS Raster Program*)

SYNOPSIS

```
r.direct
r.direct help
r.direct input=name output=name type=name
```

DESCRIPTION

r.direct generates a flow direction map from a given elevation layer.

OPTIONS

Parameters:

input=name

elevation map

output=name

flow direction map

type=name

type of flow direction map to be created.

Options: *agnps* or *grass*

The *type* is the type of format for which the user wishes to create the flow direction map. The *agnps* format gives category values from 1–8, with 1 facing north and increasing values in the clockwise direction. The *answers* format gives category values from 0–360 degrees, with 0 (360) facing east and values increasing in the counter clockwise direction at 45 degree increments. The *grass* format gives the same category values as the [r.slope.aspect](#) program.

Example

```
r.direct input=ansi.elev output=ansi.asp type=grass
```

will create a flow direction map *ansi.asp* for the type *grass*

NOTE

r.direct is sensitive to the **current window setting**. Thus the program can be used to generate a flow direction map for any sub-area within the full map layer. Also, *r.direct* is sensitive to any *mask* in effect.

SEE ALSO

[*r.fill.dir*](#)

[*r.slope.aspect*](#)

AUTHOR

Raghavan Srinivasan, Agricultural Engineering, Purdue University

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.distance – Locates the closest points between objects in two raster maps.
(GRASS Raster Program)

SYNOPSIS

```
r.distance
r.distance help
r.distance [-lq] maps=map1,map2 [fs=name]
```

DESCRIPTION

Locates the closest points between "objects" in two raster maps. An "object" is defined as all the grid cells that have the same category number, and closest means having the shortest "straight-line" distance.

The output is an ascii list, one line per pair of objects

```
cat1:cat2:distance:east1:north1:east2:north2
```

Explanation:

```
cat1
    Category number from map1
cat2
    Category number from map2
distance
    The distance in meters between "cat1" and "cat2"
east1,north1
    The coordinates of the grid cell "cat1" which is closest to "cat2"
east2,north2
    The coordinates of the grid cell "cat2" which is closest to "cat1"
```

COMMAND LINE OPTIONS

Flags:

```
-l
    Include category labels in the output
-q
    Run quietly
```

Parameters:*maps*

Maps for computing inter-class distances

fs

Output field separator

Default is a colon (:)

NOTES

The output format lends itself to filtering. For example, to "see" lines connecting each of the category pairs in two maps, filter the output using awk and then into *d.map* graph:

```
r.distance maps=map1,map2 /\
```

```
awk -F: '{print "move",$4,$5,"draw",$6,$7}' | d.mapgraph
```

To create a site list of all the "map1" coordinates, filter the output into awk and then into *s.in.ascii*:

```
r.distance maps=map1,map2 /\
```

```
awk -F: '{print $4,$5}' | s.in.ascii sites=name
```

SEE ALSO

[r.buffer](#), [r.cost](#), [r.drain](#)

AUTHOR

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

Last changed: \$Date: 2002/12/11 18:00:43 \$



NAME

r.dma2ll.sh – Import USGS 1:250,000 DEM files (DMA formatted) into a lat–lon region
(*GRASS Shell Script*)

SYNOPSIS

r.dma2ll.sh dem_file raster_name

DESCRIPTION

r.dma2ll.sh is a UNIX Bourne shell to ease importing of DMA–formatted USGS 1:250,000–scale Digital Elevation Models (DEM) data into a raster map in a lat–lon database. *r.dma2ll.sh* extracts the boundary coordinates from the DEM file automatically.

OPTIONS

None.

Parameters:

dem_file

The name of the DMA–formatted DEM file to be imported.

raster_name

The name of the resulting raster map. The new raster map is created in the currently selected mapset and database.

NOTES

Little error checking is performed. A temporary file is created and deleted during import. If **\$TMPDIR** is set, the temporary file will be created in the directory specified. Otherwise, the temporary file is created in **/tmp**.

Environment variables **GISBASE** , **GISDBASE** , **LOCATION_NAME** and **MAPSET** are used to specify the GRASS database, location, and mapset.

Standard UNIX programs *dd* , *sed* and *awk* are assumed.

FILES

DMA–formatted DEM data files can be obtained from:

<http://edcwww.cr.usgs.gov/doc/edchome/ndcdb/ndcdb.html>

<http://edcftp.cr.usgs.gov/pub/data/DEM/250/>

<ftp://edcftp.cr.usgs.gov/pub/data/DEM/250/>

SEE ALSO

[m.dmaUSGSread](#)

[m.rot90](#)

[m.in.bin](#)

AUTHOR

Tom Poindexter, March, 1999

tpoindex@nyx.net

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 [-c] [-a] [-n] **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. By default, the output will be an integer cell map with 1 along the least cost path, and null cells elsewhere.

With the **-c** (*copy*) flag, the input map cells values are copied verbatim along the path. With the **-a** (*accumulate*) flag, the accumulated cells value from the starting point up to the current cell is written on output. With the **-c** and the **-a** flags, the output map is created with the same cell type than the input map (integer, float or double). The **-c** and **-a** flags are mutually incompatible. If both are specified, the **-a** flag will supercede the **-c** flag, and a warning message will be issued. With the **-n** (*number*) flag, the cells are numbered from the starting point to the final point.

The path is calculated by choosing the stepper "slope" between adjacent cells. The slope calculation accurately accounts for the variable scale in lat-lon projections.

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 [-c] [-a] [-n] 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.

OPTIONS

Program parameters are as follows.

Flags:

- c** Input map cells values are copied verbatim along the least cost path.
- a** Accumulated input map cells values along the least cost path are written on output.
- n** Count cell numbers along the path

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.)

EXAMPLES

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.

With the **-c** (*copy*) flag, you get the following result:

Input :	Output :
ELEVATION SURFACE	LEAST COST PATH
.
. 20 19 17. 16. 17. 16. 16.	. . . 19. 17. 16. . . .

Sometimes, when using an integer cell map and 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](#)

[parser](#)

AUTHOR

Completely rewritten by Roger S. Miller

Last changed: \$Date: 2002/03/01 00:08:31 \$



NAME

r.feathin – GRASS module that takes the feature output of `r.param.scale` and creates either a sites file or raster containing VIPs (pits, peaks and passes).

Usage:

```
r.feathin [-sc] feat=name dem=name [out=name]
```

Flags:

- s Create sites file output
- c Create corner values

Parameters:

feat=name Raster surface feature layer from `r.param.scale`

dem=name DEM that matches feature classification

out=name Output raster layer containing thinned surface features

HISTORY

Jo Wood, Dept. of Geography
19th July, 1995

Modified to include elevation in sites output (conforms with GRASS 4.2 sites API). This allows production of TIN from sites file alone.

See also:
 `r.param.scale`
 `s.delaunay`

Jo Wood,
4th September, 1996

Modified to force output of corner cells if required.

Last modified, 3rd August, 1997

jwo@le.ac.uk
[ASSIST's home](#)

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.fill.dir – Filters and generates a depressionless elevation map and a flow direction map from a given elevation layer
(*GRASS Raster Program*)

SYNOPSIS

r.fill.dir

r.fill.dir [-f] *input=name elevation=name direction=name [areas=name] [type=name]*

OPTIONS

Flags:

-f
find unresolved areas only (don't generate filled DEM except 1 pixel sinks)

Parameters:

input=name
elevation map
elevation=name
corrected elevation map (depression less)
direction=name
flow direction map
type=name
type of flow direction map to be created
areas=name
Output raster map of problem areas

DESCRIPTION

r.fill.dir filters and generates a depressionless elevation map and a flow direction map from a given elevation layer.

EXAMPLE

```
r.fill.dir input=ansi.elev elevation=ansi.fill.elev direction=ansi.asp
```

will create a depressionless (sinkless) elevation map *ansi.fill.elev* and a flow direction map *ansi.asp*

for the type "grass".

ATTENTION

The type is the type of format at which the user wishes to create the flow direction map. The *agnps* format gives category values from 1–8, with 1 facing north and increasing values in the clockwise direction. The *answers* format gives category values from 0–360 degrees, with 0 (360) facing east and values increasing in the counter clockwise direction at 45 degree increments. The *grass* format gives the same category values as the [r.slope.aspect](#) program.

The method adopted to filter the elevation map and rectify it is based on the paper titled "Software Tools to Extract Structure from Digital Elevation Data for Geographic Information System Analysis" by S.K. Jenson and J.O. Domingue (1988).

The procedure takes an elevation layer as input and initially fills all the depressions with one pass across the layer. Next the flow direction algorithm tries to find a unique direction for each cell. If the watershed program detects areas with potholes, it delineates this area from the rest of the area and once again the depressions are filled using the neighborhood technique used by the flow direction routine. The final output will be a depressionless elevation layer and a unique flow direction layer.

This (D8) flow algorithm performs as follows: At each raster cell the code determines the slope to each of the 8 surrounding cells and assigns the flow direction to the highest slope out of the cell. If there is more than one equal, non-zero slope then the code picks one direction based on preferences that are hard-coded into the program. If the highest slope is flat and in more than one direction then the code first tries to select an alternative based on flow directions in the adjacent cells. It iterates that process, effectively propagating flow directions from areas where the directions are known into the area where the flow direction can't otherwise be resolved.

The flow direction map can be encoded in either ANSWERS (Beasley et.al, 1982) or AGNPS (Young et.al, 1985) form, so that it can be readily used as input to these hydrologic models. The resulting depressionless elevation layer can further be manipulated for deriving slopes and other attributes required by the hydrologic models.

In case of local problems, those unfilled areas can be stored optionally. Each unfilled area in this maps is numbered. The flag *"-f"* instructs the program to fill single-cell pits but otherwise to just find the undrained areas and exit. With the *"-f"* flag set the program writes an elevation map with just single-cell pits filled, a direction map with unresolved problems and a map of the undrained areas that were found but not filled. This option was included because filling DEMs was often not the best way to solve a drainage problem. These options let the user get a partially-fixed elevation map, identify the remaining problems and fix the problems appropriately.

NOTE

The r.fill.dir program is sensitive to the **current window setting**. Thus the program can be used to generate a flow direction map for any sub-area within the full map layer. Also, *r.fill.dir* is sensitive to any *mask* in effect.

In some cases it may be necessary to run *r.fill.dir* repeatedly (using output from one run as input to the next run) before all of problem areas are filled.

SEE ALSO

[*r.direct*](#), [*r.fillnulls*](#), [*r.slope.aspect*](#)

Jenson, S.K., and J.O. Domingue. 1988. Extracting topographic structure from digital elevation model data for geographic information system analysis. *Photogram. Engr. and Remote Sens.* 54: 1593–1600.

Beasley, D.B. and L.F. Huggins. 1982. ANSWERS (areal nonpoint source watershed environmental response simulation): User's manual. U.S. EPA–905/9–82–001, Chicago, IL, 54 p.

Young, R.A., C.A. Onstad, D.D. Bosch and W.P. Anderson. 1985. Agricultural nonpoint surface pollution models (AGNPS) I and II model documentation. St. Paul: Minn. Pollution control Agency and Washington D.C., USDA–Agricultural Research Service.

AUTHOR

Fortran version: Raghavan Srinivasan, Agricultural Engineering Department, Purdue University
Rewrite to C with enhancements: Roger S. Miller

Last changed: \$Date: 2003/08/20 08:10:13 \$



NAME

r.fillnulls – Fills NULL (no data areas) in input map and stores filled map to a new output map. The fill data are interpolated from the no data area boundaries using *s.surf.rst* spline interpolation.

(*GRASS Shell Script*)

SYNOPSIS

r.fillnulls

r.fillnulls input=name output=name

OPTIONS

Parameters:

input=name

raster map containing NULL data area(s)/pixel(s)

output=name

raster map with NULL areas filled by RST interpolation

NOTES

During the interpolation following warning may occur:

```
Warning: strip exists with insufficient data
```

```
Warning: taking too long to find points for interpolation--please change
the region to area where your points are
```

This warning is generated if large data holes exist within the surface. As the idea of *r.fillnulls* is to fill such holes, the user may ignore the warning. The interpolation will be continued. However, the user may pay attention to below notes.

ATTENTION

The algorithm is based on [s.surf.rst](#) regularized splines with tension interpolation module which interpolates the raster cell values for NULL data areas from the boundary values of the NULL data area. An eventual raster MASK is respected during the NULL data area(s) filling. The interpolated values are patched into the NULL data area(s) of the input map and saved into a new raster map.

WARNING

Depending on the shape of the NULL data area(s) problems may occur due to an insufficient number of input cell values for the interpolation process. Most problems will occur if a NULL data area reaches a large amount of the map boundary. The user will have to carefully check the result using [r.mapcalc](#) (generating a difference map to the input map) and/or [d.what.rast](#) to query individual cell values.

SEE ALSO

[r.fill.dir](#), [s.surf.rst](#)

Mitas, L., Mitasova, H., 1999, Spatial Interpolation. In: P.Longley, M.F. Goodchild, D.J. Maguire, D.W.Rhind (Eds.), *Geographical Information Systems: Principles, Techniques, Management and Applications*, Wiley, pp.481–492

Mitasova H., Mitas L., Brown W.M., D.P. Gerdes, I. Kosinovsky, Baker, T.1995, Modeling spatially and temporally distributed phenomena: New methods and tools for GRASS GIS. *International Journal of GIS*, 9 (4), special issue on Integrating GIS and Environmental modeling, 433–446.

[Mitasova H. and Mitas L. 1993](#): Interpolation by Regularized Spline with Tension: I. Theory and Implementation, *Mathematical Geology* 25, 641–655.

[Mitasova H. and Hofierka L. 1993](#): Interpolation by Regularized Spline with Tension: II. Application to Terrain Modeling and Surface Geometry Analysis, *Mathematical Geology* 25, 657–667.

AUTHORS

r.fillnulls: Markus Neteler, University of Hannover

and authors of s.surf.rst

Last changed: \$Date: 2003/02/05 12:46:03 \$



NAME

r.flow – construction of slope curves (flowlines), flowpath lengths, and flowline densities (upslope areas) from a raster digital elevation model (DEM)

(GRASS Raster/Vector Program)

SYNOPSIS

r.flow

r.flow help

r.flow [-u3mMqh] **elevin** =name [**aspin**=name] [**barin**=name] [**skip**=value] [**bound**=value] [**offset**=value] [**flout**=name] [**lgout**=name] [**dsout** =name]

DESCRIPTION

This program generates flowlines using a combined raster–vector approach (see [Mitasova and Hofierka 1993](#) and [Mitasova et al. 1995](#)) from an input elevation raster map *elevin* (integer or floating point), and optionally an input aspect raster map *aspin* and/or an input barrier raster map *barin*. There are three possible output maps which can be produced in any combination simultaneously: a vector file *flout* of flowlines, a raster map *lgout* of flowpath lengths, and a raster map *dsout* of flowline densities (which are equal upslope contributed areas per unit width, when multiplied by resolution).

Aspect used for input must follow the same rules as aspect computed in other GRASS programs (see [s.surf.rst](#) or [r.slope.aspect](#)).

Flowline output is given in a vector map *flout*, (flowlines generated downhill). The line segments of flowline vectors have endpoints on edges of a grid formed by drawing imaginary lines through the centers of the cells in the elevation map. Flowlines are generated from each cell downhill by default; they can be generated uphill using the flag *-u*. A flowline stops if its next segment would reverse the direction of flow (from up to down or vice-versa), cross a barrier, or arrive at a cell with undefined elevation or aspect. Another option, *skip=val*, indicates that only the flowlines from every *val*-th cell are to be included in *flout*. The default *skip* is $\max(1, \lfloor \text{rows in elevin} \rfloor / 50, \lfloor \text{cols in elevin} \rfloor / 50)$. A high *skip* usually speeds up processing time and often improves the readability of a visualization of *flout*.

Flowpath length output is given in a raster map *lgout*. The value in each grid cell is the sum of the planar lengths of all segments of the flowline generated from that cell. If the flag *-3* is given, elevation is taken into account in calculating the length of each segment.

Flowline density downhill or uphill output is given in a raster map *dsout*. The value in each grid cell is the number of flowlines which pass through that grid cell, that means the number of flowlines from the entire map which have segment endpoints within that cell.

OPTIONS

The program will run non-actively if the user specifies program arguments and flag settings on the command line using the following form:

```
r.flow [ -u3mMqh ] elevin = name [ aspin = name ] [ barin = name ] [ skip = val ] [ bound = val ] [ offset = val ] [ flout = name ] [ lgout = name ] [ dsout = name ].
```

Alternatively, the user can simply type **r.flow** on the command line and the program will ask for parameter values and flag settings interactively, using the standard GRASS parser interface.

Flags:

- u** Generate flowlines uphill (default generates flowlines downhill).
- 3** Compute three-dimensional lengths (default is two-dimensional).
- m** Use less memory and compute aspect at each cell on the fly. This option incurs a severe performance penalty. If this flag is given, the aspect input map (if any) will be ignored.
- M** Use a fixed size memory and utilize page-swapping to handle large input files. This option incurs a severe performance penalty but is the only way to handle arbitrarily-large data files. If this flag is given, the **-m** flag will be ignored.
- q** Quiet operation. Do not print diagnostic messages indicating progress.
- h** Display reference information.

Parameters:

elevin=*name*

Use the existing raster file name with elevations as input (required).

aspin=*name*

Use the existing raster file name with aspects as input.

barin=*name*

Use the existing raster file name with non-zero values representing barriers as input.

skip=*val*

Set the number of cells between flowlines in the **flout** output map to *val*.

bound=*val*

Set the maximum number of segments of each flowline to *val* (default is the maximum possible).

offset=*val*

Maximum magnitude of random grid point offset (default is 0).

flout=*name*

Output coordinates of flowlines to a vector file name.

lgout=*name*

Output flowpath length values to a raster file name.

dsout=*name*

Output flowline density values to a raster file name.

NOTES

For best results, use input elevation maps with high precision units (e.g., centimeters) so that flowlines do not terminate prematurely in flat areas. To prevent the creation of tiny flowline segments with imperceivable changes in elevation, an endpoint which would land very close to the center of a grid cell is quantized to the exact center of that cell. The maximum distance between the intercepts along each axis of a single diagonal segment and another segment of 1/2 degree different aspect is taken to be "very close" for that axis. Note that this distance (the so-called "quantization error") is about 1–2% of the resolution on maps with square cells.

The values in length maps computed using the `-u` flag represent the distances from each cell to an upland flat or singular point. Such distances are useful in water erosion modeling for computation of the LS factor in the standard form of USLE. Uphill flowlines merge on ridge lines; by redirecting the order of the flowline points in the output vector map, dispersed waterflow can be simulated. The density map can be used for the extraction of ridge lines.

Computing the flowlines downhill simulates the actual flow (also known as the raindrop method). These flowlines tend to merge in valleys; they can be used for localization of areas with waterflow accumulation and for the extraction of channels. The downslope flowline density multiplied by the resolution can be used as an approximation of the upslope contributing area per unit contour width. This area is a measure of potential water flux for the steady state conditions and can be used in the modeling of water erosion for the computation of the unit stream power based LS factor or sediment transport capacity.

The program has been designed for modeling erosion on hillslopes and has rather strict conditions for ending flowlines. It is therefore not very suitable for the extraction of stream networks or delineation of watersheds unless a DEM without pits or flat areas is available ([r.fill.dir](#) can be used to fill pits).

To label the vector flowlines automatically, the user can use [v.llabel](#).

Algorithm background

1. Construction of flow-lines (slope-lines): `r.flow` uses an original vector-grid algorithm which uses an infinite number of directions between 0.0000... and 360.0000... and traces the flow as a line (vector) in the direction of gradient (rather than from cell to cell in one of the 8 directions = D-infinity algorithm). They are traced in any direction using aspect (so there is no limitation to 8 directions here). The D8 algorithm produces zig-zag lines. The value in the outlet is very similar for both `r.flow` and [r.flowmd](#) algorithms (because it is essentially the watershed area), however the spatial distribution of flow, especially on hillslopes is quite different. It is still a 1D flow routing so the dispersal flow is not accurately described, but still better than D8.

2. Computation of contributing areas: `r.flow` uses a single flow algorithm, i.e. all flow is transported to a single cell downslope.

FURTHER NOTES

Differences between `r.flow` and `r.flowmd`

1. `r.flow` has an option to compute slope and aspect internally thus making the program capable to process much larger data sets than `r.flowmd`. It has also 2 additional options for handling of large data

- sets but it is not known that they work properly.
2. the programs handle the special cases when the flowline passes exactly (or very close) through the grid vertices differently.
 3. `r.flowmd` has the simplified multiple flow addition so the results are smoother.

In conclusion, `r.flowmd` produces nicer results but is slower and it does not support as large data sets as `r.flow`.

DIAGNOSTICS

ERROR: GISRC – variable not set

The program was run outside of GRASS.

Usage: `r.flow` [-u3mMqh] elevin=name [aspin=name] [barin=name] [skip=value] [bound=value] [offset=value] [flout=name] [lgout=name] [dsout=name]

Invalid options were specified on the command line.

ERROR: `r.flow`: error getting current region

ERROR: `r.flow`: unable to reset current region

ERROR: `r.flow`: cannot open vector file

"ERROR: `r.flow`: cell file " filename " not found"

"ERROR: `r.flow`: cannot get header for " filename

"ERROR: `r.flow`: unable to create raster map " filename

Self-explanatory or beyond explanation.

"ERROR: `r.flow`: " input " file's resolution differs from current" region resolution

The resolutions of all input files and the current region must match.

"ERROR: `r.flow`: resolution too unbalanced (" val " x " val ") " The difference in length between the two axes of a grid cell is so great that quantization error is larger than one of the dimensions. Resample the map and try again.

SEE ALSO

[r.flowmd](#), [r.basins.fill](#), [r.drain](#), [r.fill.dir](#), [r.water.outlet.html](#), [r.watershed v.llabel](#)

AUTHORS

Original version of program:

Maros Zlocha and Jaroslav Hofierka, Comenius University, Bratislava, Slovakia,

The current version of the program (adapted for GRASS5.0):

Joshua Caplan, Mark Ruesink, Helena Mitsova, University of Illinois at Urbana–Champaign with support from USA CERL.

[GMSL/University of Illinois at Urbana–Champaign](#)

REFERENCES

Mitsova, H., L. Mitas, 1993, Interpolation by regularized spline with tension : I. Theory and implementation. *Mathematical Geology* 25, p. 641–655. ([online](#))

Mitsova and Hofierka 1993 : Interpolation by Regularized Spline with Tension: II. Application to Terrain Modeling and Surface Geometry Analysis. *Mathematical Geology* 25(6), 657–669. ([online](#))

Mitsova, H., Mitas, L., Brown, W.M., Gerdes, D.P., Kosinovsky, I., Baker, T., 1995: Modeling spatially and temporally distributed phenomena: New methods and tools for GRASS GIS. *International Journal of Geographical Information Systems* 9(4), 433–446.

Mitsova, H., J. Hofierka, M. Zlocha, L.R. Iverson, 1996, Modeling topographic potential for erosion and deposition using GIS. *Int. Journal of Geographical Information Science*, 10(5), 629–641. (reply to a comment to this paper appears in 1997 in *Int. Journal of Geographical Information Science*, Vol. 11, No. 6)

Mitsova, H.(1993): Surfaces and modeling. *Grassclippings* (winter and spring) p.18–19.

Last changed: \$Date: 2003/07/04 08:35:34 \$



NAME

r.flowmd – construction of slope lines (flowlines), flowpath lengths and flowline densities from a raster digital elevation model using a modified multiple directions algorithm.

(GRASS Raster/Vector Program)

SYNOPSIS

r.flowmd

r.flowmd help

r.flowmd [-d] *elevin*=name *aspin*=name [*barrierin*=name] [*skip*=val] [*flout*=name [*lgout*=name] [*dsout*=name]

DESCRIPTION

r.flowmd

generates flowlines using a combined raster–vector approach from input elevation *elevin* , aspect *aspin* and optionally barrier *barrierin* raster maps.

Input elevation raster map with integer values should have values in centimeters preventing premature flowline stop in flat areas. Aspect used for input must follow the same rules as aspect computed in other GRASS programs (see *r.slope.aspect*, *s.surf.rst*). Barriers are defined by non–zero values in an input raster map *barrierin* . Make sure that the resolution of your current region is set to the resolution of the elevation raster file.

There are three possible output maps which can be produced in any combination simultaneously: a vector file *flout* of flowlines, a raster map *lgout* of flowpath lengths, and a raster map *dsout* of flowline densities. Flowlines are constructed by straight line segments (vectors) with endpoints creating intersections of the flowline with edges of the imaginary mesh connecting centers of the grid cells. Flowlines are generated from each cell uphill by default; they can be generated downhill using the flag **-d** . A flowline stops if its next segment would reverse the direction of flow (from up to down or vice–versa), cross a barrier, or arrive at a cell with undefined elevation or aspect. Another option, **skip** =*val* , indicates that only the flowlines from every *val* th cell are to be included in *flout* . The default **skip** is 1. A high **skip** usually speeds up processing time and often improves the readability of a visualization of *flout* .

Flowpath length output *flout* gives the horizontal length of the flowline for given grid cell in units defined by your mapset. It is computed as the sum of the planar length of all straight flowline segments.

Flowline density output *dsout* represents the number of generated flowlines passing through the given cell. This program uses a modified multiple directions algorithm. Flowline density for each flowline passing between 2 grid cell centers is fractioned between these grid cells by distance. This algorithm smoothes flowline densities on convex (dispersing) slopes. Flowline lengths and densities can be computed for both uphill and downhill directions.

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:

```
r.flowmd [-d ] elevin=name aspin=name [barrierin=name] [skip=val ] [flout=name ]  
[lgout=name ] [dsout=name ]
```

Alternately, the user can simply type **r.flowmd** 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.

Flag:

[-d]

Generate flowlines downhill (default generates flowlines uphill)

Parameters:

elevin=name

Use the existing raster file with elevations *name* as input.

aspin=name

Use the existing raster file with aspect *name* as input.

barrierin=name

Use the existing raster file with non-zero values representing barriers with *name* as input.

skip=val

Set the the number of cells skipped for origins of flowlines to *val* .

flout=name

Output coordinates of flowlines to vector file named *name* .

lgout=name

Output flowpath length values to raster file named *name* .

dsout=name

Output flowline density values to raster file named *name* .

Algorithm background

1. Construction of flow-lines (slope-lines): *r.flow* uses an original vector-grid algorithm which uses an infinite number of directions between 0.0000... and 360.0000... and traces the flow as a line (vector) in the direction of gradient (rather than from cell to cell in one of the 8 directions = D-infinity algorithm). They are traced in any direction using aspect (so there is no limitation to 8 directions here). The D8 algorithm produces zig-zag lines. The value in the outlet is very similar for both *r.flowmd* and [r.flow](#) algorithms (because it is essentially the watershed area), however the spatial distribution of flow, especially on hillslopes is quite different. It is still a 1D flow routing so the dispersal flow is not accurately described, but still better than D8.

2. Computation of contributing areas: *r.flowmd* divides this flow proportionally to two downslope adjacent cells using the distance between the intersection node (flowline node) and these adjacent cell centres. (All these three points are always on one connection line). Therefore *r.flowmd* uses a multiple flow algorithm but using a flux decomposition different from Desmet & Govers (1996). D. G. split the flux vector to two ordinal components (cells) using sin, cos of the aspect value. The flux is then divided to 2 from 4 neighboring cardinal cells. The algorithm used in *r.flowmd* divides the flux to two adjacent downslope cells (always one cardinal and one diagonal cell) using a procedure as described above.

NOTES

Flowlines generated uphill are used for computation of the flowpath length (slope lengths) defining the distance between the grid cell and upland area. This can be used, for example, in water erosion modeling for the computation of LS factor in standard form of USLE. Flowlines can be also generated downhill from each grid point to simulate the topographic potential for quasi steady–state overland water flow. Downslope flowline density multiplied by grid cell area gives the upslope contributing area. This topographic parameter defines an area from which water inflows to the given cell. This area is an estimate of potential steady–state water flux and can be used in the modeling of water erosion and for computation of unit stream power based LS factor. The program has been designed for modeling erosion on hillslopes and has a very strict condition for ending the flowline. It is therefore not very suitable for the extraction of stream network or delineation of watersheds unless a DEM without pits or flat areas is available ([r.fill.dir](#) can be used to fill pits).

To label the vector flowlines automatically, the user can use [v.llabel](#).

FURTHER NOTES

Differences between *r.flow* and *r.flowmd*

1. *r.flow* has an option to compute slope and aspect internally thus making the program capable to process much larger data sets than *r.flowmd*. It has also 2 additional options for handling of large data sets but it is not known that they work properly.
2. the programs handle the special cases when the flowline passes exactly (or very close) through the grid vertices differently.
3. *r.flowmd* has the simplified multiple flow addition so the results are smoother.

In conclusion, *r.flowmd* produces nicer results but is slower and it do support as large data sets as *r.flow*.

SEE ALSO

[r.flow](#),
[r.watershed](#),
[r.drain](#),
[r.fill.dir](#),
[r.slope.aspect](#),
[s.surf.rst v.llabel](#)

AUTHOR

Jaroslav Hofierka, [GeoModel s.r.o.](#), Bratislava, Slovakia

REFERENCES

Mitasova, H., L. Mitas, 1993, Interpolation by regularized spline with tension : I. Theory and implementation. *Mathematical Geology* 25, p. 641–655. ([online](#))

[GMSL/University of Illinois at Urbana–Champaign](#)

Mitasova, H., J. Hofierka, 1993, Interpolation by regularized spline with tension : II. Application to terrain modeling and surface geometry analysis. *Mathematical Geology* 25, p. 657–669. ([online](#))

Mitasova, H., Hofierka, J., Zlocha, M., Iverson, R. L. (1996). Modelling Topographic Potential for Erosion and Deposition Using GIS. *International Journal of Geographical Information Systems*, 10, 629–641. ([online](#))

Mitasova, H.(1993): Surfaces and modeling. *Grassclippings* (winter and spring) p.18–19.

Mitasova, H., J. Hofierka, M. Zlocha, L.R. Iverson, 1996, Modeling topographic potential for erosion and deposition using GIS. *Int. Journal of Geographical Information Science*, 10(5), 629–641. (reply to a comment to this paper appears in 1997 in *Int. Journal of Geographical Information Science*, Vol. 11, No. 6)

Last changed: \$Date: 2003/07/04 08:35:34 \$



NAME

rgb.hsv.sh – Converts RGB (red, green, and blue) cell values to HSV (hue, saturation, value) 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

Last changed: \$Date: 2002/01/25 05:45:35 \$



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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.his – Produces a color image by combining hue, intensity, and saturation (his) values from user–specified input raster map layers.

(GRASS Raster Program)

SYNOPSIS

r.his

r.his help

r.his [-on] **h_map=name** [**i_map=name**] [**s_map=name**] **r_map=name** **g_map=name** **b_map=name**

DESCRIPTION

his stands for hue, intensity, and saturation. This program produces red, green and blue raster map layers 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 a color image (in the form of separate red, green and blue raster map layers) 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**), along with the names of the output maps (**r_map**, **g_map** and **b_map**).

Alternately, the user can run the program interactively by typing **r.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.

Flags:

- o** Overwrite output maps.
- n** Respect NULL values while drawing

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.

r_map=name

Name of output raster map layer to be used for RED values.

g_map=name

Name of output raster map layer to be used for GREEN values.

b_map=name

Name of output raster map layer to be used for BLUE values.

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 image 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](#), [d.colors](#) or [r.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

The name is misleading. The actual conversion used is

$$\underline{H}.i.s + \underline{G}.(1-s)$$

where

\underline{H} is the R,G,B color from the hue map
 i is the red value from the intensity map
 s is the red value from the saturation map
 \underline{G} is 50% gray (R = G = B = 0.5)

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*. The separate *red*, *green* and *blue* maps can be displayed on the graphics monitor using [d.rgb](#), or combined into a composite RGB layer using [i.composite](#). Users wishing to simply display an *his* composite image without actually generating any

layers should use the program [d.his](#).

SEE ALSO

[d.his](#), [d.colormode](#), [d.colors](#), [d.colortable](#), [d.display](#), [d.rgb](#), [i.composite](#), [r.colors](#), [blend.sh](#), [i.his.rgb](#), [i.rgb.his](#), [hsv.rgb.sh](#), [rgb.hsv.sh](#), [parser](#)

AUTHOR

Glynn Clements (based upon [d.his](#))

Last changed: \$Date: 2002/06/16 15:29:18 \$



NAME

r.hydro.CASC2D – Fully integrated distributed cascaded 2D hydrologic modeling.
(GRASS Raster Program)

Note: This is beta software: version 1.01.

SYNOPSIS:

```
r.hydro.CASC2D, r.hydro.CASC2D help, r.hydro.CASC2D [-smt oepibduq] elevation=mapname
time_step=value tot_time=value discharge=mapname outlet_eastNnorthNslope=east,north,bedslope
rain_duration=value [watershed_mask=mapname] [initial_depth=mapname] [storage_capacity=mapname]
[interception_coefficient=mapname] [roughness_map=mapname] [Manning_n=value]
[conductivity=mapname] [capillary=mapname] [porosity=mapname] [moisture=mapname]
[pore_index=mapname] [residual_sat=mapname] [lake_map=mapname] [lake_elev=mapname]
[radar_intensity_map=mapname] [links_map=mapname] [nodes_map=mapname] [channel_input=mapname]
[table_input=mapname] [dis_profile=mapname] [wat_surf_profile=mapname] [hyd_location=mapname]
[r_gage_file=mapname] [unif_rain_int=value] [num_of_raingages=value] [gage_time_step=value]
[radar_time_step=value] [write_time_step=value] [unit_el_conv=value] [unit_lake=value] [unit_space=value]
[d_thresh=value] [dis_hyd_location=mapname] [depth_map=mapname] [inf_depth_map=mapname]
[surf_moist_map=mapname] [rate_of_infil_map=mapname] [dis_rain_map=mapname]
```

NOTE

In above, the command line arguments have been rearranged so that the required parameters are placed first. When doing "*r.hydro.CASC2D help*", the user will see a different sequence of parameters.

DISCLAIMER

Users with no background in or understanding of distributed hydrology are strongly advised against using this code in any mode, particularly in operational mode. Besides knowledge of basic hydrology, experience with typical numerical techniques used in physically-based hydrodynamic models is recommended as it will help the user grasp capabilities and limitations of this model. This manual is significantly condensed for electronic distribution and is in no way comprehensive. Users are encouraged to experiment with the model and venture in hydrology textbooks and journal papers to learn more about the topics touched upon in this manual. The *r.hydro.CASC2D* code is continuously being improved. Changes in the source code of *r.hydro.CASC2D* may be made at any time, without notification. No claims are made regarding the suitability of *r.hydro.CASC2D* for any particular purpose. The model *r.hydro.CASC2D* was written for research and educational purposes.

HISTORY

CASC2D originally began with the two-dimensional overland flow routing algorithm developed in APL by Prof. P.Y. Julien at Colorado State University. The overland flow routing module was converted from APL to FORTRAN by Bahram Saghafian, then at Colorado State University, with the addition of Green & Ampt infiltration and explicit channel routing (Julien and Saghafian, 1991, Saghafian, 1992, and Julien et al., 1995). The Fortran version was reformulated, significantly enhanced, and re-written in the C programming language by Bahram Saghafian at the U.S. Army Construction Engineering Research Laboratories. Implicit channel routing code was developed and added to *r.hydro.CASC2D* by Fred L. Ogden (Ogden, 1994), formerly at Colorado State University, now Asst. Professor, Department of Civil and Environmental Engineering, University of Connecticut, Storrs, Connecticut. This version became known as *r.hydro.CASC2D*, part of the GRASS GIS for hydrologic simulations (Saghafian, 1993).

DESCRIPTION

r.hydro.CASC2D is a physically-based, distributed, raster hydrologic model which simulates the hydrologic response of a watershed subject to a given rainfall field. Input rainfall is allowed to vary in space and time. Major components of the model include interception, infiltration, and surface runoff routing. Interception is a process whereby rainfall is retained by vegetation. Interception is estimated using an empirical three parameter model. Infiltration is the process whereby rainfall or surface water is pulled into the soil by capillary and gravity forces. The Green and Ampt equation with four parameters is applied to model the event-based infiltration. For continuous soil moisture accounting, redistribution of soil moisture can also be simulated whenever the non-intercepted rainfall intensity falls below the saturated hydraulic conductivity of the soil. The redistribution option requires two more soil hydraulic parameters. Excess rainfall becomes surface runoff and is routed as overland flow and subsequently as channel flow. The overland flow routing formulation is based on a two-dimensional explicit finite difference (FD) technique, while two different FD techniques, one explicit and one implicit, provide options for routing one-dimensional channel flow. Through a step function, a depression depth may be specified, below which no overland flow will be routed.

The following sections describe various aspects of the model. In executing the model, the likelihood of making a serious mistake by an inexperienced user is unfortunately very high due to complexity of input and output options. Thus the user must thoroughly read all the details of this manual and then select appropriate options for his or her needs. Since at this time GRASS is an integer GIS, all input maps are stored in integer form. This requires conversion from integer to floating point representation via a scaling factor. The scaling factor for all floating point values is fixed. Therefore, the integer maps of parameter values are created by multiplying floating point values (e.g. volumetric water content) by a multiplier.

PARAMETERS/OPTIONS

The following input/output parameters/options control complexity of the simulation. Map and file names in square brackets [] are optional. Some maps are mutually exclusive (logical –or–), while some maps require other maps to enable proper function (logical –and–). Carefully read the NOTES section.

INPUT

TOPOGRAPHY

elevation=mapname map of elevation (DEM).

GRASS Raster Commands

outlet_eastNnorthNslope=value,easting, northing, and bed slope at the outlet

value,value (comma delimited)

TIME

time_step=value computational time step duration in seconds. (typ. 1 to 30 seconds)

tot_time=value total simulation time in sec.

OVERLAND FLOW

[Manning_n=value] spatially uniform Manning's n roughness value for overland flow.

–or–

[roughness_map=mapname] spatially varied map of Manning's n roughness coefficient (values in 1000*Manning's n).

[watershed_mask=mapname] map of watershed boundary (or mask). This option is recommended, as it speeds execution greatly.

[d_thresh=value] threshold overland depth, in meters, below which overland routing will not be performed (i.e. average depression storage).

[initial_depth=mapname] map of initial overland (not lakes) depth in mm.

RAINFALL

rain_duration=value total rainfall duration in sec.

[unif_rain_int=value] spatially uniform rainfall intensity in mm/hr.

–or–

[r_gage_file=filename] raingage rainfall input file name (ASCII).

and–

[num_of_raingages=value] number of recording raingages.

–and–

[gage_time_step=value] time step (temporal resolution) of recorded raingage data in sec.

or–

[radar_intensity_map=mapname] prefix of time series of maps of radar– (or otherwise–) generated rainfall intensities in mm/hr.

and–

GRASS Raster Commands

[*radar_time_step=value*] time increment between radar– (or otherwise–) generated rainfall maps in sec.

INTERCEPTION

[*storage_capacity=mapname*] map of vegetation storage capacity in tenths of mm.

–and–

[*interception_coefficient=mapname*] map of interception coefficient (values in 1000*actual coefficient).

INFILTRATION

[*conductivity=mapname*] map of soil saturated hydraulic conductivity in tenths of mm/hr (Req'd for G&A and Redist).

[*capillary=mapname*] map of soil capillary pressure head at the wetting front in tenths of mm (Req'd for G&A and Redist).

[*porosity=mapname*] map of soil effective porosity (values in 1000*porosity) (Req'd for G&A and Redist).

[*moisture=mapname*] map of initial soil moisture (values in 1000*moisture) (Req'd for G&A and Redist).

[*pore_index=mapname*] map of soil pore–size distribution index (Brooks & Corey lambda) in 1000*index (Req'd for Redist).

[*residual_sat=mapname*] map of soil residual saturation (values in 1000*residual saturation) (Req'd for Redist).

LAKES

[*lake_map=mapname*] map of lakes categories.

[*lake_elev=mapname*] map of lakes initial water surface elevation (also see *unit_lake*).

CHANNEL ROUTING

[*channel_input=filename*] channel input data file name (ASCII), required for explicit (EX) and implicit (IM) channel routing methods.

[*links_map=mapname*] map of channel network link numbers. (EX & IM)

[*nodes_map=mapname*] map of channel network node numbers. (EX & IM)

[*table_input=filename*] look–up table file for links with breakpoint cross section, link type 8, (ASCII) (IM)

[*dis_profile=filename*] channel initial discharge profile file name (ASCII). (IM)

[*wat_surf_profile=filename*] channel initial water surface profile file name (ASCII). (IM)

[*hyd_location=filename*] file name containing link and node numbers of internal locations where discharge hydrographs are to be saved (ASCII).

UNITS

[unit_el_conv=value] unit conversion factor by which the values in elevation must be DIVIDED to convert them into meters.

[unit_lake=value] unit conversion factor by which the values in lake surface elevation map must be DIVIDED to convert them into meters.

[unit_space=value] unit conversion factor by which all region easting and northing values must be DIVIDED to convert them into meters.

OUTPUT

discharge=filename outlet hydrograph file name (ASCII).

[dis_hyd_location=filename] output file name for discharge hydrograph at internal locations (ASCII)

[write_time_step=value] time increment for writing output raster maps in seconds.

–and–

[depth_map=mapname] output maps of surface depth in mm.

[inf_depth_map=mapname] output maps of cumulative infiltration depth in tenths of mm.

[surf_moist_map=mapname] output maps of surface soil moisture in number of fractions of a thousand.

[rate_of_infil_map=mapname] output maps of infiltration rate in mm/hr.

[dis_rain_map=mapname] output maps of distributed rainfall intensity in mm.

FLAGS

There are several flags whose utility is driven by data availability and/or user's choice.

–s do not check square tolerance of a grid cell. With this option non–square grid cell can be simulated using width of east–west resolution.

–m r_gage_file units in mm/hr. Without this option r_gage_file units in in/hr.

–t spatially interpolates raingage rainfall intensities using Thiessen polygon technique. The default technique uses inverse–distance–squared proportionality for interpolation of rainfall intensity over space.

–o routes edge–accumulated overland flow out of active region (ONLY when no mask is specified). Often the surface water accumulated at the edges of the current region creates severe backwater effects and may limit the use of longer computational time steps.

–e performs one–dimensional explicit finite difference channel routing. May be suitable for low– to medium–intensity rainstorms over small arid and semi–arid watersheds with no base flow discharge. This option often limits the computational time step to small values (<10 seconds). Channel bed smoothing is recommended to eliminate adverse slopes. No hydraulic structures, except reservoir spillways, can be

simulated.

`-p` assumes uniform channel geometry in each link (requires `-e` option). If this option is chosen, the channel input file (`channel_input`) must have only one line per fluvial link rather than the default of one line per node.

`-i` performs Preissmann double sweep implicit channel routing. Particularly suitable for watersheds with some base flow to avoid dry-bed condition with channel slopes less than 1%. Supercritical slopes cannot be handled; a warning message will be printed if supercritical flow is encountered.

`-b` initializes the channel depth and discharge files (similar to `-d`, requires `-i` option) using the standard step backwater method. This option must be used with `-i` flag and replaces `-d` option to write "dis_profile" and "wat_surf_profile" files. At present, only link types 1, 2, and 8 may exist in the channel network.

`-d` performs channel initialization for implicit routing (similar to `-b`, requires `-i` option) by flooding the entire channel network with a horizontal water surface and draining down to normal depth using a $y(t)$ outlet boundary condition (similar to `-b`). It is essential for implicit channel routing technique that a minimum initial base flow discharge exist in the channels and also the corresponding initial water surface profile at each node in the channel network have a realistic value. When the depth at the outlet reaches normal depth, the values of depth and discharge at each node is written to "dis_profile" and "wat_surf_profile" files for use in start up of actual simulations. With this option no other component of the model is executed; only the implicit channel routing is performed to create initial depth and discharge files necessary for start up of actual simulations.

`-u` writes discharges in cubic feet per second (cfs) and volumes in cubic feet in "discharge" file. The internal calculations are all in SI units regardless of this flag. The default SI unit prints the discharges in cubic meters per second (cms) and volumes in cubic meters (m3).

`-q` quietly skips printing iteration, time, and discharge values to the screen. No status report is printed.

IMPORTANT NOTES

The user must pay close attention to the following notes prior to any simulation:

1) watershed_mask: Although optional, preparation of this map is highly recommended as it cuts down on the memory requirements by the amount directly proportional to the ratio of mask area over the region area (also see `-o` flag). If the basin (mask) delineation has not been performed correctly, surface water may accumulate near the edges of the watershed. This excess water has no way out of the watershed and will accumulate, creating undesirable backwater effects within the watershed which eventually dictate use of shorter time step in order to accommodate such effects. It is recommended, in such instances, to re-delineate the watershed along the edges. An alternative would be to lower the elevation of the cells being flooded by excessive surface runoff near the edges such that no artificial backwater is created.

2) elevation: The elevation map in the form of Digital Elevation Model (DEM) is undoubtedly one of the most important inputs for distributed modeling. Thus the quality of the DEM plays a major role in success of distributed hydrologic simulations. DEMs almost always contain errors. Large flat areas in the DEM may be due to the limited vertical resolution of elevation data. Routing over such flat areas usually creates problems for the numerical techniques used in distributed physically-based models. Unreal pits in the DEM may be artifacts of interpolation scheme used to rasterize digitized contours, or due to coarse resolution in areas of concave topography. As a rule of thumb, the user must cross check the DEM with topographic maps of the area. One way to discover potential errors in the DEM is to run *r.hydro.CASC2D* iteratively with no other option except uniform rainfall and a relatively short time step. Writing surface depth maps should also be

selected (see `depth_map` and `write_time_step` options). The simulation may terminate normally, at which case the surface depth maps must be examined in order to determine where most water accumulation occurred and whether such accumulation areas is justified by the topographic map of the area. Also stream network may be checked for proper connectivity via depth maps. Alternatively, if the model crashed at a certain location (whose address is printed on the screen as well as at the bottom of discharge file) the user must zoom in and double check the DEM with the topographic map. Often times some manual editing of the DEM is necessary in order to impose the actual drainage trend of the topo map. Prior to editing it is recommended that the DEM be multiplied by 10 or 100, particularly if the original vertical resolution of the DEM is also a concern. To account for this multiplication factor use `unit_el_conv=10` or `unit_el_conv=100`, whichever appropriate. Note that only the unreal pits and depressions must be removed since they most likely trap surface runoff which would have otherwise contributed to the outlet. The real lakes and reservoirs may be simulated if delineated (see `lake_map`). In any event, non-smoothed DEMs require short computational time steps, while properly smoothed DEMs, particularly those with coarser grid space resolution, allow use of longer time steps. Another source of concern, which was briefly mentioned in above paragraph, about the quality of the DEMs is that a nicely connected stream network cannot be derived from non-smoothed DEM. If no delineated network (in the form of a vector file, for example) is available, an approach similar to what was described before may be taken in order to edit the DEM. However where a network has been delineated independent of the DEM then the elevation of stream cells should be checked so that they are not higher than those of the surrounding cells. Otherwise the stream cells will not properly collect the surface runoff.

3) `initial_depth`: This is a map which contains initial overland depth values, if any. Rarely used since prior to the storm overland planes are often dry. For channel initial depth see `dis_profile` and `wat_surf_profile` options.

4) `storage_capacity` & `interception_coefficient`: In `r.hydro.CASC2D`, the interception rate (i) is expressed as:

$$i(t)=r(t) \text{ while } I < a$$

$$i(t)=b*r(t) \text{ while } I > a$$

where $r(t)$ denotes rainfall intensity at time t ; a is the storage capacity; b is the interception coefficient; and I is the cumulative interception depth. Storage capacity map, as well as interception coefficient map, are usually a reclass of vegetation map. For table of storage capacity and interception coefficient values see Gray (1970, section 4.6) or Bras (1990) p. 233.

5) `roughness_map`: This map represents the spatial distribution of overland Manning roughness coefficient n . This map could be a reclass of vegetation cover map and/or the land use map. Tables of Manning's n are available in most hydrology textbooks. By using Manning resistance equation it is assumed that the overland flow over watersheds satisfies the conditions of turbulent flow over rough surfaces.

6) `conductivity`, `capillary`, `porosity`, & `moisture`: Four soil property maps are needed for modeling infiltration process using the Green-Ampt technique. These maps respectively contain saturated hydraulic conductivity, capillary suction head, effective porosity, and initial soil moisture content. The first three maps may be reclasses of soil texture map and the last one must be prepared considering antecedent soil moisture conditions. Based on soil textural classifications, tables of estimates of the first three parameters can be found in Rawls et al. (1983). Wherever reliable field measurements of such parameters are available they may be substituted for table values. Note that the saturated hydraulic conductivity map must be adjusted for the percent of rock fraction within the soil, if known. If these four maps are specified, Green & Ampt infiltration calculations are performed. If one or more of the maps is not specified, no infiltration calculations are made.

7) `pore_index` & `residual_sat`: These two maps are required when continuous soil moisture accounting is of primary interest. The model is capable of redistributing the soil moisture during periods of no- or low-

intensity rainfall, over which infiltration capacity may recover for the next burst of storm intensity. The technique used herein for hiatus and post-hiatus stages is primarily based on the method by Smith et al. (1993). In this model Green-Ampt equation is used for post-hiatus stage. Tables of pore-size distribution index (Brooks & Corey lambda) and residual saturation are given by Rawls et al. (1982). As in (6), measured values should be substituted where available. When the four maps specified in (6) as well as the two maps specified here are given as command line arguments, the redistribution infiltration routine is used. All six maps must be specified to enable redistribution infiltration calculations.

8) lake_map & lake_elev: The first map is the delineated lake map with different categories corresponding to individual lakes and the second map holds the initial water surface elevation in the lakes. It is best to number the lakes categories sequentially. If a lake or reservoir connects to the implicit channel network at its outlet, then the lake must also be numbered as a link as reflected in the channel_input file. See channel_input and links_map options for more details. However isolated pond areas may be simulated as well; in fact such ponds are recommended to be delineated as part of the lake_map. Such isolated lakes should not be present in the links map since their outlet is not connected to the channel network. The routing of the lakes is performed by linear reservoir technique. Rainfall is added directly to the lake inflow. No interception or infiltration is abstracted from the lake cells.

9) radar_intensity_map: This is the common prefix of radar- or otherwise- generated rainfall maps. One possible source of this type of data could be the NEXRAD system (Crum and Albery, 1993). Alternatively, such time series of raster maps may come from the output of an interpolation software, which takes in the rainfall site data corresponding to different times and generates the time series of raster maps. For instance, where raingage rainfall data is available, one could run s.surf.rst of GRASS, one time for every recording time, and save the interpolated rainfall maps to be used as input for r.hydro.CASC2D. In any case one must pay attention to the unit of intensity which is in integer millimeters per hour for map values. Also see radar_time_step, and rain_duration options. Example: If one sets radar_intensity_map=rain.map and there are a total of eight maps in regular intervals of 10 minutes (radar_time_step=600 and rain_duration= 4800, both in seconds), then the actual name of the maps in the current mapset would have to be: "rain.map.1", "rain.map.2" through "rain.map.8", respectively corresponding to time periods of 0-10, 10-20 through 70-80 minutes.

10) links_map & nodes_map: A link is a channel segment, of finite length, which is comprised of two or more computational nodes placed at the center of each grid cell. Any internal boundary condition (weirs and lakes/reservoirs, for example) is also considered a link with only two nodes, one node upstream of the internal link and one downstream. Although all links and nodes must be numbered as discussed below, the link and node maps only contain trapezoidal or look-up table cross sections (link types 1 and 8). Internal boundary condition link types (e.g. weirs) do not appear in the link or node maps. The internal boundary condition link types do appear in the channel_input file (see attachment). The purpose of the link and node maps is to provide connectivity information between overland flow grid cells and channel (link,node) pairs. This information is used to pass overland flow to the 1-D channel model as lateral inflow. The nodes map must contain the node numbers corresponding to the links map. Thus nodes corresponding to internal boundary condition links are not present in the nodes map. At a cell where a link terminates and another link begins (junctions for example) the node number must comply with the downstream link; i.e. this cell is assigned a value of 1 in the nodes map. However one must note that the last node of the upstream link also lies at such junction cells. This last node number must be accounted for in the main ascii channel data file (channel_input), in which the total number of nodes per link must be given. An exception to this rule is the most downstream link leading to the outlet. The links map essentially describes topology of the stream network and its strict conformation with the stream numbering conventions is vital. Output from the GRASS command r.watershed cannot be directly used in conjunction with r.hydro.CASC2D. The link numbers assigned by r.watershed must be re-numbered in accordance with the following rules. Additionally, the node map may be constructed by re-numbering the link map. The general rules for link and node numbering are:

GRASS Raster Commands

1. The first link is numbered 1.
2. Upstream links must have smaller link numbers than downstream links.
3. Link numbers must change at junctions.
4. Link numbers may not be skipped.
5. Node numbers increase in the downstream direction.
6. Looped reaches cannot be simulated.
7. Streams may run only in x-y directions and not diagonally.
8. Link types cannot be mixed within a link.

A small example network with three links is given below:

```
LINK MAP NODE MAP
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 2 0 0 0 0 0 3 2 0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0 0 2 2 0 0 0 0 0 4 0 0 0 0 0 3 2 0 0
0 0 0 1 1 0 0 0 2 2 0 0 0 0 0 0 5 6 0 0 0 5 4 0 0 0
0 0 0 0 1 0 0 2 2 0 0 0 0 0 0 0 7 0 0 7 6 0 0 0 0
0 0 0 0 1 1 3 2 0 0 0 0 0 0 0 0 8 9 1 8 0 0 0 0 0
0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0
0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 3 4 0 0 0 0 0
0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0
```

This network has three fluvial links, link 1 has 9 nodes in the node map, while link 2 has 8 and link 3 has 5. Note that the junction of links 1 and 2 is labeled link 3. However, links 1 and 2 have hidden nodes at this junction, which must have the same bed elevation as the first node in link 3, but different cross-sections. The junction is really not the same point in space for all three links, but really represents a short distance from the confluence. In the channel_input file, link 1 really has 10 nodes and link 2 has 9. Link 3 would only have 5 nodes, because its downstream end is not a junction, rather it is the watershed outlet. The link and node maps tell *r.hydro.CASC2D* that the overland flow in the grid cell at row 5, column 4 is passed to link 1, node 5. For more discussion of links and nodes maps, see channel_input description and the last section of this document for channel input and table file options".

11) channel_input, table_file: For a detailed description of channel data file set-up and requirements: see the last section of this document which is a modified version of the document presented at the CASC2D Workshop by F.L. Ogden, June 9-10, 1994, at the University of Memphis, Tennessee (Revision 2, 10 January 1995). Also it is recommended for the user to refer to Ogden (1994) for an overview of the implicit channel routing formulation and numerical scheme. Note that for explicit channel routing, the same channel input file may be used provided that only link types 1 and 4 are present. If explicit channel routing is used, the channel talweg elevations are taken from the channel_input file, not from the DEM.

12) `dis_profile` & `wat_surf_profile`: These two files hold, respectively, the discharge profile and the water surface profile of all nodes within the stream network, including internal boundary condition nodes. These files are created by running `r.hydro.CASC2D` with the `-i` option AND either `-d` or `-b`. After creation, these files are used to initialize the implicit channel routing scheme for actual watershed simulations. They must be present for actual simulations. The `-d` flag creates initial discharge and depth files by flooding the entire channel network with a horizontal water surface, and draining the network using a depth vs. time relation at the watershed outlet. This draining proceeds until normal depth is reached at the outlet. At this time, the `dis_profile` and `wat_surf_profile` files are written for use in channel initialization for later simulations. The `-b` flag creates the initialization files using the standard step backwater method. Note that in `dis_profile` the unit of discharge is in cubic meters per second and the unit of depth in `wat_surf_profile` is in meters. Except for the reservoirs, the water surface profile file actually holds the flow depth values and thus is measured relative to the bed. CAUTION: The implicit channel routing routine must have good first approximation of depth and discharge at each node in the network. Inaccurate values (guesses) will surely lead to a crash. Additionally, the implicit channel routine is for SUBCRITICAL (Froude number < 1) flow only. A newly created channel input file will almost always crash when first run. It is then the task of the user to identify the location of the problem. Usually, there are regions of hydraulically steep slopes which cause supercritical flow. The channel code will warn the user of supercritical flow, including node and link number when it occurs, and then exits. The user should look at that node/link combination, and alter the data file to eliminate the reach of steep slopes. One workable solution to prevent stability problems at low flows is to use the so-called "Preissmann Slot" (see Cunge et al., 1980, for example). Be patient, as getting the channel network running is a time consuming process. There are a host of possible errors including: abrupt changes in cross-section, DEM-error, etc., which can cause problems. The implicit routing method is not applicable to steep (mountainous or upland) streams. If supercritical flow is encountered in upland (1st order) streams, they can be eliminated from the network. The user should verify the suitability of this approach on each watershed.

13) `hyd_location`: If a filename is specified, this input ascii file contains the link and node numbers of the locations at which discharge hydrographs are to be written to the file named in the `dis_hyd_location` option. The first column in this file should contain link numbers and the second column must be filled with the corresponding node numbers. For a description of the output, see the `dis_hyd_location` option.

14) `r_gage_file`: This is an ascii file which must be provided when raingage rainfall data is being simulated. At the top of the file, two-column lines hold the easting and the northing for each raingage. The number of such lines is determined by the total number of recording raingages (`num_of_raingages`). The location of any of the gages does not have to be within the current region nor within the current watershed mask as long as the easting and the northing are not specified relative to the current region, but are based on absolute values in the UTM or SP coordinates. If a gage falls outside in a different zone to the left of the active region's zone, then negative values are also acceptable. Note however that raingages well outside the watershed under analysis generally provide poor rainfall estimates. The subsequent lines at the bottom of the raingage file must reflect temporal variation of rainfall intensity. The number of columns per line is equal to the number of raingages (specified via `num_of_raingages`). The columns are separated by space. The number of lines in this lower portion is equal to number of instances, separated by a constant time interval, the raingages have made a recording. As usually is the case, the unit of rainfall intensity for raingage data must be in inches per hour. Example: For three raingages, each recording rainfall intensity every 2 minutes for a total duration of, say, 10 minutes, a file called "rain.inp" may look like this:

```
205150.0 750212.0
```

```
20545.0 750104.0
```

```
205320.0 750173.0
```

GRASS Raster Commands

```
0.0 0.0 0.55  
1.75 2.25 0.80  
1.00 1.80 1.50  
0.65 0.90 0.70  
0.0 0.50 0.30
```

In above example, the eastings and northing of the first, second, and third raingages are (205150.0,750212.0), (20545.0,750104.0), and (205320.0,750173.0) respectively. The intensities recorded by the first gage, for example, are 0.0, 1.75, 1.00, 0.65, and 0.0 inches per hour, respectively, over 0–2, 2–4, 4–6, 6–8, and 8–10 minutes, etc. For this example `r_gage_file=rain.inp`, `num_of_raingages=3`, `gage_time_step=120`, and `rain_duration=600`. For state plane (SP) coordinate system the eastings and northing will have to be in feet rather than meters.

15) `outlet_eastNnorthNslope`: These three values determine the location of the outlet, in terms of its easting and northing, and the outlet bed slope. One needs to make sure that the outlet described by its easting and northing is not only within the active region but also inside watershed mask. Often times the region is not set to its original settings after zoom–in operations (`d.zoom`) are performed and this may put the outlet outside the active region thus causing the model to eventually crash. The bed slope is equal to tangent of the angle which is made between the bed profile at the outlet and horizontal plane. This slope is primarily used to calculate the outflow overland discharge at the outlet, if any, based on normal depth boundary condition, when no channel routing is performed (all surface flow treated as overland flow and channels essentially assumed wide). Normal depth is also assumed to prevail at the outlet when explicit channel routing (`-e`) has been selected.

16) `Manning_n`: The alternative to simulating spatially varied roughness coefficient is to provide a single value of Manning roughness coefficient `n` via `Manning_n` parameter. The user is warned if both `roughness_map` and `Manning_n` have been specified.

17) `unif_rain_int`: This option represents the intensity of the spatially– uniform temporally–constant (up to the rainfall duration) rainfall. Therefore this option may replace `r_gage_file` and `radar_intensity_map` options. The unit is in millimeters per hour.

18) `num_of_raingages`: The total number of recording raingage. If this variable is set to one, no spatial interpolation is performed and the rainfall is treated as uniform in space but could vary in time. The temporal variation will be then provided by `r_gage_file`. Note that when `num_of_raingages` is set to one, the easting and northing of the gage is irrelevant although two (arbitrary) values must still be provided at the top of raingage file (`r_gage_file`).

19) `time_step`: This represents the duration of computational time step in seconds and is a critical variable determining the total execution time for a particular simulation. There is no firm guide for the selection of the time step; it comes with experience and strongly depends on watershed and rainfall characteristics. The general rule for overland routing and explicit channel routing is that shorter time steps must be used for higher intensity storms, finer horizontal grid resolution (grid spacing), steeper watershed slopes, larger watershed areas, and smoother surfaces. Stability of explicit routing depends upon Courant number. Unfortunately, the critical condition for Courant number limits the length of the computational time step which must be used for the entire simulation unless variable time step algorithm is implemented in the future. Shorter time steps must be used when backwater effects are generated, mainly in flat areas which are not part of the lake map. If the time step is too long for any particular simulation the surface water depth in completely flat areas may show a

GRASS Raster Commands

checker-board pattern, i.e. oscillations are observed in the water surface level. This eventually results in a crash. As such, the time step should be decreased and the simulation repeated; or the flat areas be delineated within the lake map.

20) `gage_time_step`: This variable represents the time interval, in seconds, between recording instances of rainfall intensities by raingage(s). It is implied that the rainfall data has been recorded in regular intervals. See the notes under `r_gage_file` for an example.

21) `radar_time_step`: A value expressing the time interval between consecutive rainfall maps. A uniform time step is implied.

22) `rain_duration`: This is the total duration of rainfall in seconds. If multistorm events are simulated, the time from the beginning of the first storm to the end of the last storm constitutes the total rainfall duration. As such, selection of soil moisture redistribution capability is recommended via specifying pore index (`pore_index`) and residual saturation (`residual_sat`) maps.

23) `tot_time`: The total simulation time in seconds. If the falling limb of the discharge hydrograph is of particular interest the total simulation time must be set to a value greater than total rainfall duration plus the expected recession time.

24) `write_time_step`: This time parameter determines the frequency of writing output raster maps and is equal to the time interval, in seconds, at which output raster maps are saved. Also see `depth_map`, `inf_depth_map`, `surf_moist_map`, `rate_of_infil_map`, and `dis_rain_map` output options.

25) `unit_el_conv`: This conversion factor is used to convert elevation values in DEM (elevation) map to meters. This parameter doesn't need to be specified if the DEM is already in meters since the default is one. For DEM units in cm or ft, `unit_el_conv` must be respectively set to 100 or 3.281.

26) `unit_lake`: This conversion factor is used to convert the water surface elevation values in the `lake_elev` map to meters. The default is 1.0.

27) `unit_space`: This conversion factor is used to convert the horizontal grid spacing resolution to meters. The default is 1.0. For state plane coordinate system in ft, set `unit_space` equal to 3.281.

28) `d_thresh`: This parameter represents an average step retention storage below which the overland depth will not be routed. Another words, all depressions less than or equal to `d_thresh` are filled before any overland flow could begin. The unit must be in meters so for 2 mm of depression storage set `d_thresh`=0.002. Higher values of depression storage would reduce the total execution time of the model as the overland routing consumes most of the CPU effort.

29) `discharge`: The discharge hydrograph computed at the outlet will be saved in this ascii file under the current directory. Some other information, such as peak discharge, is also printed in this file.

30) `dis_hyd_location`: Whenever the `hyd_location` option is selected, the discharge at individual node/link pairs will be saved in this file. The discharge hydrographs at the (link,node) locations specified in `hyd_location` file are grouped in columns. The number of lines in this file is determined by the total number of iterations equal to `tot_time` divided by `time_step`.

31) `depth_map`, `inf_depth_map`, `surf_moist_map`, `rate_of_infil_map`, & `dis_rain_map`: `depth_map` is the common filename prefix given to the time series of output raster maps containing surface depth values in millimeters. At the channel cells the channel flow depth will be recorded in the maps. The surface depth maps,

and all other output maps, are saved at regular intervals determined by `write_time_step` option. If `write_time_step` is not set, no output raster map will be saved. The first map always corresponds to the initial condition and naturally shows the water surface profile corresponding to the base flow discharge within the channel network when implicit channel routing is performed. Similarly the last depth map corresponds to the end-of-simulation time, or to the time at which the program finished abnormally, for example due to selection of a long step which generated oscillations leading to a crash. Abnormal program termination caused by oscillating depths may show negative depths in the overland plane. There is a hard-coded limit of 2000 output raster maps for each simulation. `inf_depth_map` and `rate_of_infil_map` are two options to save the output raster maps of cumulative infiltration depth, in tenth of mm, and rate of infiltration in mm/hr, respectively. These two options can only be selected when infiltration is being computed via either the Green & Ampt or redistribution methods. `surf_moist_map` output contains the soil moisture values at the soil surface and it may solely be selected with continuous infiltration option (`pore_index` and `residual_sat` maps are specified). `dis_rain_map` option saves the instantaneous rainfall intensities, in mm/hr, at `write_time_step` intervals and may be set for the simulations involving spatially distributed rainstorms; i.e. raingage rainfall data or radar (or other sources of rainfall raster maps) data. Example: If one sets `depth_map=depth.out`, `tot_time=1000`, and `write_time_step= 100`, then a total of 11 raster depth maps will be saved in a normally-terminated simulation: `depth.out.00`, `depth.out.01`, `depth.out.02` through `depth.out.10`, respectively corresponding to times equal to 0, 100, 200, through 1000 seconds. These maps may be viewed sequentially in animation form using the GRASS animator program `xganim`.

GENERAL NOTES FOR CHANNEL_INPUT AND TABLE_FILE FILE OPTIONS

The naming convention associated with the Preissmann double-sweep 1-D implicit channel routing method is based on the concept to links and nodes. A link is a channel segment, or an internal boundary condition, which is comprised of two or more computational nodes. All internal boundary conditions contain two nodes, while fluvial reaches may be of any size greater than or equal to two nodes. The following discussion of input file format must first distinguish between different link types. A few of the possible link types are presented below in Table 1. As of August 1995, only link types 1, 2, 4, and 8 are supported. Development is continuing on link types 3 and 7.

Table 1. Link Types

Link Type	Description of Link Type	Number of Parameters	Number of Nodes
1	Fluvial Link, Trapezoidal Cross-Section	5	>=2
2	Overflow Weir	8	2
3	Culvert/Weir (not yet supported)	8	2
4	Reservoir	5	2
7	Bridge Crossing (not yet supported)	8	2
8	Fluvial Link, Look-up Table (Breakpoint) Cross-Section	4	>=2

At present the channel model formulation accepts cross-sectional input for only two different channel geometries, namely trapezoidal, and breakpoint via look-up table. Trapezoidal channel parameters which must be provided as input at each computation node include: Manning's n, bottom width, channel depth, side slope, and bed elevation. Look-up tables of cross sectional properties must include cross-sectional area, top width, and conveyance at equal depth intervals. Smooth transition in channel cross sectional properties within links of type 8 and between all connecting fluvial links often plays a vital role in the success of simulations. Abrupt changes in cross-sections can lead to numerical errors in mass conservation.

As far as handling the reservoirs (link type 4), flow is not routed through reservoirs in this version of the CASC2D channel routing code. Instead the linear reservoir approximation is used. Among other internal boundary conditions link types, only weirs can be simulated at present.

The channel_input file contains the channel bed elevation at each node, which constructs the longitudinal profile of channels in the drainage network. Ideally, the modeler should use surveyed cross sections and talweg profiles of the channel network. However, extensive surveys are often impossible for the entire drainage networks on large watersheds. In lieu of an extensive survey, there are existing tools for extracting channel topology from digital elevation models (DEM). However, if the channel network is extracted from a DEM, a smoothing algorithm must be applied (e.g. Ogden et al., 1994) to produce physically realistic longitudinal profiles because of errors inherent in any DEM.

EXAMPLE OF INPUT FILE FORMAT AND CONSTRUCTION

NOTE: The channel_input file and table_file both follow SI unit convention. All units of length are in meters, including elevations.

In this example, a channel input data file is constructed for a fictitious watershed. Note that this particular example is merely for demonstration and has not been tried in any simulation run. The stream network will consist of trapezoidal and look-up table fluvial links, an internal reservoir, and an overflow weir. Regarding the MASK map (watershed_mask option), note that all watershed cells must be marked with 1, while all raster cells outside the watershed boundary are marked with 0. Also all the cells in the lake should carry the value 1 on the LAKE map (lake_map option). If there was another lake in this example, it would be denoted with the number 2, etc.

The channel link numbering scheme typically employed in double-sweep routing was explained in the main text. With reference to the LINK map shown below, (links_map option), assume that links 1 and 2 drain into link 3, which in-turn drains in to the lake, which is assigned link number 4. Also links 7 and 8 drain into link 9. Link 5 originates at the outlet of the reservoir. Link 5 is split into link 6 because of a change in some cross-section property. Links 6 and 9 flow into link 10, which is immediately upstream from a weir which must be considered link 11. Finally, link 12, the most downstream link, lies below the weir (link 11). Note that the LINK map contains continuous sequence of link numbers, except for the internal boundary conditions such as the weir (link 11) and lakes (link 4) which does not appear in the LINK map.

EXAMPLE LINK MAP

```

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

GRASS Raster Commands

```
0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 7 7 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 7 7 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 7 7 0 0 0 5 5 0 0 0 0 0 0 0 0 0
0 8 8 8 0 0 7 0 0 0 0 5 6 0 0 0 0 0 0 0 0
0 0 0 8 8 8 9 9 0 0 0 0 6 6 0 0 0 0 0 0 0
0 0 0 0 0 0 0 9 9 9 0 0 0 6 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 9 9 9 9 10 10 10 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 12 12 12 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 0
```

Now, refer to the NODE map (nodes_map option) shown below. In the NODE map, each link is numbered from 1 to the number of grid cells spanned by that link, with the exception of the internal boundary conditions. Conceptually, internal boundary conditions (including reservoirs) have two nodes but they are not given any node numbers in the NODE map. This is how the program recognizes internal boundary conditions. Furthermore, all links except the one leading to the watershed outlet must have an extra node to provide connectivity to the downstream link. Therefore, even though the NODE map may show link 1 to have 13 nodes, it actually has 14. This implied extra node for link 1, shared between the most downstream node of link 1 and the most upstream node of link 3, provides the connection between link 1 and link 3. The number of nodes in each link in this example are shown below in Table 2. Note that the node map entries for the lake (link 4) are 0.

EXAMPLE NODE MAP

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 2 0 0 0 0 0 0 0 0 0 3 2 0 0 0 0 0
0 0 0 3 4 0 0 0 0 0 0 0 5 4 0 0 0 0 0 0
0 0 0 0 5 6 7 0 0 0 0 0 6 0 0 0 0 0 0 0
```

GRASS Raster Commands

```

0 0 0 0 0 0 8 9 10 0 0 0 7 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 11 0 10 9 8 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 12 13 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 4 5 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 6 7 0 0 0 2 3 0 0 0 0 0 0 0 0
0 1 2 3 0 0 8 0 0 0 0 4 1 0 0 0 0 0 0 0
0 0 0 4 5 6 1 2 0 0 0 0 2 3 0 0 0 0 0 0
0 0 0 0 0 0 0 3 4 5 0 0 0 4 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 6 7 8 9 1 2 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 4 5 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0

```

Table 2. Number of Nodes in each Link for Example Watershed Stream Network

Link Number Number of Nodes Number of Nodes

as in nodes_map in channel-input file

1 13 14

2 10 11

3 3 4

4 0 2

(reservoir)

5 4 5

GRASS Raster Commands

6 4 5

7 8 9

8 6 7

9 9 10

10 3 4

11 0 2

(weir)

12 6 6

(outlet link)

The first portion of the channel input file is used to pass physical constants and simulation parameters to the model. These include the gravitational acceleration "g", kinetic energy correction factor "alpha", the friction slope weighting factor "beta", the spatial derivative weighting coefficient "theta", the length of each node "dx", the computational time step "dt" (seconds), the total simulation time "tt" (seconds), and the discharge in all first order streams "qmin" (m³/s). At present, the time step "dt" must be identical to the computational time step used in the overland flow routing portion of r.hydro.CASC2D. The program must be told the number of links, and the largest number of nodes in any link in the network for dynamic memory allocation. In our example problem, the number of links is 12, and the maximum number of nodes is 14 (in link 1). Remember that all links which are not at the outlet or not immediately upstream of a reservoir must have an extra node for connectivity purposes. The total number of links is called "nlinks", and the largest number of nodes in any link in the network is called "maxnodes". In this example, we will use the constants and parameters given in Table 3:

Table 3. physical constants and simulation parameters

"g" 9.81 m/s²

"alpha" 1.0

"beta" 0.5

"theta" 0.55

"dx" 100.0 m

"dt" 30.0 s

"tt" 3600.0 s

"qmin" 0.07 cms

"nlinks" 12

GRASS Raster Commands

```
"maxnodes" 14
```

This data constitutes the first portion of channel_input file and is arranged into a header which must have the form (note floating point and integers):

```
9.81
1.0 0.5 0.55
100.0
30.0 3600.0
0.07
12
14
```

The second portion of the input file describes link types as well as the network topology and connectivity. This is accomplished using a line-input format, one line for each link in the network. Each line contains 6 values arranged in columns, as shown in Table 4.

Table 4. Connectivity information format

Column	Data	Description
1	Link number	(INT)
2	Link type	(INT)
3	Number of links upstream from this link	(max. 2, min. 0) (INT)
4	Upstream link #1	(INT) (0 if no upstream links)
5	Upstream link #2	(INT) (0 if no or only one upstream link)
6	Downstream link	(0 for outlet) (INT)

As far as link types in the current example, assume that links 1 through 9, except reservoir link 4 (link type 4), are well-described as trapezoidal (type 1), and we are using look-up table data to describe the cross-sections of links 10 and 12 (type 8). The weir (link 11) is of link type 2. Thus the second portion of input file for describing link types and connectivity looks like:

```
1 1 0 0 0 3
2 1 0 0 0 3
3 1 2 1 2 4
4 4 1 3 0 5
```

GRASS Raster Commands

```
5 1 1 4 0 6
6 1 1 5 0 10
7 1 0 0 0 9
8 1 0 0 0 9
9 1 2 7 8 10
10 8 2 6 9 11
11 2 1 10 0 12
12 8 1 11 0 0
```

Note that only the outlet link has 0 as a downstream dependency. It is important that this be the only link with 0 as a downstream dependency. Also, the location and topology of internal boundary condition links (4 and 11) with respect to other links are known via above connectivity information in the channel_input file. As an interpretation, examine link number 10 above. It is of link type 8 (look-up table cross-section data), has 2 upstream dependencies (links 6 and 9), and flows into link 11.

The third portion of the input file contains the individual hydraulic property information for each node in the network. Assume for now that all the trapezoidal channels in the network have the properties shown in Table 5.

Table 5. Cross-Sectional Properties of Example Trapezoidal Channels

Manning roughness coefficient varies

Bottom width (m) varies

Channel depth (m) 1.75

Side slope H:V varies

Talweg elevation dependent upon location

Now, build the input file section for link #1 which has 13 nodes on the NODE map, plus an extra for connectivity at the upstream end of link 3 (total of 14 nodes). This input file section will look like:

```
1 14
0.035 2.10 1.75 2.00 264.40
0.035 2.10 1.75 2.00 263.85
0.035 2.10 1.75 2.00 263.41
0.035 2.10 1.75 2.00 262.98
```

GRASS Raster Commands

```
0.035 2.10 1.75 2.00 262.75
0.035 2.10 1.75 2.00 262.54
0.035 2.10 1.75 2.00 262.37
0.035 2.10 1.75 2.05 262.02
0.035 2.20 1.75 2.05 261.87
0.035 2.20 1.75 2.10 261.75
0.035 2.20 1.75 2.10 261.67
0.035 2.20 1.75 2.15 261.52
0.035 2.30 1.75 2.15 261.25
0.035 2.40 1.75 2.15 261.00
```

The 1 and 14 on the first line indicate that it is link 1, with 14 nodes. The columns represent Manning's n, bottom width, channel depth, trapezoidal side slope H/V, and talweg elevation, respectively.

Link 2 might have an entry which looks like:

```
2 11
0.035 1.80 1.75 1.80 264.21
0.035 1.80 1.75 1.80 264.10
0.035 1.90 1.75 1.90 263.81
0.035 1.90 1.75 1.90 263.56
0.035 2.00 1.75 1.90 263.34
0.035 2.10 1.75 1.90 262.86
0.035 2.10 1.75 2.00 262.24
0.035 2.10 1.75 2.00 261.76
0.035 2.20 1.75 2.10 261.48
0.035 2.30 1.75 2.10 261.26
0.035 2.30 1.75 2.10 261.00
```

The cross-section entry for link 3 would then look like:

GRASS Raster Commands

```
3 4
```

```
0.035 2.60 1.75 2.20 261.00
```

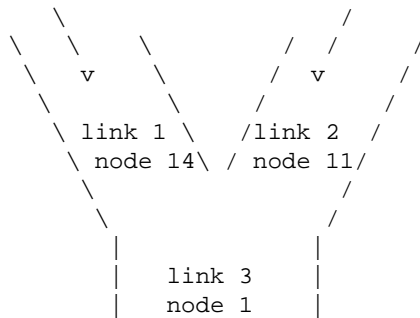
```
0.035 2.60 1.75 2.40 260.20
```

```
0.035 2.80 1.75 2.90 259.67
```

```
0.035 2.80 1.75 2.90 259.17
```

The elevation of the downstream end of links 1 and 2 must be equal to the bed elevation of the upstream end of link 3. It is required that the bed elevation of all channel inverts at each junction be equal. Even though they are not exactly the same points in space, they are assumed close enough to have the same elevation.

Flow Flow



The input for other links with trapezoidal cross section is similar to the input for link 1 (see the next section for the complete input file).

Additionally, assume that the reservoir (link 4) has a surface area of 0.498 square km, a rectangular spillway width of 12m, a spillway discharge coefficient of 0.97, an initial water surface elevation of 260.10m, and a spillway crest elevation of 260.50m. The resulting input file entry is shown below.

```
4 2
```

```
0.498 12.0 0.97 260.10 260.50
```

```
0.000 0.0 0.00 0.00 0.00
```

The number of nodes (line entries) in the channel_input file per reservoir link is two, but the numbers entered in the second row entry are for further improvements and for now their value is irrelevant. Also note that the bed elevation at the downstream limit of link 3 **MUST** be lower than the initial water surface elevation in the reservoir (link 4). This is mandatory at all downstream boundaries between channels flowing into the reservoirs. Reservoirs serve as downstream boundary conditions at upstream links. The elevation of reservoirs should therefore never be allowed to fall below critical depth in any channels which flow into the reservoir. If this happens, the code will crash.

The input for links 5, 6, 7, 8, and 9 will be similar to links 1, 2, and 3. See the constructed input file at the end

of this text. Trapezoidal channel links are simple in terms of input. The channel depth field is intended to represent the average bank–full depth of the channel. This number is not used in the present version of the code, but will be in the future when the overland flood plane and channel flows are coupled iteratively. At present the flows are not fully coupled. Water can flow only from the overland flow plane into the channels, not from the channels back to the flood plane.

If look–up table data are used (link type 8), the look–up tables are stored in a separate file. The GRASS command line option for this look–up table file is `table_file`. If there are any link type 8 in the `channel_input` data file, the program will attempt to open the file `table_file` (typically called "table.dat"). If this file is not present, the program will terminate. In `channel_input` file, all that is needed is the table number for a particular cross section, and the talweg elevation. Assume that the 4 nodes in link 10 are approximated by one cross–section, which is given as table entry 1, then the `channel_input` file entry for link 10 would look like:

```
10 4
1 244.0 1 243.5 1 243.0 1 242.5
```

In this format, the first column represents the look–up table number for the given node, and the second column represents the bed elevation for that node. Table numbers can be mixed within a link. However, abrupt changes in cross–section should be avoided because they cause significant continuity errors in the formulation. The format of the `table_file` is discussed later in this document.

Link 11 is an overflow weir link. While this link has no nodes which appear in the node map for overland flow connectivity, it does have two nodes in terms of channel topology. The meaning of the columns in the weir node input section is presented in Table 6.

Table 6. Parameter Description for Weirs

Line	Entry	Column	Parameter	Description
1	1	forward	weir discharge coefficient	
1	2	reverse	weir discharge coefficient	
1	3	0.0	(reserved for future use)	
1	4	crest length, meters		
1	5	elevation of weir crest, meters		
2	1–4	0.0	(reserved for future use)	
2	5	downstream bed elevation, meters		

The first (upstream) node represents the crest of the rectangular weir. The second (downstream) node represents the bed of the channel just downstream from the weir. Assume that this weir has a crest elevation of 244.4 m, a crest width of 8 m, a discharge coefficient in the forward direction of 0.92, a discharge coefficient for flow in the reverse direction equal to 0.85, and the channel bed elevation immediately downstream from the weir is 239.8m. The entry for this weir in the `channel_input` file would thus appear as:

```
11 2
```

GRASS Raster Commands

```
0.92 0.85 0.00 8.00 244.4
```

```
0.00 0.00 0.00 0.00 239.8
```

Link 12 uses table entry 2 for the its first four nodes and entry 4 for the remaining two nodes. Note that link 12 has 6 nodes in the node map, and 6 nodes in the channel_input file. Because link 12 is the outlet link, we do not add an extra node at the downstream end for connectivity purposes. The portion of the channel_input file for link 12 looks like:

```
12 6
```

```
2 239.80
```

```
2 239.47
```

```
2 239.33
```

```
2 239.05
```

```
3 238.54
```

```
3 238.44
```

Hence, 238.44 is the talweg elevation at the outlet of the catchment. Also notice that the upstream end of link 12 and the downstream node of link 11 have the same bed elevation.

Weirs cannot be used as the outlet boundary condition for this channel model. If your watershed has a weir at the outlet, just place a short link downstream from the weir with appropriate characteristics.

FINAL CHANNEL FILE FORMAT

The example channel data file (typically called chn.dat) developed for this example looks like:

```
-----BEGIN channel_input FILE  
HERE-----
```

```
9.81
```

```
1.0 0.5 0.55
```

```
100.0
```

```
30.0 3600.0
```

```
0.07
```

```
12
```

```
14
```

```
1 1 0 0 0 3
```

```
11/06/2003
```

GRASS Raster Commands

2 1 0 0 0 3

3 1 2 1 2 4

4 4 1 3 0 5

5 1 1 4 0 6

6 1 1 5 0 10

7 1 0 0 0 9

8 1 0 0 0 9

9 1 2 7 8 10

10 8 2 6 9 11

11 2 1 10 0 12

12 8 1 11 0 0

1 14

0.035 2.10 1.75 2.00 264.40

0.035 2.10 1.75 2.00 263.85

0.035 2.10 1.75 2.00 263.41

0.035 2.10 1.75 2.00 262.98

0.035 2.10 1.75 2.00 262.75

0.035 2.10 1.75 2.00 262.54

0.035 2.10 1.75 2.00 262.37

0.035 2.10 1.75 2.05 262.02

0.035 2.20 1.75 2.05 261.87

0.035 2.20 1.75 2.10 261.75

0.035 2.20 1.75 2.10 261.67

0.035 2.20 1.75 2.15 261.52

0.035 2.30 1.75 2.15 261.25

0.035 2.40 1.75 2.15 261.00

11/06/2003

GRASS Raster Commands

2 11

0.035 1.80 1.75 1.80 264.21

0.035 1.80 1.75 1.80 264.10

0.035 1.90 1.75 1.90 263.81

0.035 1.90 1.75 1.90 263.56

0.035 2.00 1.75 1.90 263.34

0.035 2.10 1.75 1.90 262.86

0.035 2.10 1.75 2.00 262.24

0.035 2.10 1.75 2.00 261.76

0.035 2.20 1.75 2.10 261.48

0.035 2.30 1.75 2.10 261.26

0.035 2.30 1.75 2.10 261.00

3 4

0.035 2.60 1.75 2.20 261.00

0.035 2.60 1.75 2.40 260.20

0.035 2.80 1.75 2.90 259.67

0.035 2.80 1.75 2.90 259.17

4 2

0.498 12.0 0.97 260.10 260.50

0.000 0.0 0.00 0.00 0.00

5 5

0.035 2.50 1.75 2.10 249.20

0.035 2.50 1.75 2.10 248.65

0.035 2.50 1.75 2.15 247.94

0.035 2.50 1.75 2.20 247.31

0.035 2.50 1.75 2.20 246.97

11/06/2003

GRASS Raster Commands

0.035 2.50 1.75 2.25 246.42

6 5

0.035 2.50 1.75 2.00 246.42

0.035 2.50 1.75 2.00 245.97

0.035 2.50 1.75 2.00 245.24

0.035 2.50 1.75 2.00 244.73

0.035 2.50 1.75 2.00 244.00

7 9

0.035 1.75 1.75 2.00 254.20

0.035 1.75 1.75 2.00 253.55

0.035 1.75 1.75 2.00 252.92

0.035 1.90 1.75 2.00 252.29

0.035 1.90 1.75 2.00 251.71

0.035 1.90 1.75 2.00 251.24

0.035 2.00 1.75 2.00 250.82

0.035 2.00 1.75 2.00 250.34

0.035 2.00 1.75 2.00 250.08

8 7

0.035 1.60 1.75 2.00 253.71

0.035 1.90 1.75 2.00 253.21

0.035 1.90 1.75 2.00 252.94

0.035 2.00 1.75 2.00 252.55

0.035 2.10 1.75 2.00 251.80

0.035 2.10 1.75 2.00 251.11

0.035 2.10 1.75 2.00 250.08

9 10

11/06/2003

GRASS Raster Commands

```
0.035 3.10 1.75 2.90 250.08
0.035 3.20 1.75 2.90 249.16
0.035 3.30 1.75 3.00 248.63
0.035 3.40 1.75 3.00 248.02
0.035 3.40 1.75 3.10 247.33
0.035 3.20 1.75 3.10 246.65
0.035 3.10 1.75 3.10 246.05
0.035 3.20 1.75 3.10 245.88
0.035 3.20 1.75 3.10 245.52
0.035 3.50 1.75 3.10 245.24
0.035 3.70 1.75 3.40 244.74
0.035 3.50 1.75 3.40 244.51
0.035 3.10 1.75 3.40 244.00
10 4
1 244.00
1 243.50
1 243.00
1 242.50
11 2
0.92 0.85 0.00 8.00 244.40
0.00 0.00 0.00 0.00 239.80
12 6
2 239.80
2 239.47
2 239.33
2 239.05
11/06/2003
```

```
3 238.54
```

```
3 238.44
```

```
----- END OF channel_input FILE
-----
```

TABLE_FILE INPUT FORMAT

If link type 8 (look-up table x-sections) are present in the channel_input file, we require a file typically called "table.dat", to store all look-up table values. This file must begin with an integer equal to the number of tables contained within the file. In our example this number is 3. Then we require 3 tables. The first line of each table is an integer equal to the table number. The second line in the table entry contains two numbers, an integer, and a floating point (real) value. The first number is equal to the number of entries (rows) in each particular table, designated by "numhts". The second number is the vertical distance between hydraulic property points, which must be a constant for each table. For instance, if you describe the variation of cross-sectional area, topwidth, and conveyance at 0.5 m vertical intervals, this number will be 0.5.

Each table entry must then contain "numhts" lines, each with four entries. The first line must be:

```
0.0 0.0 0.0 0.0
```

subsequent lines must contain the following:

```
height topwidth x-section_area x-section_conveyance.
```

For instance, assume we know geometric variables for table entries 1, 2, and 3. The file "table.dat" (table_file option) may look like:

```
3
```

```
1 5 1.0
```

```
0.0 0.0 0.0 0.0
```

```
1.0 2.1 5.9 13.5
```

```
2.0 6.9 22.3 122.2
```

```
3.0 14.7 72.1 312.8
```

```
4.0 19.6 145.6 789.4
```

```
2 8 0.5
```

```
0.0 0.0 0.0 0.0
```

```
0.5 1.1 1.5 7.5
```

```
1.0 1.7 3.4 17.1
```

GRASS Raster Commands

1.5 3.1 11.2 43.9

2.0 5.2 32.4 98.5

2.5 7.2 49.2 187.4

3.0 9.2 86.4 312.5

3.5 12.1 143.1 624.9

3 5 1.0

0.0 0.0 0.0 0.0

1.0 2.2 5.4 15.5

2.0 6.6 15.3 132.2

3.0 15.7 81.8 352.8

4.0 21.4 155.2 839.6

For further clarification, consider table 2 above, corresponding to cross sectional properties of the four most upstream nodes of link 12. Table 2 has 8 entries, each separated by 0.5 meter of depth. At a depth of 3.0 m, for example, the channel has a top-width of 9.2m, a cross-sectional area of 86.4 m², and a conveyance of 312.5 m³/s. The conveyance K is used to calculate the discharge using:

$$Q = K * \sqrt{\text{Slope}}$$

Where using Manning's equation, K is defined (in SI units) as:

$$K = 1/n * (\text{Area}) * (\text{Hydraulic_radius})^{2/3}$$

These tables can be produced easily using a spreadsheet using measured channel cross-section data.

REFERENCES

Bras, R. L., 1990, Hydrology: An introduction to hydrologic science, Addison-Wesley, Reading, Mass., 643 p.

Crum, T. D., and R. L. Alberty, 1993, The WSR-88D and the WSR-88D operational support facility, Bulletin of the American Meteorological Soc., 74(9), pp. 1669-1687.

Cunge, J.A., F.M. Holly, and A. Verwey, 1980, Practical Aspects of Computational River Hydraulics, Iowa Institute of Hydraulic Research, 404HL The University of Iowa, Iowa City, IA 52242. 420 p.

Gray, D.M., 1970, Handbook on the Principles of Hydrology, National Research Council of Canada, Water Information Center Inc., Water Research Building, Manhasset Isle, Port Washington, N.Y., 11050.

GRASS Raster Commands

Julien, P. Y., and B. Saghafian, 1991, CASC2D users manual – A two dimensional watershed rainfall–runoff model, Civil Engr. Report, CER90–91PYJ–BS–12, Colorado State University, Fort Collins, CO.

Julien, P. Y., Saghafian, B., and F. L. Ogden, 1995, "Raster–Based Hydrologic Modeling of Spatially–Varied Surface Runoff", Water Resources Bulletin, AWRA, 31(3), pp. 523–536.

[Ogden, F.L.](#), 1994, de–St Venant channel routing in distributed hydrologic modeling., Proc. Hydraulic Engineering `94, ASCE Hydraulics Specialty Conference, G.V. Cotroneo and R.R. Rumer, eds., Vol. 1, pp. 492–496.

Ogden, F.L., Saghafian, B., and W.F. Krajewski, 1994, GIS–based channel extraction and smoothing algorithm for distributed hydrologic modeling, Proc. Hydraulic Engineering `94, ASCE Hydraulics Specialty Conference, G.V. Cotroneo and R.R. Rumer eds., August 1–5, 1994, Buffalo, N.Y., pp. 237–241.

Rawls, W. J., Brakensiek, D. L., and N. Miller, 1983, Green–Ampt infiltration parameters from soils data, J. of Hydraulic Engineering, ASCE, 109(1), pp. 62–70.

Rawls, W. J., Brakensiek, D. L., and K. E. Saxton, 1982, Estimation of soil water properties, Trans. of ASAE, pp. 1316–1320.

Saghafian, B., 1992, Hydrologic analysis of watershed response to spatially varied infiltration, Ph.D. Dissertation, Civil Engr. Dept., Colorado State University, Fort Collins, CO.

Saghafian, B., 1993, Implementation of a distributed hydrologic model within Geographic Resources Analysis Support System (GRASS), Proceedings of the Second International Conference on Integrating Environmental Models and GIS, Breckenridge, CO.

Smith, R. E., Corradini, C., and F. Melone, 1993, Modeling infiltration for multistorm runoff events, Water Resources Research, 29(1), pp. 133–144.

[CASC 2D page](#)

AUTHORS

B. Saghafian and F. L. Ogden, Colorado State University

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.in.arc – Convert an ESRI ARC/INFO ascii raster file (GRID) into a (binary) raster map layer.
(*GRASS Raster Data Import Program*)

SYNOPSIS

```
r.in.arc
r.in.arc help
r.in.arc input=name output=name [title="phrase"] [mult=multiplier]
```

DESCRIPTION

r.in.arc allows a user to create a (binary) GRASS raster map layer from an ESRI ARC/INFO ascii GRID file with (optional) title.

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:

```
ncols:
nrows:
xllcorner:
yllcorner:
cellsize:
```

or alternatively (not supported in *r.in.arc*, but in [r.in.gdal](#)):

```
ncols:  
nrows:  
xllcenter:  
yllcenter:  
cellsize:
```

NOTES

r.in.arc handles floating point cell values. The **mult** option allows the number of significant figures of a floating point cell to be increased before importing. Multiples of ten are the most functional multipliers.

SEE ALSO

[*r.in.gdal r.out.arc*](#)

AUTHOR

Unknown German author, updated by Bill Brown to floating point support.

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 [-ifds] **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.

OPTIONS

Flags:

-i integer values are imported
-f floating point values are imported
-d double floating point values are imported
-s SURFER (Golden Software) ascii grid file will be imported

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.

GRASS Raster Commands

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.

Optionally the following parameters can be defined in the header section:

```
null: nn
type: float
multiplier: 2.
```

"null" defines a string or number to be converted to NULL value (no data).

"type" defines the data type (int, float double) and is not required.

"multiplier" is an optional parameter to multiply each cell value.

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
null: -9999

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 $r \times x \times c$ 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.

The header information in ESRI Raster ASCII files differs from GRASS. To convert an Arc/Info (ArcView) ASCII grid file into GRASS, click [here](#). Also, see [r.in.arc](#) to import ESRI Raster ASCII files.

SEE ALSO

[m.flip](#), [r.out.ascii](#), [r.in.arc](#), [r.out.arc](#), [r.in.bin](#), [GRASS ASCII formats](#)

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
Surfer support by Roger Miller

Last changed: \$Date: 2003/07/04 08:35:34 \$



NAME

r.in.bin – Import a binary raster file into a GRASS raster map layer.
(*GRASS Raster Data Import Program*)

SYNOPSIS

r.in.bin

r.in.bin help

r.in.bin [-s] [-f] [-b] [-h] **input=name** **output=name** **bytes=value** [**title="phrase"**] [**north=value**] [**south=value**] [**east=value**] [**west=value**] [**r=value**] [**c=value**] [**anull=value**]

DESCRIPTION

r.in.bin allows a user to create a (binary) GRASS raster map layer from a variety of binary raster data formats.

OPTIONS

Flags:

- s Signed one byte data (range -128 – 127). Interpret high bit (> 127) as a negative value.
- f Import the data to a Float type GRASS raster map layer (default is Integer)
- b Byte swap the binary data during import. This includes the header if -h specified.
- h Read region information from a Generic Mapping Tools (GMT) type binary header. Is compatible with GMT binary grid types 1 and 2.

Parameters:

input=name

Name of an existing binary raster file to be imported.

output=name

Name to be assigned to resultant GRASS raster map layer.

bytes=value

Number of bytes per cell (1, 2, 4)

title="phrase"

Title to be assigned to resultant raster map layer.

north=value

Northern limit of geographic region

south=value
Southern limit of geographic region

east=value
Eastern limit of geographic region

west=value
Western limit of geographic region

r=value
Number of rows

c=value
Number of columns

anull=value
Set Value to NULL

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. If input is a [GMT](#) binary array (-h flag), the six dimension fields are obtained from the GMT header. If the bytes field is entered incorrectly an error will be generated suggesting a closer bytes value.

r.in.bin can be used to import numerous binary arrays including: etopo30, etopo-5, Globe DEM, BIL, AVHRR and GMT binary arrays (ID 1 & 2)

EXAMPLE

The following is a sample call of *r.in.bin* to import GTOPO30 DEM data:

```
r.in.bin -s input=E020N90.DEM output=gtopo30 bytes=2 north=90 south=40
east=60 west=20 r=6000 c=4800
```

(you can add "anull=-9999" if you want sea level to have a NULL value)

The following is a sample call of *r.in.bin* to import a GMT type 1 (float) binary array:

```
r.in.bin -hf input=sample.grd output=sample.grass
```

(-b could be used to swap bytes if required)

The following is a sample call of *r.in.bin* to import an AVHRR image:

```
r.in.bin in=p07_b6.dat out=avhrr c=128 r=128
```

NOTES

If optional parameters are not supplied, **r.in.bin** attempts to calculate them. For example if the rows and columns parameters are not entered, **r.in.bin** automatically calculates them by subtracting south from north and west from east. This will only produce correct results if the raster resolution equals 1. Also, if the north, south, east, and west parameters are not entered, **r.in.bin** assigns them from the rows and columns parameters. In the above AVHRR example, the raster would be assigned a north=128, south=0, east=128, west=0.

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.

Notes on (non)signed data:

If you use the `-s` flag the highest bit is the sign bit. If this is 1 the data is negative, and the data interval is half of the unsigned (not exactly).

This flag is only used if `bytes= 1`. If `bytes=` is greater than 1 the flag is ignored.

SEE ALSO

[r.out.bin](#), [r.in.ascii](#), [r.out.ascii](#), [r.in.arc](#), [r.out.arc](#), [r.in.gdal](#)

AUTHORS

Jacques Bouchard, France (bouchard@onera.fr)

Bob Covill, Canada (bcovill@tekmap.ns.ca)

Man page: Zsolt Felker (felker@c160.pki.mata.v.hu)

Last changed: \$Date: 2002/03/01 00:08:31 \$



NAME

r.in.elas – Import an ELAS raster file into a GRASS raster map layer.
(*GRASS Raster Data Import Program*)

SYNOPSIS

r.in.elas

r.in.elas help

r.in.elas [-h] **channel**=value **zone**=value **proj**=value **nbpe**=value **x-spot-size**=value **y-spot-size**=value
input=name **output**=name

DESCRIPTION

r.in.elas imports ELAS raster files to GRASS cell files.

OPTIONS

Flags:

-h

Create GRASS histogram file.

Parameters:

input=name

Name of an existing binary ELAS raster file to be imported.

output=name

Name to be assigned to resultant GRASS raster map layer.

channel=value

ELAS channel number.

default: 1

zone=value

UTM zone number. (1–60)

default: 0

proj=value

GRASS "proj" value. (0 for XY and 1 for UTM)

default: 1

nbpe=value

(number of bytes per element) for ELAS file. "nbpe" is an OPTIONAL value, only to be inputted to override the ELAS header value.

default: 0

x-spot-size=value

ELAS X Spot size number. "x-spot-size" is an OPTIONAL value, only to be inputted to override the ELAS header value.

default: 0

y-spot-size=value

ELAS Y Spot size number. "y-spot-size" is an OPTIONAL value, only to be inputted to override the ELAS header value.

default: 0

This program "r.in.elas" copies and converts an ELAS raster file to a GRASS cell file. The ELAS raster file must be a 1 to 4 byte integer per element file. The GRASS file created from the ELAS file will be either an unreferenced XY or a UTM projection. The ELAS color values will be transferred to GRASS only if there are color values in the ELAS header. If no color values are transferred from ELAS then a GRASS color file will automatically be created. This color file will be a RGB color ramp. The area to be transferred from ELAS to GRASS will be determined by the ELAS values "il" (initial line), "ll" (last line), "ie" (initial element), and "le" (last element). Multi-channel ELAS raster files may be transferred to GRASS one channel at a time. Input variables: nbpe, x-spot-size, and y-spot-size are OPTIONAL variables and can only be inputted non-interactively, since these values will usually already be stored in the ELAS header. But if these values are not in the ELAS header, then the user may provide those values by entering the non-interactive command. This program will produce a GRASS range file and if the flag "-h" is passed to the program then a histogram file will be created as well. Using the "-h" flag with larger files is not recommended since creating a histogram file will increase processing time. You can create a histogram file after this program is finished executing by running "r.support". This program will produce a new "WIND" (region) that will correspond to the new "cellhd" file that has been created. If the user should want to save their original "WIND" (region) then they should run "g.region" prior to executing this program.

SEE ALSO

[r.out.bin](#), [r.in.ascii](#), [r.out.ascii](#), [r.in.arc](#), [r.out.arc](#), [r.in.gdal](#)

AUTHOR

Bruce Powell, National Park Service

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.in.erdas – Creates raster files from ERDAS files. It creates one raster file for each band, and creates color support if an ERDAS trailer file is specified.

"(SCS GRASS Raster Program)

SYNOPSIS

r.in.erdas

r.in.erdas help

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

DESCRIPTION

This command prompts the user twice:

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

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

Note:

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

COMMAND LINE OPTIONS

Parameters:

erdas=name

Input ERDAS file name.

trl=name

Input ERDAS trailer file name.

prefix=name

Prefix of the GRASS raster files to be created.

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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.

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.

Parameters:

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*:

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 conclusion\hypothesis, 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

[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*.

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

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

SYNOPSIS

r.info
r.info help
r.info [-r] 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.

OPTIONS

Flags:

-r
 print range only and exit

Parameters:

map=name
 name of an existing raster map layer

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
```

GRASS Raster Commands

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. 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](#)

Last changed: \$Date: 2003/05/06 13:35:00 \$



NAME

r.in.gdal – Import GDAL supported raster file into a binary raster map layer.
(*GRASS Raster Data Import Program*)

SYNOPSIS

```
r.in.gdal
r.in.gdal help
r.in.gdal [-o] [-e] input=name [band=number] output=name [target=name] [TITLE="phrase"]
[location=name]
```

DESCRIPTION

r.in.gdal allows a user to create a (binary) GRASS raster map layer, or imagery group, from any GDAL supported raster file format, with an optional title. The imported file may also be optionally used to create a new location.

OPTIONS

Flags:

- o*
Override projection (use locations projection)
- e*
Extend the DEFAULT_WIND in PERMANENT mapset to include the region of the new map layer. Old resolution is preserved, but the region, and rows/cols are updated. This will fail if the user doesn't have write access to the PERMANENT mapset.

Parameters:

- input=name*
Name of an existing GDAL supported raster file to be imported.
- band=number*
Number of input band to extract (default is all bands). Number between 1 and number of bands.
- output=name*
Name to be assigned to resultant binary raster map layer or imagery group. An imagery group is generated if more than one band is imported.
- target=name*
Name of location to read projection info from for GCPs transformation
- TITLE="phrase"*
Title to be assigned to resultant raster map layer(s).

location=name

Name of new location to create based on the file, if so desired.

GDAL Supported Formats

Full details on GDAL supported formats are available at:

http://www.remotesensing.org/gdal/formats_list.html

Selected formats:

Format Name	Georeferencing
Arc/Info Binary Grid	Yes
CEOS (Spot for instance)	No
First Generation USGS DOQ	Yes
New Labelled USGS DOQ	Yes
Military Elevation Data (DTED)	Yes
Eosat Fast Format	No
ESRI .hdr Labelled	Yes
TIFF / GeoTIFF	Yes
Erdas Imagine .hfa	No
JPEG JFIF	No
PCI .aux Labelled	Yes
Portable Network Graphics (PNG)	No
USGS SDTS DEM	Yes
SAR CEOS	No

If GDAL doesn't support the indicated file it will report it with a message like `ERROR 4: `abc.e00' not recognised as a supported file format`. This means either that the file doesn't exist, that it isn't supported by GDAL, or that the format wasn't built into the version of libgdal.1.1.so being used.

r.in.gdal will only operate correctly if the GDAL shared library is installed on the system and accessible (normally via `LD_LIBRARY_PATH`). If not, `r.in.gdal` will report `Unable to initialize GDAL bridge`. If this occurs it means that you either don't have GDAL installed (ie. libgdal.1.1.so), it isn't in the `LD_LIBRARY_PATH`, or it is the wrong version. Recent builds of libgdal may be available at <http://gdal.velocet.ca/projects/grass>, and the shared library should be copied into `$GISBASE/lib`. Binary distributions of GRASS will normally include libgdal.

`r.in.gdal` uses a number of strategies to find the GDAL shared library. It first looks in `$GISBASE/lib/`. Then it looks for `$GDAL_HOME` and finally it searches without a path letting all the standard Unix `dlopen()` search rules apply (which normally includes `LD_LIBRARY_PATH`).

Location Creation

r.in.gdal attempts to preserve projection information when importing datasets if the source format includes projection information, and if the GDAL driver supports it. If the projection of the source dataset does not match the projection of the current location **r.in.gdal** will report an error message (`Projection of dataset does not appear to match current location`) and then report the `PROJ_INFO` parameters of the source dataset.

If the user wishes to ignore the difference between the apparent coordinate system of the source data and the current location, they may pass the `-o` flag to override the projection check.

If the user wishes to import the data with the full projection definition, it is possible to have `r.in.gdal` automatically create a new location based on the projection and extents of the file being read. This is accomplished by passing the name to be used for the new location via the **location** parameter. Upon completion of the command, a new location will have been created (with only a PERMANENT mapset), and the raster will have been imported with the indicated **output** name into the PERMANENT mapset.

Support for GCPs: In case the image contains GCPs they are written to a POINTS file within an imagery group. They can directly be used for [i.rectify](#). The *target* option allows to automatically re-project the GCPs from their own projection into another projection read from the PROJ_INFO file of the location name *target*.

NOTES

I plan to make a variety of improvements to **r.in.gdal** in the future including support for reporting everything known about a dataset if the **output** parameter is not set.

The **r.in.gdal** command does support the following features, as long as the underlying format driver supports it:

Color Table

Bands with associated colortables will have the color tables transferred. Note that if the source has no colormap, `r.in.gdal` in GRASS 5.0 will emit no colormap. Use `r.colors map=... color=grey` to assign a greyscale colormap. In a future version of GRASS `r.in.gdal` will likely be upgraded to automatically emit greyscale colormaps.

Data Types

Most GDAL data types are supported. Float32 and Float64 type bands are translated as GRASS floating point cells (but not double precision ... this could be added if needed), and most other types are translated as GRASS integer cells. This includes 16bit integer data sources. Complex (some SAR signal data formats) data bands are translated to two floating point cell layers (*.real and *.imaginary).

Georeferencing

If the dataset has affine georeferencing information, this will be used to set the north, south, east and west edges. Rotational coefficients will be ignored, resulting in incorrect positioning for rotated datasets.

Projection

The datasets projection will be used to compare to the current location or to define a new location. Internally GDAL represents projections in OpenGIS Well Known Text format. A large subset of the total set of GRASS projections are supported.

Null Values

Raster bands for which a null value is recognised by GDAL will have the null pixels transformed into GRASS style nulls during import. Many generic formats (and formats poorly supported by GDAL) do not have a way of recognising null pixels in which case `r.null` should be used after the import.

GCPs

Datasets that have Ground Control Points will have them imported as a POINTS file associated with the imagery group. Datasets with only one band that would otherwise have been translated as a simple raster map will also have an associated imagery group if there are ground control points. The coordinate system of the ground control points is reported by `r.in.gdal` but not preserved. It is up to the user to ensure that the location established with `i.target` has a compatible coordinate system before using the points with `i.rectify`.

SEE ALSO

[*r.in.ascii*](#), [*r.in.arc*](#), [*r.in.bin*](#), [*r.null*](#)

REFERENCES

GDAL Pages: <http://www.remotesensing.org/gdal/>,

Precompiled libgdal for GRASS: <http://gdal.velocet.ca/projects/grass>

AUTHOR

[Frank Warmerdam](mailto:warmerdam@pobox.com) (warmerdam@pobox.com).

Last changed: \$Date: 2002/08/12 21:45:18 \$



NAME

r.in.gif – Converts a GIF87 (8bit) raster file to a GRASS raster file.
(*GRASS Raster Data Import Program*)

SYNOPSIS

r.in.gif [-v] **input=name** **output=name** [**title=name**]

DESCRIPTION

This program converts a GIF raster file (8bit) to a GRASS raster file. Output is placed in the */cell* directory under the user's current GRASS mapset.

OPTIONS

Flag:

-v
 Verbose mode

Parameters:

input=name
 Name of an existing GIF 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.

The program prompts the user to enter the name of the GIF raster file to be converted and the name to be assigned to the GRASS raster file to contain the resultant image. The user should adjust boundary coordinates stored in the cell header after import using [r.support](#) .

The user must, of course, first create the GIF raster file to be converted (e.g., from a scanning system that produces GIF raster file format).

NOTE

The GIF89 format is currently not accepted.

SEE ALSO

[*r.support*](#) and [*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.in.hdf – Convert data in HDF format into a (binary) raster map layer.
(*GRASS Raster Data Import Program*)

GRASS VERSION

4.x

SYNOPSIS

r.in.hdf

r.in.hdf help

r.in.hdf [-qal] **input**=name [**output**=name] [**mult**=value] [**dsets**=value[,value,...]]

DESCRIPTION

r.in.hdf allows a user to create a GRASS raster map layer from a file in NCSA Hierarchical Data Format (HDF).

OPTIONS

Flags:

- q Run quietly.
- a Convert ALL data sets in the HDF file.
- l Only list the contents of the HDF file (no conversion).

Parameters:

input

HDF file to be converted.

output

Name of new raster file.

Default: hdf.rast

mult

Floating point multiplier. (rastfile = (int)(file.hdf * multiplier))

dsets

GRASS Raster Commands

A list of reference numbers for datasets to be extracted.

The raster file created will have its southwest origin at 0 East and 0 North, with resolution of 1. To view the new GRASS raster file, use *g.region rast=newfile*, then use *d.rast* as normal. You may wish to manually change the new raster's cellhd file.

NCSA HDF is a multi-object file format developed by The National Center for Supercomputing Applications at Champaign, Illinois for the transfer of graphical and floating-point data between machines. NASA Pathfinder AVHRR data is stored in HDF format. The HDF format defines both a raster type and a SDS (scientific data set) type. The later is basically a highly structured multi-dimensional array of values. A single HDF file may contain more than one SDS or raster; using the *-a* option will extract all data sets from the HDF file, creating a separate GRASS raster file from each data set and naming the raster files outputname01, outputname02, outputname03.... If the *-a* option is not specified and the HDF file contains more than one data set, only the first data set is extracted unless the user specifies specific reference numbers for desired data sets in the file using the *dssets* option. To see a list of reference numbers for data sets in an HDF file, use the *-l* flag. If the HDF file contains labels or descriptions of the data, these will be shown when using the *-l* flag. Labels and descriptions from the HDF file will also be written to the GRASS history file when creating the raster map. If a multiplier is given using the *mult* option, every file created will be the product of the input data set and the multiplier. If the HDF file contains an SDS, it must be only two dimensional in order for *r.in.hdf* to accept it as input. If the HDF file contains an 8-bit raster image with an associated palette, a GRASS color file will be created. This program will not import 24-bit HDF raster images.

NCSA distributes the HDF library and several public domain visualization applications which use the HDF format. Some commercial applications also support HDF. For more information, use the NCSA anonymous ftp server <ftp.ncsa.uiuc.edu> or contact:

NCSA
152 Computing Applications Building
605 E. Springfield Ave.
Champaign, IL 61820
(217) 244-0072

BUGS

If a data set contains long labels and descriptions, some of it may be truncated when writing to the GRASS history file. The GRASS history structure currently allows 50 lines of 80 characters each. But when editing history using *r.support*, only 20 lines of 65 characters may be used.

SEE ALSO

[r.out.hdf](#), [r.support](#), [g.region](#)

AUTHOR

Bill Brown, U.S. Army Construction Engineering Research Laboratory

Last changed: \$Date: 2002/01/25 05:45:34 \$



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.

OPTIONS

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

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}

GRASS Raster Commands

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](#), [r.in.ascii](#), [r.in.bin](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.in.miads – Converts a MIADS output ASCII text file into an GRASS raster import (*r.in.ascii*) formatted file.

(SCS GRASS Raster Program)

SYNOPSIS

r.in.miads

r.in.miads help

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

DESCRIPTION

r.in.miads allows a user to create a *r.in.ascii* formatted ASCII file from a MIADS output printer format ASCII file.

The program will actively read the MIADS data file, selectively remove and process each strip, creating a individual *r.in.ascii* formatted file for each strip, and finally create one category file for all strips. The program also produces a report file summarizing all pertinent information for each strip.

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

COMMAND LINE OPTIONS

Parameters:

input=name

MIADS input file name.

output=name

GRASS raster data output file name.

strip=value

MIADSs strip number of reference cell.

line=value

MIADSs line number of reference cell.

cell=value

MIADS cell number of reference cell.

Northing=value

UTM Easting at the cell reference.

Easting=value

UTM Northing at the cell reference.

size=value

Cell size (length one side) in meters.

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

run.miads –

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

getstrip –

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

strip.99s –

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

SEE ALSO

[*d.display*](#), [*d.rast*](#), [*r.in.ascii*](#), [*r.patch*](#), [*r.support*](#)

AUTHOR

r.in.miads – R.L.Glenn, USDA, SCS, NHQ–CGIS

run.miads, *strip.99s* – Harold Kane, USDA, SCS, Oklahoma State Office

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.in.pbm – Converts a PBM image file to a GRASS raster file.
(*GRASS Raster Data Import Program*)

SYNOPSIS

r.in.pbm [-v] **input**=*name* **output**=*name*

DESCRIPTION

This program converts a PBM raster file to a GRASS raster map.

OPTIONS

Flags:

-v
 Verbose mode.

Parameters:

input=*name*
 Name of an existing PBM raster file to be imported.
output=*name*
 Name to be assigned to resultant raster map layer.

The user should adjust boundary coordinates stored in the cell header after import using *r.support* or *r.region*.

SEE ALSO

[*r.in.pgm*](#)
[*r.in.ppm*](#)
[*r.support*](#)
[*r.region*](#)
[*parser*](#)

AUTHOR

Glynn Clements

Last changed: \$Date: 2002/04/13 10:52:46 \$



NAME

r.in.pgm – Converts a PGM image file to a GRASS raster file.
(*GRASS Raster Data Import Program*)

SYNOPSIS

r.in.pgm [-vf] *input=name output=name*

DESCRIPTION

This program converts a PGM raster file to a GRASS raster map.

OPTIONS

Flags:

- v Verbose mode.
- f Create a floating-point map (range 0.0 – 1.0).

Parameters:

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

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

The user should adjust boundary coordinates stored in the cell header after import using *r.support* or *r.region*.

SEE ALSO

[r.in.pbm](#)
[r.in.ppm](#)
[r.support](#)
[r.region](#)
[parser](#)

AUTHOR

Glynn Clements

Last changed: \$Date: 2002/04/13 10:52:46 \$



NAME

r.in.png – Converts a PNG image file to a GRASS raster file.
(*GRASS Raster Data Import Program*)

SYNOPSIS

r.in.png [-hv] **input=name** **output=name** [**title=string**] [**gamma=value**] [**alpha=value**]

DESCRIPTION

This program converts a PNG raster file to a GRASS raster file.

OPTIONS

Flags:

- f Create floating-point maps (range 0.0 – 1.0).
- h Output image file header only; don't create map(s).
- v Verbose mode.

Parameters:

input=name

Name of an existing PNG raster file to be imported.

output=name

Name to be assigned to resultant raster map layer(s).

title=string

Title to be assigned to resultant raster map layer(s).

gamma=value

Perform gamma correction using this value for the display gamma.

alpha=value

Pixels with an alpha (transparency) value less than or equal to this value are imported as no-data cells.

NOTES

The output depends upon the format of the PNG image. Gray-scale and paletted (indexed, color-mapped)

images will result in a single output map, with the name specified by the **output** option.

True-color (RGB) images will result in three output maps, whose names are formed by appending ".r", ".g" and ".b" to the name specified by the **output** option.

For both gray-scale and true-color images, if transparency information is present, an additional map will be created whose name is formed by appending ".a" to the name specified by the **output** option.

For paletted images, the resulting cell values will be the color indices from the image, with the map's color table generated from the image's palette. For gray-scale and true-color images, the cell values will be the intensity values from the image. The map's color table will map these values to the normalized intensity levels, incorporating any gamma correction.

However, when the **-f** flag is used, any scaling or gamma correction is performed on the actual cell values. The map's color table will be a linear gray scale. Also, paletted images will first be expanded to true-color.

The user should adjust boundary coordinates stored in the cell header after import using *r.support* or *r.region*.

The **gamma** option is only used if the image contains a gamma specification.

The **alpha** option is only relevant if the image contains transparency information.

SEE ALSO

[*r.composite*](#)

[*r.support*](#)

[*r.region*](#)

[*parser*](#)

AUTHOR

Glynn Clements

Last changed: \$Date: 2002/05/10 08:20:22 \$



NAME

r.in.poly – Create raster maps from ascii polygon/line data files in the current directory.
(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 binary raster maps from ASCII files in the current directory containing polygon and linear features.

OPTIONS

Parameters:

input=name

Unix input file, in ascii format, containing the polygon and linear features. The format of this file is described in the section [INPUT FORMAT](#) below.

output=name

Raster output file

TITLE="phrase"

Title for resultant raster map (optional)

rows=value

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
```

GRASS Raster Commands

```
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).

EXAMPLE

An area described by four points:

```
A
591316.80  4926455.50
591410.25  4926482.40
591434.60  4926393.60
591341.20  4926368.70
= 42 stadium
```

SEE ALSO

[r.digit](#) for interactive on–screen polygon/line digitizing for raster maps

[r.colors](#) for creates color tables for raster maps

AUTHOR

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

Last changed: \$Date: 2003/07/04 08:35:34 \$



NAME

r.in.ppm – Converts a PPM image file to a GRASS raster file.
(*GRASS Raster Data Import Program*)

SYNOPSIS

r.in.ppm [-vfb] *input=name output=name*

DESCRIPTION

This program converts a PPM raster file to a GRASS raster file.

OPTIONS

Flags:

- v* Verbose mode.
- f* Create floating-point maps (range 0.0 – 1.0). Not applicable to composite maps created by the *-c* flag.
- b* Create 3 separate raster maps of the red/green/blue levels
- c* Create a composite color map

Parameters:

- input=name*
Name of an existing PPM raster file to be imported.
- output=name*
Name to be assigned to resultant raster map layer(s).

NOTES

R/G/B levels: each output raster map layer is given a linear gray scale color table. Each output raster map layer is assigned the user-specified *output* name with (.r, .g, .b) suffix.

The user should adjust boundary coordinates stored in the cell header after import using *r.support* or *r.region*.

Composite color maps are generated by generating the separate red, green and blue maps, then invoking [r.composite](#) to generate the composite map. Unless the **-b** flag was used, the red, green and blue maps will then be deleted.

Greater control may be exercised over the creation of composite color maps by invoking [r.composite](#) manually.

SEE ALSO

[r.in.pbm](#)

[r.in.pbm](#)

[r.composite](#)

[r.support](#)

[r.region](#)

[parser](#)

AUTHOR

Glynn Clements

Last changed: \$Date: 2002/04/14 06:55:19 \$



NAME

r.in.shape – Read an ArcView Shapefile (polygons and lines)

SYNOPSIS

r.in.shape *in=name* *out=name* [*cat=category column*] [*lab=category label column*]

DESCRIPTION

The *r.in.shape* program is designed to import ArcView Shapefiles (polygons and lines) into GRASS binary raster maps.

OPTIONS

Parameters:

in=name

Name of input shape file. Provide a full path name or the name of a file in the current directory. The *.shp* extension is required.

out=name

Raster output file.

cat=category_column

Name of the attribute to use as the category number in *dig_att*. Category column must be **integer**.

lab=label_column

Name of the attribute to use as the category label in *dig_cats*.

NOTE

r.in.shape creates ASCII file first and then run *r.in.poly*.

SEE ALSO

[v.in.shape](#), [v.to.rast](#).

AUTHORS

Radim Blazek (Radim.Blazek@dhv.cz)

Based on Shapelib (<http://gdal.velocet.ca/projects/shapelib/>)

Last changed: \$Date: 2002/01/25 05:45:34 \$



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

GRASS Raster Commands

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.in.tiff – Imports a TIFF (8 or 24 bit) raster file into GRASS raster file(s).
(*GRASS Raster Data Import Program*)

SYNOPSIS

r.in.tiff [-v] **input=name output=name**

DESCRIPTION

This program imports a TIFF raster file (8 or 24 bit) to GRASS raster file(s). Output is placed in the */cell* directory under the user's current GRASS mapset.

Flag:

-v

Verbose mode

Parameters:

input=name

Name of an existing TIFF raster file to be imported.

output=name

Name to be assigned to resultant binary raster map layer.

The program prompts the user to enter the name of the TIFF raster file to be converted and the name to be assigned to the GRASS raster file to contain the resultant image. Currently "TIFF/uncompressed", "TIFF/PackBits-compression" and "TIFF/deflate" formats are supported. Other compression schemes may not be read correctly. LZW compression is not supported due to UNISYS patent.

If the image is a 24-bit RGB TIFF, then the output will be split into respective red, green and blue band files; each having the *output* name as its prefix followed by *.r*, *.g* or *.b*, respectively. Color composites can be displayed with [d.rgb](#).

If a TIFF World file exists with an extension of "tfw", "tifw", "TFW", or "TIFW", it will be used to set the boundary coordinates and cell size. Rotation in the TIFF World file is not supported (second and third numbers), and if present the TIFF will be imported as an XY (unreferenced) image. If no TIFF World file exists, the user should adjust boundary coordinates stored in the cell header after import using [r.support](#).

A sample TFW file opened up would look like:

GRASS Raster Commands

```
        6.00  
        0.00  
        0.00  
       -6.00  
1709053.00  
807714.00
```

```
Line 1 - Cell size in the "X" direction  
Line 2 - "Rotation" term for the row  
Line 3 - "Rotation" term for the column  
Line 4 - Cell size in the "Y" direction  
Line 5 - Easting value of insertion point "X"  
Line 6 - Northing value of insertion point "Y"
```

Note: The y-scale (4) is negative because the origins of an image and a geographic coordinates system are different. The origin of an image is located in the upper-left corner, whereas the origin of the map coordinate system is located in the lower-left corner. Row values in the image increase from the origin downward, while y-coordinate values in the map increase from the origin upward.

NOTE

This module only supports TIFF's conforming to the base TIFF 6 specification. These include monochrome, grayscale, palette and RGB ("True Color"), with at most 8 bits per "sample". Other features, like larger bits/sample, floating point (or rational) samples, alpha blending, transparency masks, and multiband images, are not currently handled.

This module also doesn't handle true GeoTIFF's (those with coordinate reference information actually embedded in the TIFF file). So the assumption with the TIFF world file is that it make sense in whatever coordinate system the current region settings return.

SEE ALSO

[r.support.parser](#) and [r.out.tiff, r.in.gdal](#)

AUTHOR

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

GRASS 5.0 team

Last changed: \$Date: 2002/05/10 08:06:07 \$



NAME

r.kappa – Calculate error matrix and kappa parameter for accuracy assessment of classification result.
(GRASS Raster Program)

SYNOPSIS

r.kappa

r.kappa help

r.kappa [-mwqzh] **classification**=name **reference**=name **output**=name [TITLE="name"]

DESCRIPTION

r.kappa tabulates the error matrix of classification result by crossing classified map layer with respect to reference map layer. Both overall *kappa* (accompanied by its *variance*) and conditional *kappa* values are calculated. This analysis program respects the current geographic region and mask settings.

OPTIONS

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

r.kappa [-mwqzh] **classification**=name **reference**=name **output**=name [TITLE="name"]

Flags:

- m** Report zero values due to mask.
- w** Print a wide report, 132 columns (default: 80)
- q** Run quietly,
- z** Report non-zero values only.
- h** Report without header.

Parameters:

classification=name

Name of classified raster map layer.

reference=name

Name of reference raster map layer.

output=name

Name of ascii file in which to output report results.

TITLE=name

Title of the report file which is the first line in output file.

Default: *ACCURACY ASSESSMENT*

Alternately, the user can run *r.kappa* interactively by simply typing *r.kappa* without command line arguments; in this case, the user will be prompted for the names of classified result map and reference map, which will be the subjects of the analysis. *r.kappa* then calculates the error matrix of the two map layers and prepares the table from which the report is to be created. *kappa* values for overall and each classes are computed along with their variances. Also percent of commission and omission error, total correct classified result by pixel counts, total area in pixel counts and percentage of overall correctly classified pixels are tabulated.

The report will be write to an output file which is in plain text format and named by user at prompt of running the program.

The body of the report is arranged in panels. The classified result map layer categories is arranged along the vertical axis of the table, while the reference map layer categories along the horizontal axis. Each panel has a maximum of 5 categories (9 if wide format) across the top. In addition, the last column of the last panel reflects a cross total of each column for each row. All of the categories of the map layer arranged along the vertical axis, i.e., the reference map layer, 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.

NOTES

It is recommended to reclassify categories of classified result map layer into a more manageable number before running *r.kappa* on the classified raster map layer. Because *r.kappa* calculates and then reports information for each and every category.

NA's in output file mean non-applicable in case *MASK* exists.

SEE ALSO

[*g.region*](#)

[*m.ipf*](#)

[*r.cats*](#)

[*r.mask*](#)

[*r.reclass*](#)

[*r.report*](#)

[*r.stats*](#)

AUTHOR

Tao Wen, University of Illinois at Urbana-Champaign, Illinois

Last changed: \$Date: 2003/03/16 12:25:19 \$

A GRASS Program for Determining the Topology of Stream Networks

John F. Stamm

Department of Geological Sciences, Case Western Reserve University,

Cleveland, OH 44106–7216, email: jfs5@po.cwru.edu

Introduction

This paper presents a GRASS (geographical resource analysis support system; Shapiro et al., 1992; United States Army Corps of Engineers, 1993) program for determining the topology of stream networks. The program inputs raster files generated by the GRASS program *r.watershed*. Because it determines the relationships of tributary streams, it is called *r.tribs*. The input files required are:

stream: a raster map of stream segments; stream segments are labelled using integer values greater than or equal to 2,

accumulation: a raster map of the number of cells that drain through each cell; absolute values is the amount of overland flow that is routed through a pixel,

drainage: a raster map of drainage directions; if there is no flow direction, a value of –1 is assigned. Otherwise, the integers 1 through 8 are assigned to the compass directions shown below:

5	6	7
4	–1	8
3	2	1

For example, a value of 2 means that the pixel drains south, a value of 5 means the pixel drains to the northwest.

Two tables are output by *r.tribs*. The first table list the tributaries associated with each stream segment. The second table list streams and tributaries in an order such that a stream and its tributaries are printed only if the tributaries have been previously listed. That is, first order streams are listed first, streams with tributaries that are first order streams are listed second, and so on. The program can also be run in debugging mode, in which more detailed information is printed.

The program *r.tribs* was written so that GRASS can be used to generate input for a runoff and erosion model called KINEROS (Smith et al., in press; Woolhiser et al., 1990). KINEROS represents a watershed as a set of related elements. Elements may be hillslopes, channels or ponds. A computational order must be specified so that boundary conditions for an element, such as the amount of water contributed by lateral hillslopes and upstream tributaries, are available. The program *r.tribs* provides KINEROS with that computational order.

Input Files

We assume that the program *r.watershed* has been run, and that the input files *stream*, *accumulation* and *drainage* are available. Assume these files are called *stream.0*, *accum.0* and *drain.0*, respectively. The command–line version of *r.tribs* is:

```
GRASS 4.1 > r.tribs st=stream.0 ac=accum.0 dr=drain.0
```

Alternately, the command *r.tribs* can be entered on the command line:

```
GRASS 4.1 > r.tribs
```

and the user will be prompted for these files.

Methodology and Example of Output

This is followed by a listing of the tributaries associated with each stream. The tributaries of a stream are determined by using the following methodology.

- a. The *stream* and *accumulation* files are scanned to determine the locations of pixels that are stream segments and have the lowest accumulation of runoff for that segment. These points mark the locations of the heads of the stream segments. Locations of the head of a stream are stored for the next step.
- b. Points within one pixel of the head are scanned to determine if they are part of a different stream segment and if they drain into the head of this stream. The *drainage* raster file is used to determine the drainage direction. Any point meeting this criteria are listed as tributary streams.
- c. All points with no tributaries are first order. Tributary values are set to zero.
- d. The label of the stream and its tributaries are stored and printed to the screen.

We have used *r.tribs* to generate output for a small watershed (17.5 km²) in Idaho (Horse Creek) in which we used *r.watershed* to define a total of 50 stream segments. Note that stream segments and tributaries are labelled using even integers ranging from 2 to 102. Stream segment 2 is located at the mouth of the watershed.

The *r.tribs* program first outputs the number of rows (*nrows*) and columns (*ncols*) in the raster files. Then the maximum and minimum labels of stream segments are printed. This is followed by two tables. The first table lists streams and their tributaries. An example of this table is given below.

```
nrows: 138
```

```
ncols: 273
```

```
Max stream index = 102
```

```
Min stream index = 2
```

```
Stream: 2 Tributary 0: 6 Tributary 1: 4
```

```
Stream: 4 Tributary 0: 0 Tributary 1: 0
```

```
Stream: 6 Tributary 0: 14 Tributary 1: 8
```

```
Stream: 8 Tributary 0: 12 Tributary 1: 10
```

```
Stream: 10 Tributary 0: 0 Tributary 1: 0
```

```
Stream: 12 Tributary 0: 0 Tributary 1: 0
```


GRASS Raster Commands

Stream: 14 Tributary 0: 22 Tributary 1: 16
Stream: 16 Tributary 0: 20 Tributary 1: 18
Stream: 18 Tributary 0: 0 Tributary 1: 0
Stream: 20 Tributary 0: 0 Tributary 1: 0
Stream: 22 Tributary 0: 102 Tributary 1: 24
Stream: 24 Tributary 0: 36 Tributary 1: 26
Stream: 26 Tributary 0: 30 Tributary 1: 28
Stream: 28 Tributary 0: 0 Tributary 1: 0
Stream: 30 Tributary 0: 34 Tributary 1: 32
Stream: 32 Tributary 0: 0 Tributary 1: 0
Stream: 34 Tributary 0: 0 Tributary 1: 0
Stream: 36 Tributary 0: 100 Tributary 1: 38
Stream: 38 Tributary 0: 42 Tributary 1: 40
Stream: 40 Tributary 0: 0 Tributary 1: 0
Stream: 42 Tributary 0: 98 Tributary 1: 44
Stream: 44 Tributary 0: 96 Tributary 1: 46
Stream: 46 Tributary 0: 50 Tributary 1: 48
Stream: 48 Tributary 0: 0 Tributary 1: 0
Stream: 50 Tributary 0: 58 Tributary 1: 52
Stream: 52 Tributary 0: 56 Tributary 1: 54
Stream: 54 Tributary 0: 0 Tributary 1: 0
Stream: 56 Tributary 0: 0 Tributary 1: 0
Stream: 58 Tributary 0: 90 Tributary 1: 60
Stream: 60 Tributary 0: 88 Tributary 1: 62
Stream: 62 Tributary 0: 66 Tributary 1: 64
Stream: 64 Tributary 0: 0 Tributary 1: 0

GRASS Raster Commands

```
Stream: 66 Tributary 0: 74 Tributary 1: 68
Stream: 68 Tributary 0: 72 Tributary 1: 70
Stream: 70 Tributary 0: 0 Tributary 1: 0
Stream: 72 Tributary 0: 0 Tributary 1: 0
Stream: 74 Tributary 0: 82 Tributary 1: 76
Stream: 76 Tributary 0: 80 Tributary 1: 78
Stream: 78 Tributary 0: 0 Tributary 1: 0
Stream: 80 Tributary 0: 0 Tributary 1: 0
Stream: 82 Tributary 0: 86 Tributary 1: 84
Stream: 84 Tributary 0: 0 Tributary 1: 0
Stream: 86 Tributary 0: 0 Tributary 1: 0
Stream: 88 Tributary 0: 0 Tributary 1: 0
Stream: 90 Tributary 0: 94 Tributary 1: 92
Stream: 92 Tributary 0: 0 Tributary 1: 0
Stream: 94 Tributary 0: 0 Tributary 1: 0
Stream: 96 Tributary 0: 0 Tributary 1: 0
Stream: 98 Tributary 0: 0 Tributary 1: 0
Stream: 100 Tributary 0: 0 Tributary 1: 0
Stream: 102 Tributary 0: 0 Tributary 1: 0
```

If the program is run in debugging mode, then the values of *stream*, *accumulation* and *direction* of the head of the stream and the surrounding 8 pixels are also printed. Set the value of `DB_FIND_TRIBS` (defined in the beginning of the routine `find_tribs`) to a value of 1 and recompile the program to activate debugging mode.

The second table that is output by *r.tribs* list the streams and their associated tributaries in their proper computational order. The program loops through the data listed above several times. First streams with no tributaries are listed (LOOP 1). Then streams with only first-order streams as tributaries are listed (LOOP 2). Then, streams with tributaries listed in previous loops are listed. The program continues until all streams have been listed. The variable "Order" is also listed, which can be interpreted as the computational order. This is the order in which programs, such as KINEROS, must consider streams in the network such that data for tributaries will be available when considering the listed stream. An example listing of the computational order of streams is given below.

GRASS Raster Commands

Computational Order of Stream Segments:

LOOP: 1

Order: 0 Stream: 4 Tributary 0: 0 Tributary 1: 0

Order: 1 Stream: 10 Tributary 0: 0 Tributary 1: 0

Order: 2 Stream: 12 Tributary 0: 0 Tributary 1: 0

Order: 3 Stream: 18 Tributary 0: 0 Tributary 1: 0

Order: 4 Stream: 20 Tributary 0: 0 Tributary 1: 0

Order: 5 Stream: 28 Tributary 0: 0 Tributary 1: 0

Order: 6 Stream: 32 Tributary 0: 0 Tributary 1: 0

Order: 7 Stream: 34 Tributary 0: 0 Tributary 1: 0

Order: 8 Stream: 40 Tributary 0: 0 Tributary 1: 0

Order: 9 Stream: 48 Tributary 0: 0 Tributary 1: 0

Order: 10 Stream: 54 Tributary 0: 0 Tributary 1: 0

Order: 11 Stream: 56 Tributary 0: 0 Tributary 1: 0

Order: 12 Stream: 64 Tributary 0: 0 Tributary 1: 0

Order: 13 Stream: 70 Tributary 0: 0 Tributary 1: 0

Order: 14 Stream: 72 Tributary 0: 0 Tributary 1: 0

Order: 15 Stream: 78 Tributary 0: 0 Tributary 1: 0

Order: 16 Stream: 80 Tributary 0: 0 Tributary 1: 0

Order: 17 Stream: 84 Tributary 0: 0 Tributary 1: 0

Order: 18 Stream: 86 Tributary 0: 0 Tributary 1: 0

Order: 19 Stream: 88 Tributary 0: 0 Tributary 1: 0

Order: 20 Stream: 92 Tributary 0: 0 Tributary 1: 0

Order: 21 Stream: 94 Tributary 0: 0 Tributary 1: 0

Order: 22 Stream: 96 Tributary 0: 0 Tributary 1: 0

Order: 23 Stream: 98 Tributary 0: 0 Tributary 1: 0

GRASS Raster Commands

Order: 24 Stream: 100 Tributary 0: 0 Tributary 1: 0

Order: 25 Stream: 102 Tributary 0: 0 Tributary 1: 0

LOOP: 2

Order: 26 Stream: 8 Tributary 0: 12 Tributary 1: 10

Order: 27 Stream: 16 Tributary 0: 20 Tributary 1: 18

Order: 28 Stream: 30 Tributary 0: 34 Tributary 1: 32

Order: 29 Stream: 52 Tributary 0: 56 Tributary 1: 54

Order: 30 Stream: 68 Tributary 0: 72 Tributary 1: 70

Order: 31 Stream: 76 Tributary 0: 80 Tributary 1: 78

Order: 32 Stream: 82 Tributary 0: 86 Tributary 1: 84

Order: 33 Stream: 90 Tributary 0: 94 Tributary 1: 92

LOOP: 3

Order: 34 Stream: 26 Tributary 0: 30 Tributary 1: 28

Order: 35 Stream: 74 Tributary 0: 82 Tributary 1: 76

LOOP: 4

Order: 36 Stream: 66 Tributary 0: 74 Tributary 1: 68

LOOP: 5

Order: 37 Stream: 62 Tributary 0: 66 Tributary 1: 64

LOOP: 6

Order: 38 Stream: 60 Tributary 0: 88 Tributary 1: 62

GRASS Raster Commands

LOOP: 7

Order: 39 Stream: 58 Tributary 0: 90 Tributary 1: 60

LOOP: 8

Order: 40 Stream: 50 Tributary 0: 58 Tributary 1: 52

LOOP: 9

Order: 41 Stream: 46 Tributary 0: 50 Tributary 1: 48

LOOP: 10

Order: 42 Stream: 44 Tributary 0: 96 Tributary 1: 46

LOOP: 11

Order: 43 Stream: 42 Tributary 0: 98 Tributary 1: 44

LOOP: 12

Order: 44 Stream: 38 Tributary 0: 42 Tributary 1: 40

LOOP: 13

Order: 45 Stream: 36 Tributary 0: 100 Tributary 1: 38

LOOP: 14

Order: 46 Stream: 24 Tributary 0: 36 Tributary 1: 26

LOOP: 15

Order: 47 Stream: 22 Tributary 0: 102 Tributary 1: 24

11/06/2003

LOOP: 16

Order: 48 Stream: 14 Tributary 0: 22 Tributary 1: 16

LOOP: 17

Order: 49 Stream: 6 Tributary 0: 14 Tributary 1: 8

LOOP: 18

Order: 50 Stream: 2 Tributary 0: 6 Tributary 1: 4

Obtaining *r.tribs* via FTP or email

The C programs that are required to generate *r.tribs* are available via anonymous ftp to moon.cecer.army.mil. I have printed out the main segment of the *r.tribs* program in Appendix I. This was done to illustrate how routines in the GRASS library are used to read in raster data. Routines from the GRASS library begin with "G_". Two utility programs are also used: imatrix and ivector. These are discussed by Press and others (1989), and are used to allocate space for arrays. Comments in the code discuss the details of each routine. Appendix II lists the makefile used to compile *r.tribs*. Note that the variable GIS must be edited so that the proper path to the grass directory is specified.

As previously mentioned, the program can be obtained by anonymous ftp to moon.cecer.army.mil. Change to the "incoming/r.tribs" directly to get the files. The program can also be obtained by contacting the author via email (jfs5@po.cwru.edu). The code has been commented to help the user to understand the program structure.

References

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., 1989, Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, 735 pp.

Shapiro, M., Westervelt, J., Gerdes, D., Larson, M., and Brownfield, K.R., 1992, GRASS 4.0 Programmer's Manual, U.S. Army Construction Engineering Research Laboratory, Champaign, Illinois, 292 pp.

Smith, R.E., Goodrich, D.C., Woolhiser, D.A., and Unkrich, C.L., in press, A KINematic Runoff and EROsion Model, in: V.P. Singh (Ed.), Computer Models of Watershed Hydrology, Water Resources Pub., Highlands Ranch, Colorado

United States Army Corps of Engineers, 1993, GRASS4.1 Users's Reference Manual, U.S. Army Construction Engineering Research Laboratories, Champaign, Illinois, 556 pp.

Woolhiser, D.A., Smith, R.E., Goodrich, D.C., 1990, KINEROS, A Kinematic Runoff and Erosion Model: Documentation and Users Manual, U.S. Department of Agriculture, Agricultural Research Service, ARS-77, 130 pp.

Appendix I: main.c code for r.tribs

```

#include "gis.h"

#include <stdio.h>

int **imatrix();

int *ivector();

/*
 * Program to determining the topology of a stream network. A
 * table is generated that reports the tributaries that are
 * at the head of each stream segment. The computational order
 * of streams is also determined.
 *
 * Written by:
 * Dr. John F. Stamm
 * Department of Geological Sciences
 * Case Western Reserve University
 * Cleveland, OH 44106-7216
 * email: jfs5@po.cwru.edu
 */

main(argc,argv)

int argc;

char *argv[];

{
/*
 * Matricies
 */

int **accum;

```

GRASS Raster Commands

```
int **chann;

int **aspect;

CELL *cell;

char *chann_name;

char *accum_name;

char *aspect_name;

char *mapset;

int col;

int fd_accum;

int fd_chann;

int fd_aspect;

int ncols;

int nrows;

int row;

struct {

struct Option *accum ;

struct Option *chann ;

struct Option *aspect ;

} parm;

/*

* Allocate memory for the Option structure and return a

* pointer to this structure. Do this for the structured

* variables parm.accum and parm.chann.

*

* Set values for parm.accum

*/
```


GRASS Raster Commands

```
parm.accum = G_define_option() ;
parm.accum->key = "accumulation";
parm.accum->type = TYPE_STRING;
parm.accum->required = YES;
parm.accum->gisprompt = "old,cell,raster" ;
parm.accum->description= "Name of the ACCUMULATION map" ;
/*
 * Set values for parm.chann
 */
parm.chann = G_define_option() ;
parm.chann->key = "stream";
parm.chann->type = TYPE_STRING;
parm.chann->required = YES;
parm.chann->gisprompt = "old,cell,raster" ;
parm.chann->description= "Name of the STREAM map" ;
/*
 * Set values for parm.aspect
 */
parm.aspect = G_define_option() ;
parm.aspect->key = "drainage";
parm.aspect->type = TYPE_STRING;
parm.aspect->required = YES;
parm.aspect->gisprompt = "old,cell,raster" ;
parm.aspect->description= "Name of the DRAINAGE DIRECTION
map";
/*
```

GRASS Raster Commands

```
* Initailize GIS library for this program
*/
G_gisinit(argv[0]);
/*
* Parse values from the command line. If this is not
* successful, then display a usage statement and exit.
*/
if (G_parser(argc, argv))
    exit (-1);
accum_name = parm.accum->answer;
chann_name = parm.chann->answer;
aspect_name = parm.aspect->answer;
/*
* Find the name of mapset that we are going to use.
*/
mapset = G_find_cell2 (accum_name, "");
if (mapset == NULL) {
    char msg[100];
    sprintf (msg, "%s: <%s> cellfile not found\n",
G_program_name(), accum_name);
    G_fatal_error (msg);
    exit(1);
}
/*
* Open the cell files in "mapset".
*/
```

GRASS Raster Commands

```
fd_accum = G_open_cell_old (accum_name, mapset);
if (fd_accum < 0)
exit(1);
fd_chann = G_open_cell_old (chann_name, mapset);
if (fd_chann < 0)
exit(1);
fd_aspect = G_open_cell_old (aspect_name, mapset);
if (fd_aspect < 0)
exit(1);
/*
* Open up a vector that is just long enough to hold one
* row of data.
*/
cell = G_allocate_cell_buf();
/*
* Determine the number of rows and columns.
*/
nrows = G_window_rows();
ncols = G_window_cols();
printf ("\n", nrows);
printf ("nrows: %d\n", nrows);
printf ("ncols: %d\n", ncols);
/*
* Allocate memory for matrices.
*/
accum = imatrix(0,nrows,0,ncols);
```

GRASS Raster Commands

```
chann = imatrix(0,nrows,0,ncols);
aspect = imatrix(0,nrows,0,ncols);
/*
 * Process DEM and Channel files.
 */
for (row=(nrows-1); row>=0; row--) {
if(G_get_map_row (fd_accum, cell, row) < 0)
exit(1);
for (col = 0; col < ncols; col++) {
accum[row][col] = (int)cell[col];
}
if(G_get_map_row (fd_chann, cell, row) < 0)
exit(1);
for (col = 0; col < ncols; col++) {
chann[row][col] = (int)cell[col];
}
if(G_get_map_row (fd_aspect, cell, row) < 0)
exit(1);
for (col = 0; col < ncols; col++) {
aspect[row][col] = (int)cell[col];
}
}
/*
 * Compute the topology of the network.
 */
(void)find_tribs(nrows, ncols, accum, chann, aspect);
```

GRASS Raster Commands

```
exit(0);
```

```
}
```

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.lags – GRASS module to calculate various spatial dependence measures for all possible lags within a given image. Each measure is made by comparing two cells only for each lag azimuth and distance. The whole image is read once for each cell in the image, therefore is very computationally expensive. A Sparc 2 would typically take 1 hour to calculate a 150x150 image.

Usage:

```
r.lags [-sc] in=name out=name [measure=name]
```

Flags:

- n Calculate non-zero values only
- v Calculate semi-variogram instead

Parameters:

in=name Raster surface layer to measure

out=name Output raster containing lag map

measure=name Spatial dependence measure (Moran's I or texture)

options: moran,texture

default: moran

HISTORY

V1.0 written 17.6.92 to calculate Moran autocorrelation statistic.

V2.0 written 18.7.92 to also calculate Haralick's grey-tone spatial dependence textural measures.

V2.1 modified 18.9.95 to conform to standard GRASS program structure.

Future improvements could include (i) Option of faster, memory intensive option; (ii) Further lag dependent measures; (iii) Cumulative lags under some distance weighting function. (iv) Option of ignoring/including zero values in the calculation.

Jo Wood, November, 1995

jwo@le.ac.uk
[ASSIST's home](#)

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.le.dist – The *r.le.dist* program can be used to measure distances between patches and report those distances using several methods.

(GRASS Raster Program)

GRASS VERSION

4.x

SYNOPSIS

r.le.dist

r.le.dist help

r.le.dist [-bntu] **map**=name [**sam**=name] [**reg**=name] [**ski**=value] [**can**=value] [**di1**=name[,name,...]]
 [**di2**=name[,name,...]] [**out**=name]

OPTIONS

Flags:

- b** Run in background
- n** Output map 'num' with patch numbers.
- t** Use 4 neighbor tracing instead of 8 neighbor.
- u** Output maps 'units_x' with sampling units for each scale x.

Parameters:

map

Raster map to be analyzed.

sam

Sampling method (choose only 1 method): w=whole map, u=units, m=moving window, r=regions
 Options: w, u, m, r
 Default: w

reg

Name of regions map, only when sam = r; omit otherwise.

ski

Skip m boundary cells to speed up nearest neighbor search.

Options: 0–10

Default: 0

can

Use only 'can' candidate patches for faster nearest neighbor search.

Options: 1–30

Default: 30

di1

Distance methods (Choose only 1 method):

Options: m0, m1, m2, m3, m4, m5, m6, m7, m8, m9

(CC=Center–Center, EE=Edge–Edge, CE=Center–Edge):

m0 = each patch to all adjacent neighbors CC

m1 = each patch to all adjacent neighbors CE

m2 = each patch to nearest patch of same gp CC

m3 = each patch to nearest patch of same gp CE

m4 = each patch to nearest patch of same gp EE

m5 = each patch to nearest patch of any diff. gp CC

m6 = each patch to nearest patch of any diff. gp CE

m7 = patches of 1 gp to nearest of specific gp CC

m8 = patches of 1 gp to nearest of specific gp CE

m9 = patches of 1 gp to nearest of specific gp EE

di2

Distance measures:

Options: n1, n2, n3, n4, n5, n6

n1 = mean dist.

n2 = st. dev. dist.

n3 = mean dist. by gp

n4 = st. dev. dist. by gp

n5 = no. of dist. by dist. class

n6 = no. of dist. by dist. class by gp

out

Name of output file for individual patch measures, when sam=w, u, r; if out=head, then column headings will be printed

SEE ALSO

The [r.le Programs](#) Users Guide

[r.le.null](#), [r.le.patch](#), [r.le.pixel](#), [r.le.rename](#), [r.le.setup](#), [r.le.trace](#)

AUTHOR

William L. Baker, Department of Geography and Recreation, University of Wyoming

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.le.null – The *r.le.null* program is designed to generate a neutral structure map layer. This map layer can be used as a "null hypothesis" layer for testing the statistical significance of landscape measures.
(*GRASS Raster Program*)

GRASS VERSION

4.x

SYNOPSIS

```
r.le.null
r.le.null help
r.le.null map=name num=value att=value[,value,...] pro=value[,value,...]
```

Parameters:

map
Raster map, with neutral structure, to be created.

num
Number of attributes desired in the map (max = 24).

att
Attribute i (an integer) of 'num' attributes.

pro
Probability (as a % between 0–100) for attribute i of 'num' attributes.

SEE ALSO

The [r.le Programs](#) Users Guide
[r.le.dist](#), [r.le.patch](#), [r.le.pixel](#), [r.le.rename](#), [r.le.setup](#), [r.le.trace](#)

AUTHOR

William L. Baker, Department of Geography and Recreation, University of Wyoming

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.le.patch – This program can be used to calculate attribute, patch size, core (interior) size, shape, fractal dimension, and perimeter measures for sets of patches in a landscape
(*GRASS Raster Program*)

SYNOPSIS

r.le.patch

r.le.patch help

r.le.patch [-bcnptu] **map**=name [sam=name] [reg=name] [att=name[,name,...]] [siz=name[,name,...]] [co1=value] [co2=name[,name,...]] [sh1=name] [sh2=name[,name,...]] [fra=name] [per=name[,name,...]] [out=name]

OPTIONS

Flags:

- b** Run in background.
- c** Output map 'interior' with patch cores.
- n** Output map 'num' with patch numbers.
- p** Include sampling area boundary as perimeter.
- t** Use 4 neighbor tracing instead of 8 neighbor.
- u** Output maps 'units_x' with sampling units for each scale x.

Parameters:

- map** Raster map to be analyzed.
- sam** Sampling method (choose only 1 method): w=whole map, u=units, m=moving window, r=regions
Default: w
- reg** Name of regions map, only when sam = r; omit otherwise.
- att** Attribute measures:

Options: a1, a2, a3, a4, a5, a6, a7

a1 = mean pixel attribute
a2 = st. dev. pixel attribute
a3 = mean patch attribute
a4 = st. dev. patch attribute
a5 = cover by gp
a6 = density by gp
a7 = total density

siz

Patch size measures:

Options: s1, s2, s3, s4, s5, s6
s1 = mean patch size
s2 = st. dev. patch size
s3 = mean patch size by gp
s4 = st. dev. patch size by gp
s5 = no. by size class
s6 = no. by size class by gp

co1

Edge width in pixels (integer) for use with co2

co2

Core size measures (required if co1 was specified):

Options: c1, c2, c3, c4, c5, c6, c7, c8, c9, c10
c1 = mean core size
c2 = st. dev. core size
c3 = mean edge size
c4 = st. dev. edge size
c5 = mean core size by gp
c6 = st. dev. core size by gp
c7 = mean edge size by gp
c8 = st. dev. edge size by gp
c9 = no. by size class
c10 = no. by size class by gp

sh1

Shape method (choose only 1 method):

m1 = perim./area
m2 = corr. perim./area
m3 = rel. circum. circle

sh2

Shape measures (required if sh1 was specified):

Options: h1, h2, h3, h4, h5, h6
h1 = mean patch shape
h2 = st. dev. patch shape
h3 = mean patch shape by gp
h4 = st. dev. patch shape by gp
h5 = no. by shape class
h6 = no. by shape class by gp

fra

Fractal dimension measures:

Options: f1
f1 = perim.-area fractal dim.

per

Perimeter measures (required if pe1 was specified):

Options: p1, p2, p3, p4, p5, p6

p1 = sum of perims.

p2 = mean perim.

p3 = st. dev. perim.

p4 = sum of perims. by gp

p5 = mean perim. by gp

p6 = st. dev. perim. by gp

out

Name of output file for individual patch measures, when sam=w, u, r; if out=head, then column headings will be printed

SEE ALSO

The [r.le Programs](#) Users Guide

[r.le.dist](#), [r.le.null](#), [r.le.pixel](#), [r.le.rename](#), [r.le.setup](#), [r.le.trace](#)

AUTHOR

William L. Baker, Department of Geography and Recreation, University of Wyoming

Last changed: \$Date: 2002/04/22 17:36:27 \$



NAME

r.le.pixel – The *r.le.pixel* program contains a set of measures for attributes, diversity, texture, juxtaposition, and edge.

(GRASS Raster Program)

SYNOPSIS

r.le.pixel

r.le.pixel.help

r.le.pixel [-beuz] **map**=name [**sam**=name] [**reg**=name] [**att**=name[,name,...]] [**div**=name[,name,...]]
 [**te1**=name] [**te2**=name[,name,...]] [**jux**=name[,name,...]] [**edg**=name[,name,...]]

OPTIONS

Flags:

- b Run in background.
- e Output map 'edge' of edges given a '1' in r.le.para/edge file.
- u Output maps 'units_x' with sampling units for each scale x.
- z Output map 'zscores' with standardized scores.

Parameters:

map

Raster map to be analyzed.

sam

Sampling method (choose only 1 method): w=whole map, u=units, m=moving window, r=regions
 Default: w

reg

Name of regions map, only when sam = r; omit otherwise.

att

Attribute measures:

Options: b1, b2, b3, b4

b1 = mean pixel attribute

b2 = st. dev. pixel attribute

b3 = minimum pixel attribute

b4 = maximum pixel attribute

div

Diversity measures:
 Options: d1, d2, d3, d4
 d1 = richness
 d2 = Shannon
 d3 = dominance
 d4 = inverse Simpson

te1

Texture method (choose only 1 method):
 Options: m1, m2, m3, m4, m5, m6, m7
 m1 = 2N-H
 m2 = 2N-45
 m3 = 2N-V
 m4 = 2N-135
 m5 = 4N-HV
 m6 = 4N-DIAG
 m7 = 8N

te2

Texture measures (required if te1 was specified):
 Options: t1, t2, t3, t4, t5
 t1 = contagion
 t2 = ang. sec. mom.
 t3 = inv. diff. mom.
 t4 = entropy
 t5 = contrast

jux

Juxtaposition measures (weight file in r.le.para needed):
 Options: j1, j2
 j1 = mean juxtaposition
 j2 = standard deviation of juxtaposition

edg

Edge measures:
 Options: e1, e2
 e1 = sum of edges
 e2 = sum of edges by type (edge file in r.le.para needed)

SEE ALSO

The [r.le Programs](#) Users Guide
[r.le.dist](#), [r.le.null](#), [r.le.patch](#), [r.le.rename](#), [r.le.setup](#), [r.le.trace](#)

AUTHOR

William L. Baker, Department of Geography and Recreation, University of Wyoming

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.le.rename – The *r.le.rename* program is used to either add a suffix to all the files in the *r.le.out* subdirectory, or to rename the files one by one.
(GRASS Raster Program)

GRASS VERSION

4.x

SYNOPSIS

r.le.rename

r.le.rename help

r.le.rename [-a] [ext=name] [old=name[,name,...]] [new=name[,name,...]]

DESCRIPTION

It is necessary to rename the files before running an *r.le* program over again, because the *r.le* program will overwrite existing files each time the program is started.

OPTIONS

Flag:

-a

All files in *r.le.out* that have extension *.out* change their extension to parameter *ext*; others not affected.

Parameters:

ext

New extension with which to replace the *.out* extension of files in *r.le* out directory.

old

Old file name *i* in *r.le.out* directory to be changed.

new

New file name for old file name *i* in *r.le.out* directory.

SEE ALSO

The [r.le Programs](#) Users Guide

[r.le.dist](#), [r.le.null](#), [r.le.patch](#), [r.le.pixel](#), [r.le.setup](#), [r.le.trace](#)

AUTHOR

William L. Baker, Department of Geography and Recreation, University of Wyoming

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.le.setup – The *r.le.setup* program is used to set up the sampling and analysis framework that will be used by the other *r.le* programs.

(GRASS Raster Program)

SYNOPSIS

r.le.setup

DESCRIPTION

The first menu allows the user to define a rectangular sampling frame, select how sampling will be done (regions, sampling units, moving window), setup the limits for groups and classes, and change the color table. Use the left mouse button to make your choice.

Information about the structure of the landscape is obtained by overlaying a set of sampling areas on top of a specified part (the sampling frame of a map layer, and then calculating specific structural measures for the part of the map layer that corresponds to the area in each sampling area.

To setup a *sampling frame* click on SAMPLING FRAME in the main menu. The program will ask "Will the sampling frame (total area within which sampling units are distributed) be the whole map? (y/n) [y]" Just hit a carriage return to accept the default, which is to use the whole map. You do not need to setup a sampling frame if you want to use the whole map, as this is the default. To setup a different sampling frame type "n" in response to this question. Then use the mouse and a rubber band box to outline a rectangular sampling frame on screen. This box will be moved to the nearest row and column of the map. You will be asked last whether you want to "Refresh the screen before choosing more setup?" If you don't like the sampling frame you just setup, answer yes to this question, then click on SAMPLING FRAME again to redo this part of the setup. This sampling frame will be used in all subsequent setup procedures unless you change it. You can change it at any time by simply clicking on SAMPLING FRAME again.

A *sampling area* may be one of four things. First, it is possible to treat the entire map layer as the one (and only) sampling area. Second, if the map layer can be divided into meaningful geographical regions, then it is possible to treat the regions themselves as sampling areas. The third option is that the sampling areas may be sampling units of fixed shape and size (also called scale) that are placed within the map layer as a whole. The fourth and final option is that the sampling area may be moved systematically across the map as a moving window.

If regions are to be used as the sampling areas, then the user can use *r.le.setup* to draw regions or any existing map of regions can simply be used directly. To draw regions and create a new regions map in *r.le.setup* select "REGIONS" from the first *r.le.setup* menu, and the user is asked to do the following:

1. "ENTER THE NEW REGION MAP NAME:". Only a new raster map name is acceptable. The user

GRASS Raster Commands

can type LIST to find out the existing raster map names in this location and mapset.

2. "PLEASE OUTLINE REGION # 1". The user should move the mouse cursor into the graphic monitor window and use the mouse buttons as instructed:
Left button: where am I. to display the current coordinates of the cursor.
Middle button: Mark start (next) point. to enter a vertex of the region boundary.
Right button: Finish region-connect to 1st point to close the region boundary by setting the last vertex to be equal to the first one.
3. A "REGION OPTIONS:" menu is displayed and the user should use the mouse to select one of the options:

"DRAW MORE": repeat the above process and setup another region.
"START OVER": abandon the previous setup and start all over again.
"DONE-SAVE": save the regions outlined so far and exit this procedure.
"QUIT-NO SAVE": quit the procedure without saving the regions.

Once the "DONE-SAVE" option is selected, the new raster map of the sampling regions is generated. It is displayed on the monitor window for several seconds, the monitor window is refreshed, the main menu is displayed again, and the program is ready for other setup work. Note that you cannot draw regions in areas outside the mask, if a mask is present (see *r.mask* command).

The user can also use the GRASS *r.digit* or *v.digit* programs to digitize circular or polygonal regions and to create a sampling regions map without using *r.le.setup*. Or, as mention above, an existing raster map can be used as a regions map.

If sampling units are to be used as the sampling areas (Fig. 2), then choose "SAMPLING UNITS" from the first *r.le.setup* menu. The program checks the *r.le.para* subdirectory for an existing "units" file from a previous setup session and allows the user to rename this file (to save it) before proceeding. The *r.le.setup* program will otherwise overwrite the "units" file. Then the following choice is displayed followed by a series of other choices:

```
Which do you want to do?
(1) Use the keyboard to enter sampling unit parameters
(2) Draw the sampling units with the mouse
Enter 1 or 2:
```

When sampling units are defined using the keyboard, the user inputs the shape and size (scale) of the sampling units by specifying dimensions in pixels using the keyboard. When sampling units are drawn with the mouse, the user clicks the mouse to define the sampling units in the GRASS monitor window, and then actually places the sampling units for each scale onto the map. By placing the units with the mouse the user can directly determine the method of sampling unit distribution as well as the shape, size, and number of sampling units.

If the choice is made to define sampling units using the keyboard, the following series of questions must be answered:

```
How many different SCALES do you want (1-15)?
```

The user is asked to specify the number of scales that will be used. The *r.le* programs allow the user to simultaneously sample the same map with the same measures using sampling areas of different sizes. Currently there can be between 1 and 15 scales that can be sampled simultaneously. Substantial output can be produced if many scales are used.

Sampling units must be placed spatially into the landscape. There are five options for doing this :

Random nonoverlapping

Sampling units are placed in the landscape by randomly choosing numbers that specify the location of the upper left corner of each sampling unit, subject to the constraint that successive sampling units not overlap other sampling units or the edge of the landscape, and that they must be entirely within the area defined by the mask (see *r.mask* command) if one exists.

Systematic contiguous

Sampling units are placed side by side across the rows. The user will be able to enter a row and column to indicate where the upper left corner of the systematic contiguous framework should be placed. Rows are numbered from the top down beginning with row 1 of the sampling frame. Columns are numbered from left to right, beginning with column 1 of the sampling frame. A random starting location can be obtained by using a standard random number table to choose the starting row and column. The *r.le.setup* program does not avoid placing the set of sampling units over areas outside the mask. The user will have to make sure that sampling units do not extend outside the mask by choosing a particular starting row and column or by drawing a sampling frame before placing the set of sampling units.

Systematic noncontiguous

The user must specify the starting row and column as in #2 above and the amount of spacing (in pixels) between sampling units. Horizontal and vertical spacing are identical. Sampling units are again placed side by side (but spaced) across the rows. As in #2 the program does not avoid placing sampling units outside the masked area; the user will have to position the set of units to avoid areas outside the mask.

Stratified random

The strata are rectangular areas within which single sampling units are randomly located. The user must first specify the starting row and column as in #2 above. Then the user must specify the number of strata in the horizontal and vertical directions. As in #2 the program does not avoid placing sampling units outside the masked area; the user will have to position the set of units to avoid areas outside the mask.

Centered over sites

The user must specify the name of a sitefile containing point locations. A single sampling unit is placed with its center over each site in the site file. This is a useful approach for determining the landscape structure around points, such as around the location of wildlife observations.

The user is prompted to enter a ratio that defines the shape of the sampling units. Sampling units may have any rectangular shape, including square as a special case of rectangular. Rectangular shapes are specified by entering the ratio of columns/rows (horizontal dimension/vertical dimension) as a real number. For example, to obtain a sampling unit 10 columns wide by 4 rows long specify the ratio as 2.5 (10/4).

Recommended maximum SIZE is m in x cell total area.

What size (in cells) for each sampling unit of scale n?

The user is then given the recommended maximum possible size for a sampling unit (in pixels) and asked to input the size of sampling units at each scale. Sampling units can be of any size, but the maximum size is the size of the landscape as a whole. All the sampling units, that make up a single sampling scale, are the same size. After specifying the size, the program determines the nearest actual number of rows and columns, and hence size, that is closest to the requested size, given the shape requested earlier.

The nearest size is x cells wide X y cells high = xy cells

Is this size OK? (y/n) [y]

Maximum NUMBER of units in scale n is p?

What NUMBER of sampling units do you want to try to use?

GRASS Raster Commands

The maximum number of units that can be placed over the map, given the shape and size of the units, is then given. The user can then choose the number of sampling units to be used in the map layer. It may not always be possible to choose the maximum number, depending upon the shape of the sampling units. In the case of systematic contiguous and noncontiguous, the program will indicate how many units will fit across the columns and down the rows. The user can then specify a particular layout (e.g., 6 units could be placed as 2 rows of 3 per row or as 3 rows of 2 per row).

```
Is this set of sampling units OK? (y/n) [y]
```

Finally, the set of sampling units is displayed on the screen (e.g., Fig. 1) and the user is asked whether it is acceptable. If the answer is no, then the user is asked if the screen should be refreshed before redisplaying the menu for "Methods of sampling unit distribution" so that the user can try the sampling unit setup again.

The choice is made to define sampling units using the mouse, then the following menu for use with the mouse is displayed:

```
Outline the standard sampling unit of scale n.  
Left button: Check unit size  
Middle button: Move cursor  
Right button: Lower right corner of unit here
```

The user can then use the mouse and the rubber band box to outline the standard sampling unit. Once it has been outlined, the number of columns and rows in the unit, the ratio of width/length and the size of the unit, in cells, will be displayed. After this first unit is outlined, then a new menu is displayed:

```
Outline more sampling units of scale n?  
Left button: Exit  
Middle button: Not used  
Right button: Lower right corner of next unit here
```

The user can then place more units identical to the standard unit by simply clicking the right mouse button where the lower right corner of the unit should be placed. The rest of the rubber band box can be ignored while placing additional units. The program is set up so that units cannot be placed so they overlap one another, so they overlap the area outside the mask, or so they overlap the edge of the sampling frame. Warning messages are issued for all three of these errors and a sampling unit is simply not placed.

Using this procedure a rectangular "window" or single sampling area is moved systematically across the map to produce a new map (Fig. 2,3). This sampling procedure can only be used with the measures that produce a single value or with a single class or group when measures produce distributions of values (Table 1). The first class or group specified when defining class or group limits (section 2.3.2.) is used if distributional measures are chosen with the moving window sampling method. In this case, the user should manually edit the *r.le.para/recl_tb* file so that the desired group is listed as the first group in this file.

Sampling begins with the upper left corner of the window placed over the upper left corner of the sampling frame. It is strongly recommended that the user read the section on the GRASS mask (section 2.2.2) prior to setting up the moving window, as this mask can be used to speed up the moving window operation. The value of the chosen measure is calculated for the window area. This value is assigned to the location on the new map layer corresponding to the center pixel in the window if the window has odd (e.g. 3 X 3) dimensions. The value is assigned to the location on the new map layer corresponding to the first pixel below and to the right of the center if the window has even dimensions (e.g. 6 X 10). If this pixel has the value "0," which means "no data" in GRASS, then this pixel is skipped and a value of "0" is assigned to the corresponding location in the new map. The window is then moved to the right (across the row) by one pixel, and the process is repeated. At the end of the row, the window is moved down one pixel, and then back across the row. This option produces a new map layer, whose dimensions are smaller by approximately $(m-1)/2$ rows and columns, where m is the number of rows or columns in the window.

If the "MOVE-WINDOW" option in the main menu is selected, first the program checks for an existing "move_wind" file, in the *r.le.para* subdirectory, containing moving window specifications from a previous session. The user is given the option to avoid overwriting this file by entering a new file name for the old "move_wind" file. Users should be aware that moving window analyses are very slow, because a large number of sampling units are, in effect, used. See the appendix on "Time needed to complete analyses with the *r.le* programs" for some ideas about how moving window size and sampling frame area affect the needed time to complete the analyses.

The *r.le* programs *r.le.dist* and *r.le.patch* allow the attribute categories in the input map to be reclassified into several attribute groups, and reports the analysis results by each of these attribute groups. It is necessary to setup group limits for all measures that say "by gp" when typing "*r.le.dist help*" or "*r.le.patch help*" at the GRASS prompt. The same reclassification can be done with the measurement indices (e.g., size), except that each "cohort" (class) of the reclassified indices is called an index class instead of a group. It is also necessary to setup class limits for all measures that say "by class" when typing "*r.le.dist help*" or "*r.le.patch help*" at the GRASS prompt.

Group/class limits are setup by choosing "GROUP/CLASS LIMITS" from the main menu upon starting *r.le.setup*, or you can create the files manually using a text editor. The program checks for existing group/class limit files in subdirectory *r.le.para* and allows the user to rename these files prior to continuing. If the files are not renamed the program will overwrite them. The files are named *recl_tb* (attribute group limits), *size* (size class limits), *shape_PA* (shape index class limits for perimeter/area index), *shape_CPA* (shape index class limits for corrected perimeter/area index), *shape_RCC* (shape index class limits for related circumscribing circle index), and *from_to* (for the *r.le.dist* program distance methods m7–m9).

Attribute groups and index classes are defined in a different way. In the *r.le* programs attribute groups are defined as in the following example:

```
1, 3, thru, 20 = 1 (comment)
thru 30 = 2 (comment)
end
```

In this example, the existing categories 1, 3, 5, 7, {9, 10, ... 20, 21} are included in the new group 1, while {31, 32, 33, ..., 49, 50} are included in the new group 2. The characters in bold are the "key words" that are required in the definition. Each line is called one "reclass rule".

The GRASS reclass convention is adopted here with a little modification (see "*r.reclass*" command in the GRASS User's Manual). The difference is that *r.le* only allows one rule for each group while the GRASS *r.reclass* command allows more than one. The definition of "from" and "to" groups is simply the extension of the GRASS reclass rule. The advantage of using the GRASS reclass convention is that the user can generate a permanent reclassified map, using GRASS programs, directly from the *r.le* setup results.

The *r.le* measurement index classes are defined by the lower limits of the classes, as in the following example:

```
0.0, 10.0, 50.0, -999.0,
```

This means:

```
if v >= 0.0 and v < 10.0 then v belongs to index class 1;
if v >= 10.0 and v < 50.0 then v belongs to index class 2;
if v >= 50.0 and v < 200.0 then v belongs to index class 3;
if v >= 200.0 then v belongs to index class 4;
```

where *v* is the calculated index value and **-999** marks the end of the index class definition. The measurement index can be the size index, one of the three shape indices, or one of the three distance indices. The program is currently designed to allow no more than 25 attribute groups, 25 size classes, 25 shape index classes, and 25

distance index classes. As an alternative, the user may want to permanently group certain attributes prior to entering the *r.le* programs. For example, the user may want to group attributes 1–10, in a map whose attributes are ages, into a single attribute representing young patches. The user can do this using the GRASS *r.reclass* and *r.resample* commands, which will create a new map layer that can then be analyzed directly (without setting up group limits) with the *r.le* programs.

SEE ALSO

The [r.le Programs](#) Users Guide

[r.digit](#), [r.le.dist](#), [r.le.null](#), [r.le.patch](#), [r.le.pixel](#), [r.le.rename](#), [r.le.trace](#), [r.reclass](#), [r.resample](#), [v.digit](#)

AUTHOR

William L. Baker, Department of Geography and Recreation, University of Wyoming

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.le.trace – the *r.le.trace* program can be used to display the boundary of each patch and show how the boundary is traced, display the attribute, size, perimeter, and shape indices for each patch, and save the data in an output file.

(GRASS Raster Program)

GRASS VERSION

4.x

SYNOPSIS

r.le.trace

r.le.trace help

r.le.trace [-npt] map=name [out=name]

DESCRIPTION

When sampling the whole map (sam=W), the *r.le.trace* program can be used to do three things: (1) display the boundary of each patch and show how the boundary is traced, (2) display the attribute, size, perimeter, and shape indices for each patch, and (3) save these data in an output file.

Flags:

- n** Output map 'num' with patch numbers.
- p** Include sampling area boundary as perimeter.
- t** Use 4 neighbor tracing instead of 8 neighbor.

Parameters:

- map** Raster map to be analyzed.
- out** Name of output file to store patch data.

SEE ALSO

The [r.le Programs](#) Users Guide

[r.le.dist](#), [r.le.null](#), [r.le.patch](#), [r.le.pixel](#), [r.le.rename](#), [r.le.setup](#)

AUTHOR

William L. Baker, Department of Geography and Recreation, University of Wyoming

Last changed: \$Date: 2002/01/25 05:45:34 \$



ATTENTION:

This page is not yet properly corrected! It was taken from *r.rational.regression*. So be careful and send a better version to neteler@itc.it

NAME

r.linear.regression – linear regression calculation from data stored in ASCII file
(GRASS Image Processing Program)

SYNOPSIS

r.linear.regression
r.linear.regression help
r.linear.regression input=*name* output=*name*

DESCRIPTION

The *r.linear.regression* program calculates the linear regression model. If it is used as an image processing tool, the multispectral space remote sensing data will be the regression variables (ASCII file) and the ground vegetation coverage measurements will be the response variables (also ASCII file) and this command will be useful for obtaining linear regression models from the remote-sensing data which have corresponding ground measurement and for predicting vegetation coverage using other remote-sensing data which have no corresponding ground truth records. The input file has the following format

regression variables x1, x2, ... response variable y
 channel 1 (x1) channel 2 (x2) ... coverage

For a three channel remote-sensing data the following is an example of input ASCII file

```
0.4350 0.2616 0.7016 0.98
0.4140 0.2620 0.6520 0.99
0.4940 0.3500 0.5580 0.34
0.5983 0.5350 0.5650 0.10
0.4883 0.3733 0.5533 0.88
0.4150 0.2916 0.5116 0.60
0.5566 0.5250 0.5466 0.09
0.4420 0.2820 0.6800 0.86
0.4220 0.2620 0.6260 0.88
0.4766 0.3666 0.5933 0.61
0.5180 0.4300 0.5140 0.60
0.4416 0.2700 0.7383 0.96
```

```
0.4583 0.3116 0.5133 0.76
0.4300 0.2750 0.7233 0.98
0.4320 0.2760 0.6460 1.00
0.4733 0.3566 0.5616 0.53
0.4200 0.2450 0.7966 1.00
0.4850 0.3533 0.7216 0.99
0.4360 0.2620 0.7620 0.99
0.4283 0.2650 0.6783 0.91
0.4633 0.3200 0.6750 0.94
```

The resulted regression model (coefficient numbers) and related information about the confidential test, goodness or utility test (e.g., correlation coefficient r between observed and calculated coverage, F value and t value) are put on the output file (ASCII file also).

The formular of linear regression can be:

$$\begin{aligned}y &= a[0]x[0] + a[1], \\y &= a[0]x[0] + a[1]x[1] + a[2] \\y &= a[0]x[0] + a[1]x[1] + a[2]x[2] + a[4]\end{aligned}$$

r.linear.regression will be run non- interactively if the user specifies program arguments on the command line, using the form:

r.linear.regression input=name output=name

But after run, the computer will prompt the user to select model number. Alternately, the user can simply type: *r.linear.regression* 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*.

SEE ALSO

[*i.ndvi, r.rational.regression*](#)

AUTHORS

Hong C. Zhuang, U.S. Army Construction Engineering Research Laboratory Department of Electrical Computer Engineering, University of Illinois at Urbana-Champaign.

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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.

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*](#)

[*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 (viewshed).

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

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

Alternately, the user can type simply **r.los** on the command line; in this case, the program will prompt the user for parameter values using the standard GRASS [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: 1000

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](#)

[parser](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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
^	exponentiation	Arithmetic	5
%	modulus	Arithmetic	4
/	division	Arithmetic	4

GRASS Raster Commands

*	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
#	color separator operator	Arithmetic	

(modulus is the remainder upon division)

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:

$$0.10 * \text{red} + 0.81 * \text{green} + 0.01 * \text{blue}$$

Alternatively, you can use:

```
result = y#map
```

to use the NTSC weightings:

$$0.30 * \text{red} + 0.59 * \text{green} + 0.11 * \text{blue}$$

Or, you can use:

```
result = i#map
```

to use equal weightings:

$$0.33 * \text{red} + 0.33 * \text{green} + 0.33 * \text{blue}$$

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
atan(x,y)	inverse tangent of y/x (result is in degrees)	F
cos(x)	cosine of x (x is in degrees)	F
double(x)	convert x to double-precision floating point	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 single-precision 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
isnull(x)	check if x = NULL	
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	*
mode(x,y[,z...])	mode value of those listed	*
not(x)	1 if x is zero, 0 otherwise	
rand(a,b)	random value between a and b	
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

Internal variables:

```
row()      current row of moving window
col()      current col of moving window
x()        current x-coordinate of moving window
y()        current y-coordinate of moving window
ewres()    current east-west resolution
nsres()    current north-south resolution
null()     NULL value
```

Note, that the row() and col() indexing starts with 1.

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.

Note: If you calculate with integer numbers, the resulting map will be integer. If you want to get a float result, add the decimal point to integer number(s).

NULL support

9. Division by zero should result in NULL.
10. Modulus by zero should result in NULL.
11. NULL-values in any arithmetic or logical operation should result in NULL.
12. NULL-values in function arguments should result in NULL (however, if(), eval() and isnull() are treated specially, as described below).
13. The eval() function always returns its last argument
14. The situation for if() is:

```
if(x)
  NULL if x is NULL; 0 if x is zero; 1 otherwise
if(x,a)
  NULL if x is NULL; a if x is non-zero; 0 otherwise
if(x,a,b)
  NULL if x is NULL; a if x is non-zero; b otherwise
if(x,n,z,p)
  NULL if x is NULL; n if x is negative;
  z if x is zero; p if x is positive
```

15. The (new) function isnull(x) returns: 1 if x is NULL; 0 otherwise. The (new) function null() (which has no arguments) returns an integer NULL.
16. Non-NULL, but invalid, arguments to functions should result in NULL.

```
Examples:
log(-2)
sqrt(-2)
pow(a,b) where a is negative and b is not an integer
```

NULL support: Please note that any math performed with NULL cells always results in a NULL value for these cells. If you want to replace a NULL cell on-the-fly, use the isnull() test function in a if-statement.

Example: The users wants the NULL-valued cells to be treated like zeros. To add maps A and B (where B contains NULLs) to get a map C the user can use a construction like:

```
C=A + if(isnull(B),0,B)
```

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)
```

To change all map values below 5 to NULL:

```
newmap = if(map<5, null(), 5)
```

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 `\e` 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. This is important for the `eval()` function.

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.

r.mapcalc does not print a warning in case of operations on NULL cells. It is left to the user to utilize the `isnull()` function.

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).

[Performing Map Calculations on GRASS Data: r.mapcalc Program Tutorial](#), by Marji Larson, Michael Shapiro and Scott Tweddale, U.S. Army Construction Engineering Research Laboratory (December 1991)

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.fillnulls](#), [r.infer](#), [r.mask](#), [r.weight](#)

AUTHOR

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

Last changed: \$Date: 2003/02/09 12:07:23 \$



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.

GRASS Raster Commands

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

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

SYNOPSIS

r.mask.points

r.mask.points help

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

DESCRIPTION

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

```

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

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

OPTIONS

Flags

-r

Coordinates are reversed: north east

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

Parameters

mask=name

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=name

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=character

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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.

OPTIONS

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

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

<u><i>g.region</i></u>	<u><i>r.average</i></u>
<u><i>r.cats</i></u>	<u><i>r.clump</i></u>
<u><i>r.describe</i></u>	<u><i>r.mapcalc</i></u>
<u><i>r.mask</i></u>	<u><i>r.mfilter</i></u>
<u><i>r.mode</i></u>	<u><i>r.neighbors</i></u>
<u><i>r.reclass</i></u>	<u><i>r.stats</i></u>
<u><i>parser</i></u>	

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 it 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 ($n \times n$) 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 $n \times n$ 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*](#)
[*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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*](#)

[*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.moran – GRASS module that calculates Moran's I for a raster surface and optionally creates a surface of local moran values.

Usage:

r.moran [-zb] in=name [out=name] [size=value]

Flags:

-z Exclude zeros in calculation

-b Brief output only

Parameters:

in Raster surface layer to measure

out Output raster layer containing local Moran measures

size Size of processing window (odd number only)
default: 3

HISTORY

Jo Wood 21st October, 1994

Modified to give brief version of output with increased significant figures. Also reports on average local variance - roughness measure.

Jo Wood, 3rd April, 1995

jwo@le.ac.uk
[ASSIST's home](#)

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.neighbors – Makes each cell value a function of the 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 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 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 value equal to the average of the 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 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 value a center cell in a neighborhood will have after examining 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 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

GRASS Raster Commands

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

mode

The most frequently occurring value in the neighborhood.

minimum

The minimum value within the neighborhood.

maximum

The maximum 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 values within the neighborhood (rounded to the nearest integer).

sum

The sum of values within the neighborhood.

variance

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

diversity

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

interspersion

The percentage of cells containing values which differ from the values 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). ERROR .RE

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 values, no *color* files are copied for these map layers. (The user should note that although the *color* file is copied for *averaged* neighborhood function output, whether or not the color file makes sense for the output will be dependent on the input data values.)

SEE ALSO

[*g.region*](#)

[*r.clump*](#)

[*r.mapcalc*](#)

[*r.mask*](#)

[*r.mfilter*](#)

[*r.support*](#)

AUTHOR

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

Last changed: \$Date: 2003/02/09 12:07:23 \$

NAME

r.nntool – Neural Network classification tool for raster maps.
(GRASS Raster Program)

SYNOPSIS

r.nntool

DESCRIPTION

r.nntool is a fully interactive, menu-driven GRASS routine that uses a neural network to train landuse classifications and build a classified image. *r.nntool* facilitates the use of neural networks and linear classifiers in supervised classification of raster cell files.

Neural networks are composed of simple non-linear computational units called neurons that are linked together and work cooperatively to solve complex mapping problems. Neural network output is validated by the use of a controlled training dataset. For application in the GRASS framework, each input unit (equal to one satellite image pixel) to the neural network is assigned a raster map layer, and training data for the network is collected on a cell by cell basis. Typically, a single map layer is used for selecting training sites, although this requirement can be relaxed so that output units can be assigned to more than one map layer, and the map layer used in selecting training sites need not be used as output to the neural network. Since a maximum likelihood classifier (*i.maxlik*) already exists in GRASS, many of the utilities of *i.maxlik* in selecting and analyzing training data were used in the neural network tool. Among these utilities is the ability to visualize and, if necessary, change histograms from each training site.

The program for the neural network tool is structured in such a way that training classes selected in the neural network tool could also be used in the maximum likelihood classifier. This facilitates the implementation of *i.maxlik* within GRASS for validation of neural network output.

In GRASS, the maximum likelihood classifier assumes a Gaussian distribution for the training data, which is a widely used method for satellite imagery classification. In the use of the tool, the user is asked to enter the name of the output map layer, the number of output classes, and the names of the input map layers.

Using the lump option of the menu, the tool selects the "dominant" category within a specified window and generates a new map layer. The user can reset the resolution to the specified window, or retain the old resolution in which he entered the tool. In existing GRASS routines, when resolution (window) of a region is enlarged, the middle pixel of the window in the lower resolution is selected.

Training areas are selected using the define areas option.

Using the zoom option the user can zoom out parts of the output map in which he wishes to delineate training areas. Training areas can be delineated by clicking on points, drawing circles, or by drawing polygons.

Using the delete function, the users can interactively select and delete polygons, with the number of samples after deletion shown in the window.

Histograms of training sites can be examined and signatures saved so that the user can use *i.maxlik*. Once the user is satisfied with the training sites selected, all input map layers are sampled for their data. At intersections of training areas with input map layers, training data for the neural network are gathered.

Training data are stored as an ASCII file so that the user may examine and change it, if necessary. Input data to the network is obtained cell-wise from all areas of the input maps.

The classes option of the neural network tool lets a user examine the distribution of data when two input map layers are used. For higher input dimensions, it is necessary to link the tool to a more sophisticated program such as *xgobi*. The user may eliminate outliers, and data conflicts by drawing rectangular boxes around data points. If necessary a whitening and diagonalization operation can be done on the data so that better class

separability is achieved. Unlike in using traditional classifiers, careful preprocessing of the training data should be performed since neural networks give equal consideration to all data.

Once the user is satisfied with the class distributions, the configure option is selected. Here the user selects a quick propagation network, or the traditional back propagation. The quick propagation network uses gradient descent to adjust weights and assumes a parabolic shape for global minimum. Iterations of the network are performed by the number of training cycles set by the user. Back propagation uses gradient descent and converges to a root mean square error value set by the user.

In `r.nntool`, performance of the network as training progresses is shown on the left half of the GRASS screen. Once training of the neural network is complete, the user propagates cell values of the input map layers through the network. The new map layer generated by the neural network can then be queried. Upon completion of network training, the user may save the neural network structure such as the number of input, hidden, and output units, and the network weights.

Notes

The training site I/O data is stored in a file called `o_train` (`o_` for "old" training file). This is the default file used for training the neural network. If the random option is used to rearrange the training data, the network training data is stored in a file called `TRAIN`. The `classes` option can only be used right now for visualization of two input vectors. The color scheme isn't all that exciting and future upgrades are working on using a better color scheme to represent the training data in each class. [Users may wish to also ftp `xgobi.tar.gz`, and use that tool to visualize training samples in the `o_train/TRAIN` file]. The Bayes' classifier right now can only be used if you have access to IMSL (otherwise, you'll have to wait until routines are written for diagonalizing a covariant matrix etc in C). Users will have to do a priori determine the attribute value for each class in the output map:

Ex., Say there are 5 classes. Extend the attribute values for the classes from 0 to 100. So that,

```
0 - class 1
25 - class 2
50 - class 3
75 - class 4
100 - class 5.
```

This is a limitation of GRASS since the color intensities of a map are determined by the attribute values. The input values to the network are scaled by the highest attribute of each input. Users may wish to try other schemes such as $\text{sgn}(x)(1 + \ln|x|)$, or transform the data using a squashing function such as $\text{tahn}(x)$. Users will have to look at the source code to do this (see `nntool.c`).

SEE ALSO

[imagery](#)
[i.maxlik](#)
[r.reclass](#)

AUTHORS

Ranjan Muttiyah, TAES, Blackland Research Center
Bruce Byars, GRASS Research Group, Baylor University

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.null – The function of *r.null* is to explicitly create the NULL-value bitmap file.
(GRASS 5 Raster Program)

SYNOPSIS

r.null

r.null *-fincr map=name [setnull=val [-val][,val[-val],...]] [null=value]*

DESCRIPTION

The function of *r.null* is to explicitly create the NULL-value bitmap file. The intended usage is to fix "old" maps that don't have a NULL-value bitmap file (i.e. to indicate if zero is valid value or is to be converted to NULL). The module does not work with reclassified maps.

The design is flexible. Ranges of values can be set to NULL and/or the NULL value can be eliminated and replace with a specified value.

Flags:

- f* Only do the work if the map is floating-point.
- i* Only do the work if the map is integer.
- n* Only do the work if the map doesn't have a NULL-value bitmap file.
- c* Create NULL-value bitmap file which validates all data cells.
- r* Remove NULL-value bitmap file.

Parameters:

map

raster map for which to edit null file

setnull=range[,range...]

The values specified in the ranges are to be set to NULL. A range is either a single value (e.g., 5.3), or a pair of values (e.g., 4.76–34.56). Existing NULL-values are left NULL, unless the null argument is requested.

null=value

Eliminate the NULL value and replace it with value. This argument is applied only to existing NULL

values, and not to the NULLs created by the setnull argument.

NOTES

Note that value is restricted to integer if the map is an integer map.

r.null and reclassified maps:

The problem is, if r.null was run on the reclass raster it would alter the original and any other reclass rasters of the original. Therefore r.null doesn't allow to recode reclassified maps (products of r.reclass). So, the way to recode such a map is: The user makes a raster out of the reclass that isn't a reclass by copying it:

```
r.mapcalc newmap = reclass
```

Then he/she recodes this copy.

SEE ALSO

[r.support](#) and [r.quant](#)

AUTHOR

U.S.Army Construction Engineering Research Laboratory

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.out.arc – Converts a raster map layer into an ESRI ARC–GRID file.
(*GRASS Raster Data Export Program*)

SYNOPSIS

r.out.arc
r.out.arc help
r.out.arc [-h] [-1] input=name output=name [dp=value]

DESCRIPTION

r.out.arc converts a user–specified raster map layer (**input=name**) into an ESRI ARC–GRID ascii file (**output=name**) suitable for export to other computer systems. The *dp=value* option (where *value* is a number of the user's choice) can be used to request that numbers after decimal points are limited. However, to use this, the user should know the maximum number of digits that will occur in the output file. The user can find the maximum number of digits occurring in the output file by running *r.out.arc* without the **dp=value** option.

The GRASS program [r.in.arc](#) can be used to perform the reverse function, converting an ESRI ARC–GRID ascii file in suitable format to GRASS raster file format. The order of cell values in file is from lower left to upper right (reverse to GRASS).

OPTIONS

Flags:

- h** Suppress printing of header information.
- 1** List one entry per line.

Parameters:

- input=name**
Name of an existing raster map layer.
- output**
Name of an output ARC–GID map (use **output=-** for stdout)
- dp=value**
The minimum number of decimals (per cell) to be printed.

r.out.arc 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 *dp*, using the form

```
r.out.arc input=inname output=outname [dp=value]
```

where *inname* is the name of a raster map layer to be converted to ARC-GRID format, *outname* is the name of a raster map layer to be exported, 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.arc** on the command line, without program arguments. In this case, the user will be prompted for parameter values using the standard GRASS [parser](#) interface.

NOTES

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

```
r.out.arc input=soils output=- > out.grd
```

The output file *out.grd* can then be copied onto a CDROM or floppy disk for export purposes.

SEE ALSO

[r.in.arc](#)

[parser](#)

AUTHOR

Markus Neteler, University of Hannover, Germany,
based on r.out.ascii from
Michael Shapiro, U.S.Army Construction Engineering Research Laboratory

Last changed: \$Date: 2002/02/25 08:56:25 \$



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 [-hsMi] input=name output=name [dp=value] [null=string]
```

DESCRIPTION

r.out.ascii converts a user-specified raster map layer (**input=name**) into an ASCII text file (**output=name**) suitable for export to other computer systems.

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.

OPTIONS

Flags:

- h** Suppress printing of header information.
- s** Write SURFER .grd ASCII GRID instead of GRASS ASCII GRID (with reverted row order, different header)
- M** Write MODFLOW (USGS) free-format array instead of GRASS ASCII GRID
- i** Output integer category values, not cell values.

Parameters:

- input=name**
Name of an existing raster map layer.
- output**
Name of an output ASCII file (use output=- for stdout)
- dp=value**
Number of decimal places.
- null=string**

Character string to represent no data cell.

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 *dp*, using the form

```
r.out.ascii input=inname output=outname [dp=value]
```

where *inname* is the name of a raster map layer to be converted to ASCII format, *outname* is the name of the exported file, and *value* is the 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.

To write a SURFER .grd ASCII GRID file (with reverted row order and different header) use the *-s* flag:

```
r.out.ascii -s input=inname output=outname.grd [dp=value]
```

NULL data are coded to "1.70141e+038" for SURFER ASCII GRID files (ignoring the *null=* parameter).

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.

NOTES

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

```
r.out.ascii input=soils output=- > out.file
```

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

SEE ALSO

[r.in.ascii](#), [r.in.arc](#), [parser](#)

AUTHOR

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

Surfer support by Markus Neteler

Last changed: \$Date: 2003/02/05 11:06:51 \$



NAME

r.out.bin – exports a GRASS raster to a binary array
(*GRASS Raster Export Program*)

SYNOPSIS

r.out.bin [-i] [-h] [-b] [-s] **input=***name* **output=***name* [**null=***value*]

DESCRIPTION

The *r.out.bin* program exports a GRASS raster map to a binary array file. Optionally, output can be sent to standard output (stdout) for direct input (pipe) into other applications. Data is exported according to the original GRASS raster type (e.g. float). If the "-i" flag is specified, an integer array is output. The region parameters are printed to stderr.

Flags:

- i** Output integer category values, not raster cell values
- h** Include a Generic Mapping Tools (GMT) binary header on the file
- b** Generate separate BIL world and header description files
- s** Byte swap the data and optional header during output

Parameters:

- input** Name of an existing raster map
- output** Name of an output binary file (use output=- for stdout)
- null** Value to write out for null
Default: 0

NOTES

With the -h flag, data can be directly used by [GMT](#) as Grid Format 1 (float) or 2 (short). For example:

```
r.out.bin -h input=grass.raster output=new.grd
```


GRASS Raster Commands

```
grdinfo new.grd=1 (if float)
```

Exported data can be piped directly into the GMT program xyz2grd.

```
r.out.bin input=grass.raster output=- | xyz2grd -R.... -ZTLf -F
```

The example uses the GMT program xyz2grd with the `-ZTLf` flag indicating that a float array was output.

AUTHOR

This program is derived from [r.out.ascii](#) with a few modifications.

Author: [Bob Covill](#)

SEE ALSO

[r.in.bin](#), [r.in.ascii](#), [r.out.ascii](#), [r.in.arc](#), [r.out.arc](#)

Last changed: \$Date: 2002/03/01 00:08:31 \$



NAME

r.out.elas – Export a GRASS raster map layer to an ELAS raster file
(*GRASS Raster Data Export Program*)

SYNOPSIS

```
r.out.elas
r.out.elas help
r.out.elas [-h] input=name output=name
```

DESCRIPTION

r.out.elas exports a GRASS cell file to an ELAS raster file.

OPTIONS

Flags:

-f
Floating-point ELAS output file will be created.

Parameters:

*input=**name*
GRASS input cell file name.
*output=**name*
ELAS output raster file name.

"*r.out.elas*" will transfer and convert a GRASS cell file to an ELAS raster file. The GRASS cell file will be a 1 to 4 byte integer per element file. This GRASS cell file may be converted to either a 1 to 4 byte integer per element ELAS file (default) or a 4 byte floating-point per element ELAS file. Color values will be transferred from GRASS to ELAS for GRASS 1 byte integer per element files only. "*r.out.elas*" only transfers GRASS cell files that have either an unreferenced XY or UTM projection. Whether the GRASS cell file is either an unreferenced XY or UTM projection, the ELAS file created will be a UTM projection. "*r.out.elas*" will transfer the GRASS cell file based on the selected region and will only transfer the portion of the GRASS cell file that is within that selected region. "*r.out.elas*" will notify the user as to whether the current region has the same values as the "cellhd" file for the GRASS cell file being transferred to ELAS. Since the area transferred to ELAS is determined by the region (WIND file) and not the cellhd file then the notification of a difference between these files will indicate that not all of the GRASS cell file will be transferred to ELAS. Also, this program will notify the user as to whether the raster "MASK" is on or not.

SEE ALSO

[r.out.bin](#), [r.in.ascii](#), [r.out.ascii](#), [r.in.arc](#), [r.out.arc](#), [r.in.gdal](#)

AUTHOR

Bruce Powell, National Park Service

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.out.hdf – Convert data in a GRASS raster map layer into HDF format. (GRASS Raster Data Export Program)

SYNOPSIS

```
r.out.hdf
r.out.hdf help
r.out.hdf [-q] input=name [output=name] [mult=value]
```

DESCRIPTION

r.out.hdf allows a user to create a data file in NCSA Hierarchical Data Format (HDF) from an existing GRASS raster file.

COMMAND LINE OPTIONS

Flag:

-q Run quietly

Parameters:

input Raster file to be converted.

output Name for new hdf file.
Default: <rasterfilename>.hdf

mult Floating point multiplier. (file.hdf = cellfile * multiplier)

NCSA HDF is a multi-object file format developed by The National Center for Supercomputing Applications at Champaign, Illinois for the transfer of graphical and floating-point data between machines. The format defines both a raster type and a SDS type. The later is basically a highly structured multi-dimensional array of floating point values. A single HDF file may contain more than one SDS, but using *r.out.hdf* will create a single HDF file containing a single 2-dimensional SDS. The HDF raster format consists of a palette and a 2D array of 8-bit numbers, but *r.out.hdf* does not yet create this format file, so color tables are not transferrable.

NCSA distributes the HDF library and several public domain visualization applications which use the HDF format. Some commercial applications also support HDF. For more information, use the NCSA anonymous

ftp server ftp.ncsa.uiuc.edu or contact:

NCSA
152 Computing Applications Building
605 E. Springfield Ave.
Champaign, IL 61820
(217) 244-0072

SEE ALSO

[r.in.hdf](#)

AUTHOR

Bill Brown, U.S. Army Construction Engineering Research Laboratory

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.out.mpeg – Raster File Series to MPEG Conversion Program

GRASS VERSION

4.x, 5.x

SYNOPSIS

r.out.mpeg

r.out.mpeg help

r.out.mpeg [-qc]

view1=name[,name,...][view2=name[,name,...]][view3=name[,name,...]][view4=name[,name,...]]
[output=name] [qual=value]

DESCRIPTION

r.out.mpeg is a tool for combining a series of GRASS raster files into a single MPEG-1 (Motion Pictures Expert Group) format file. MPEG-1 is a "lossy" video compression format, so the quality of each resulting frame of the animation will be much diminished from the original raster image. The resulting output file may then be viewed using your favorite mpeg-format viewing program.

The user may define up to four "views", or sub-windows, to animate simultaneously. e.g., View 1 could be rainfall, View 2 flooded areas, View 3 damage to bridges or levees, View 4 other economic damage, all animated as a time series. A black border 2 pixels wide is drawn around each view. There is an arbitrary limit of 100 files per view (100 animation frames). Temporary files are created in the conversion process, so lack of adequate tmp space could also limit the number of frames you are able to convert.

The environment variable GMPEG_SIZE is checked for a value to use as the dimension, in pixels, of the longest dimension of the animation image. If GMPEG_SIZE is not set, the animation size defaults to the rows & columns in the current GRASS region, scaling if necessary to a default minimum size of 200 and maximum of 500. These size defaults are overridden when using the *-c* flag (see below). The resolution of the current GRASS region is maintained, independent of image size. Playback programs have to decode the compressed data "on-the-fly", therefore smaller dimensioned animations will provide higher frame rates and smoother animations.

UNIX – style wild cards may be used with the command line version in place of a raster file name, but it must be quoted.

Example:

```
r.out.mpeg view1="rain[1-9]","rain[0-2]" view2="temp*"
```

If the number of files differs for each view, the view with the fewest files will determine the number of frames in the animation.

OPTIONS

Flags:

-q

Quiet – suppress progress report.

-c

Convert "on the fly", uses less disk space by using *r.out.ppm* with *stdout* option to convert frames as needed instead of converting all frames to ppm before encoding. Only use when encoding a single view. Use of this option also overrides any size defaults, using the CURRENTLY DEFINED GRASS REGION for the output size. So be careful to set region to a reasonable size prior to encoding.

Parameters:

view1

Raster file(s) for View1.

view2

Raster file(s) for View2.

view3

Raster file(s) for View3.

view4

Raster file(s) for View4.

output

Name for MPEG output file.

Default: gmovie.mpg

qual

Quality factor (1–5).

Default: 3

A quality value of *qual=1* will yield higher quality images, but with less compression (larger MPEG file size). Compression ratios will vary depending on the number of frames in the animation, but an MPEG produced using *qual=5* will usually be about 60% the size of the MPEG produced using *qual=1*.

BUGS

MPEG images must be 16–pixel aligned for successful compression, so if the rows & columns of the calculated image size (scaled, with borders added) are not evenly divisible by 16, a few rows/columns will be cut off the bottom & right sides of the image. The MPEG format is optimized to recognize image MOTION, so abrupt changes from one frame to another will cause a "noisy" encoding.

NOTES

This program requires the program *mpeg_encode*:

GRASS Raster Commands

MPEG-1 Video Software Encoder
(Version 1.3; March 14, 1994)

Lawrence A. Rowe, Kevin Gong, Ketan Patel, and Dan Wallach Computer Science Division-EECS,

Univ. of Calif. at Berkeley

Available by anonymous ftp from: s2k-ftp.CS.Berkeley.EDU

Use of the `-c` flag also requires the program `r.out.ppm` with the `stdout` option.

AUTHOR

Bill Brown, U.S. Army Construction Engineering Research Laboratories

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

`r.out.pov` – Converts a raster map layer into a height–field file for POVray.

(GRASS Raster Data Export Program)

SYNOPSIS

`r.out.pov`

`r.out.pov help`

`r.out.pov` [`-h`] `map=name` `tga=name` [`hftype=value`] [`bias=value`] [`scale=value`]

DESCRIPTION

`r.out.pov` converts a user–specified raster map layer (`map==name`) into a height–field file for POVray (`tga==name`). The `hftype==value` option (where `value` is either 0 or 1) specifies the height–field type. When the user enters 0 the output will be actual heights. If entered 1 the cell–values will be normalized. If `hftype` is 0 (actual heights) the `bias==value` can be used to add or subtract a value from heights. Use `scale==value` to scale your heights by value. The GRASS program `r.out.pov` can be used to create height–field files for Persistence of Vision (POV) raytracer. POV can use a height–field defined in Targa (.TGA) image file format where the RGB pixel values are 24 bits (3 bytes). A 16 bit unsigned integer height–field value is assigned as follows: RED = high byte, GREEN = low byte, BLUE = empty.

Parameters:

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

`tga=name` Name of TARGA outputfile (one should add the extension .tga).

`hftype=value` 0=actual heights, 1=normalized heights.

`bias=value` Bias which is added or subtracted to heights.

`scale=value` Value to stretch or shrink elevations.

`r.out.pov` 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 a name for tga (output), using the form

```
r.out.pov map=inname tga=outname
```

where inname is the name of a raster map layer to be converted to POV format, and outname is the name of the outputfile. Further optional values can be entered.

Alternately, the user can simply type *r.out.pov* 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.out.pov map=elevation tga=out.tga
```

EXAMPLE

An example Povray script file may look like this:

```
#include "shapes.inc"
#include "colors.inc"
#include "textures.inc"

#declare Scale = 7;

light_source { color MainLight }

camera {
  location <23000, Scale*2000, 0>
  angle 90
  look_at <23000, Scale*1400, 5000>
}

height_field {
  tga "dem.lr.tga"
  smooth
  water_level 0.11 // 726 / 6553.6 = 0.111
  texture {
    pigment {
      image_map { // image is always projected from -z, with front facing +z, top to +Y
        ppm "map.lr.ppm"
        once
      }
      rotate x*90 // align map to height_field
    }
  }
}

finish {
  ambient 0.2 // Very dark shadows
  diffuse 0.8 // Whiten the whites
  phong 0.2 // shiny
  phong_size 100.0 // with tight highlights
  specular 0.5
  roughness 0.05
}

scale <14500, Scale*6553.6, 13000 >
translate
}
```

AUTHOR

Klaus Meyer, GEUM.tec GbR, eMail: GEUM.tec@geum.de

NOTICE

This program is part of the contrib section of the GRASS distribution. As such, it is externally contributed code that has not been examined or tested by the Office of GRASS Integration.

Last changed: \$Date: 2003/03/16 12:25:19 \$



NAME

r.out.ppm3 – converts 3 GRASS raster layers (R,G,B) to a PPM image file, using the current region.

Usage:

r.out.ppm3 [-qc] **red**=name **green**=name **blue**=name **output**=name

Flags:

-q Run quietly

-c Store the region in the image as comments

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.

output=filename.ppm

Name of PPM file to be written (use out=- for stdout).

This program converts a GRASS raster file to a PPM image file using the the current region.

To get the full area of the raster map, do:

```
g.region rast=[mapname]
```

before running **r.out.ppm3**.

NOTES

One pixel is written for each cell value, so if `ew_res` and `ns_res` differ, the aspect ratio of the resulting image will be off.

SEE ALSO

[r.out.ppm](#)

[r.in.ppm](#)

[d.rgb](#)

AUTHOR

Glynn Clements Based upon *r.out.ppm* and *d.rgb*.

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.out.ppm – converts a GRASS raster file to a PPM image file at the pixel resolution of the CURRENTLY DEFINED REGION.

Usage:

r.out.ppm [-qG] **input**=name **output**=name

Flags:

-q Run quietly

-G Output greyscale instead of color

Parameters:

input Raster file to be converted.

output Name for new PPM file. (use out=- for stdout)

default: <rasterfilename>.ppm

This program converts a GRASS raster file to a PPM image file at the pixel resolution of the CURRENTLY DEFINED REGION. e.g., to get the resolution of the raster map, do:

```
g.region rast=[mapname]
```

before running r.out.ppm.

The PPM file created is 24bit color, rawbits storage by default. Using -G, you may force r.out.ppm to use 8bit greyscale instead. The greyscale conversion uses the NTSC conversion:

$$Y = .30*Red + .59*Green + .11*Blue$$

One pixel is written for each cell value, so if ew_res and ns_res differ, the aspect ratio of the resulting image will be off.

NOTE

A few ppm file comments are written – the name of the GRASS raster file, resolution, etc. Although these are perfectly legal, I've found one PD image utility that chokes on them, so if you need a commentless ppm file, use out=- > outfile.ppm. (When sending output to stdout, no comments are written.)

TODO

Allow combination of 3 grass files to represent R, G, B components of ppm file?

Bill Brown <brown@gis.uiuc.edu>

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.out.rlc – converts a GRASS raster file to a RLC encoded image file

Usage:

r.out.rlc [-v] **input**=name **output**=name

Flags:

-q Run verbose

Parameters:

input Raster map to export

output export file name

This program converts a GRASS raster file to a RLC encoded image file at the pixel resolution of the CURRENTLY DEFINED REGION. e.g., to get the resolution of the raster map, do:

```
g.region rast=[mapname]
```

before running r.out.rlc.

SEE ALSO

[r.in.bin](#), [r.in.ascii](#), [r.out.ascii](#), [r.in.arc](#), [r.out.arc](#)

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.out.tiff – Exports a GRASS raster file to a 8/24bit TIFF image file at the pixel resolution of the CURRENTLY DEFINED REGION.
(GRASS Raster Data Export Program)

GRASS VERSION

4.x, 5.x (versions differ)

SYNOPSIS

r.out.tiff [-ptlv] **input**=name **output**=name [**compression**=name

DESCRIPTION

This program converts a GRASS raster file to a TIFF raster file. Output may be 8 or 24 bit (TrueColor). Optionally, a TIFF World file compatible with ESRI's and other's products may be output.

OPTIONS

Flags:

- p TIFF Palette output (8bit instead of 24bit)
- t Output TIFF world file (**output.tfw**)
- l Output Tiled TIFF
- v Verbose mode

Parameters:

- input*=name
Name of an existing GRASS raster file to be exported.
- output*=name
Name of new TIFF image file.
- compression*=name
Type of compression to use for TIFF file.
Options: none, packbit, deflate

Default: none

The program prompts the user for the name of a GRASS raster file, an output TIFF file, whether an 8 or 24 bit format is desired, and whether or not to create a TIFF world file. Currently only uncompressed, packbit, or deflate TIFF files are written. These output formats are known to be compatible with r.in.tiff.

When writing with "-l" option, tiles are written at 128x128 pixels. For programs that can utilize tiles, it can help speed up some drawing operations.

The user may adjust region and resolution before export using *g.region*.

SEE ALSO

[*g.region r.in.tiff.html*](#)

AUTHOR

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

GRASS 5.0 team

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.out.xyz – Export GRASS raster files into xyz format
(*GRASS Raster Data Export Command*)

SYNOPSIS

r.out.xyz help
r.out.xyz input=*name*

Parameter:

input
Name of raster file

Example:

```
3566625.220431 5763805.084400 145.010000
3566675.164037 5763805.084400 148.240000
3566725.107644 5763805.084400 151.480000
3566775.051250 5763805.084400 NULL
3566824.994856 5763805.084400 163.660000
3566874.938462 5763805.084400 165.900000
```

NOTE

Using unix piping the result can be directed into a file

```
r.out.xyz in=dgm12 > dgm12.txt
```

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.param.scale – GRASS module that extracts terrain parameters from a DEM. Uses a multi-scale approach by fitting a bivariate quadratic polynomial to a given window size using least squares.

Usage:

```
r.param.scale [-c] in=name out=name [s_tol=value] [c_tol=value] [size=value] [param=name] [exp=value]
[zscale=value]
```

Flags:

-c Constrain model through central window cell

Parameters:

in Input elevation raster

out Output raster representing the selected morphometric parameter

s_tol Slope tolerance that defines a 'flat' surface (degrees)
default: 1.0

c_tol Curvature tolerance that defines 'planar' surface
default: 0.0001

size Size of processing window (odd numbers only)
default: 3

param Morphometric parameter to calculate
options: elev,slope,aspect,profc,planc,longc,crosc,minic,maxic, feature
default: elev

exp Exponent for distance weighting (0.0–4.0)
default: 0.0

zscale Vertical scaling factor
default: 1.0

The module calculates the following parameters (terminology is from Wood, 1996 with related terminology)

used in other GRASS modules listed in brackets)

- *elev*: Generalised elevation value (for resampling purposes at different scale)
- *slope*: Magnitude of maximum gradient (steepest slope angle)
- *aspect*: Direction of maximum gradient (steepest slope direction=flow direction)
- *profc* or profile curvature (curvature intersecting with the plane defined by Z axis and maximum gradient direction). Positive values describe convex profile curvature, negative values concave profile curvature.
- *planc* or plan curvature (horizontal curvature, intersecting with the XY plane)
- *longc* or longitudinal curvature (profile curvature intersecting with the plane defined by the surface normal and maximum gradient direction)
- *crosc* or cross-sectional curvature (tangential curvature intersecting with the plane defined by the surface normal and a tangent to the contour – perpendicular to maximum gradient direction)
- *maxic* or maximum curvature (can be in any direction)
- *minic* or minimum curvature (in direction perpendicular to the direction of of maximum curvature)
- *meanc* or mean curvature (average of maximum and minimum curvatures).
- Morphometric features: peaks, ridges, passes, channels, pits and planes

NOTES

In *r.param.scale* the direction of maximum gradient (considered downslope) is stored as (West is 0 degree, East is +/- 180 degree):

- 0..+180 degree from West to North to East
- 0..-180 degree from West to South to East

Note that the aspect map is calculated differently from [r.slope.aspect](#).

Still to do

Fix bug when 'constrain through central cell' option selected. Create color tables for all output files (presently only on features).

SEE ALSO

[d.param.scale](#)

See also Java Code in [LandSerf](#) that implements the same procedure

REFERENCE

Wood, J. (1996): The Geomorphological characterisation of Digital Elevation Models. Diss., Department of Geography, University of Leicester, U.K.

online at:

www.geog.le.ac.uk/jwo/research/dem_char/thesis/index.html

AUTHOR

jwo@le.ac.uk – [ASSIST's home](#)

Update to FP 3/2002: L. Potrich, M. Neteler, S. Menegon (ITC-irst)

Last changed: \$Date: 2003/03/16 12:25:19 \$



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 [-qz] **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 NULL/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 current geographic region definition and mask settings are respected.

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

r.patch [-qz] **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.

Flag:

-q

GRASS Raster Commands

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

-z

Use zero (0) for transparency instead of NULL.

Parameters:

input=name,name,...

The name(s) of between one and 200 existing raster map layers to be patched together. The first of the maximal 200 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](#)

[parser](#)

AUTHOR

Michael Shapiro, U.S. Army Construction Engineering Research Laboratory
-z flag by Huidae Cho

Last changed: \$Date: 2002/03/04 10:32:05 \$



NAME

r.plane – Creates a tilted plane.

GRASS VERSION

5.x

SYNOPSIS

r.plane

r.plane help

r.plane [*dip=value*][*azimuth=value*][*easting=value*]

OPTIONS

Parameters:

dip=value

Please give the value for the dip (inclination) in degrees. The value must be between -90 and 90 from horizontal, with positive values pointing down. Real numbers are valid. No value results in $dip=0$ and horizontal surface.

azimuth=value

Please give the value for the azimuth in degrees counterclockwise from north. The value must be between 0 and 360 . Real numbers are valid.

easting=value

Please enter easting, northing for one point on the plane. Real numbers are valid.

EXAMPLE

r.plane help – it gets the current region

```
n=5777911.25
s=5773623.75
w=3552072.25
e=3557172.25
nsres=12.5
ewres=12.5
```

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 [-lcb] **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 [-lcb] 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.

Flags:

- l* Smooth corners.
- c* Output values as category labels
- b* Output only area borders as vector lines

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. Using *-b* the area borders (vector line type) are written instead of the areas (vector area type). This may be useful for buffering of area borders and other analysis.

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](#)

[parser](#)

AUTHOR

Original version of *r.poly*:

Jean Ezell and Andrew Heekin,

U.S. Army Construction Engineering Research Laboratory

Modified program for smoothed lines:

David Satnik, Central Washington University

Updated 2001 by Andrea Aime, Modena, Italy

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 [-g] [-i] **input**=name **output**=name **profile**=east,north[,east,north,east,north,...] **res**=resolution

DESCRIPTION

This program outputs a two or four column (with -g) ASCII file (or stdout). The default two column output consists of cumulative profile length in meters and raster value. The optional four column output consists of an easting, northing, cumulative profile length (m), and raster value. Profile end or "turning" points can be set manually with the *profile* argument or selected interactively from the GRASS monitor by setting the -i flag. The profile resolution, or distance between profile points, is obtained from the current region resolution, or can be manually set with the *res* argument.

OPTIONS

Flags:

-g

Output easting and northing in first two columns of four column output

-i

Interactive mode

The user selects the profile from the GRASS monitor by clicking the left mouse button along the profile, clicking the right mouse button ends the profile

Parameters:

input

The name of the GRASS raster file from which to generate the profile info

output

The name of the 2 or 4 column ASCII file to send profile results to
Use "-" to send output to standard output (stdout)

profile

The comma separated geographic coordinates for profile line endpoints
The interactive flag (-i) overrides this option

res

This sets the distance between each profile point (resolution)

The resolution must be provided in GRASS database units (i.e. decimal degrees for Lat Long database and meters for UTM)

By default *r.profile* uses the resolution of the current GRASS region

OUTPUT FORMAT

The 2 or 4 column output from *r.profile* is intended for easy use in other programs. The output can be piped (|) directly into other programs or saved to file for later use. The 4 column output is compatible with *s.in.ascii* and can be piped directly into this program for later plotting.

```
r.profile -ig input=elev.rast output=- | s.in.ascii sites=elev.profile
```

The 2 column output is compatible with most plotting programs.

NOTES

The profile resolution is measured exactly from the supplied end or "turning" point along the profile. The end of a profile segment will be an exact multiple of the profile resolution and will therefore not always match the end point coordinates entered for the segment.

To extract the numbers in scripts, following parameters can be used:

```
r.profile dgm12.5 prof=3570631,5763556 out=- 2>/dev/null
```

This filters out the everything except the numbers.

SEE ALSO

[d.profile](#)

[r.what](#)

[r.transect](#)

AUTHOR

[Bob Covill](#)

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.proj – Allows the user to re-project a raster map from one location to the current location.
(GRASS Raster Program)

SYNOPSIS

r.proj [-l] input=name location=name [output=name] [mapset=name] [dbase=name] [method=name]
[resolution=value]

Flags:

-l
List raster files in input location and exit

Parameters:

input
input raster map from source location

location
source location of input map

output
output raster map for current location

mapset
mapset of input map

dbase
path to GRASS database of input location

method
interpolation method to use
options: nearest,bilinear,cubic
default: nearest

resolution
resolution of output map

NOTES

If **output** 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 **mapset** is not specified, its name is assumed to be the same as the current mapset's name.

r.proj supports transformations between nad27 and nad83. It performs the transformation automatically based on the datum specified in the mapset region. See BUGS section.

DESCRIPTION

r.proj projects a raster map in a specified mapset of a specified location from the projection of the input location to a raster map in the current location. The projection information is taken from the momentary PROJ_INFO files.

Introduction

Map projections

Map projections are a method of representing information from a curved surface (usually a spheroid) in two dimensions, typically to allow indexing through cartesian coordinates. There are a wide variety of projections, with common ones divided into a number of classes, including cylindrical and pseudo-cylindrical, conic and pseudo-conic, and azimuthal methods, each of which may be conformal, equal-area, or neither.

The particular projection chosen depends on the purpose of the project, and the size, shape and location of the area of interest. For example, normal cylindrical projections are good for maps which are of greater extent east-west than north-south and in equatorial regions, while conic projections are better in mid-latitudes; transverse cylindrical projections are used for maps which are of greater extent north-south than east-west; azimuthal projections are used for polar regions. Oblique versions of any of these may also be used. Conformal projections preserve angular relationships, and better preserve arc-length, while equal-area projections are more appropriate for statistical studies and work in which the amount of material is important.

Projections are defined by precise mathematical relations, so the method of projecting coordinates from a geographic reference frame (latitude-longitude) into a projected cartesian reference frame (eg metres) is governed by these equations. Inverse projections can also be achieved. The public-domain Unix software package *proj* [1] has been designed to perform these transformations, and the user's manual contains a detailed description of over 100 useful projections. This also includes a programmers library of the projection methods to support other software development.

Thus, converting a "vector" map – in which objects are located with arbitrary spatial precision – from one projection into another is usually accomplished by a simple two-step process: first the location of all the points in the map are converted from the source through an inverse projection into latitude-longitude, and then through a forward projection into the target. (Of course the procedure will be one-step if either the source or target is in geographic coordinates.)

Converting a "raster map", or image, between different projections, however, involves additional considerations. A raster may be considered to represent a sampling of a process at a regular, ordered set of locations. The set of locations that lie at the intersections of a cartesian grid in one projection will not, in general, coincide with the sample points in another projection. Thus, the conversion of raster maps involves an interpolation step in which the values of points at intermediate locations relative to the source grid are estimated.

Projecting maps within the GRASS GIS

GIS data capture, import and transfer often requires a projection step, since the source or client will frequently be in a different projection to the working projection.

In some cases it is convenient to do the conversion outside the package, prior to import or after export, using software such as *proj* [1]. This is certainly the easiest method for **site-lists**, since there is no topology to be

preserved, and *proj* can be used to process simple lists with a one-line command.

The format of files describing maps containing **lines** and **arcs** is generally more complex, as even in ascii parts of the files will describe topology, and not just locations. In the GRASS GIS package a program *v.proj* is provided to convert "vector" maps, transferring topology and attributes as well as node locations. This program uses the projection definition and parameters which are stored in the PROJ_INFO and PROJ_UNITS files in the PERMANENT mapset directory for every GRASS location.

However, although it is oriented mainly towards operations on **raster maps**, the standard GRASS distribution includes this *r.proj* module to convert raster maps. That is the purpose of the program described here.

Design of r.proj

As discussed briefly above, the fundamental step in re-projecting a raster is resampling the source grid at locations corresponding to the intersections of a grid in the target projection. The basic procedure for accomplishing this, therefore, is as follows:

r.proj converts a map to a new geographic projection. It reads a map from a different location, projects it and write it out to the current location.

The projected data is resampled with one of three different methods: nearest neighbor, bilinear and cubic convolution.

The nearest option, which performs a nearest neighbor assignment is the fastest of the three resampling methods. It is primarily used for categorical data such as a land use classification, since it will not change the values of the data cells. The bilinear option determines the new value of the cell based on a weighted distance average of the 4 surrounding cells in the input map. The cubic option determines the new value of the cell based on a weighted distance average of the 16 surrounding cells in the input map.

The bilinear and cubic interpolation methods are most appropriate for continuous data and cause some smoothing. Both options shouldn't be used with categorical data, since the cell values will be altered. If nearest neighbor assignment is used, the output map has the same raster format as the input map. If any of the both interpolations is used, the output map is written as floating point.

Note that, following normal GRASS conventions, the coverage and resolution of the resulting grid is set by the current region settings, which may be adjusted using *g.region*. The target raster will be relatively unbiased for all cases if its grid has a similar resolution to the source, so that the resampling/interpolation step is only a local operation. If the resolution is changed significantly, then the behaviour of the generalisation or refinement will depend on the model of the process being represented. This will be very different for categorical versus numerical data. Note that three methods for the local interpolation step are provided.

BUGS

Currently *r.proj* does not support general datum conversions. It only supports conversion between nad27 and nad83, and only within the CONUS conversion grid; 20 degrees to 50 degrees north latitude, 63 degrees to 131 degrees west longitude. That covers all of the conterminous USA plus Mexico north of Mexico City and most of Canada farther south than Winnipeg, Manitoba.

The entire input map is read into memory even if portions only are transformed. This requires a large amount of memory if large raster layers where projected.

References:

[1]Evenden, G.I. (1990) Cartographic projection procedures for the UNIX environment – a user's manual. USGS Open–File Report 90–284 (Also see Interim Report and 2nd Interim Report on Release 4, Evenden 1994).

Richards, John A. (1993), Remote Sensing Digital Image Analysis, Springer–Verlag, Berlin, 2nd edition.

SEE ALSO

[m.proj](#), [s.proj](#), [v.proj](#), [g.setproj](#), [i.rectify](#), [i.rectify3](#), [r.support](#), [r.stats](#), [s.sample](#), [s.surf.idw](#), [s.surf.rst](#)

AUTHOR

Martin Schroeder, University of Heidelberg, Germany

Man page text from S.J.D. Cox, AGCRC, CSIRO Exploration & Mining, Nedlands, WA

Updated by [Morten Hulden](#)

Last changed: \$Date: 2002/04/22 16:07:49 \$



NAME

r.quant – This routine produces the quantization file for a floating–point map.
(GRASS 5 Raster Program)

SYNOPSIS

r.quant

r.quant [**-tr**] **input=name[,name...]** [**basemap=map**] [**fprange=dmin,dmax**] [**range=min,max**]

DESCRIPTION

r.quant produces the quantization file for a floating–point map.

map=name The map for which the rules be to be created. If more than one map is specified, then this implies that the floating–point range is the minimum and maximum of all the maps together, unless either *basemap=map* or *fprange=min,max* is specified.

basemap=map The quant rules of this map set the quantization.

fprange=min,max This sets the floating–point range for the quantization.

-t flag Truncate floating point data

-r flag Round floating point data

range=min,max This sets the integer range for the quantization. Otherwise a default of 1–255 is used.

Quant rules

The quant rules have to be entered interactively.

If rules is specified, the input has the form:

```
value1:value2:cat1:[cat2]
```

where *value1* and *value2* are floating point values and *cat1* and *cat2* are integers. If *cat2* is missing, it is taken to be equal to *cat1*. All values can be "*" which means infinity.

NOTE

It is an error for both *basemap* and *fprange* to be specified.

SEE ALSO

[*r.support*](#) and [*r.null*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



r.random.cells – generates random sampling of cells.

(GRASS Raster Program)

SYNOPSIS

```
r.random.cells
r.random.cells help
r.random.cells [-q] output=name [distance=value] [seed=value]
```

DESCRIPTION

r.random.cells generates a random sets of cells that are at least *distance* apart. The cells are numbered from 1 to the numbers of cells generated. Random cells will not be generated in areas masked off.

OPTIONS

-q This flag prevents the program from sending a verbose description to standard output of what the program is doing.

output Output map: Random cells. Each random cell has a unique non-zero cell value ranging from 1 to the number of cells generated. The heuristic for this algorithm is to randomly pick cells until there are no cells outside of the chosen cell's buffer of radius *distance*.

distance Input value(s) [default 0.0]: *distance* determines the minimum distance the centers of the random cells will be apart.

seed Input value [default: random]: Specifies the random seed that **r.random.cells** will use to generate the cells. If the random seed is not given, **r.random.cells** will get a seed from the process ID number.

NOTES

The original purpose for this program was to generate independent random samples of cells in a study area. The *distance* value is the amount of spatial autocorrelation for the map being studied. The amount of spatial autocorrelation can be determined by using **r.2Dcorrelogram** with **r.2Dtoid**, or **r.1Dcorrelogram**. With *distance* set to zero, the *output* map will number each non-masked cell from 1 to the number of non-masked cells in the study region.

REFERENCES

Random Field Software for GRASS by Chuck Ehlschlaeger

As part of my dissertation, I put together several programs that help GRASS (4.1 and beyond) develop uncertainty models of spatial data. I hope you find it useful and dependable. The following papers might

clarify their use:

["Visualizing Spatial Data Uncertainty Using Animation \(final draft\)"](#), by Charles R. Ehlschlaeger, Ashton M. Shortridge, and Michael F. Goodchild. Submitted to Computers in GeoSciences in September, 1996, accepted October, 1996 for publication in June, 1997.

["Modeling Uncertainty in Elevation Data for Geographical Analysis"](#), by Charles R. Ehlschlaeger, and Ashton M. Shortridge. Proceedings of the 7th International Symposium on Spatial Data Handling, Delft, Netherlands, August 1996.

["Dealing with Uncertainty in Categorical Coverage Maps: Defining, Visualizing, and Managing Data Errors"](#), by Charles Ehlschlaeger and Michael Goodchild. Proceedings, Workshop on Geographic Information Systems at the Conference on Information and Knowledge Management, Gaithersburg MD, 1994.

["Uncertainty in Spatial Data: Defining, Visualizing, and Managing Data Errors"](#), by Charles Ehlschlaeger and Michael Goodchild. Proceedings, GIS/LIS'94, pp. 246–253, Phoenix AZ, 1994.

SEE ALSO

[r.mask](#), [r.1Dcorrelogram](#), [r.2Dcorrelogram](#), [r.2Dto1D](#), [r.random.surface](#), [r.random.model](#), [r.random](#)

AUTHOR

Charles Ehlschlaeger; National Center for Geographic Information and Analysis, University of California, Santa Barbara.

Last changed: \$Date: 2003/02/09 12:07:23 \$



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 [-iqz] **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 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 with NULL values).

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. If the **-z** is specified, then a unique entry is made for the value used where the *input* was NULL. This value is at least 1 less than the smallest value in the *input* raster and is given a medium gray color.

The *site_lists* file created by *r.random* contains a listing of the sites' coordinates; these coordinates are the *center points* of the randomly selected cells. A double attribute contains the cell value of the *input* raster (or the assigned value when **-z** is used).

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 cells* (e.g., 10%, or 3.05%). The number of cells considered for the percentage reflects whether or not the **-z** flag was given. Options are 0–100; percentages less than one percent may be stated as decimals.

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 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 NULL values (using the **-z** option).

The **-i** can be used to get a count of the total cells and NULL cells given the current region settings.

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 [parser](#) interface.

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 NULL cells in the pool from which *r.random* will randomly generate site locations.
- i** Print the raster map's name and location, the total number of cells under the current region settings, and the number of NULL valued cells under the current region settings. Then exit without doing anything. Useful for deciding on the number of sites to have *r.random* create.

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

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 NULL cells in the *input* value will be assigned the category values assigned these cells in the input layer; sites falling on NULL cells in the *input* layer will be assigned to a newly created category in the *raster_output* layer with at least one integer value less than the minimum value in the *input* 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 reclassified raster map layer as

input to *r.random*.

BUGS

It's not possible to use the **-i** flag and not also specify the **nsites** parameter.

SEE ALSO

[*g.region*](#)

[*r.mask*](#)

[*r.reclass*](#)

[*s.random*](#)

[*parser*](#)

AUTHOR

Dr. James Hinthorne, GIS Laboratory, Central Washington University

Modified for GRASS 5.0 by Eric G. Miller

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.random.surface – generates random surface(s) with spatial dependence.
(GRASS Raster Program)

SYNOPSIS

r.random.surface

r.random.surface help

r.random.surface [-uq] **output**=name(s) [**distance**=value(s)] [**exponent**=value(s)] [**flat**=value(s)]
[**weight**=value(s)] [**seed**=value] [**high**=value]

DESCRIPTION

r.random.surface generates a spatially dependent random surface. The random surface is composed of values representing the deviation from the mean of the initial random values driving the algorithm. The initial random values are independent Gaussian random deviates with a mean of 0 and standard deviation of 1. The initial values are spread over each output map using filter(s) of diameter distance. The influence of each random value on nearby cells is determined by a distance decay function based on exponent. If multiple filters are passed over the output maps, each filter is given a weight based on the weight inputs. The resulting random surface can have "any" mean and variance, but the theoretical mean of an infinitely large map is 0.0 and a variance of 1.0. Description of the algorithm is in the **NOTES** section.

The random surface generated are composed of floating point numbers, and saved in the category description files of the output map(s). Cell values are uniformly or normally distributed between 1 and high values inclusive (determined by whether the *-u* flag is used). The category names indicate the average floating point value and the range of floating point values that each cell value represents.

r.random.surface's original goal is to generate random fields for spatial error modeling. A procedure to use *r.random.surface* in spatial error modeling is given in the **NOTES** section.

OPTIONS

Flags:

-q

This flag prevents the program from sending information to standard output.

-u

This flag will make the output map's cell values have a uniform distribution instead of a normal distribution. Uniform distributions are useful in situations where the user wants random values for a range of integer values. The user then uses the cell values instead of the random values in the category description file. The uniform flag is also useful when the random values near the mean are more

meaningful to the user. The flag on will cause the cell values from low to high to have an equal chance of being used. Thus, not using the flag is useful in situations where you want to use the floating point numbers in the category description in your analysis. Keeping this flag off will allow the outliers of the random distribution to be more accurately represented. See the **NOTES** section for example use of this flag.

Parameters:

output

Output map(s): Random surface(s). The cell values are a random distribution between the low and high values inclusive. The category values of the output map(s) are in the form "#.# #.# to #.#" where each #.# is a floating point number. The first number is the average of the random values the cell value represents. The other two numbers are the range of random values for that cell value. The "average" mean value of generated \fIoutput\fr map(s) is 0. The "average" variance of map(s) generated is 1. The random values represent the standard deviation from the mean of that random surface.

distance

Input value(s) [default 0.0]: distance determines the spatial dependence of the output map(s). The distance value indicates the minimum distance at which two map cells have no relationship to each other. A distance value of 0.0 indicates that there is no spatial dependence (i.e., adjacent cell values have no relationship to each other). As the distance value increases, adjacent cell values will have values closer to each other. But the range and distribution of cell values over the output map(s) will remain the same. Visually, the clumps of lower and higher values gets larger as distance increases. If multiple values are given, each output map will have multiple filters, one for each set of distance, exponent, and weight values.

exponent

Input value(s) [default 1.0]: exponent determines the distance decay exponent for a particular filter. The exponent value(s) have the property of determining the "texture" of the random surface. Texture will decrease as the exponent value(s) get closer to 1.0. Normally, exponent will be 1.0 or less. If there are no exponent values given, each filter will be given an exponent value of 1.0. If there is at least one exponent value given, there must be one exponent value for each distance value.

flat

Input value(s) [default 0.0]: flat determines the distance at which the filter

weight

Input value(s) [default 1.0]: weight determines the relative importance of each filter. For example, if there were two filters driving the algorithm and weight=1.0, 2.0 was given in the command line: The second filter would be twice as important as the first filter. If no weight values are given, each filter will be just as important as the other filters defining the random field. If weight values exist, there must be a weight value for each filter of the random field.

high

Input value [default 255]: Specifies the high end of the range of cell values in the output map(s). Specifying a very large high value will minimize the "errors" caused by the random surface's discretization. The word errors is in quotes because errors in discretization are often going to cancel each other out and the spatial statistics are far more sensitive to the initial independent random deviates than any potential discretization errors.

seed

Input value(s) [default random]: Specifies the random seed(s), one for each map, that *r.random.surface* will use to generate the initial set of random values that the resulting map is based on. If the random seed is not given, *r.random.surface* will get a seed from the process ID number.

NOTES

While most literature uses the term random field instead of random surface, this algorithm always generates a surface. Thus, its use of random surface.

r.random.surface builds the random surface using a filter algorithm smoothing a map of independent random deviates. The size of the filter is determined by the largest distance of spatial dependence. The shape of the filter is determined by the distance decay exponent(s), and the various weights if different sets of spatial parameters are used. The map of independent random deviates will be as large as the current region PLUS the extent of the filter. This will eliminate edge effects caused by the reduction of degrees of freedom. The map of independent random deviates will ignore the current mask for the same reason.

One of the most important uses for *r.random.surface* is to determine how the error inherent in raster maps might effect the analyses done with those maps. If you wanted to check to see how sensitive your analysis is to the errors in the DEMs in your study area, see:

["Visualizing Spatial Data Uncertainty Using Animation \(final draft\)"](#), by Charles R. Ehlschlaeger, Ashton M. Shortridge, and Michael F. Goodchild. Submitted to Computers in GeoSciences in September, 1996, accepted October, 1996 for publication in June, 1997.

["Modeling Uncertainty in Elevation Data for Geographical Analysis"](#), by Charles R. Ehlschlaeger, and Ashton M. Shortridge. Proceedings of the 7th International Symposium on Spatial Data Handling, Delft, Netherlands, August 1996.

["Dealing with Uncertainty in Categorical Coverage Maps: Defining, Visualizing, and Managing Data Errors"](#), by Charles Ehlschlaeger and Michael Goodchild. Proceedings, Workshop on Geographic Information Systems at the Conference on Information and Knowledge Management, Gaithersburg MD, 1994.

["r.random, r.mapcalc, r.mask, s.random"](#)

REFERENCES

Random Field Software for GRASS by Chuck Ehlschlaeger

As part of my dissertation, I put together several programs that help GRASS (4.1 and beyond) develop uncertainty models of spatial data. I hope you find it useful and dependable. The following papers might clarify their use:

["Visualizing Spatial Data Uncertainty Using Animation \(final draft\)"](#), by Charles R. Ehlschlaeger, Ashton M. Shortridge, and Michael F. Goodchild. Submitted to Computers in GeoSciences in September, 1996, accepted October, 1996 for publication in June, 1997.

["Modeling Uncertainty in Elevation Data for Geographical Analysis"](#), by Charles R. Ehlschlaeger, and Ashton M. Shortridge. Proceedings of the 7th International Symposium on Spatial Data Handling, Delft, Netherlands, August 1996.

["Dealing with Uncertainty in Categorical Coverage Maps: Defining, Visualizing, and Managing Data Errors"](#), by Charles Ehlschlaeger and Michael Goodchild. Proceedings, Workshop on Geographic Information Systems at the Conference on Information and Knowledge Management, Gaithersburg MD, 1994.

["Uncertainty in Spatial Data: Defining, Visualizing, and Managing Data Errors"](#), by Charles Ehlschlaeger and Michael Goodchild. Proceedings, GIS/LIS'94, pp. 246–253, Phoenix AZ, 1994.

AUTHORS

Charles Ehlschlaeger, Michael Goodchild, and Chih–chang Lin; National Center for Geographic Information and Analysis, University of California, Santa Barbara.

Last changed: \$Date: 2003/02/09 12:07:23 \$



NAME

r.rational.regression – linear and nonlinear regression calculation from data stored in ASCII file
(GRASS Raster Processing Program)

SYNOPSIS

r.rational.regression

r.rational.regression help

r.rational.regression **input=name** **output=name** [**check="phrase"**] [**predict="phrase"**] [**plot="phrase"**]
[**calculat="phrase"**]

DESCRIPTION

The *r.rational.regression* program calculates the linear or nonlinear regression model. If it is used as an image precessing tool, the multispectral space remote sensing data will be the regression variables (ascii file) and the ground vegetation coverage measurements will be the response variables (also ascii file) and this command will be useful for obtaining linear or nonlinear regression models from the remote-sensing data which have corresponding ground mearurement and for predicting vegetation coverage using other remote-sensing data which have no corresponding ground truth records. The input file has the following format

regression valuables x1, x2, ... response variable y

channel 1 (x1) channel 2 (x2) ... coverage

For a three channel remote-sensing data the following is an example of input ascii file

```
0.4350 0.2616 0.7016 0.98
0.4140 0.2620 0.6520 0.99
0.4940 0.3500 0.5580 0.34
0.5983 0.5350 0.5650 0.10
0.4883 0.3733 0.5533 0.88
0.4150 0.2916 0.5116 0.60
0.5566 0.5250 0.5466 0.09
0.4420 0.2820 0.6800 0.86
0.4220 0.2620 0.6260 0.88
0.4766 0.3666 0.5933 0.61
0.5180 0.4300 0.5140 0.60
0.4416 0.2700 0.7383 0.96
0.4583 0.3116 0.5133 0.76
0.4300 0.2750 0.7233 0.98
0.4320 0.2760 0.6460 1.00
0.4733 0.3566 0.5616 0.53
```

```

0.4200 0.2450 0.7966 1.00
0.4850 0.3533 0.7216 0.99
0.4360 0.2620 0.7620 0.99
0.4283 0.2650 0.6783 0.91
0.4633 0.3200 0.6750 0.94

```

The resulted regression model (coefficient numbers) and related information about the confidential test, goodness or utility test (e.g., correlation coefficient r between observed and calculated coverage, F value and t value) are put on the output file (ascii file also).

Eight models can be chosen by user after prompted by the program. They are:

1) usual linear model:

$$y=a[0]x[0]+a[1],$$

$$y=a[0]x[0]+a[1]x[1]+a[2]$$

$$y=a[0]x[0]+a[1]x[1]+a[2]x[2]+a[4]$$

2) linear model with remote-sensing data normalized by the data in the first spectral band; normalized by x_1 :

$$y=a[0]x[1]/x[0] + a[1]$$

$$y=a[0]x[1]/x[0] + a[1]x[2]/x[0] + a[2]$$

3) linear model with normalization by the second band; normalized by x_2 :

$$y=a[0]x[0]/x[1] + a[1]$$

$$y=a[0]x[0]/x[1] + a[1]x[2]/x[1] + a[2]$$

4) NDVI (normalized differential vegetation index) model;

5) NDVI model for intensity instead of radiance;

6) NDVI model for reflectance;

7) semi-relaxation vegetation index model;

8) RVI (relaxation vegetation index) nonlinear model.

The user has three options for check: `multx1`, `multx2`, and `multx3`. These options check multi-collinearity of the input data. `multx1` calculates R^2 for channel x_1 being replaced by coverage y . `multx2` for channel x_2 if input data are composed of two remote-sensing channels. And `multx3` for channel x_3 if input data are composed of at least three channels.

The program provides three methods to conduct the nonlinear regression calculation. These methods can be invoked by the user as options of `calculat`. The first one is rational fraction method. If no any option of `calculat` is given by the user, the program will adopt the first method. If `calculat = rvi_linear` the linearization method is taken to conduct the nonlinear relaxation index regression. If the user set `calculat = rvi_nonlinear` the general nonlinear regression method is used. If `calculat = all`, the program will conduct nonlinear regression first using the rational fraction method and then using the obtained coefficients as initial values to conduct the second method calculation and finally using the third method to improve the results. For nonlinear regression calculation there may exist multi-minimums. We can not be in reliance on the existing usual algorithms which can find one minimum only. The computer will not scan the possible minimums in order to save computer time. The user should judge and select a least minimum during the iteration. The necessary parameters which will facilitate the judgement for each iteration are displayed in the screen and also output to the output file.

For plots of relevant features of the regression calculation the user can set option for plot. There are four options for the plot. If plot = radiance-coverage computer will send radiance data against coverage data for further plotting radiance-coverage curves or figures to show the scatter of input data. This data will be stored in ASCII files named "curve.radiance_coverage_x1y", "curve.radiance_coverage_x2y" and "curve.radiance_coverage_x3y". If plot = vegetation-soil three ASCII data files will be generated by the program for further figures. Their names are "curve.veget_soil_x1x2", "curve.veget_soil_x1x3" and "curve.veget_soil_x2x3". If the user set plot = adequacy the program will generate residual data files for plotting figures of residuals against predictors and dependent variable to show the adequacy. These files named curve_residual_x1, "curve_residual_x2", "curve_residual_x3" and "curve_residual_y". If plot = all the program will generate all these above mentioned data files for further plotting of different figures.

The program not only can calculate regression models but also can make prediction to new remote sensing data using the obtained model. The user can set option of predict = same to calculate regression model using part of one imagery data and predict vegetation coverage for another part of remote sensing data in the same imagery. If predict = other the program will use one imagery to get regression model and predict vegetation coverage for other imagery. If the user did not give any option for predict the program calculates model based on the whole input data and not conduct any prediction. The program still generate a set of "calculated vegetation coverage" using the obtained model and the same input data in order for the user to check the utility, goodness and confidential status of the regression. *r.rational.regression* will be run non-interactively if the user specifies program arguments on the command line, using the form:

```
r.rational.regression input=name output=name [check="phrase"] [predict="phrase"] [plot="phrase"]
[calculat="phrase"]
```

But after run, the computer will prompt the user to select model number.

Alternately, the user can simply type:

```
r.rational.regression
```

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*.

OPTIONS

Parameters:

check="phrase"

For check of multi-collinearity.

Options: multx1, multx2, multx3

calculat="phrase"

Method of calculation for nonlinear regression.

Options: rvi_linear, rvi_nonlinear, all

plot="phrase"

Name of data files generated by the program for further plotting.

Options: radiance-coverage, vegetation-soil, adequacy, all

predict=

Fashion of prediction.

Options: same, other

SEE ALSO

[*i.ndvi, r.linear.regression*](#)

AUTHORS

Hong C. Zhuang, U.S. Army Construction Engineering Research Laboratory Department of Electrical Computer Engineering, University of Illinois at Urbana–Champaign.

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 reclassified 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 reclassified 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 **reclassified** categories.

```

ENTER NEW CATEGORY NAMES FOR THESE CATEGORIES

TITLE:  Roads Reclassified
CAT     NEW CATEGORY NAME
NUM
0       no data
1       Paved Roads
2       Unpaved Roads
3       Railroad, single track

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

```

Based upon the information supplied by the user in the above sample screens, the new output map, supporting category, color, history, and header files are created.

NON-INTERACTIVE PROGRAM USE: RECLASS RULES

In non-interactive program use, the names of an input map, output map, and output map TITLE are given on the command line. However, the reclass rules are still read from standard input (i.e., from the keyboard, redirected from a file, or piped through another program).

Once the user has specified an input raster map layer, output map layer name, and (optionally) output map layer TITLE by typing

```
r.reclass input=name output=name [TITLE=name]
```

Each line of input must have the following format:

```
input_categories=output_category [label]
```

where the input lines specify the category values in the input raster map layer to be reclassified to the new *output_category* category value. Specification of a *label* to be associated with the new output map layer category is optional. If specified, it is recorded as the category label for the new category value. The equal sign = is required. The *input_category(ies)* may consist of single category values or a range of such values in the format "*low* thru *high*." The word "thru" must be present.

To include all (remaining) values the asterix "*" can be used. This rule has to be set as last rule. No further rules are accepted after setting this rule.

No data have to be specified with NULL.

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, 2 and 3 in the input raster map layer "roads" to category 1 with category label "good quality" in the output map layer, and reclassifies input raster map layer categories 4 and 5 to category 2 with the label "poor quality" in the output map layer.

```
1 2 3 = 1 good quality
4 5 = 2 poor 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. The range from 30 to 40 is reclassified as NULL.

```
1 thru 10 = 1
11 thru 20 = 2
21 thru 30 = 3
30 thru 40 = NULL
```

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. 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. All other values are reclassified to NULL.

```
1 3 5  = 1    poor quality
2 4 6  = 2    good quality
*      = NULL
```

5. 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 (reclassified) 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 will be stored in GRASS as a reclass of the *original* raster map on which the first reclassified 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 reclassified 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. Alternatively *r.recode* can be used.

Category values which are not explicitly reclassified to a new value by the user will be reclassified to NULL.

SEE ALSO

[*r.resample*](#)

[*r.rescale*, *r.recode*](#)

AUTHORS

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.reclass.pg – Generate a new raster reclass map based on the results of multiple queries to the currently selected database. (GRASS–RDBMS Raster Interface Program)

SYNOPSIS

r.reclass.pg

r.reclass.pg help

r.reclass.pg sql=file input=name key=name output=name [label=name]

DESCRIPTION

r.reclass.pg generates reclass rules for a new raster map layer based on the results of multiple queries to the currently selected database. The user constructs a series of mutually exclusive SQL retrieve statements designed to return groups of records from the database. Each group of records should be interanally consistent in terms of attribute characteristics specified by the user in the retrieve clause. These groups should also be mutually exclusive, thereby insuring that a row returned by one retrieve clause is not also returned by a subsequent retrieve clause. Each group of records therefore forms the basis for a single category in the resulting GRASS raster reclass map. *r.reclass.pg* processes each retrieve statement in order generating the GRASS reclass rules needed to create a new raster map. As each retrieve statement is processed the group of records returned receives a common category value. The category value is incremented by one for each subsequent retrieve statement which is processed. The resulting reclass map will have one category for each of the original retrieve statements and an additional category for no data areas. No data in this case includes actual no data areas and areas for which no rows were returned by the database queries.

COMMAND LINE OPTIONS

Parameters:

sql=filename

Name of file containing SQL query statements.

input=map

Name of an existing raster map layer.

key=databasecolumnname

Name of the database column linked to GRASS via the categories in the input map layer.

output=map

Name of new raster (reclass), file.

label=name

Label for new categories.

EXAMPLE:

```
r.reclass.pg sql=dbrr.sql key=grasscat input=seco.soils output=tfact.recl label="Tfact="
```

dbrr.sql:

```
retrieve (secosoilcats.grasscat)
```

```
where layer.tfact >0 and layer.tfact <=2 and layer.muid=secosoilcats.muid
```

```
retrieve (secosoilcats.grasscat)
```

```
where layer.tfact >0 and layer.tfact <=2 and layer.muid=secosoilcats.muid
```

```
retrieve (secosoilcats.grasscat)
```

```
where layer.tfact = 5 and layer.muid=secosoilcats.muid
```

BUGS

None known.

NOTE

This program requires the Postgres database software.

SEE ALSO

[g.column.pg](#), [g.select.pg](#), [g.stats.pg](#), [g.table.pg](#), [d.rast.pg](#), [d.site.pg](#), [d.vect.pg](#), [d.what.r.pg](#), [d.what.s.pg](#), [d.what.v.pg](#), [r.rescale.pg](#), [v.reclass.pg](#)

AUTHOR

Original Informix SQL–tools: James A. Farley, Wang Song, and W. Fredrick Limp University of Arkansas, CAST

Postgres modifications: Janne Soimasuo, Faculty of Forestry, University of Joensuu, Finland.

Updated to GRASS 5 by Alex Shevlakov (sixote@yahoo.com)



NAME

r.reclass.scs – Create a new raster map layer based on an existing raster map.
(SCS GRASS Raster Program)

SYNOPSIS

`r.reclass.scs`
`r.reclass.scs help`

DESCRIPTION

r.reclass.scs is an interface to the GRASS *r.reclass* program. The program will reclassify the category values in a raster map layer based on reclass instructions entered by the user. The user can enter map reclassification rules to *r.reclass.scs* either from standard input or from a file. The program then issues *r.reclass* commands to produce the new reclassified raster map layer.

Input to *r.reclass.scs* consists of a list of category names or category values that will be grouped into the same category in the output (reclassified) map. Only one category name should appear on each line of input. Input can be entered either interactively, or from a file.

A file containing these reclass rules can be created using a text editor, word–processor, DBMS, etc. It is no more than a list of category names which will have the same category value after the reclassification.

SEE ALSO

[*r.reclass*](#)
[*r.resample*](#)
[*r.rescale*](#)

AUTHOR

R.L. Glenn, USDA, SCS, NHQ–CGIS

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.recode – recode raster maps
(GRASS Raster Program)

SYNOPSIS

r.recode [-ad] **input**=name **output**=name [**title**=name]

DESCRIPTION

r.recode creates an output map layer based on an input raster map layer. The output map layer will be a recoding of the input map layer based on recode rules input to *r.recode*. A *title* for the output map layer may be (optionally) specified by the user.

The recode 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 recoded, the *name* of an output layer to hold recoded map, and (optionally) the name of a title for the output map.

Flag:

- a Align the current region to the input map
- d Force output to double map type (DCELL)

Parameters:

- input* Raster map to be recoded
- output* Name for the resulting raster map
- title* Title for the resulting raster map

Enter a rule in one of these formats:

```
old_low:old_high:new_low:new_high
old_low:old_high:new_val  (i.e. new_high == new_low)
*:old_val:new_val        (interval [inf, old_val])
old_val:*:new_val        (interval [old_val, inf])
```

GRASS Raster Commands

r.recode is loosely based on *r.reclass* and uses the GRASS reclass library to convert the rasters. It has routines for converting to every possible combination of raster (eg. int to double, double to float, etc). Standard floating point raster precision is float, with *-d* double precision will be written.

There are four basic routines that it accepts:

1. old-low to old-high is reclassified to new-low to new high , where the user provides all four values. The program figures on the fly what type of raster should be created.
2. old-low to old-high is reclassified to a single new value. Anything outside the range is null.
3. * to old-high will reclass everything less than old-high to a single new value.
4. old-low to * will reclass everything greater than old-low to a single new value.

These four sets of arguments can be given on the command line, or piped via stdin or a file. More than one set of arguments is accepted.

EXAMPLES

Map type conversion

To simply convert a raster between formats (eg. int to float) the user would use the first argument. For example

```
10:1500:0.1:15.0
```

would convert an old raster with range between 10 and 1500 to a float raster with range between 0.1 and 15.0.

Value replacement

r.recode can be used to replace existing cell values by others. The formatting is as described above. In following example the values 1, 2 and 3 are replaced by 1.1, 7.5 resp. 0.4:

```
r.recode in=oldmap out=newmap << EOF
1:1:1.1:1.1
2:2:7.5:7.5
3:3:0.4:0.4
EOF
```

AUTHOR

CERL

Last changed: \$Date: 2002/03/01 00:27:45 \$



NAME

r.region – Program to manage the boundary definitions for a raster map.
(GRASS Raster Program)

SYNOPSIS

r.region

r.region help

r.region [-cd] **map**=name [**region**=name] [**raster**=name] [**vector**=name] [**3dview**=name] [**n**=value]
[**s**=value] [**e**=value] [**w**=value] [**align**=name]

DESCRIPTION

The *r.region* program allows the user to manage the boundaries of a raster map. These 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), a raster or vector map, or a 3dview file.

Flags:

- c
Set the boundaries to those of the current region.
- d
Set the boundaries to those of the default region.

Parameters:

map=name

Raster map whose boundaries are to be set

region=name

Obtain the boundaries from the specified region file

raster=name

Obtain the boundaries from the specified raster map

vector=name

Obtain the boundaries from the specified vector map

3dview=name

Obtain the boundaries from the specified 3dview file

n=value

Set map coordinate value for the map's northern edge to *value*

s=value

Set map coordinate value for the map's southern edge to *value*

e=value

Set map coordinate value for the map's eastern edge to *value*
w=value

Set map coordinate value for the map's western edge to *value*
align=name

Set the current resolution equal to that of the named raster map, and align the boundaries to a row and column edge in the named map. Alignment only moves the existing boundaries 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.

NOTE

After all updates have been applied, the raster map's resolution settings are recomputed from the boundaries and the number of rows and columns in the raster map.

SEE ALSO

[*r.support*](#)

[*g.region*](#)

AUTHOR

Glynn Clements
Based upon *g.region*

Last changed: \$Date: 2002/04/08 01:30:44 \$



NAME

r.report – Reports statistics for raster map layers.
(GRASS Raster Program)

SYNOPSIS

r.report

r.report help

r.report [-hmfqznNCi] **map**=name[,name,...] [**units**=name[,name,...]] [**pl**=value] [**pw**=value]
[**output**=name] [**null**=string][**nsteps**=value]

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 [-hmfqznNCi] map=name[,name,...] [units=name[,name,...]] [pl=value] [pw=value]
[output=name]
```

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**, **-q**, **-z**, **-n**, **-N**, **-C**, and **-i** 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.

- n** Filter out all no data cells.
- N** Filter out cells where all maps have no data.
- C** Report for cats fp ranges (fp maps only).
- i** Read fp map as integer (use map's quant rules).

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.

null=*string*

String representing no data cell value.

nsteps=*value*

Number of fp subranges to collect stats from.

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).

GRASS Raster Commands

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	
REGION:	north: 4928000.00	east: 609000.00	
	south: 4914000.00	west: 590000.00	
	res: 100.00	res: 100.00	
MASK:	none		
MAP:	geology in PERMANENT		
#	Category Information description	Acres	Square 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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 (suppressing 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.

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*](#)

[*parser*](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.resamp.rst – reinterpolates and computes topographic analysis from input raster file to a new raster file (possibly with different resolution) using regularized spline with tension and smoothing.
(GRASS Raster Program)

SYNOPSIS

r.resamp.rst

r.resamp.rst help

r.resamp.rst [-r] [-d] [-t] **input** = name **ew_res** = val **ns_res** = val **elev** = name [**slope** = name] [**aspect** = name] [**pcurv** = name] [**tcurv** = name] [**mcurv** = name] [**smooth** = name] [**maskmap** = name] [**overlap** = val] [**zmult** = val] [**tension** = val]

DESCRIPTION

r.resamp.rst

This program reinterpolates the values a from given raster file named *input* to a new raster file named *elev*. If *-r* flag is specified, all zero elevations in *input* file are treated as elevations, otherwise they are ignored. Reinterpolation (resampling) is done to higher, same or lower resolution which is specified by parameters *ew_res* and *ns_res*. All resulting raster files are created for the given region (which might be different from the header of the input raster file). 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. If *-d* flag is set the program outputs partial derivatives *fx*, *fy*, *fxx*, *fxy*, *fyy* instead of slope, aspect and curvatures.

For noisy data, it is possible to define spatially variable smoothing by providing a raster file *smooth* containing smoothing parameters. With the smoothing parameter set to zero (*smooth* is not given or contains zero data), the resulting surface passes exactly through the data points. 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. Parameter *zmult* allows the user to rescale the z-values (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. The flag *-t* can be set to use "dnorm independent tension". The interpolation is performed for overlapping rectangular segments. The user can define the width of overlap (in number of cells) by option *overlap*.

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:

```
r.resamp.rst [-r] [-d] [-t] input = name ew_res = val ns_res = val elev = name [slope = name] [aspect = name] [pcurv = name] [tcurv = name] [mcurv = name] [smooth = name] [maskmap = name] [overlap = val] [zmult = val ] [tension = val]
```

Alternatively, the user can simply type **r.resamp.rst** 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* .

Flags

-r Indicates that zeroes in input map represent elevation.

-d Output partial derivatives instead of aspect, slope and curvatures

Parameters:

input =*name*

Use the existing site file *name* as input.

ew_res = *val*

Set desired east-west resolution to *val* .

ns_res = *val*

Set desired north-south resolution to *val* .

elev = *name*

Output elevation values to raster file named *name* .

slope = *name*

Output slope or fx values to raster file named *name* .

aspect = *name*

Output aspect or fy values to raster file named *name* .

pcurv = *name*

Output profile curvature or fxx values to raster file named *name* .

tcurv=*name*

Output tangential curvature values or fyy to raster file named *name* .

mcurv=*name*

Output mean curvature values or fxy to raster file named *name*

.

smooth=*name*

18 Set smoothing parameter from file *name* .

maskmap=*name*

Use the existing raster file *name* as a mask.

overlap =*val*

Use overlap *val* cells to get additional points for interpolation for a given segment. Default value is 3.

zmult =*val*

Convert z-values using conversion factor *val* . Default value is 1.

tension = *val*

Set tension to *val* .

NOTES

r.resamp.rst uses regularized spline with tension for interpolation (as described in Mitasova and Mitas, 1993). Region is temporarily changed while writing output files with desired resolution. Topographic parameters are computed the same way as in *s.surf.rst*. (See also Mitasova and Hofierka, 1993) Raster file *smooth* should contain variable smoothing parameters that can be derived from errors, slope, etc. using *r.mapcalc*. 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. When overshoots occur the resulting *elev* file will have white color in the locations of overshoots since the color table for the output file is the same as colortable for raster input file. 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* . 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 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. The program gives warning when the user wants to interpolate outside the region given by the header of the input raster 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 *r.resamp.rst* .

SEE ALSO

[g.region](#), [r.resample](#), [r.surf.contour](#), [s.surf.rst](#)

AUTHORS

Original version of program (in FORTRAN):

Lubos Mitas, NCSA, University of Illinois at Urbana Champaign, IL
Helena Mitasova, US Army CERL, Champaign, Illinois

Modified program (translated to C, adapted for GRASS, segmentation procedure):

Irina Kosinovsky, US Army CERL .
Dave Gerdes, US Army CERL .

REFERENCES

Mitas, L., Mitasova, H., 1999, Spatial Interpolation. In: P.Longley, M.F. Goodchild, D.J. Maguire, D.W.Rhind (Eds.), Geographical Information Systems: Principles, Techniques, Management and Applications, Wiley, 481–492.

Mitasova, H. and Mitas, L., 1993. Interpolation by regularized spline with tension: I. Theory and implementation, Mathematical Geology No.25 p.641–656.

Mitasova, H. and Hofierka, L., 1993. Interpolation by regularized spline with tension: II. Application to terrain modeling and surface geometry analysis, Mathematical Geology No.25 p.657–667.

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.

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.rescale.eq – Rescales histogram equalized the range of category values in a raster map layer.
(GRASS Raster Program)

SYNOPSIS

```
r.rescale.eq
r.rescale.eq help
r.rescale.eq [-qz] input=name [from=min,max] output=name to=min,max [title="phrase"]
```

DESCRIPTION

The *r.rescale.eq* program rescales the range of category values appearing in a raster map layer with equalized histogram. 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](#).

OPTIONS

r.rescale.eq will be run non- interactively if the user specifies program arguments on the command line, using the form:

```
r.rescale.eq [-qz] input=name [from=min,max] output=name to=min,max [title="phrase"]
```

Alternately, the user can simply type:

```
r.rescale.eq
```

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.
- z** Use zero (0) for transparency instead of NULL, that means set values outside range to zero (0) instead of NULL.

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).

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 0–255, the following command line could be used:

```
r.rescale.eq input=elevation from=1090,1800 output=elevation.255 to=0,255
```

NOTES

Category values that fall beyond the input range will become NULL. 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.eq* program to determine this range, or can obtain it using the [r.describe](#) or [r.info](#) command. If the category value range is determined using *r.rescale.eq*, the input raster map layer is examined, and the minimum and maximum non-NULL category values are selected as the input range.

SEE ALSO

[r.coin](#), [r.describe](#), [r.info](#), [r.mapcalc](#), [r.reclass](#), [r.report](#), [r.resample](#), [r.stats](#), and [parser](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 [-qz] **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](#).

OPTIONS

r.rescale will be run non-interactively if the user specifies program arguments on the command line, using the form:

r.rescale [-qz] **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.

-z

Use zero (0) for transparency instead of NULL, that means set values outside range to zero (0) instead of NULL.

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).

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 0–255, the following command line could be used:

```
r.rescale input=elevation from=1090,1800 output=elevation.255 to=0,255
```

NOTES

Category values that fall beyond the input range will become NULL. 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](#) or [r.info](#) command. If the category value range is determined using *r.rescale*, the input raster map layer is examined, and the minimum and maximum non-NULL category values are selected as the input range.

SEE ALSO

[r.coin](#), [r.describe](#), [r.info](#), [r.mapcalc](#), [r.reclass](#), [r.report](#), [r.resample](#), [r.stats](#), and [parser](#)

AUTHOR

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

Last changed: \$Date: 2002/03/01 00:08:31 \$



NAME

r.rescale.inf – Generate a raster map layer in which the categories represent values in a database column which have been divided into equal interval units.
(*GRASS-RDBMS Raster Interface Program*)

SYNOPSIS

r.rescale.inf

r.rescale.inf help

r.rescale.inf *tab=name key=name col=name cats=name input=name output=name [join=tab,tabkey,pkey]*

DESCRIPTION

r.rescale.inf creates a reclassified raster map layer by dividing the values in a numeric column in the currently selected database into equal interval units. The number of resulting categories is determined by the user via the command line parameter [*cats=*]. *r.rescale.inf* evaluates the range of values for the database column and subsets these values into equal interval groups of records returned by the query. For example, if the database column contains values which range from 1–1000 and the [*cats*] value is equal to 10 the resulting raster map layer will contain the 10 categories: 1=1–100, 2=101–200 etc. In other words, each category in the new raster map layer will represent a range of 100 values from the database column used in the rescale operation. The database column being evaluated must be numeric in type. To identify the data types of columns in a database table use the *g.column.inf* command with the [*-v*] flag. *r.rescale.inf* does not take outlying data values into account. Therefore, if the range of values for a database column contains a limited number of extreme values the resulting rescale operation will be skewed in the direction of these values.

COMMAND LINE OPTIONS

Parameters

tab=database_table_name

Table containing a column linked to category values in an existing raster map.

key=database_column_name

Column corresponding to category values in an existing raster map.

col=database_column_name

Column to base rescale operation on which is numeric in type.

cats=value

Number of categories to define in the resulting reclass map.

input=map

Name of an existing raster file with category values linked to a column in the currently selected database.

output=map

Name of new raster map.

join=tab,tabkey,pkey

Tab is the table used to develop the current SQL query. Tabkey is the database column used to relate information in this table with data in the table linked to the GRASS category file. Pkey is the associated column in the table linked to the GRASS category file which is related to tabkey in the current table.

For instance, assume that stf1_main is a table containing column values associated with category values in a the GRASS raster file blkgrp.ids. In addition, assume that stf1_main is a table containing attribute data on age in the column pop100. In this example stf1_main is the table associated with the GRASS raster map and tract_blk is the column linking stf1_main to the GRASS category file. The column pop100 in stf1_main will be the basis for the rescale effort. To specify the rescale:

```
r.rescale.inf tab=stf1_main key=tract_blk col=pop100  
cats=5 input=blkgrp.ids output=pop100.rescale
```

Specifying these conditions would insure that all rows from table stf1_main which satisfy the query criteria would be related to the spatial features in the GRASS data layer via the GRASS category values.

BUGS

None known.

NOTE

This program requires the Informix database software.

SEE ALSO

g.column.inf, g.select.inf, g.stats.inf, g.table.inf, d.rast.inf, d.site.inf, d.vect.inf, d.what.r.inf, d.what.s.inf, d.what.v.inf, r.reclass.inf, [v.reclass.inf](#)

AUTHOR

James A. Farley, Wang Song and W. Fredrick Limp University of Arkansas, CAST

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.rescale.pg – Generate a raster map layer in which the categories represent values in a database column which have been divided into equal interval units. (GRASS–RDBMS Raster Interface Program)

SYNOPSIS

r.rescale.pg

r.rescale.pg help

r.rescale.pg **tab**=name **key**=name **col**=name **cats**=name **input**=name **output**=name [**join**=tab,tabkey,pkey]

DESCRIPTION

r.rescale.pg creates a reclassed raster map layer by dividing the values in a numeric column in the currently selected database into equal interval units. The number of resulting categories is determined by the user via the command line parameter [cats=]. *r.rescale.pg* evaluates the range of values for the database column and subsets these values into equal interval groups of records returned by the query. For example, if the database column contains values which range from 1–1000 and the [cats] value is equal to 10 the resulting raster map layer will contain the 10 categories: 1=1–100, 2=101–200 etc. In other words, each category in the new raster map layer will represent a range of 100 values from the database column used in the rescale operation. The database column being evaluated must be numeric in type. To identify the data types of columns in a database table use the *g.column.pg* command with the [-v] flag. *r.rescale.pg* does not take outlying data values into account. Therefore, if the range of values for a database column contains a limited number of extreme values the resulting rescale operation will be skewed in the direction of these values.

COMMAND LINE OPTIONS

Parameters:

tab=databasetablename

Table containing a column linked to category values in an existing raster map.

key=databasecolumnname

Column corresponding to category values in an existing raster map.

col=databasecolumnname

Column to base rescale operation on which is numeric in type.

cats=value

Number of categories to define in the resulting reclass map.

input=map

GRASS Raster Commands

Name of an existing raster file with category values linked to a column in the currently selected database.

output=map

Name of new raster map

join=tab,tabkey,pkey

Tab is the table used to develop the current postQUEL query. Tabkey is the database column used to relate information in this table with data in the table linked to the GRASS category file. Pkey is the associated column in the table linked to the GRASS category file which is related to tabkey in the current table.

For instance, assume that stf1main is a table containing column values associated with category values in a the GRASS raster file blkgrp.ids. In addition, assume that stf1main is a table containing attribute data on age in the column pop100. In this example stf1main is the table associated with the GRASS raster map and tractblk is the column linking stf1main to the GRASS category file. The column pop100 in stf1main will be the basis for the rescale effort. To specify the rescale:

```
r.rescale.pg tab=stf1main key=tractblk col=pop100 cats=5 input=blkgrp.ids output=pop100.rescale
```

Specifying these conditions would insure that all rows from table stf1main which satisfy the query criteria would be related to the spatial features in the GRASS data layer via the GRASS category values.

BUGS

None known.

NOTE

This program requires the Postgres database software.

SEE ALSO

[*g.column.pg*](#), [*g.select.pg*](#), [*g.stats.pg*](#), [*g.table.pg*](#), [*d.rast.pg*](#), [*d.site.pg*](#), [*d.vect.pg*](#), [*d.what.r.pg*](#), [*d.what.s.pg*](#), [*d.what.v.pg*](#), [*r.reclass.pg*](#), [*v.reclass.pg*](#)

AUTHOR

Original Informix SQL–tools: James A. Farley, Wang Song, and W. Fredrick Limp University of Arkansas, CAST

Postgres modifications: Janne Soimasuo, Faculty of Forestry, University of Joensuu, Finland.

Updated to GRASS 5 by Alex Shevlakov (sixote@yahoo.com)



NAME

r.ros (for wildfire spread simulation) – Generates three, or four raster map layers showing 1) the base (perpendicular) rate of spread (ROS), 2) the maximum (forward) ROS, 3) the direction of the maximum ROS, and optionally 4) the maximum potential spotting distance.

(GRASS Raster Program)

SYNOPSIS

r.ros

r.ros help

r.ros [-vs] **model**=name [moisture_1h=name] [moisture_10h=name] [moisture_100h=name]
moisture_live=name [velocity=name] [direction=name] [slope=name] [aspect=name] [elevation=name]
output=name

DESCRIPTION

r.ros generates the base ROS value, maximum ROS value, direction of the maximum ROS, and optionally the maximum potential spotting distance of a wildfire for each raster cell in the current geographic region. The calculation of the two ROS values for each raster cell is based on the Fortran code by Pat Andrews (1983) of the Northern Forest Fire Laboratory, USDA Forest Service. The direction of the maximum ROS results from the vector addition of the forward ROS in wind direction and that in upslope direction. The spotting distance, if required, will be calculated by a separate function, *spot_dist()*, which is based on Lathrop and Xu (in preparation), Chase (1984) and Rothermel (1991). These three or four raster map layers serve as inputs for another GRASS raster program *r.spread*. More information on *r.ros* and *r.spread* can be found in Xu (1994).

Flags:

- v Run verbosely, printing information about program progress to standard output.
- s Calculate the maximum potential spotting distance.

Parameters:

model=name Name of an existing raster map layer in the user's current mapset search path containing the standard fuel models defined by the USDA Forest Service. Valid values are 1–13; other numbers are recognized as barriers by *r.ros*.

moisture_1h=name Name of an existing raster map layer in the user's current mapset search path containing

GRASS Raster Commands

the 1-hour (<.25") fuel moisture (percentage content multiplied by 100).

moisture_10h=name Name of an existing raster map layer in the user's current mapset search path containing the 10-hour (.25–1") fuel moisture (percentage content multiplied by 100).

moisture_100h=name Name of an existing raster map layer in the user's current mapset search path containing the 100-hour (1–3") fuel moisture (percentage content multiplied by 100).

moisture_live=name Name of an existing raster map layer in the user's current mapset search path containing live (herbaceous) fuel fuel moisture (percentage content multiplied by 100).

velocity=name Name of an existing raster map layer in the user's current mapset search path containing wind velocities at half of the average flame height (feet/minute).

direction=name Name of an existing raster map layer in the user's current mapset search path containing wind direction, clockwise from north (degree).

slope=name Name of an existing raster map layer in the user's current mapset search path containing topographic slope (degree).

aspect=name Name of an existing raster map layer in the user's current mapset search path containing topographic aspect, counter-clockwise from east (GRASS convention) (degree).

elevation=name Name of an existing raster map layer in the user's current mapset search path containing elevation (meters).

output=name Prefix of new raster map layers in the user's current mapset to contain

- 1) the base (perpendicular) ROS (cm/minute);
- 2) the maximum (forward) ROS (cm/minute),
- 3) the direction of the maximum ROS, clockwise from north (degree), and optionally
- 4) the maximum potential spotting distance (meters).

If 'my_ros' is given as the output name, then r.ros automatically assigns 'my_ros.base', 'my_ros.max', 'my_ros.maxdir', and optionally, 'my_ros.spotdist' as the names for the actual output map layers.

OPTIONS

r.ros can be run either non-interactively or interactively. The program is run interactively if the user types r.ros without specifying flag settings and parameter values on the command line. In this case, the user will be prompted for input. 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.ros [-vs] model=name [moisture_1h=name] [moisture_10h=name] [moisture_100h=name]
moisture_live=name [velocity=name] [direction=name] [slope=name] [aspect=name]
[elevation=name] output=name
```

If the options **moisture_1h**=name, **moisture_10h**=name, and **moisture_100h**=name are partially given, the program will assign values to the missing option using the formula:

$$\text{moisture_100h} = \text{moisture_10h} + 1 = \text{moisture_1h} + 2.$$

However at least one of them should be given. Options **velocity**=name and **direction**=name must be in pair, whether given or not. If none is given, the program will assume a no-wind condition. Options **slope**=name and **aspect**=name must be in pair, whether given or not. If none is given, the program will assume a topographically flat condition. Option **elevation**=name must be given if -s option is used.

EXAMPLE

Assume we have inputs, the following generates ROSEs and spotting distances:

```
r.ros -vs model=fire_model moisture_1h=1hour_moisture moisture_live=live_moisture
velocity=wind_speed direction=wind_direction slope=slope aspect=aspect elevation=elevation
output=my_ros
```

NOTES

1. r.ros is supposed to be run before running another GRASS program r.spread. The combination of the two forms a simulation of the spread of wildfires.
2. The inputs to r.ros should be in proper units.
3. The output units for the base and maximum ROSEs are in cm/minute rather than ft/minute, which is due to that a possible zero ft/minute base ROS value and a positive integer ft/minute maximum ROS would result in calculation failure in the r.spread program.
4. The user needs to provide only ONE output name even the program actually generates THREE or FOUR map layers.
5. The rules for optional parameters must be followed.

SEE ALSO

[g.region](#), [r.mask](#), [r.slope.aspect](#), [r.spread](#)

REFERENCES

- Albini**, F. A., 1976, Computer-based models of wildland fire behavior: a user's manual, USDA Forest Service, Intermountain Forest and Range Experiment Station, Ogden, Utah.
- Andrews**, P. L., 1986, BEHAVE: fire behavior prediction and fuel modeling system -- BURN subsystem, Part 1, USDA Forest Service, Intermountain Research Station, Gen. Tech. Rep. INT-194, Ogden, Utah.
- Chase**, Carolyn, H., 1984, Spotting distance from wind-driven surface fires -- extensions of equations for pocket calculators, US Forest Service, Res. Note INT-346, Ogden, Utah.
- Lathrop**, Richard G. and Jianping Xu, A geographic information system-based approach for calculating spotting distance. (in preparation)
- Rothermel**, R. E., 1972, A mathematical model for predicting fire spread in wildland fuels, USDA Forest Service, Intermountain Forest and Range Experiment Station, Res. Pap. INT-115, Ogden, Utah.
- Rothermel**, Richard, 1991, Predicting behavior and size of crown fires in the northern Rocky Mountains, US Forest Service, Res. Paper INT-438, Ogden, Utah.
- Xu**, Jianping, 1994, Simulating the spread of wildfires using a geographic information system and remote

sensing, Ph. D. Dissertation, Rutgers University, New Brunswick, New Jersey.

AUTHOR

Jianping Xu, Center for Remote Sensing and Spatial Analysis, Rutgers 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.

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.runoff (GRASS raster command) Generates an SCS curve number runoff map layer. It is a command–line interface for generating the runoff map by the SCS method.

SYNOPSIS

```
r.runoff rf=rainfall_map cn=cn_map ro=runoff_map
```

Parameters:

rf=map
rainfall map name in mm

cn=map
CN map name

ro=map
output runoff map name in *100 mm

The command–line ordering can be in any form but all key words must be there to run the program.

EXAMPLE

```
r.runoff rf=rainfall_map cn=cn_map ro=runoff_map
```

will create a runoff map runoff_map

NOTE

The *r.runoff* program is sensitive to the current window setting. Thus the program can be used to generate a runoff map of any sub–area within the full map layer. Also, *r.runoff* is sensitive to any mask in effect.

AUTHORS

Raghavan Srinivasan and Dr. Bernie A. Engel, Agricultural Engineering Department, Purdue University

Last changed: \$Date: 2002/05/31 15:56:08 \$



NAME

r.slope.aspect – Generates raster map layers of slope, aspect, curvatures and partial derivatives from a raster map layer of true elevation values. Aspect is calculated counterclockwise from east.

(GRASS Raster Program)

SYNOPSIS

r.slope.aspect

r.slope.aspect help

r.slope.aspect [-aq] **elevation=name** [**slope=name**] [**aspect=name**] [**format=name**] [**zfactor=value**]
 [**prec=name**] [**pcurv=name**] [**tcurv=name**] [**dx=name**] [**dy=name**] [**dxx=name**] [**dyy=name**] [**dxy=name**]
 [**min_slp_allowed=value**]

DESCRIPTION

r.slope.aspect generates raster maps of slope, aspect, curvatures and first and second order partial derivatives 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. 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).

OPTIONS

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] [format=name] [zfactor=value]  

  [prec=name] [pcurv=name] [tcurv=name] [dx=name] [dy=name] [dxx=name] [dyy=name]  

  [dxy=name] [min_slp_allowed=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.

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).

pcurv =name

Output profile curvature filename

tcurv =name

Output tangential curvature filename

dx=name

Output E-W slope filename

dy=name

Output N-S slope filename

dxx=name

Output partial derivative dxx filename

dyy=name

Output partial derivative dyy filename

dxy =name

Output partial derivative dxy filename

min_slp_allowed =value

Minimum slope value (in percent) for which aspect is computed. (default: 0.0)

prec=name

type of output aspect and slope maps
options: default,double,float,int

The raster *elevation* map layer specified by the user must contain true elevation values, *not* rescaled or categorized data.

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. The aspect categories represent the number degrees of east and they increase counterclockwise: 90deg is North, 180 is West, 270 is South and 360 is East.

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. Profile and tangential curvatures are the curvatures in the direction of steepest slope and in the

GRASS Raster Commands

direction of the contour tangent respectively.

For some 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.

Horn's formula is used to find the derivatives in x and y directions.

Only when using integer elevation models, 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. Because most cells with a very small slope end up having category 0, 45, ..., 360 it is sometimes possible to reduce bias in these directions by filtering out computation of aspect in areas where terrain is almost flat. 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).

When working with floating point elevation models, no such aspect bias occurs.

WARNING

(the following may not be true anymore because of introduction of NULL, but it has not been tested)

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.

REFERENCE

Horn, B. K. P. (1981). Hill Shading and the Reflectance Map, Proceedings of the IEEE, 69(1):14–47.

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

`r.spread` – Simulates elliptically anisotropic spread on a graphics window and generates a raster map of the cumulative time of spread, given raster maps containing the rates of spread (ROS), the ROS directions and the spread origins. It optionally produces raster maps to contain backlink UTM coordinates for tracing spread paths. (GRASS Raster/Display Program)

SYNOPSIS

`r.spread`

`r.spread help`

`r.spread` [-vds] **max**=name **dir**=name **base**=name **start**=name [**spot_dist**=name] [**w_speed**=name] [**f_mois**=name] [**least_size**=odds int] [**comp_dens**=decimal] [**init_time**=int (>=0)] [**lag**=int (>= 0)] [**backdrop**=name] **output**=name [**x_output**=name] [**y_output**=name]

DESCRIPTION

Spread phenomena usually show uneven movement over space. Such unevenness is due to two reasons:

- 1) the uneven conditions from location to location, which can be called SPATIAL HETEROGENEITY, and
- 2) the uneven conditions in different directions, which can be called ANISOTROPY.

The anisotropy of spread occurs when any of the determining factors have directional components. For example, wind and topography cause anisotropic spread of wildfires.

One of the simplest spatial heterogeneous and anisotropic spread is elliptical spread, in which, each local spread shape can be thought as an ellipse. In a raster setting, cell centers are foci of the spread ellipses, and the spread phenomenon moves fastest toward apogees and slowest to perigees. The sizes and shapes of spread ellipses may vary cell by cell. So the overall spread shape is commonly not an ellipse.

r.spread simulates elliptically anisotropic spread phenomena, given three raster map layers about ROS (base ROS, maximum ROS and direction of the maximum ROS) plus a raster map layer showing the starting sources. These ROS layers define unique ellipses for all cell locations in the current geographic region as if each cell center was a potential spread origin. For some wildfire spread, these ROS layers can be generated by another GRASS raster program `r.ros`. The actual locations reached by a spread event are constrained by the actual spread origins and the elapsed spread time.

r.spread optionally produces raster maps to contain backlink UTM coordinates for each raster cell of the spread time map. The spread paths can be accurately traced based on the backlink information by another GRASS raster program `r.spreadpath`.

Part of the spotting function in `r.spread` is based on Chase (1984) and Rothermel (1983). More information on *r.spread*, [r.ros](#) and [r.spreadpath](#) can be found in Xu (1994).

Flags:

- v Run verbosely, printing information about program progress to standard output.
- d Display the "live" simulation on screen. A graphics window must be opened and selected before using this option.
- s For wildfires, also consider spotting.

Parameters:

max=name Name of an existing raster map layer in the user's current mapset search path containing the maximum ROS values (cm/minute).

dir=name Name of an existing raster map layer in the user's current mapset search path containing directions of the maximum ROSes, clockwise from north (degree).

base=name Name of an existing raster map layer in the user's current mapset search path containing the ROS values in the directions perpendicular to maximum ROSes' (cm/minute). These ROSes are also the ones without the effect of directional factors.

start=name Name of an existing raster map layer in the user's current mapset search path containing starting locations of the spread phenomenon. Any positive integers in this map are recognized as starting sources.

spot_dist=name Name of an existing raster map layer in the user's current mapset search path containing the maximum potential spotting distances (meters).

w_speed=name Name of an existing raster map layer in the user's current mapset search path containing wind velocities at half of the average flame height (feet/minute).

f_mois=name Name of an existing raster map layer in the user's current mapset search path containing the 1-hour (<.25") fuel moisture (percentage content multiplied by 100).

least_size=odd int An odd integer ranging 3 – 15 indicating the basic sampling window size within which all cells will be considered to see whether they will be reached by the current spread cell. The default number is 3 which means a 3x3 window.

comp_dens=decimal A decimal number ranging 0.0 – 1.0 indicating additional sampling cells will be considered to see whether they will be reached by the current spread cell. The closer to 1.0 the decimal number is, the longer the program will run and the higher the simulation accuracy will be. The default number is 0.5.

init_time=int A non-negative number specifying the initial time for the current spread simulation (minutes). This is useful when multiple phase simulation is conducted. The default time is 0.

lag=int A non-negative integer specifying the simulating duration time lag (minutes). The default is infinite, but the program will terminate when the current geographic region/mask has been filled. It also controls the computational time, the shorter the time lag, the faster the program will run.

GRASS Raster Commands

backdrop=name Name of an existing raster map layer in the user's current mapset search path to be used as the background on which the "live" movement will be shown.

output=name Name of the new raster map layer to contain the results of the cumulative spread time needed for a phenomenon to reach each cell from the starting sources (minutes).

x_output=name Name of the new raster map layer to contain the results of backlink information in UTM easting coordinates for each cell.

y_output=name Name of the new raster map layer to contain the results of backlink information in UTM northing coordinates for each cell.

OPTIONS

The user can run `r.spread` either interactively or non- interactively. The program is run interactively if the user types `r.spread` 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.spread` non- interactively, by specifying the names of raster map layers and desired options on the command line, using the form:

```
r.spread [-vds] max=name dir=name base=name start=name [spot_dist=name] [w_speed=name]
[f_mois=name] [least_size=odds int] [comp_dens=decimal] [init_time=int (>=0)] [lag=int (>= 0)]
[backdrop=name] output=name [x_output=name] [y_output=name] The -d option can only be used after a
graphics window is opened and selected.
```

Options `spot_dist=name`, `w_speed=name` and `f_mois=name` must all be given if the `-s` option is used.

EXAMPLE

Assume we have inputs, the following simulates a spotting- involved wildfire on the graphics window and generates three raster maps to contain spread time, backlink information in UTM northing and easting coordinates:

```
r.spread -ds max=my_ros.max dir=my_ros.maxdir base=my_ros.base start=fire_origin
spot_dist=my_ros.spotdist w_speed=wind_speed f_mois=1hour_moisture backdrop=image_burned
output=my_spread x_output=my_spread.x y_output=my_spread.y
```

NOTES

1. `r.spread` is a specific implementation of the shortest path algorithm. `r.cost` GRASS program served as the starting point for the development of `r.spread`. One of the major differences between the two programs is that `r.cost` only simulates ISOTROPIC spread while `r.spread` can simulate ELLIPTICALLY ANISOTROPIC spread, including isotropic spread as a special case.

2. Before running `r.spread`, the user should prepare the ROS (base, max and direction) maps using appropriate models. For some wildfire spread, a separate GRASS program `r.ros` based on Rothermel's fire equation does such work. The combination of the two forms a simulation of wildfire spread.
3. The relationship of the start map and ROS maps should be logically correct, i.e. a starting source (a positive value in the start map) should not be located in a spread BARRIER (zero value in the ROS maps). Otherwise the program refuses to run.
4. `r.spread` uses the current geographic region settings. The output map layer will not go outside the boundaries set in the region, and will not be influenced by starting sources outside. So any change of the current region may influence the output. The recommendation is to use slightly larger region than needed. Refer to `g.region` to set an appropriate geographic region.
5. The inputs to `r.spread` should be in proper units.
6. `r.spread` is a computationally intensive program. The user may need to choose appropriate size of the geographic region and resolution.
7. A low and medium (i.e. ≤ 0.5) sampling density can improve accuracy for elliptical simulation significantly, without adding significantly extra running time. Further increasing the sample density will not gain much accuracy while requiring greatly additional running time.

SEE ALSO

[`g.region`](#), [`r.cost`](#), [`r.mask`](#), [`r.spreadpath`](#), [`r.ros`](#).

REFERENCES

- Chase, Carolyn, H., 1984, Spotting distance from wind-driven surface fires -- extensions of equations for pocket calculators, US Forest Service, Res. Note INT-346, Ogden, Utah.
- Rothermel, R. C., 1983, How to predict the spread and intensity of forest and range fires. US Forest Service, Gen. Tech. Rep. INT-143. Ogden, Utah.
- Xu, Jianping, 1994, Simulating the spread of wildfires using a geographic information system and remote sensing, Ph. D. Dissertation, Rutgers University, New Brunswick, New Jersey.

AUTHOR

Jianping Xu and Richard G. Lathrop, Jr., Center for Remote Sensing and Spatial Analysis, Rutgers 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.

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.spreadpath – Recursively traces the least cost path backwards to cells from which the cumulative cost was determined.

(GRASS Raster Program)

SYNOPSIS

r.spreadpath

r.spreadpath help

r.spreadpath [-v] **x_input**=name **y_output**=name [**coordinate**=x,y[x,y,...]]**output**=name

DESCRIPTION

r.spreadpath recursively traces the least cost path backwards to the origin, given backlink information input map layers and target locations from where paths are to be traced. The backlink information map layers record each cell's backlink UTM northing (the *y_input*) and easting (the *x_input*) coordinates from which the cell's cumulative cost was determined.

The backlink inputs can be generated from another GRASS raster program *r.spread*. One of the major applications of *r.spreadpath* along with *r.spread* is to accurately find the least cost corridors and/or paths on a raster setting. More information on *r.spread* and *r.spreadpath* can be found in Xu (1994).

Flags:

-v

Run verbosely.

Parameters:

x_input=name

Name of input raster map layer containing backlink UTM easting coordinates.

y_input=name

Name of input raster map layer containing backlink UTM northing coordinates.

coordinate=x,y[,x,y,x,y, ...]

Each x,y coordinate pair gives the easting and northing (respectively) geographic coordinates of a target point from which to backwards trace the least cost path. As many points as desired can be entered by the user.

output=name

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

OPTIONS

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

Alternately, the user can run *r.spreadpath* non- interactively, by specifying the names of raster map layers and desired options on the command line, using the form:

```
r.spreadpath [-v] x_input=name y_output=name [coordinate=x,y[x,y,...]]output=name
```

SEE ALSO

[r.spread](#), [r.ros](#).

REFERENCES

Xu, Jianping, 1994, Simulating the spread of wildfires using a geographic information system and remote sensing, Ph. D. Dissertation, Rutgers University, New Brunswick, New Jersey.

AUTHOR

Jianping Xu and Richard G. Lathrop, Jr., Center for Remote Sensing and Spatial Analysis, Rutgers 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.

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.stage3 – NEXRAD weather radar input tool.
(GRASS Raster Program)

GRASS VERSION

4.x

SYNOPSIS

r.stage3 [-v] **input=name** [**output=name**]

DESCRIPTION

A Stage III precipitation product is a mosaic of all the Stage II precipitation estimates within the RFC (River Forecast Center) area. The Stage III products are produced by merging radar precipitation estimates (Stage I) with ground truth data provided by rain gages. In the Arkansas–red River Forecast Center (ABRFC) this includes 15 88–D radars and approximately 500 rain gages. The 13 RFC's within the United States will eventually produce Stage III products. As of 4–94 the ABRFC is the only RFC currently producing Stage III products. As more radars come on line other RFC's will begin producing this product as well. The 88–D radars have difficulty detecting precipitation when the raindrops are very small or when the precipitation is in the form of snow. Under these conditions precipitation estimates tend to be under estimated. Large Hail within a storm causes the radar to over estimate precipitation amounts. The grid cell size is approximately 4 km by 4 km. The NetCDF files containing the precipitation values in the units of 1/100 of mm can be obtained from ABRFC's gopher server. *r.stage3* program generates the GRASS raster map layers only in the Lat/Lon coordinate system. UTM is not utilized because the area being referenced is too large for a UTM zone. The region settings for the ABRFC's coverage area is:

```
projection: 3 (Latitude–Longitude)
zone: 0
north: 41:16:06N
south: 32:48:28N
east: 90:53:28W
west: 107:08:03W
```

You may use the clark66 spheroid and use any resolution that may be suitable as far as your other data requirements are concerned. (start with esres= nsres = 0:00:30).

GRASS Raster Commands

This version of *r.stage3* will work only on NetCDF files created after the 6th of April 1994.

You will need the NetCDF libraries created by the Unidata Program Center. The NetCDF libraries can be obtained via anonymous ftp from unidata.ucar.edu under pub/netcdf/netcdf.tar.Z. You will need to compile the libraries. *r.stage3* can be only run within GRASS. *r.stage3* needs both the NetCDF libraries as well as the GRASS libraries. If you have all the libraries then *r.stage3* can be compiled simply by running "gmake" (For GRASS 4.2 it is gmake4.2) in this directory.

In the above, if the output parameter is not specified then a mapname with the same name as that of the input parameter is created. The rainbow colortable is created for the resultant map. The resulting maps have a resolution of 0:02:59.

A few sample NetCDF files and the corresponding GIF files are provided in the sample directory. The vector directory contains the vector boundary output from the command *v.out.ascii* of the ABRFC basin.

To run *r.stage3* on the sample files just go into the sample directory and type:

```
r.stage3 -v in=04059418.nc
```

This will create a raster map by the name 04059418.nc. Similarly *r.stage3* can be run on other sample files. You can use the corresponding GIF files to compare them. Since the GIF files are in a different projection system, there is some distortion in between the raster maps and the GIF file. However the values in the raster are properly georeferenced.

OPTIONS

Flag:

-v
Verbose

Parameters:

input = name
Name of the netcdf file to be converted.
output = name
Name of the resultant raster map.

AUTHORS

Nalneesh Gaur, Arkansas Red Basin River Forecast Center
Norman L. Bingham, Arkansas Red Basin River Forecast Center

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.statistics – Category or object oriented statistics (GRASS Raster Program)

SYNOPSIS

r.statistics **r.statistics help** **r.statistics** [-c] **base**=name **cover**=name **method**=name [**output**=name]

DESCRIPTION

r.statistics is a tool, where different class, resp. object oriented statistical analysis methods are possible. The methods are the following:

[english] – [german]

distribution – Verteilung in Prozent

average – Durchschnitt

mode – Modalwert

median – Median

average deviation – Durchschnittliche Abweichung

standard deviation – Standardabweichung

variance – Varianz

skewness – Schiefe der Verteilung

kurtosis – Relative Abflachung (-) oder Zuspitzung (+)

minimum – Minimum

maximum – Maximum

sum – Summe

COMMAND LINE OPTIONS

Flag:*-c*

Values are taken from category labels of the covering-layer, instead of covering-layer itself.

Parameters:*base*

Base raster map

cover

Cover raster map

method

Statistical method

Options: distribution, average, mode, median, avedev, stddev, variance, skewness, kurtosis, min, max, sum

output

Name of the new layer (not when using 'distribution')

The calculations will be performed for areas with data of the covering-layers which belong in the base-layer or the same category.

Setting the *-c* flag the category labels of the covering raster layer will be used. This is nice to avoid the GRASS limitation to interger in raster maps because using category values floating point numbers can be stored.

The output-layer is a reclassified version of the base-layer with identical category values. The results of the calculations are stored in the category labels of the output-layer.

The described output is valid for all calculations with exception of distribution. Here the output is given to stdout. A file name eventually specified will be ignored. The result will be a table with three columns. In the 1. column are the category values of the base-layer (a), in the 2. column the associated value of the cover-layers (b) and in the 3. column the percentage of b from a. Example:

1	124	23
1	201	47
1	273	30
2	101	5
2	152	16
2	167	60
2	187	19
	.	
	.	
	.	

REMARK

The source code of average, mode and median was taken from some GRASS raster modules.

BUGS

unkown

SEE ALSO

[*r.mode, r.median, r.average, r.neighbors*](#)

AUTHOR

Martin Schroeder, Geographisches Institut Heidelberg, Germany

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.stats – Generates area statistics for raster map layers.
(GRASS Raster Program)

SYNOPSIS

r.stats

r.stats help

r.stats [-1aclmqgxnNCri] **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 [-1aclmqgxnNCri] **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 *-l*, *-a*, *-c*, *-l*, *-m*, *-q*, *-g*, *-x*, *-n*, *-N*, *-C*, *-r*, and *-i* 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:

-l

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.
- g** Print the grid coordinates (easting and northing), for each cell. This option works only if the **-l** option is also specified.
- x** Print the x and y (column and row) values, for each cell. This option works only if the **-l** option is also specified.
- n** Suppress reporting of any nulls.
- N** Suppress reporting of nulls when all values are null.
- C** Report for cats fp ranges (fp maps only).
- r** Print raw indexes of fp ranges (fp maps only).
- i** Read fp map as integer (use map's quant rule).

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.

GRASS Raster Commands

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](#), [parser](#)

AUTHOR

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

Last changed: \$Date: 2002/10/10 21:06:38 \$



NAME

r.sum – sums up the raster cell values
(*GRASS Raster Program*)

SYNOPSIS

r.sum rast=name

Parameter:

rast
Name of incidence or density file.

AUTHOR

[Bill Brown](#), UIUC GIS Laboratory

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.sun – Computes beam (direct), diffuse and ground reflected solar irradiation raster maps for given day, latitude, surface and atmospheric conditions. Solar parameters (e.g. time of sunrise and sunset, declination, extraterrestrial irradiance, daylight length) are saved in a local text file. Alternatively, the local time can be specified to compute solar incidence angle and/or irradiance raster maps. The shadowing effect of the topography is optionally incorporated.

SYNOPSIS

r.sun

r.sun help

r.sun [-s] **elevin**=name **aspin**=name **slopein**=name [**linkein**=name] [**lin**=value] [**albedo**=name] [**alb**=value] [**latin**=name] [**lat**=value] [**coefbh**=name] [**coefd**=name] [**incidout**=name] [**beam_rad**=name] [**insol_time**=name] [**diff_rad**=name] [**refl_rad**=name] **day**=value [**step**=value] [**declin**=value] [**time**=value] [**dist**=value]

DESCRIPTION

The solar geometry of the model is based on the works of Krcho (1990), later improved by Jenco (1992). The equations describing Sun–Earth position as well as an interaction of the solar radiation with atmosphere were originally based on the formulas suggested by Kitler and Mikler (1986). This component was considerably updated by the results and suggestions of the working group co-ordinated by Scharmer and Greif (2000) (this algorithm might be replaced by SOLPOS algorithm–library included in GRASS within [r.sunmask](#) command). The model computes all three components of global radiation (beam, diffuse and reflected) for the clear sky conditions, i.e. not taking into consideration the spatial and temporal variation of clouds. The extent and spatial resolution of the modelled area, as well as integration over time, are limited only by the memory and data storage resources. The model is built to fulfil user needs in various fields of science (hydrology, climatology, ecology and environmental sciences, photovoltaics, engineering, etc.) for continental, regional up to the landscape scales.

As an option the model considers a shadowing effect of the local topography. The **r.sun** works in two modes. In the first mode it calculates for the set local time a solar incidence angle [degrees] and solar irradiance values [W.m⁻²]. In the second mode daily sums of solar radiation [Wh.m⁻².day⁻¹] are computed within a set day. By a scripting the two modes can be used separately or in a combination to provide estimates for any desired time interval. The model accounts for sky obstruction by local relief features. Several solar parameters are saved in a local text file named *r.sun_out.txt*.

The solar incidence angle raster map *incidout* is computed specifying elevation raster map *elevin*, aspect raster map *aspin*, slope steepness raster map *slopin*, given the day *day* and local time *time*. There is no need to define latitude for locations with known and defined projection/coordinate system (check it out using [g.projinfo](#) command). If you have undefined projection, (x,y) system, etc. then the latitude can be defined explicitly for large areas by input raster map *latin* with interpolated latitude values or, for smaller areas, a

single region latitude value *lat* can be used instead. All input raster maps must be floating point (FCELL) raster maps. Null data in maps are excluded from the computation (and also speeding-up the computation), so each output raster map will contain null data in cells according to all input raster maps. The user can use [r.null](#) command to create/reset null file for your input raster maps.

The specified day *day* is the number of the day of the general year where January 1 is day no.1 and December 31 is 365. Time *time* must be a local (solar) time (i.e. NOT a zone time, e.g. GMT, CET) in decimal system, e.g. 7.5 (= 7h 30m A.M.), 16.1 = 4h 6m P.M..

Setting the solar declination *declin* by user is an option to override the value computed by the internal routine for the day of the year. The value of geographical latitude can be set as a constant for the whole computed region or, as an option, a grid representing spatially distributed values over a large region. The geographical latitude must be also in decimal system with positive values for northern hemisphere and negative for southern one. In similar principle the Linke turbidity factor (*linkein*, *lin*) and ground albedo (*albedo*, *alb*) can be set.

Besides clear-sky radiations, user can compute a real-sky radiation (beam, diffuse) using *coefbh* and *coefd* input raster maps defining the fraction of the respective clear-sky radiations reduced by atmospheric factors (e.g. cloudiness). Usually these coefficients can be obtained from a long-terms meteorological measurements.

The solar irradiation or irradiance raster maps *beam_rad*, *diff_rad*, *refl_rad* are computed for a given day *day*, latitude *lat* (*latin*), elevation *elevin*, slope *slopein* and aspect *aspin* raster files. The program uses the Linke atmosphere turbidity factor and ground albedo coefficient. A default, single value of Linke factor is *lin*=3.0 and is near the annual average for rural-city areas. The Linke factor for an absolutely clear atmosphere is *lin*=1.0. See notes below to learn more about this factor. The incidence solar angle is the angle between horizon and solar beam vector. The solar radiation maps for given day are computed integrating the relevant irradiance between sunrise and sunset times for given day. The user can set finer or coarser time step *step* used for all-day radiation calculations. A default value of *step* is 0.5 hour. Larger steps (e.g. 1.0–2.0) can speed-up calculations but produce less reliable results. The output units are in Wh per squared meter per given day [Wh/(m*m)/day]. The incidence angle and irradiance/irradiation maps can be computed without shadowing influence of relief by default or they can be computed with this influence using the flag *-s*. In mountainous areas this can lead to very different results! The user should be aware that taken into account the shadowing effect of relief can slow down the speed of computing especially when the sun altitude is low. When considering shadowing effect (flag *-s*) speed and precision computing can be controlled by a parameter *dist* which defines the sampling density at which the visibility of a grid cell is computed in the direction of a path of the solar flow. It also defines the method by which the obstacle's altitude is computed. When choosing *dist* less than 1.0 (i.e. sampling points will be computed at *dist* * cellsize distance), *r.sun* takes altitude from the nearest grid point. Values above 1.0 will use the maximum altitude value found in the nearest 4 surrounding grid points. The default value *dist*=1.0 should give reasonable results for most cases (e.g. on DEM). *Dist* value defines a multiplying coefficient for sampling distance. This basic sampling distance equals to the arithmetic average of both cell sizes. The reasonable values are in the range 0.5–1.5. The values below 0.5 will decrease and values above 1.0 will increase the computing speed. Values greater than 2.0 may produce estimates with lower accuracy in highly dissected relief. The fully shadowed areas are written to the output maps as zero values. Areas with NULL data are considered as no barrier with shadowing effect .

The text file *r.sun_out.txt* is generated with the following listed parameters used in the computation:

- Solar constant 1367 W.m⁻²
- Extraterrestrial irradiance on a plane perpendicular to the solar beam [W.m⁻²]
- Day of the year
- Declination [radians]
- Decimal hour (Alternative 1 only)
- Sunrise and sunset (min–max) over a horizontal plane

- Daylight lengths
- Geographical latitude (min–max)
- Linke turbidity factor (min–max)
- Ground albedo (min–max)

The user can use a nice shellcript with variable day to compute radiation for some time interval within the year (e.g. vegetation or winter period). Elevation, aspect and slope input values should not be reclassified into coarser categories. This could lead to incorrect results.

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:

```
r.sun [-s] elevin=name aspin=name slopein=name [linkein=name] [lin=value] [albedo=name] [alb=value]
[latin=name] [lat=value] [coefbh=name] [coefdh=name] [incidout=name] [insol_time=name]
[beam_rad=name] [diff_rad=name] [refl_rad=name] day=value [step=value] [declin=value] [time=value]
[dist=value]
```

Alternately, the user can simply type *r.sun* on the command line without program arguments. In this case, the user will be prompted for parameter values using the standard GRASS parser interface. Currently, there are two modes of *r.sun*. In the first mode it calculates solar incidence angle and solar irradiance raster maps using the set local time. In the second mode daily sums of solar irradiation [kWh.m–2.day–1] are computed for a specified day.

Flag:

- s
takes into account the shadowing effect of relief

Parameters:

elevin=name

Use the existing raster file with elevationsname as input [metres].

aspin=name

Use the existing raster file with aspectname as input (aspect or azimuth of the solar panel) [decimal degrees].

slopein=name

Use the existing raster file with slopename as input (slope angle or solar panel inclination raster file) [decimal degrees].

linkein=name

Use the existing raster file with Linke atmospheric turbidity factor [–].

lin=val

Set the single value of the Linke atmospheric turbidity factor (default: 3.0).

albedo=name

Use the existing raster file with ground albedo raster file [–].

alb=val

Set the single value of the ground albedo (default: 0.2).

latin=name

Use the existing raster file with latitude raster file [decimal degrees].

lat=name

Set the single value of latitude [decimal degrees] to val.

coefbh=name

The real-sky beam radiation coefficient file.

coefdh=name

The real-sky diffuse radiation coefficient file.

incidout=name

Output solar incidence angle raster file [decimal degrees] (mode 1 only).

insol_time=name

Output time duration of the beam irradiation raster file [min.] (mode 2 only).

beam_rad=name

Output beam irradiance [W.m⁻²] (mode 1) or irradiation raster file [Wh.m⁻².day⁻¹] (mode 2).

diff_rad=name

Output diffuse irradiance [W.m⁻²] (mode 1) or irradiation raster file [Wh.m⁻².day⁻¹] (mode 2).

refl_rad=name

Output ground reflected irradiance [W.m⁻²] (mode 1) or irradiation raster file [Wh.m⁻².day⁻¹] (mode 2).

day=val

Set the number of day of the year [1–365] to val.

step=val

Set the time step when computing radiation daily sums [decimal hours] (default 0.5 hour) to val.

declin=val

Set the declination value (overriding the internally computed value) [radians] to val.

time=val

Set the decimal value of local (solar) time [decimal hours] (to be set for mode 1 only) to val.

dist=val

Set the step coefficient defining the sampling distance (0.2–1.5) in relative values of a cell size to val (default: 1.0).

NOTES

Solar energy is an important input parameter in different models concerning energy industry, landscape, vegetation, evapotranspiration, snowmelt or remote sensing. Solar rays incidence angle maps can be effectively used in radiometric and topographic corrections in mountainous and hilly terrain where very accurate investigations should be performed.

The clear-sky solar radiation model applied in the `r.sun` is based on the work undertaken for development of European Solar Radiation Atlas (Scharmer and Greif 2000, Page et al. 2001, Rigollier 2001). The clear sky model estimates the global radiation from the sum of its beam, diffuse and reflected components. The main difference between solar radiation models for inclined surfaces in Europe is the treatment of the diffuse component. In the European climate this component is often the largest source of estimation error. Taking into consideration the existing models and their limitation the European Solar Radiation Atlas team selected the Muneer (1990) model as it has a sound theoretical basis and thus more potential for later improvement.

Details of underlying equations used in this program can be found in the reference literature cited below or book published by Neteler and Mitasova: *Open Source GIS: A GRASS GIS Approach* (published in Kluwer Academic Publishers in 2002).

Average monthly values of the Linke turbidity coefficient for a mild climate (see reference literature for your study area):

Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	annual
mountains	1.5	1.6	1.8	1.9	2.0	2.3	2.3	2.3	2.1	1.8	1.6	1.5	1.90
rural	2.1	2.2	2.5	2.9	3.2	3.4	3.5	3.3	2.9	2.6	2.3	2.2	2.75
city	3.1	3.2	3.5	4.0	4.2	4.3	4.4	4.3	4.0	3.6	3.3	3.1	3.75
industrial	4.1	4.3	4.7	5.3	5.5	5.7	5.8	5.7	5.3	4.9	4.5	4.2	5.00

It should be noted that *r.sun* currently uses a lot of memory, some improvements are under way. Planned improvements may include the use of SOLPOS algorithm for solar geometry calculations and internal computation of aspect and slope.

SEE ALSO

[s.surf.rst](#), [r.slope.aspect](#), [r.sunmask](#), [g.projinfo](#), [r.null](#)

AUTHORS

Jaroslav Hofierka, GeoModel, s.r.o. Bratislava, Slovakia

Marcel Suri, GeoModel, s.r.o. Bratislava, Slovakia

© 2002, Jaroslav Hofierka, Marcel Suri hofierka@geomodel.sk suri@geomodel.sk

REFERENCES

Hofierka, J., Suri, M. (2002): The solar radiation model for Open source GIS: implementation and applications. Manuscript submitted to the International GRASS users conference in Trento, Italy, September 2002.

Hofierka, J. (1997). Direct solar radiation modelling within an open GIS environment. Proceedings of JEC-GI'97 conference in Vienna, Austria, IOS Press Amsterdam, 575–584.

Jenco, M. (1992). Distribution of direct solar radiation on georelief and its modelling by means of complex digital model of terrain (in Slovak). *Geograficky casopis*, 44, 342–355.

Kasten, F. (1996). The Linke turbidity factor based on improved values of the integral Rayleigh optical thickness. *Solar Energy*, 56 (3), 239–244.

Kasten, F., Young, A. T. (1989). Revised optical air mass tables and approximation formula. *Applied Optics*, 28, 4735–4738.

Kittler, R., Mikler, J. (1986): Basis of the utilization of solar radiation (in Slovak). VEDA, Bratislava, p. 150.

Krcho, J. (1990). Morphometric analysis and digital models of georelief. VEDA, Bratislava (in Slovak).

Muneer, T. (1990). Solar radiation model for Europe. *Building services engineering research and technology*, 11, 4, 153–163.

GRASS Raster Commands

Neteler, M., Mitasova, H. (2002): Open Source GIS: A GRASS GIS Approach, Kluwer Academic Publishers.

Page, J. ed. (1986). Prediction of solar radiation on inclined surfaces. Solar energy R&D in the European Community, series F Solar radiation data, Dordrecht (D. Reidel), 3, 71, 81–83.

Page, J., Albuissou, M., Wald, L. (2001). The European solar radiation atlas: a valuable digital tool. Solar Energy, 71, 81–83.

Rigollier, Ch., Bauer, O., Wald, L. (2000). On the clear sky model of the ESRA – European Solar radiation Atlas – with respect to the Heliosat method. Solar energy, 68, 33–48.

Scharmer, K., Greif, J., eds., (2000). The European solar radiation atlas, Vol. 2: Database and exploitation software. Paris (Les Presses de l'École des Mines).

Last changed: \$Date: 2003/08/20 08:10:13 \$



NAME

r.sunmask – calculate cast shadow areas from sun position and DEM. Either A: exact sun position is specified, or B: date/time to calculate the sun position by *r.sunmask* itself.

(GRASS Raster Program)

SYNOPSIS

```
r.sunmask [-zvs] elev=name output=name [altitude=value] [azimuth=value] [year=value]
[month=value] [day=value] [hour=value] [minute=value] [second=value] [timezone=value] [east=value]
[north=value]
```

DESCRIPTION

r.sunmask creates an output map layer based on an input elevation raster map layer and the sun position. The output map layer contains the cast shadow areas arising from sun shine and elevations. The user can define the sun position either directly or the module calculates it from given location and date/time parameters using the [NREL](#) sun position algorithm. So either "A:"-parameters to specify the exact known sun position or "B:-parameters" to specify date/time for sun position calculation by *r.sunmask* itself have to be used.

Flag:

- z
Zero is a real elevation
- v
verbose output
displays sun related information like sunset/sunrise etc.
- s
calculate sun position only and exit

Parameters:

- elev*
Elevation raster map to be used for shadow calculation
- output*
Name for the resulting binary cast shadow raster map (1=shadow)
- A: *altitude*
altitude of the sun above horizon, degrees (options: 0–89.999)
- A: *azimuth*
azimuth of the sun from the north, degrees (options: 0–360)
- B: *year*
year (1950..2050) for sun position calculation

B: month

month (0..12) for sun position calculation

B: day

day (0..31) for sun position calculation

B: hour

hour (0..12) for sun position calculation

B: minute

minutes (0..60) for sun position calculation

B: second

seconds (0..60) for sun position calculation

B: timezone

timezone (east positive, offset from GMT, also use to adjust daylight savings)

east

east coordinate (point of interest, default: map center)

north

north coordinate (point of interest, default: map center)

The module performs sunset/sunrise checks and refraction correction for sun position calculation. Local coordinate systems are internally transformed to latitude/longitude for the SOLPOS algorithm. The elevation is not considered in the sunset/sunrise calculations.

Notes

r.sunmask and daylight savings: Rather than convert time to GMT, the solpos algorithm uses what is called Local Standard Time, which is generally defined politically as an offset from GMT. So the key is the offset from GMT, which the solpos Time Zone parameter. If the user specifies clock time (different for winter and summer), he/she would have to change the Time Zone parameter seasonally in r.sunmask (timezone parameter).

Note: In latitude/longitude locations the position coordinates pair (east/west) has to be specified in decimal degree (not D:M:S). If not specified, the map center's coordinates will be used. Also *g.region -l* displays the map center's coordinates.

Acknowledgements

Acknowledgements: National Renewable Energy Laboratory for their [SOLPOS 2.0](#) sun position algorithm.

SEE ALSO

[g.region](#), [r.sun](#), [r.slope.aspect](#)

AUTHOR

Janne Soimasuo, Finland 1994
update to FP by Huidae Cho 2001
added solpos algorithm feature by Markus Neteler 2001

Last changed: \$Date: 2003/03/28 17:33:55 \$



NAME

r.support – Allows the user to create and/or modify raster map layer support files.
(GRASS Raster Program)

SYNOPSIS

r.support [-r] map=*name*

OPTIONS

Flag:

-r Update the stats (histogram, range)

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.

Most options have to be used interactively, so 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. This option is (currently) the only option which can be used on command line (*-r* flag).

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.surf.area – Surface area estimation for rasters.
(*GRASS Raster Program*)

SYNOPSIS

r.surf.area
r.surf.area help
r.surf.area input=name [vscale=value]

DESCRIPTION

r.surf.area Calculates area of regular 3D triangulated points (centers of cells) in current region by adding areas of triangles. Therefore, area of a flat surface will be reported as $(rows + cols - 1) * (area\ of\ cell)$ less than area of flat region due to a half row and half column missing around the perimeter.

NOTE: This calculation is heavily dependent on data resolution (think of it as a fractal shoreline problem, the more resolution the more detail, the more area, etc). This program uses the *CURRENT GRASS REGION*, not the resolution of the map. This is especially important for surfaces with NULL values and highly irregular edges. The program does not [currently] attempt to correct for the error introduced by this *edge effect*.

This version actually calculates area twice for each triangle pair, keeping a running minimum and maximum area depending on the direction of the diagonal used.

Reported totals are:

1. "Plan" area of NULL values within the current GRASS region
2. "Plan" area within calculation region $(rows-1 * cols-1 * cellarea)$
3. Average of the minimum and maximum calculated 3d triangle area within this region
4. "Plan" area within current GRASS region $(rows * cols * cellarea)$
5. Scaling of calculated area to current GRASS region (see [NOTE](#))

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.area input=name [vscale=value]

Alternately, the user can simply type **r.surf.area** 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](#).

Parameters:

input=name

Name of an existing raster map layer to use for the calculation.

vscale=value

Optional vertical scaling value to use in surface area calculation.

NOTES

r.surf.area works best when the surface being evaluated extends to the edges of the current GRASS region and the cell resolution is small. Surfaces which are especially long and thin and have highly irregular boundaries will tend to have underestimated surface areas. Setting a high cell resolution (small area) will greatly reduce this impact, but will cause longer processing times.

r.surf.area

SEE ALSO

[*r.surf.idw*](#)

[*r.surf.idw2*](#)

[*s.surf.idw*](#)

[*r.surf.fractal*](#)

[*r.surf.gauss*](#)

[*r.volume*](#)

[*v.to.rast*](#)

[*r.slope.aspect*](#)

[*g.region*](#)

[*parser*](#)

AUTHOR

Bill Brown, USACERL December 21, 1994

Modified for floating point rasters and NULL values by Eric G. Miller (October 17, 2000)

Last changed: \$Date: 2002/12/11 18:00:43 \$



NAME

r.surf.contour – Surface generation program from rasterized contours.
(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. So the *r.surf.contour* linearly interpolates between contour lines. 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](#).

Flags:

-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*](#)

[*parser*](#)

AUTHOR

Chuck Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.surf.fractal – GRASS module to create a fractal surface of a given fractal dimension. Uses spectral synthesis method. Can create intermediate layers showing the build up of different spectral coefficients (see Saupe, pp.106–107 for an example of this).

Use this module to generate naturally looking synthetical elevation models (DEM).

Usage:

r.surf.fractal *out=name* [*d=value*] [*n=value*]

Parameters:

out
Name of fractal surface raster layer to be produced

d
Fractal dimension of surface ($2 < D < 3$)
Default: 2.05

n
Number of intermediate images to produce
Default: 0

SEE ALSO

[r.mask](#)
[r.surf.contour](#)
[r.surf.idw](#)
[s.surf.idw](#)
[s.surf.rst](#)
[r.surf.gauss](#)
[r.surf.random](#)
[r.surf.idw2](#)
[v.surf.rst](#)
[parser](#)

REFERENCE

Saupe, D. (1988) Algorithms for random fractals, in Barnsley M., Devaney R., Mandelbrot B., Peitgen, H–O., Saupe D., and Voss R. (1988) The Science of Fractal Images, Ch. 2, pp.71–136. London: Springer–Verlag.

jwo@le.ac.uk
[ASSIST's home](#)

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.surf.gauss – GRASS module to produce a raster map layer of gaussian deviates whose mean and standard deviation can be expressed by the user. It uses a gaussian random number generator.

SYNOPSIS

r.surf.gauss out=*name* [mean=*value*] [sigma=*value*]

Parameters:

out

Name of fractal surface raster layer.

mean

Distribution mean

Default: 0.0

sigma

Standard deviation

Default: 1.0

SEE ALSO

[r.mask](#), [r.surf.contour](#), [r.surf.idw](#), [s.surf.idw](#), [s.surf.rst](#), [r.surf.fractal](#), [r.surf.random](#), [r.surf.idw2](#), [v.surf.rst](#), [parser](#)

jwo@le.ac.uk
[ASSIST's home](#)

Last changed: \$Date: 2002/01/25 05:45:34 \$



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](#)

[s.surf.idw](#)

[s.surf.rst](#)

to compare this surface generation program with others available in GRASS.

SEE ALSO

[r.mask](#)

[r.surf.contour](#)

[r.surf.idw](#)

[s.surf.idw](#)

[s.surf.rst](#)

[r.surf.gauss](#)

[r.surf.fractal](#)

[r.surf.random](#)

[r.surf.idw2](#)

[v.surf.rst](#)

[parser](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 determine 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.

OPTIONS

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.mask](#)
[r.surf.contour](#)
[r.surf.idw2](#)
[s.surf.idw](#)
[s.surf.rst](#)
[r.surf.gauss](#)
[r.surf.fractal](#)
[r.surf.random](#)
[v.surf.rst](#)
[parser](#)

AUTHOR

Greg Koerper
 Global Climate Research Project

GRASS Raster Commands

U.S. EPA Environmental Research Laboratory
200 S.W. 35th Street, JSB
Corvallis, OR 97333

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.surf.random – GRASS module to produce a raster map layer of uniform random deviates whose range can be expressed by the user. It uses the random number generator described in Press, Flannery, Teukolsky and Vetterling (1988) – Numerical Recipes in C.

SYNOPSIS

r.surf.random *out=name* [**min=value**] [**max=value**]

Parameters:

out
Name of random surface raster layer to be produced.

min
Minimum random value.
Default: 0

max
Maximum random value.
Default: 100

SEE ALSO

[r.mask](#), [r.surf.contour](#), [r.surf.idw](#), [s.surf.idw](#), [s.surf.rst](#), [r.surf.gauss](#), [r.surf.fractal](#), [r.surf.idw2](#), [v.surf.rst](#), [parser](#)

jwo@le.ac.uk
[ASSIST's home](#)

Last changed: \$Date: 2002/01/25 05:45:34 \$



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.

OPTIONS

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.

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 skeleton."

The skeleton is finally thinned when the thinning process converges; i.e., "no further pixels can be removed without altering the connectivity or shortening the skeleton legs" (p. 541). The authors prove that the thinning process described always converges and produces one-pixel thick skeletons. 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 $\leq n$ pixels thick, the algorithm should converge in $\leq n$ iterations.

SEE ALSO

[g.region](#)

[r.line](#)

[v.digit](#)

[v.support](#)

[v.trim](#)

[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).

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.timestamp print/add/remove a timestamp for a raster map
(GRASS Raster Program)

SYNOPSIS

```
r.timestamp
r.timestamp help
r.timestamp map=name [date=timestamp[/timestamp]]
```

DESCRIPTION

This command has 2 modes of operation. If no date argument is supplied, then the current timestamp for the raster map is printed. If a date argument is specified, then the timestamp for the raster map is set to the specified date(s). See EXAMPLES below.

OPTIONS

Parameters:

map=name
Name of raster map.

date=timestamp[/timestamp] or none
String specifying timestamp, range, or *none*. See FORMAT below.

NOTES

Strings containing spaces should be quoted. For specifying a range of time, the two timestamps should be separated by a forward slash. To remove the timestamp from a raster map, use **date=none**.

EXAMPLES

r.timestamp map=soils
Prints the timestamp for the "soils" raster map. If there is no timestamp for soils, nothing is printed. If there is a timestamp, one or two time strings are printed, depending on if the timestamp for the map consists of a single date or two dates (ie start and end dates).

r.timestamp map=soils date='15 sep 1987'
Sets the timestamp for "soils" to the single date "15 sep 1987"

r.timestamp map=soils date='15 sep 1987/20 feb 1988'

Sets the timestamp for "soils" to have the start date "15 sep 1987" and the end date "20 feb 1988"

r.timestamp map=soils date=none

Removes the timestamp for the "soils" raster map

TIMESTAMP FORMAT

The timestamp values must use the format as described in the GRASS datetime library. The source tree for this library should have a description of the format. For convenience, the formats as of Feb, 1996 are reproduced here:

There are two types of datetime values: absolute and relative. Absolute values specify exact dates and/or times. Relative values specify a span of time. Some examples will help clarify:

Absolute

The general format for absolute values is:

```
day month year [bc] hour:minute:seconds timezone
```

day is 1–31

month is jan,feb,....,dec

year is 4 digit year

[bc] if present, indicates dates is BC

hour is 0–23 (24 hour clock)

minute is 0–59

second is 0–59.9999 (fractions of second allowed)

timezone is +hhmm or –hhmm (eg, –0600)

parts can be missing

1994 [bc]

Jan 1994 [bc]

15 jan 1000 [bc]

15 jan 1994 [bc] 10 [+0000]

15 jan 1994 [bc] 10:00 [+0100]

15 jan 1994 [bc] 10:00:23.34 [–0500]

Relative

There are two types of relative datetime values, year– month and day–second. The formats are:

```
[–] # years # months
```

```
[–] # days # hours # minutes # seconds
```

The words years, months, days, hours, minutes, seconds are literal words, and the # are the numeric values. Examples:

2 years
5 months
2 years 5 months
100 days
15 hours 25 minutes 35.34 seconds
100 days 25 minutes
1000 hours 35.34 seconds

The following are *illegal* because it mixes year–month and day–second (because the number of days in a month or in a year vary):

3 months 15 days
3 years 10 days

SEE ALSO

[v.timestamp](#)

BUGS

Spaces in the timestamp value are required.

AUTHOR

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

Last changed: \$Date: 2003/03/16 12:25:19 \$



NAME

r.to.gnuplot – outputs a raster map in *gnuplot* format
(*GRASS Shell Script*)

SYNOPSIS

r.to.gnuplot help
r.to.gnuplot name

DESCRIPTION

r.to.gnuplot is a Bourne shell script that converts a raster map into a format suitable for plotting with *gnuplot* and writes the results to standard output.

OPTIONS

This program runs non-actively; the user must state all parameter values on the command line.

Parameter:

name
Name of a raster map layer.

EXAMPLE

Typing the following at the command line:

```
r.to.gnuplot elevation > elev.dat
```

will write the raster data to *elev.dat*. The following dialogue:

```
gnuplot  
gnuplot> set parametric  
gnuplot> set contour base  
gnuplot> set nosurface  
gnuplot> set view 180,0  
gnuplot> splot 'elev.dat' noTITLE with lines
```

will plot a contour map of *elevation*.

NOTES

Similar procedures may be used to plot wire-mesh surfaces.

gnuplot may be used to simultaneously plot surfaces and contours from multiple raster maps.

Output may be saved as PostScript, FrameMaker, TeX, etc (approximately 2 dozen output formats).

FILES

`$GISBASE/scripts/r.to.gnuplot`.

SEE ALSO

[*r.stats*](#)

[*v.to.gnuplot*](#)

[*gnuplot*](#)

AUTHOR

James Darrell McCauley, Agricultural Engineering, Purdue University

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.to.pg – exports GRASS raster map to PostgreSQL database into a three column table.
(*GRASS Raster Script*)

SYNOPSIS

r.to.pg
r.to.pg help
r.to.pg rastermap [*table_name*]

DESCRIPTION

The *r.to.pg* program exports GRASS raster map to PostgreSQL database into a three column table. A connection to PostgreSQL database is needed. The new table will be created in the currently connected database, if the table name does not exist – please check first.

Parameters:

rastermap
 Name of an existing raster map to be exported to PostgreSQL.

table_name
 Name of new table to create in currently connected PostgreSQL database (*Default is rastermap name*)

EXAMPLE

A PostgreSQL table with three columns will be created:

ID	column_name	label
1	45.2525	text
2	45.5	label text
3	44.124	text
4	46	text
...		

with:

id:	Cell ID	(int8)
column_name:	Value of rastermap cells	(float)
label:	Category label	(text)

KNOWN BUGS

The number check of imported values sometimes does not work properly because of some shell problems with 'asterix'(*).

SEE ALSO

[*db.connect*](#)

AUTHOR

Markus Neteler
category and ID support by Otto Dassau

Last changed: \$Date: 2002/07/27



NAME

r.topidx – creates topographic index, $\ln(a/\tan(\beta))$, map from elevation map
(*GRASS Raster Program*)

SYNOPSIS

r.topidx [-ov] input=*name* output=*name*

DESCRIPTION

r.topidx creates topographic index, $\ln(a/\tan(\beta))$, map from elevation map.

where

a: the area of the hillslope per unit contour length that drains through any point

$\tan(\beta)$: the local surface topographic slope (delta vertical) / (delta horizontal)

Elevation map need not to be rectangular if study basin was already determined. Non-interesting region may have NULL values. Study basin map can be taken from *r.water.outlet* then the following *r.mapcalc* statements will create study basin extracted elevation map.

```
r.mapcalc "belev = if(isnull(basin), basin, elev)"
```

```
r.avgstats prints out averaged statistics for topographic index.
```

Flags:

-o Overwrite output map
-v Output verbosely

Parameters:

input Elevation map
output Topographic index $\ln(a/\tan B)$ map

SEE ALSO

[*r.topmodel*](#), [*r.water.outlet*](#), [*r.mapcalc*](#)

AUTHORS

Main algorithm sources are rewritten from GRIDATB.FOR.
GRIDATB.FOR is available from www.es.lancs.ac.uk/hfdg/topmodel.html.
Thanks to Keith Beven.

GRASS port by [Huidae Cho](#)
Hydro Laboratory, Kyungpook National University, South Korea

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.topmodel – simulates TOPMODEL which is physically based hydrologic model
(GRASS Raster Program)

SYNOPSIS

```
r.topmodel [-iow] [elevation=name] [basin=name] [depressionless=name] [direction=name]
[belevation=name] [topidx=name] [nidxclass=value] idxstats=name parameters=name input=name
output=name [Qobs=name] [timestep=value] [idxclass=value]
```

Flags:

-i

Input data given for (i/o). Without this flag, all input data, (i) or (i/o), should be given and intermediate output can not be input for run.

In example, [belevation] map will be created from [elevation] and [basin] every run. However, every [belevation] output map is the same and this is useless process if [belevation] map is already given.

With this flag, intermediate output can be input for run. (i/o) parameters can be affected by this flag.

-o

Overwrite outputs

-w

Wide output. No limit for 80 columns output

Parameters:

Note: (i) means input; (o) means output; (i/o) means input or output

elevation

(i) Rectangular elevation map

basin

(i) Basin map created by *r.water.outlet*

depressionless

(o) Depressionless elevation map

This can be omitted if [elevation] map is already depressionless map. If this is given, [depressionless] map will be used as [elevation] map.

direction

(o) Direction map with depressionless elevation map

belevation

(i/o) Basin elevation map (extracted). Study basin extracted elevation map

topidx

- (i/o) Topographic index $\ln(a/\tan B)$ map (extracted). Study basin extracted topographic index map
- nidxclass***
(i) Number of topographic index classes
Default: 30
- idxstats***
(i/o) Topographic index statistics file
- parameters***
(i) TOPMODEL parameters file
- input***
(i) Rainfall and potential evapotranspiration data file
- output***
(o) Output file
- Qobs***
Observed Q file. Compare Q_t with Q_{obs} and calculate model efficiency
- timestep***
OPTIONAL output for given time step
- idxclass***
OPTIONAL output for given topographic index class

SEE ALSO

[*r.topidx*](#)

AUTHORS

Main algorithm sources are rewritten from TMOD9502.FOR.
TMOD9502.FOR is available from www.es.lancs.ac.uk/hfdg/topmodel.html.
Thanks to Keith Beven.

GRASS port by [Huidae Cho](#)
Hydro Laboratory, Kyungpook National University, South Korea

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.to.sites – Converts point data in a GRASS raster map layer into a GRASS site_lists file.
(GRASS Raster Program)

SYNOPSIS

```
r.to.sites
r.to.sites help
r.to.sites [-az] input=name output=name [label=name]
```

DESCRIPTION

The *r.to.sites* program extracts data from a GRASS raster map layer and stores output in a new GRASS *site_lists* file. The resulting sites map layer can be used with such programs as *d.sites*.

The user can run the program non-actively by specifying the names of an existing raster 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 *r.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*.

OPTIONS:

Flags:

```
-a      Output as decimal attribute rather than cat
-z      Output as third dimension rather than cat
```

Parameters:

```
input      Name of an existing raster map layer from which site data are to be extracted
output     Name of new sites file
label      Label for site values
             Default: No Label
```

SEE ALSO

[d.sites.parser](#)

AUTHOR

Bill Brown

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.transect – Outputs raster map layer values lying along user defined transect line(s).
(GRASS Raster Program)

SYNOPSIS

```
r.transect
r.transect help
r.transect map=name [result=type] [width=value]
line=east,north,azimuth,distance[,east,north,azimuth,distance,...]
```

DESCRIPTION

This program outputs, in ASCII, the values in a raster map which lie along one or more user-defined transect lines. The transects are described by their starting coordinates, azimuth, and distance. The transects may be single-cell wide lines, or multiple-cell wide lines. The output, for each transect, may be the values at each of the cells, or a single aggregate value (e.g., average or median value).

OPTIONS

Parameters:

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

result=type
Type of results to be output.
Options: raw, median, average
Default: raw
If raw results are output, each of the category values assigned to cells falling along the transect are output. Median and average results output a single value per transect: average outputs the average category value of all cells along the transect; median outputs the median category value of these cells.

*line=**east,north,azimuth,distance*[,*east,north,azimuth,distance*,...]
A definition of (each) transect line, specified by the geographic coordinates of its starting point (*easting*, *northing*), the angle and direction of its travel (*azimuth*), and its distance (*distance*). The *azimuth* is an angle, in degrees, measured to the east of north. The *distance* is in map units (meters for a metered database, like UTM).

width=value
Profile width, in cells (odd number).
Default: 1

Wider transects can be specified by setting the width to 3, 5, 7, etc. The transects are then formed as rectangles 3, 5, 7, etc., cells wide.

OUTPUT FORMAT

The output from this command is printed to standard output in ASCII. The format of the output varies slightly depending on the type of result requested. The first number printed is the number of cells associated with the transect. For raw output, this number is followed by the individual cell values. For average and median output, this number is followed by a single value (i.e., the average or the median value).

The following examples use the *elevation.dem* raster map layer in the *spearfish* sample data set distributed with GRASS 4.0. (To reproduce these examples, first set the geographic region as shown:

```
g.region rast=elevation.dem
```

Single-cell transect:

```
r.transect map=elevation.dem line=593655,4917280,45,100  
4 1540 1551 1557 1550
```

3-cell wide transect:

```
r.transect map=elevation.dem line=593655,4917280,45,100 width=3  
22 1556 1538 1525 1570 1555 1540 1528 1578 1565  
1551 1536 1523 1569 1557 1546 1533 1559 1550 1542  
1552 1543 1548
```

(Output appears as multiple lines here, but is really one line)

3-cell wide transect average:

```
r.transect map=elevation.dem line=593655,4917280,45,100 width=3 result=average  
22 1548.363636
```

3-cell wide transect median:

```
r.transect map=elevation.dem line=593655,4917280,45,100 width=3 result=median  
22 1549.000000
```

NOTES

This program is a front-end to the [r.profile](#) program. It simply converts the azimuth and distance to an ending coordinate and then runs [r.profile](#).

SEE ALSO

[r.profile](#)
[parser](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.tribs – A GRASS program for determining the topology of stream networks.

GRASS VERSION

4.x

SYNOPSIS

r.tribs

r.tribs *st=name ac=name dr=name*

DESCRIPTION

This paper presents a GRASS (geographical resource analysis support system; Shapiro et al., 1992; United States Army Corps of Engineers, 1993) program for determining the topology of stream networks. The program inputs raster files generated by the GRASS program *r.watershed*. Because it determines the relationships of tributary streams, it is called *r.tribs*. The input files required are:

Parameters:

st=name

A raster map of stream segments; stream segments are labeled using integer values greater than or equal to 2. Stream file from *r.watershed*.

ac=name

A raster map of the number of cells that drain through each cell; absolute values is the amount of overland flow that is routed through a pixel. Accumulation file from *r.watershed*.

dr=name

Drainage file from *r.watershed*. A raster map of drainage directions; if there is no flow direction, a value of -1 is assigned. Otherwise, the integers 1 through 8 are assigned to the compass directions shown below:

```

5 6 7
4 -1 8
3 2 1.
```

For example, a value of 2 means that the pixel drains south, a value of 5 means the pixel drains to the northwest. Two tables are output by *r.tribs*. The first table lists the tributaries associated with each stream segment. The second table lists streams and tributaries in an order such that a stream and its tributaries are printed only if the tributaries have been previously listed. That is, first order streams are listed first, streams with tributaries that are first order streams are listed second, and so on. The program can also be run in

debugging mode, in which more detailed information is printed.

The program *r.tribs* was written so that GRASS can be used to generate input for a runoff and erosion model called KINEROS (Smith et al., in press; Woolhiser et al., 1990). KINEROS represents a watershed as a set of related elements. Elements may be hillslopes, channels or ponds. A computational order must be specified so that boundary conditions for an element, such as the amount of water contributed by lateral hillslopes and upstream tributaries, are available. The program *r.tribs* provides KINEROS with that computational order. We are developing a program called *r.kineros* which will output a file that can be read by KINEROS.

Input Files

We assume that the program *r.watershed* has been run and that the input files: stream, accumulation and drainage, are available. Assume these files are called stream.0, accum.0 and drain.0, respectively. The command-line version of *r.tribs* is:

```
GRASS 4.1 > r.tribs st=stream.0 ac=accum.0 dr=drain.0
```

Alternately, the command *r.tribs* can be entered on the command line:

```
GRASS 4.1 > r.tribs
```

and the user will be prompted for these files.

Methodology and Example of Output

Output of *r.tribs* is a listing of the tributaries associated with each stream. The tributaries of a stream are determined by using the following methodology.

- a. The stream and accumulation files are scanned to determine the locations of pixels that are stream segments and have the lowest accumulation of runoff for that segment. These points mark the locations of the heads of the stream segments. Locations of the head of a stream are stored for the next step.
- b. Points within one pixel of the head are scanned to determine if they are part of a different stream segment and if they drain into the head of this stream. The drainage raster file is used to determine the drainage direction. Any point meeting this criteria are listed as tributary streams.
- c. All points with no tributaries are first order. Tributary values are set to zero.
- d. The label of the stream and its tributaries are stored and printed to the screen.

We have used *r.tribs* to generate output for a small watershed (17.5 km²) in Idaho (Horse Creek) in which we used *r.watershed* to define a total of 50 stream segments. Note that stream segments and tributaries are labeled using even integers ranging from 2 to 102. Stream segment 2 is located at the mouth of the watershed.

The *r.tribs* program first outputs the number of rows (nrows) and columns (ncols) in the raster files. Then the maximum and minimum labels of stream segments are printed. This is followed by two tables. The first table lists streams and their tributaries. An example of this table is given below.

```
nrows: 138
ncols: 273
Max stream index = 102
Min stream index = 2
Stream: 2 Tributary 0: 6 Tributary 1: 4
Stream: 4 Tributary 0: 0 Tributary 1: 0
```

GRASS Raster Commands

```
Stream: 6 Tributary 0: 14 Tributary 1: 8
Stream: 8 Tributary 0: 12 Tributary 1: 10
Stream: 10 Tributary 0: 0 Tributary 1: 0
Stream: 12 Tributary 0: 0 Tributary 1: 0
Stream: 14 Tributary 0: 22 Tributary 1: 16
Stream: 16 Tributary 0: 20 Tributary 1: 18
Stream: 18 Tributary 0: 0 Tributary 1: 0
Stream: 20 Tributary 0: 0 Tributary 1: 0
Stream: 22 Tributary 0: 102 Tributary 1: 24
Stream: 24 Tributary 0: 36 Tributary 1: 26
Stream: 26 Tributary 0: 30 Tributary 1: 28
Stream: 28 Tributary 0: 0 Tributary 1: 0
Stream: 30 Tributary 0: 34 Tributary 1: 32
Stream: 32 Tributary 0: 0 Tributary 1: 0
Stream: 34 Tributary 0: 0 Tributary 1: 0
Stream: 36 Tributary 0: 100 Tributary 1: 38
Stream: 38 Tributary 0: 42 Tributary 1: 40
Stream: 40 Tributary 0: 0 Tributary 1: 0
Stream: 42 Tributary 0: 98 Tributary 1: 44
Stream: 44 Tributary 0: 96 Tributary 1: 46
Stream: 46 Tributary 0: 50 Tributary 1: 48
Stream: 48 Tributary 0: 0 Tributary 1: 0
Stream: 50 Tributary 0: 58 Tributary 1: 52
Stream: 52 Tributary 0: 56 Tributary 1: 54
Stream: 54 Tributary 0: 0 Tributary 1: 0
Stream: 56 Tributary 0: 0 Tributary 1: 0
Stream: 58 Tributary 0: 90 Tributary 1: 60
Stream: 60 Tributary 0: 88 Tributary 1: 62
Stream: 62 Tributary 0: 66 Tributary 1: 64
Stream: 64 Tributary 0: 0 Tributary 1: 0
Stream: 66 Tributary 0: 74 Tributary 1: 68
Stream: 68 Tributary 0: 72 Tributary 1: 70
Stream: 70 Tributary 0: 0 Tributary 1: 0
Stream: 72 Tributary 0: 0 Tributary 1: 0
Stream: 74 Tributary 0: 82 Tributary 1: 76
Stream: 76 Tributary 0: 80 Tributary 1: 78
Stream: 78 Tributary 0: 0 Tributary 1: 0
Stream: 80 Tributary 0: 0 Tributary 1: 0
Stream: 82 Tributary 0: 86 Tributary 1: 84
Stream: 84 Tributary 0: 0 Tributary 1: 0
Stream: 86 Tributary 0: 0 Tributary 1: 0
Stream: 88 Tributary 0: 0 Tributary 1: 0
Stream: 90 Tributary 0: 94 Tributary 1: 92
Stream: 92 Tributary 0: 0 Tributary 1: 0
Stream: 94 Tributary 0: 0 Tributary 1: 0
Stream: 96 Tributary 0: 0 Tributary 1: 0
Stream: 98 Tributary 0: 0 Tributary 1: 0
Stream: 100 Tributary 0: 0 Tributary 1: 0
Stream: 102 Tributary 0: 0 Tributary 1: 0
```

If the program is run in debugging mode, then the values of stream, accumulation and direction of the head of the stream and the surrounding 8 pixels are also printed. Set the value of `DB_FIND_TRIBS` (defined in the beginning of the routine `find_tribs`) to a value of 1 and recompile the program to activate debugging mode.

The second table that is output by *r.tribs* list the streams and their associated tributaries in their proper computational order. The program loops through the data listed above several times. First streams with no tributaries are listed (LOOP 1). Then streams with only first-order streams as tributaries are listed (LOOP 2). Then, streams with tributaries listed in previous loops are listed. The program continues until all streams have been listed. The variable `Order` is also listed, which can be interpreted as the computational order. This is the order in which programs, such as `KINEROS`, must consider streams in the network such that data for

GRASS Raster Commands

tributaries will be available when considering the listed stream. An example listing of the computational order of streams is given below.

Computational Order of Stream Segments:

```
LOOP: 1 Order: 0 Stream: 4 Tributary 0: 0 Tributary 1: 0 Order: 1 Stream: 10 Tributary 0: 0 Tributary 1: 0
Order: 2 Stream: 12 Tributary 0: 0 Tributary 1: 0 Order: 3 Stream: 18 Tributary 0: 0 Tributary 1: 0 Order: 4
Stream: 20 Tributary 0: 0 Tributary 1: 0 Order: 5 Stream: 28 Tributary 0: 0 Tributary 1: 0 Order: 6 Stream: 32
Tributary 0: 0 Tributary 1: 0 Order: 7 Stream: 34 Tributary 0: 0 Tributary 1: 0 Order: 8 Stream: 40 Tributary
0: 0 Tributary 1: 0 Order: 9 Stream: 48 Tributary 0: 0 Tributary 1: 0 Order: 10 Stream: 54 Tributary 0: 0
Tributary 1: 0 Order: 11 Stream: 56 Tributary 0: 0 Tributary 1: 0 Order: 12 Stream: 64 Tributary 0: 0
Tributary 1: 0 Order: 13 Stream: 70 Tributary 0: 0 Tributary 1: 0 Order: 14 Stream: 72 Tributary 0: 0
Tributary 1: 0 Order: 15 Stream: 78 Tributary 0: 0 Tributary 1: 0 Order: 16 Stream: 80 Tributary 0: 0
Tributary 1: 0 Order: 17 Stream: 84 Tributary 0: 0 Tributary 1: 0 Order: 18 Stream: 86 Tributary 0: 0
Tributary 1: 0 Order: 19 Stream: 88 Tributary 0: 0 Tributary 1: 0 Order: 20 Stream: 92 Tributary 0: 0
Tributary 1: 0 Order: 21 Stream: 94 Tributary 0: 0 Tributary 1: 0 Order: 22 Stream: 96 Tributary 0: 0
Tributary 1: 0 Order: 23 Stream: 98 Tributary 0: 0 Tributary 1: 0 Order: 24 Stream: 100 Tributary 0: 0
Tributary 1: 0 Order: 25 Stream: 102 Tributary 0: 0 Tributary 1: 0
----- LOOP: 2 Order:
26 Stream: 8 Tributary 0: 12 Tributary 1: 10 Order: 27 Stream: 16 Tributary 0: 20 Tributary 1: 18 Order: 28
Stream: 30 Tributary 0: 34 Tributary 1: 32 Order: 29 Stream: 52 Tributary 0: 56 Tributary 1: 54 Order: 30
Stream: 68 Tributary 0: 72 Tributary 1: 70 Order: 31 Stream: 76 Tributary 0: 80 Tributary 1: 78 Order: 32
Stream: 82 Tributary 0: 86 Tributary 1: 84 Order: 33 Stream: 90 Tributary 0: 94 Tributary 1: 92
----- LOOP: 3 Order:
34 Stream: 26 Tributary 0: 30 Tributary 1: 28 Order: 35 Stream: 74 Tributary 0: 82 Tributary 1: 76
----- LOOP: 4 Order:
36 Stream: 66 Tributary 0: 74 Tributary 1: 68
----- LOOP: 5 Order:
37 Stream: 62 Tributary 0: 66 Tributary 1: 64
----- LOOP: 6 Order:
38 Stream: 60 Tributary 0: 88 Tributary 1: 62
----- LOOP: 7 Order:
39 Stream: 58 Tributary 0: 90 Tributary 1: 60
----- LOOP: 8 Order:
40 Stream: 50 Tributary 0: 58 Tributary 1: 52
----- LOOP: 9 Order:
41 Stream: 46 Tributary 0: 50 Tributary 1: 48
----- LOOP: 10 Order:
42 Stream: 44 Tributary 0: 96 Tributary 1: 46
----- LOOP: 11 Order:
43 Stream: 42 Tributary 0: 98 Tributary 1: 44
----- LOOP: 12 Order:
44 Stream: 38 Tributary 0: 42 Tributary 1: 40
----- LOOP: 13 Order:
45 Stream: 36 Tributary 0: 100 Tributary 1: 38
----- LOOP: 14 Order:
46 Stream: 24 Tributary 0: 36 Tributary 1: 26
----- LOOP: 15 Order:
47 Stream: 22 Tributary 0: 102 Tributary 1: 24
----- LOOP: 16 Order:
48 Stream: 14 Tributary 0: 22 Tributary 1: 16
```

----- LOOP: 17 Order:
49 Stream: 6 Tributary 0: 14 Tributary 1: 8

----- LOOP: 18 Order:
50 Stream: 2 Tributary 0: 6 Tributary 1: 4

Obtaining *r.tribs* via FTP or email

The C programs that are required to generate *r.tribs* are available via anonymous ftp to moon.cecer.army.mil. I have printed out the main segment of the *r.tribs* program in Appendix I. This was done to illustrate how routines in the GRASS library are used to read in raster data. Routines from the GRASS library begin with G_. Two utility programs are also used: *imatrix* and *ivector*. These are discussed by Press and others (1989), and are used to allocate space for arrays. Comments in the code discuss the details of each routine. Appendix II lists the makefile used to compile *r.tribs*. Note that the variable GIS must be edited so that the proper path to the grass directory is specified.

As previously mentioned, the program can be obtained by anonymous ftp to moon.cecer.army.mil. Change to the incoming/r.tribs directly to get the files. The program can also be obtained by contacting the author via email (jfs5@po.cwru.edu). The code has been commented to help the user to understand the program structure. Also, a developmental version of *r.kineros* is in the incoming/r.kineros directory on moon.cecer.army.mil.

REFERENCES

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., 1989, Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, 735 pp.

Shapiro, M., Westervelt, J., Gerdes, D., Larson, M., and Brownfield, K.R., 1992, GRASS 4.0 Programmers Manual, U.S. Army Construction Engineering Research Laboratory, Champaign, Illinois, 292 pp.

Smith, R.E., Goodrich, D.C., Woolhiser, D.A., and Unkrich, C.L., in press, A KINematic Runoff and EROsion Model, in: V.P. Singh (Ed.), Computer Models of Watershed Hydrology, Water Resources Pub., Highlands Ranch, Colorado

United States Army Corps of Engineers, 1993, GRASS4.1 Users Reference Manual, U.S. Army Construction Engineering Research Laboratories, Champaign, Illinois, 556 pp.

Woolhiser, D.A., Smith, R.E., Goodrich, D.C., 1990, KINEROS, A Kinematic Runoff and Erosion Model: Documentation and Users Manual, U.S. Department of Agriculture, Agricultural Research Service, ARS-77, 130 pp.

Appendix I: main.c code for r.tribs

```
#include "/GRASS4.1/src/include/gis.h"
#include <stdio.h>
int **imatrix();
int *ivector();
/*
 * Program to determining the topology of a stream network. A
 * table is generated that reports the tributaries that are
 * at the head of each stream segment. The computational order
 * of streams is also determined.
 *
 * Written by:
```

GRASS Raster Commands

```
* Dr. John F. Stamm
* Department of Geological Sciences
* Case Western Reserve University
* Cleveland, OH 44106-7216
* email: jfs5@po.cwru.edu
*/
main(argc,argv)
int argc;
char *argv[];
{
/*
* Matricies
*/
int **accum;
int **chann;
int **aspect;
CELL *cell;
char *chann_name;
char *accum_name;
char *aspect_name;
char *mapset;
int col;
int fd_accum;
int fd_chann;
int fd_aspect;
int ncols;
int nrows;
int row;
struct {
struct Option *accum ;
struct Option *chann ;
struct Option *aspect ;
} parm;
/*
* Allocate memory for the Option structure and return a
* pointer to this structure. Do this for the structured
* variables parm.accum and parm.chann.
*
* Set values for parm.accum
*/
parm.accum = G_define_option() ;
parm.accum->key = "accumulation";
parm.accum->type = TYPE_STRING;
parm.accum->required = YES;
parm.accum->gisprompt = "old,cell,raster" ;
parm.accum->description= "Name of the ACCUMULATION map" ;
/*
* Set values for parm.chann
*/
parm.chann = G_define_option() ;
parm.chann->key = "stream";
parm.chann->type = TYPE_STRING;
parm.chann->required = YES;
parm.chann->gisprompt = "old,cell,raster" ;
parm.chann->description= "Name of the STREAM map" ;
/*
* Set values for parm.aspect
*/
parm.aspect = G_define_option() ;
parm.aspect->key = "drainage";
parm.aspect->type = TYPE_STRING;
parm.aspect->required = YES;
```


GRASS Raster Commands

```
parm.aspect->gisprompt = "old,cell,raster" ;
parm.aspect->description= "Name of the DRAINAGE DIRECTION map";
/*
 * Initailize GIS library for this program
 */
G_gisinit(argv[0]);
/*
 * Parse values from the command line. If this is not
 * successful, then display a usage statement and exit.
 */
if (G_parser(argc, argv))
exit (-1);
accum_name = parm.accum->answer;
chann_name = parm.chann->answer;
aspect_name = parm.aspect->answer;
/*
 * Find the name of mapset that we are going to use.
 */
mapset = G_find_cell2 (accum_name, "");
if (mapset == NULL) {
char msg[100];
sprintf (msg, "%s: <%s> cellfile not found\n",
G_program_name(), accum_name);
G_fatal_error (msg);
exit(1);
}
/*
 * Open the cell files in "mapset".
 */
fd_accum = G_open_cell_old (accum_name, mapset);
if (fd_accum < 0)
exit(1);
fd_chann = G_open_cell_old (chann_name, mapset);
if (fd_chann < 0)
exit(1);
fd_aspect = G_open_cell_old (aspect_name, mapset);
if (fd_aspect < 0)
exit(1);
/*
 * Open up a vector that is just long enough to hold one
 * row of data.
 */
cell = G_allocate_cell_buf();
/*
 * Determine the number of rows and columns.
 */
nrows = G_window_rows();
ncols = G_window_cols();
printf ("\n", nrows);
printf ("nrows: %d\n", nrows);
printf ("ncols: %d\n", ncols);
/*
 * Allocate memory for matrices.
 */
accum = imatrix(0,nrows,0,ncols);
chann = imatrix(0,nrows,0,ncols);
aspect = imatrix(0,nrows,0,ncols);
/*
 * Process DEM and Channel files.
 */
for (row=(nrows-1); row>=0; row--) {
if(G_get_map_row (fd_accum, cell, row) < 0)
```

```

exit(1);
for (col = 0; col < ncols; col++) {
accum[row][col] = (int)cell[col];
}
if(G_get_map_row (fd_chann, cell, row) < 0)
exit(1);
for (col = 0; col < ncols; col++) {
chann[row][col] = (int)cell[col];
}
if(G_get_map_row (fd_aspect, cell, row) < 0)
exit(1);
for (col = 0; col < ncols; col++) {
aspect[row][col] = (int)cell[col];
}
}
}
/*
* Compute the topology of the network.
*/
(void)find_tribs(nrows, ncols, accum, chann, aspect);
exit(0);
}

```

Appendix II: makefile

```

PGM = r.tribs
GIS = /GRASS4.1
HOME = .
SRC = $(GIS)/src
LIBDIR = $(SRC)/libes/LIB
GISLIB = $(LIBDIR)/libgis.a
OFILES = debug.o \
find_tribs.o \
imatrix.o \
ivector.o \
main.o \
neighbors.o \
stream_order.o
$(HOME)/$(PGM): $(OFILES) $(GISLIB)
$(CC) $(LDFLAGS) $(OFILES) $(GISLIB) -o $(PGM)
$(GISLIB): #

```

AUTHOR

John F. Stamm, Department of Geological Sciences, Case Western Reserve University, Cleveland, OH 44106-7216, email: jfs5@po.cwru.edu

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.univar – Univariate statistics for a GRASS raster map.
(*GRASS Script*)

SYNOPSIS

r.univar
r.univar help
r.univar name

DESCRIPTION

r.univar calculates univariate statistics of a raster map. This includes the number of cells counted, minimum and maximum cell values, range, arithmetic mean, variance, standard deviation and coefficient of variation.

Parameters:

name
Name of an existing raster map.

SEE ALSO

[*s.univar*](#)

AUTHOR

Markus Neteler, University of Hannover
neteler@itc.it

Last changed: \$Date: 2002/01/25 05:45:34 \$



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. The centroid easting of each clump.
6. The centroid northing of each clump. These values are output under the "Centroid Easting" and "Centroid Northing" columns (fields 5 and 6) in the output.
7. The total "volume" of each clump. For each clump, the volume is calculated by multiplying the area of each cell by its category value, and taking the sum of this value for all cells within the clump. Since, in GRASS, each cell in the *data* map will have the same cell dimensions (i.e., the same area), this is equivalent to multiplying the area of one cell by the "Data Total" column (field 3). (The area of each cell is equal to the product of its east-west resolution by its north-south resolution. See [g.region](#).)

Results are sent to standard output in the form of a table. If the user sets the **-f** flag, this table will be output in a form suitable for input to such UNIX programs as *awk* and *sed*; the table's columns are stored as colon-separated fields. The user can also (optionally) elect to store clump centroids in a GRASS *site_lists* file. A sample output report is shown below.

r.volume works with the current geographic region definitions and respects the current MASK.

The user can run *r.volume* non-interactively by specifying parameter values on the command line. If the user omits parameter values from the command line, the program will prompt the user for input using the standard interface described in the manual entry for [parser](#).

OPTIONS

Flags:

- f** Generate unformatted output. Output is in a form suitable for input to UNIX programs like *awk*; each column in the output is separated by a colon. Results are sent to standard output.
- q** Run quietly, suppressing the printing of debugging messages to standard output.

Parameters:

data=name

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

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=name

The name to be assigned to a new GRASS **site_list** file, in which clump centroids can be stored.

EXAMPLE OF REPORT

The following report might be generated by the command:

```
r.volume d=elevation c=fields.only s=field.centers
```

Cat #	Average in clump	Data Total	# Cells in clump	Centroid		Total Volume
				Easting	Northing	
1	1181.09	75590	64	595500.0	4927700.0	755900000.0
2	1163.50	69810	60	597100.0	4927700.0	698100000.0
3	1146.83	34405	30	598300.0	4927700.0	344050000.0
4	1193.20	366311	307	599400.0	4927300.0	3663110000.0
.....						
60	1260.08	351563	279	603100.0	4921000.0	3515630000.0
61	1213.93	35204	29	603700.0	4921500.0	352040000.0
62	1207.71	33816	28	604100.0	4921500.0	338160000.0
Total Volume =						67226740000.0

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:

```
eastings|northings|#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](#)
[parser](#)

AUTHOR

Dr. James Hinthorne, GIS Laboratory, Central Washington University

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.water.fea – Finite element analysis program for hydrologic simulations.

(GRASS Raster Program)

SYNOPSIS

r.water.fea

OVERVIEW

r.water.fea is an interactive program that allows the user to simulate storm water runoff analysis using the finite element numerical technique. Infiltration is calculated using the Green and Ampt formulation.

r.water.fea computes and draws hydrographs for every basin as well as at stream junctions in an analysis area. It also draws animation maps at the basin level.

DATA REQUIREMENTS

The maps required by **r.water.fea** are:

- 1) Basin map
- 2) Stream map
- 3) Drainage map
- 4) Accumulation map
- 5) Slope map

The other data requirements of **r.water.fea** are the parameters needed to calculate infiltration and the channel roughness parameter. Model parameters may be provided either in the form of maps or as values:

- 1) Manning roughness coefficient map or basin value
- 2) Saturated hydraulic conductivity map or basin value
- 3) Suction head at wetting front map or basin value
- 4) Effective porosity map or basin value
- 5) Degree of saturation basin value

DESCRIPTION

On running **r.water.fea** for the first time, the directory "*r.water.fea*" is created under \$LOCATION. When the user runs **r.water.fea**, the program will prompt the user for the project name. The project name refers to the directory that is created under the "*r.water.fea*" directory. All files (not maps) related to the analysis carried out by **r.water.fea** are stored under this directory. If the project does not exist then the user is further

requested for the input maps. If the project already exists, then the program looks for the proper project related files to proceed with stopped work.

Configuration

The user is asked for the following configuration modes: I) Rainfall mode: The rainfall mode is defined as follows:

- 1 = spatially uniform and constant in time
- 2 = spatially uniform but varying in time

If the user decides to use mode 2, then a mechanism is provided to allow creation of a rainfall rate file (described in step 4).

II) Basin-level hydrographs: This configuration mode allows the user to view intermediate hydrographs for every basin. The hydrographs will be displayed on the graphics monitor. Each basin is considered to be independent of every other basin in the analysis area.

III) Basin-level animation maps: This configuration mode allows the user to create time-series maps for later animation of flow depth for all the basins that have been analyzed. All animation maps use a multiplication factor of 1000. The map cell value divided by the multiplication factor yields the actual value of flow depth in meters. The user will require enough file space in the GRASS database for this configuration. All animation map names have the following naming scheme:

fea.<project_name>.#

Here '#' represents the time step. One time step refers to 1/40th of the total monitoring time. Twenty maps are created at every other time step.

Program flow

The basin and stream maps are displayed on the monitor when the user starts working on a project. The entire analysis is divided into a number of steps. The user is presented with a menu to proceed through the set of steps to facilitate easy changes to the simulation for a given analysis area or to stop analysis between menu steps and continue at a more convenient time.

The main menu is shown below:

Choose from the menu:

- * 1. Process steps without breaks.
- *-> 2. Select basins for simulation.
- X 3. Extract topographical data.
- X 4. Select hydraulic parameters and simulation time.
- X 5. Basin simulation.
- X 6. Simulate any particular basin.
- X 7. Channel routing of basin hydrographs.
- * 8. Stop.

You are starting from the beginning.

Choice:

The "X", "*", and "*->" above have the following meaning.

* 8

The user can select this part from the menu.

*-> 8

This is the step that the user must select in order to proceed in a sequence.

X 8

The user can not select these parts of the menu (until previous steps have been executed).

Throughout the program the symbols described above change as the user moves from step to step. The message just above the Choice prompt signals the status of the program, and guides the user as to what should be the next step.

"1. Process steps without breaks." 8

This part of the menu will not prompt for steps 2–5, 7 and will carry out all the analysis. The user will find it advantageous to use this step when analyzing a small area or a few basins.

"2. Select basins for simulation." 8

This part of the menu draws the basin and stream maps for the user to select the area of analysis. The mouse is then activated to provide a point and click environment for the user to select basins in the area of analysis. Upon successfully selecting the basins the user is given the choice of deleting basins from the selected area. The basin topology is then determined and information on basin statistics is gathered. The information on connectivity between basins is stored in the *"input.basin"* file and the information on basin statistics is stored in the *"basin_info"* file. Two reclass maps describing the analysis area are also created. The maps have the following naming scheme:

"fea.stream.<project_name>"

"fea.basin.<project_name>"

The user should avoid using these names to create other maps. Once these maps are created the user should not destroy them, if the user wishes to continue working on the project.

"3. Extract topographical data." 8

This part of the menu generates information about the connectivity between cells and boundary conditions. This information is stored in the *"input.file"* file in the project directory. If the animation configuration has been set then another file called *"coordinates"* is created in the project directory. This file contains information on the coordinates of every cell in the analysis area.

"4. Select hydraulic parameters and simulation time." 8

This part of the menu carries out two tasks. The first task involves querying the user for simulation parameters. The simulation parameters include duration of rainfall, maximum intensity, time step for simulation, monitoring time and names of simulation maps if any. The second task involves querying channel characteristics assuming the channels as trapezoidal cross-sections. The user is required to provide channel side slope, channel bottom width and the channel roughness coefficient values for every stream category.

The program creates a file *"timedata"* in the project directory to store information from this part of the menu. If the user has selected the "Spatially uniform and time varying rainfall" mode (mode = 2), then the user is queried for the the name of a time file. If the file does not exist then a screen like the one shown below appears:

GRASS Raster Commands

.if t .ne 15

```
center;  
l l l.
```

Rainfall data

[The time column must be filled in increasing order.]

	Time[minutes]	Intensity[cm/hr]
1	_____	_____
2	_____	_____
3	_____	_____
4	_____	_____
5	_____	_____
6	_____	_____
7	_____	_____
8	_____	_____
9	_____	_____
10	_____	_____
11	_____	_____
12	_____	_____
13	_____	_____
14	_____	_____
15	_____	_____

.T&

c s s s

c s s s

l l l l.

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

It is important to note that the values in the time column should be in an increasing order. It is also not necessary to fill all the rows and the user can stop after filling only a few rows. The number of lines are limited to fifteen. If more than fifteen lines are required then the user will have to create the file using an editor. In that case the user should just type the time since commencement of rainfall (minutes) in the first column followed by the rainfall intensity (cm/hr) in the second column as shown below:

```
center;  
l l  
l c.  
10      2.54  
30      4.52  
60      5.62
```

"5. Basin simulation." 8

This part of the menu carries out the simulation for each basin in the area of analysis. Every basin is analyzed as independent of every other basin. The user is shown the independent basin hydrograph for every basin on the graphics monitor. The file "*disch.basin*" is created towards the end of simulation of all the basins. This file contains columns of discharge for each basin. The column number corresponds to the basin category value in the legend. If the animation configuration mode was set

then a file is created in the project directory called "*disch_file*". This file holds basin discharge values at every point in every basin of the analysis area.

"6. Simulate any particular basin." 8

This part of the menu provides the facility of changing basin characteristics such as the Manning roughness coefficient and other infiltration parameters. This choice can be used only after the successful completion of choice 5. If the user has provided maps for the parameters then the user should select "stop" from the menu to make changes to the parameter maps provided to the model.

"7. Channel routing of basin hydrographs." 8

This part of the menu connects the basins that were considered independent in the previous step. Routing of the basin-level hydrographs is done based on the connectivity between basins. This can generate hydrographs only for seven stream junctions. If there are more than seven stream junctions then the first seven stream junction hydrographs are shown. The process of drawing individual basin animation maps follows the drawing of the hydrographs at stream junctions. After completing this choice a file "*disch.junction*" is created. This file contains discharge values at different steps for every stream junction.

"8. Stop." 8

This part of the menu is available to the user at any time between the different choices described in the menu and exits the user out of the program.

LIMITATIONS

- 1.) Negative values of drainage direction inside basins maps cannot be accepted. Negative values are generated as a result of incomplete information regarding the basin drainage pattern (e.g., **r.watershed** produces negative values as a result of outflowing drainage basins).
- 2.) The drainage map should route the water and not form pits, lakes, or ponds. Note that this does not imply that the DEM by itself should not have any pits.
- 3) Interstorm modeling, interflow, or baseflow are not considered.
- 4) Backwater effects are not considered.
- 5) The kinematic wave analogy is appropriate where the land surface slope and channel slope are large. This may not be true in flat, marshy terrain and in slow, meandering river channels.

OUTPUT FILES OF INTEREST

These are ASCII files that can be found in the "\$LOCATION/r.water.fea/<project_name>" directory. The files have a format such that it can be imported to various analysis packages.

disch.basin 8

This file contains multiple columns which contain the individual basin discharge values in order of first column containing the discharge values for basin one and the second for basin two and so on.

disch.junction 8

This file contains the results of the discharge values at stream junctions specified by the icons. The first column in this file shows the time step in minutes. The remaining columns specify the discharge values. The

first row specifies the stream junction icon numbers.

NOTES

1. **r.water.fea** alters the region in the *WIND* file. This is done by making a systems call to:

```
g.region align=name
```

just when the program **r.water.fea** is run.

2. A small watershed can be analyzed by providing values of model parameters. However it is advisable to provide maps of various model parameters if there are many basins in the watershed. In the former case the value provided by the user shall be constant for the entire basin for which the value is queried. The user shall create the infiltration maps using the following set of rules. The map value divided by the multiplication factor yields the actual value in the described units:

```
center tab(:);
ll
lc.
Parameter:Multiplication factor
```

Soil saturated hydraulic conductivity map (meters/sec):10,000,000

Manning roughness coefficient:1000

Soil suction at wetting front (m):1000

Soil porosity map (m³/m³):1000

3. If the user has provided a slope map that has zero slope value anywhere in the map, then a slope value of 5% is assumed for that cell.

4. It is preferred that none of the basins in the analysis area has more than 750 cells, as this will increase the computation time drastically. The number of cells in a basin can be controlled by setting the threshold value to small values when running **r.watershed**.

5. Using larger cells can speed up the analysis process significantly. It is important to note that **r.watershed** should be run for the resolution at which the user desires to run the **r.water.fea** analysis.

FILES

All the files listed below are ASCII files. None of these files should be deleted if the user wishes to continue working on the same project.

\$LOCATION/r.water.fea/<project_name>/input.basin 8

This file contains information on connectivity between different basins in the analysis area.

\$LOCATION/r.water.fea/<project_name>/input.file 8

This file contains information on connectivity between different cells for every basin in the analysis area.

\$LOCATION/r.water.fea/<project_name>/disch.basin 8

This file contains discharge values for every basin in the analysis area.

\$LOCATION/r.water.fea/<project_name>/disch.junction 8

This file contains discharge values at every stream junction in the analysis area.

\$LOCATION/r.water.fea/<project_name>/timedata 8

This file contains the response queried from the user in choice 4 of the menu.

\$LOCATION/r.water.fea/<project_name>/basin_info 8

This file contains information on statistics of the basins in the analysis area.

\$LOCATION/r.water.fea/<project_name>/coordinates 8

This file contains information on coordinates of every cell in the analysis area.

\$LOCATION/r.water.fea/<project_name>/control 8

This file contains information on map names, configuration modes, and stopped choice in the menu.

\$LOCATION/r.water.fea/<project_name>/disch_file 8

This file contains the discharge values at every point in the analysis area.

\$LOCATION/r.water.fea/<project_name>/timefiles/<file_name> 8

This file contains the spatially constant and time variant rainfall mode file.

REFERENCES

Finite element methodology used by *r.water.fea* is described in Vieux, B. E., Bralts, V. F., Segerlind, L. J., Wallace, R. B., (1990), "FINITE ELEMENT WATERSHED MODELING: ONE-DIMENSIONAL ELEMENTS", *J. of Water Resources Planning and Management*, Vol. 116, No. 6, p803–819.

SEE ALSO

[*r.slope.aspect*](#)

[*r.watershed*](#)

[*r.mask*](#)

[*r.reclass*](#)

[*r.stats*](#)

[*r.colors*](#)

AUTHOR

Baxter E. Vieux, University of Oklahoma, Norman

Nalneesh Gaur, University of Oklahoma, Norman

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.water.outlet – Watershed basin creation program.
(GRASS Raster Program)

SYNOPSIS

r.water.outlet
r.water.outlet help
r.water.outlet drainage=name basin=name easting=value northing=value

DESCRIPTION

r.water.outlet generates a watershed basin from a drainage direction map (from [r.watershed](#)) and a set of coordinates representing the outlet point of watershed.

OPTIONS

Parameters

drainage=name

Input map: drainage direction. Indicates the "aspect" for each cell. Multiplying positive values by 45 will give the direction in degrees that the surface runoff will travel from that cell. The value -1 indicates that the cell is a depression area. Other negative values indicate that surface runoff is leaving the boundaries of the current geographic region. The absolute value of these negative cells indicates the direction of flow. This map is generated from [r.watershed](#).

basin=name

Output map: Values of one (1) indicate the watershed basin. Values of zero are not in the watershed basin.

easting=value

Input value: Easting value of outlet point.

northing=value

Input value: Northing value of outlet point.

NOTES

In the context of this program, a watershed basin is the region upstream of an outlet point. Thus, if the user chooses an outlet point on a hill slope, the resulting map will be a thin silver of land representing the overland slope uphill of the point.

SEE ALSO

[d.where](#), [r.watershed](#), [r.topidx](#)

AUTHOR

Charles Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 can prepare inputs to lumped–parameter hydrologic models. After a verbose interactive session, 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 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 's case, the first two letters of each name sufficiently distinguishes parameter names. For example, the two expressions below are equivalent inputs to :

el=elev.map **th**=100 **st**=stream.map **ba**=basin.map

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

GRASS Raster Commands

can be limiting, but is relatively fast. Setting the flag causes the program to manage memory on disk which allows larger maps to be processed 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. The algorithm is D8 flowaccumulation.

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, 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

GRASS Raster Commands

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 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\ut\d 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 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 will be 90 meters: use the minimum function for a 3 by 3 neighborhood. After going to the resolution at which will be run, 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 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\|d least-cost search algorithm used by is described in *Using the A\|T\|d 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.

Reference: "Using the AT Search Algorithm to Develop Hydrologic Models from Digital Elevation Data," International Geographic Information System Symposium, Baltimore MD, 1989

<http://geography.hunter.cuny.edu/~chuck/IGIS/paper.html>

[r.cost](#), [r.drain](#), [r.flow](#), [r.flowmd](#), [r.mask](#), [r.param.scale](#), [r.topidx](#), [r.water.outlet](#)

AUTHOR

Charles Ehlschlaeger, U.S. Army Construction Engineering Research Laboratory

Last changed: \$Date: 2003/02/09 12:07:23 \$



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
```

GRASS Raster Commands

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

r.weighted.cn

Generates a weighted SCS curve number map layer
(*GRASS Raster Program*)

SYNOPSIS

r.weighted.cn **input**=cn_map **output**=weighted_cn_map

Parameters:

input=map

curve number map name

output=map

weighted curve number map name

The command–line ordering can be in any form but all key words must be there to run the program.

NOTE

The *r.weighted.cn* program is sensitive to the current window settings. Thus the program can be used to generate a weighted CN map of any sub–area within the full map layer. Also, *r.weighted.cn* is sensitive to any mask in effect.

SEE ALSO

[*r.cn*](#), [*r.cn2*](#)

AUTHORS

Raghavan Srinivasan and Dr. Bernie A. Engel,
Agricultural Engineering Department, Purdue University

Last changed: \$Date: 2002/01/25 05:45:34 \$



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

<i>recover</i>	Recover old analysis
<i>add</i>	Request that weights be added (this is the default)
<i>mult</i>	Request that weights be multiplied
<i>execute</i>	Run analysis
<i>erase</i>	Erase the screen
<i>color grey</i>	Set the graphics montior colors to a grey scale (this is the default)
<i>color wave</i>	Set the graphics monitor colors to a color wave.
<i>color ramp</i>	Set the graphics monitor colors to a color ramp.
<i>quit</i>	Leave <i>r.weight</i>

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

GRASS Raster Commands

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



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 [-fci] **input**=name[,name,...] [**cache**=value] [**null**=string]

r.what [-fci] **input**=name[,name,...] [< inputfile] [**cache**=value] [**null**=string]

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](#).

OPTIONS

Flags:

- f** Also output the category label(s) associated with the cell(s) at the user-specified location(s).
- c** Turn on cache reporting.
- i** Output integer category values, not cell values.

Parameters:

input=name[,name,name,...]

The name(s) of one or more existing raster map layers to be queried.

cache=value

Size of point cache.

null=string

Character string to represent no data cell.

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 regular font) 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](#), [parser](#)

AUTHOR

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

Last changed: \$Date: 2002/01/25 05:45:34 \$



NAME

wrat – Water Resource Assessment Tool

USAGE

This module must be used interactively.

INPUT MAP CODES

elevation: meters, as well as cell resolution

rainfall maps: 100ths of an inch

K factor: K factor times 100 K of .35 = 35

Introduction

The Water Resource Assessment Tool is a collection of programs run within the GRASS GIS. These programs are an aid in understanding the nature of runoff in a study area based on information typically available for a GIS. The programs analyze the terrain to define drainage direction and areas, simulate runoff and peak runoff and model nonpoint source pollution and map contaminant source areas and contaminant pathways. This users guide outlines the typical analytical sequence, describes input data requirements and possible output. Some suggestion for interpreting output are included. It is assumed the user is familiar with the basics of GRASS.

Input

Map layers required: digital elevation model, land cover, soil maps of hydrologic soil group, soil texture, and erodibility factor (K factor). Optional maps describe best management practices for sediment, nitrogen phosphorous and chemical oxygen demand (COD).

Typical Analysis

Three interrelated areas of analysis are available: terrain analysis, hydrology and pollution (see figure X1). Because these areas are not distinct, analysis should proceed in a logical order since some portions build on previous results. Terrain analysis and runoff generation must proceed routing peak discharge, which in turn must proceed contaminant routing. The following steps provide a natural sequence of analysis through all of the tools. However, a great deal of valuable information may be gained executing only a portion of the available programs. This is a tool kit, not a prescription for studying every area.

GRASS Raster Commands

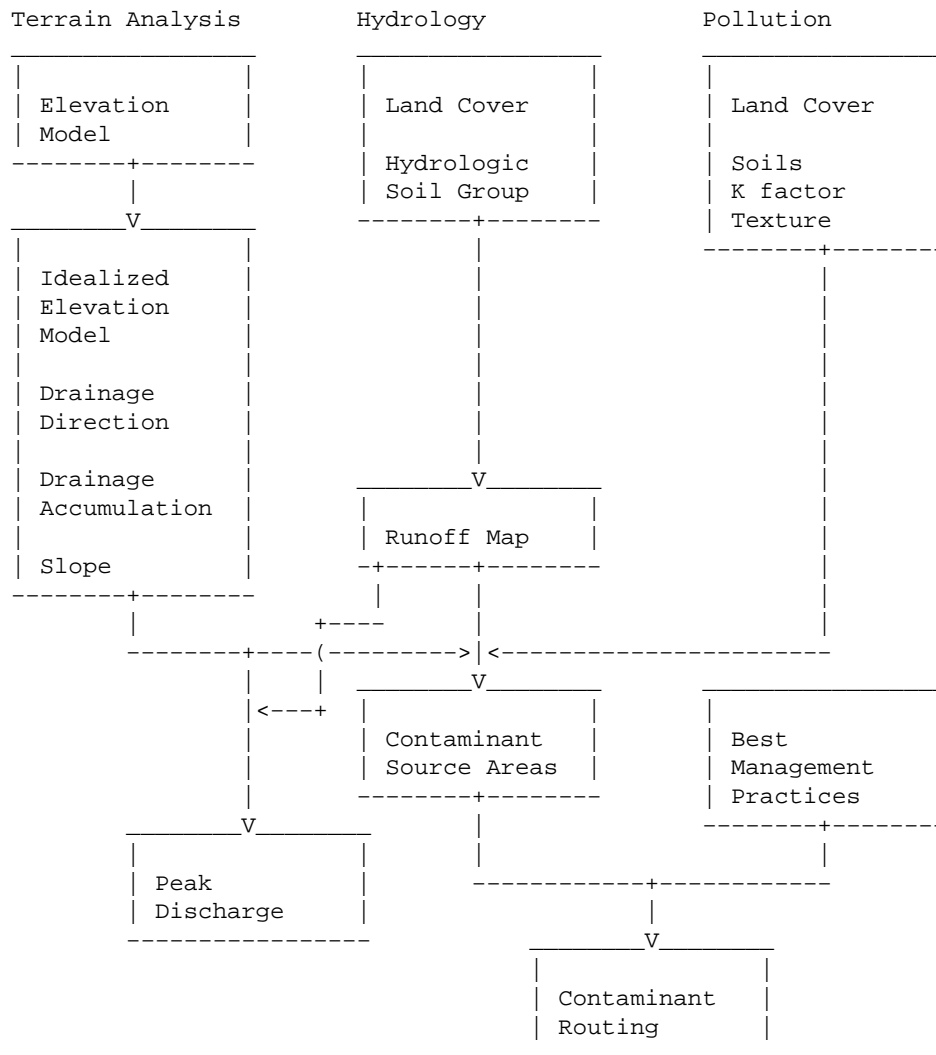


figure X1.

Defining the study area

GRASS looks at the world through a rectangular "window". The window should be set to include the entire area of interest with several cells to spare in all directions. Because of difficulty interpreting terrain at the edge of the window, the analysis moves in from the window edge. The extra area is particularly important if the top of a watershed of interest is not marked by a steep decline into the adjacent watershed.

Routing routines assume that the entire watershed is within the current window. This is not always true or even practical. For instance, an analysis along the side of a long river will provide valuable information about local contributions to the larger resource. However, calculations in the long river itself can not take into account upstream effects outside the current window and are thus invalid. Yet those local contributions may be of considerable interest. Great care must be taken interpreting results of watersheds only partially contained within the operating window.

Cell resolution, the size of each cell, should be set as large as is appropriate for your purposes. In no case should the resolution be smaller than the spatial resolution of your best input layer. Doubling the cell length

and width will cut computation times by 3/4s! The east–west resolution need not match the north–south resolution. However, large deviations from square cells will distort some of the neighbor concepts used by some algorithms to "feel their way" across the terrain.

Once the analysis is begun, the same window, including cell resolution, must be maintained throughout the project. The interface program tries to insure this. Changing the boundaries invalidates the drainage accumulation maps. Changing the cell resolution can have disastrous effects on the drainage direction map, such as creating dead end and circular drainages. The idealized elevation model will lose its "ideal" character if resampled. If the window must be changed, start the analysis from the beginning.

Some of the output maps may be resampled to a smaller window or different cell size after the analysis is finished. Contaminant source area maps are produced in units of mass per unit area and may be resampled. This may be useful for presentation of results or incorporation within another GIS analysis.

Temporal scale

Hydrologic simulations are based on single storms, presumably one day events. Actual or design storms may be employed.

Terrain analysis

The actual analysis may begin using the digital elevation model to produce an idealized elevation model. Real data has, for a number of reasons, sections which are difficult to drain. The idealized elevation model has no cells which can not drain to the edge of the window. This assumes complete surface drainage is possible. This condition is never completely true and in some regions, such as those with karst topography, so far from the case as to invalidate the peak discharge and contaminant routing results.

The idealized elevation model is used to determine the direction in which each cell will drain. This is done by searching the edge of the window for the lowest point and draining the watershed that leads there. Then the next lowest outlet is found and its watershed drained, and so on until the entire area is drained.

The drainage direction map is used to produce a drainage accumulation map. The values in a drainage accumulation map are the total number of cells including that cell, which drain through that cell. If multiplied by the area of a cell, the drainage accumulation map will yield the area of the watershed above and including each cell.

Hydrology

Two programs contribute to the hydrology section: runoff and peak discharge. The runoff section is based on the Soil Conservation Services Curve number method (Soil Conservation Service, USDA, 1971). Curve numbers, from 0 to 100, describe the potential for runoff. Curve numbers are inferred from the land cover map and the map of hydrologic soil groups. A depth of precipitation for a design storm or a rainfall map is required to produce a runoff map.

To this point the terrain analysis and the hydrology could progress independently. From this point on the distinctions between terrain analysis, hydrology and pollution, as separate lines of analysis begin to fade. Actually pollution analysis will always be dependent to some extent on hydrology but that will be dealt with in a later section. Peak discharge combines the results of the terrain analysis and runoff sections.

Peak discharge, the volume of water passing out of a cell per unit time, is calculated with an empirical relationship developed by Smith and Williams (1980) and employed in CREAMS and AGNPS. Watershed characteristics are interpreted from several all of the results from the terrain analysis. The resulting map of peak discharge, significant by itself, is also used in the routing of contaminants.

Pollution

Contaminant sources are mapped as the contaminant originating per unit area. Sediment, nitrogen, phosphorous and COD may be modeled. The algorithms used to predict contaminant mobilization depend on the landcover of the cell involved. The universal soil loss equation and adaptations from AGNPS and CREAMS are employed for rural, agricultural and open lands. An algorithm from SWMM is adopted for urban areas. Land cover, soil texture and erodibility factors, rainfall, runoff and peak discharge maps are the required inputs for this section.

The final part of the analysis routes contaminants. It requires the results of the preceding sections. Sediment, nitrogen and phosphorous nutrients and COD may be modeled. The routing is done as a mass balance accounting for imports of contaminants into a cell, contaminants originating within the cell, losses to infiltration and deposition and exports to the next cell. Figure X2 is a schematic representation.

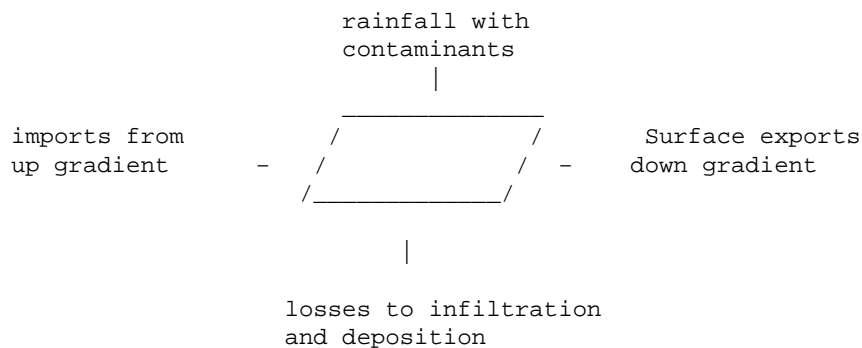


figure X2.

Some practical considerations:

GRASS only stores integers, assuming values less than one to be equal to zero. Therefore small units of measure have been employed. Rainfall and runoff are described in hundredths of inches, while quantities of sediment are in kilograms with nutrients in grams. Flows are measured in cubic feet per second. The mixing of english and metric units employs those units in common usage in the U.S. Maps may be converted to any appropriate set of units using "[r.mapcalc](#)" to multiply the map by the appropriate conversion factor.

Preparing map layers

GRASS associates numeric values with cells as their attributes. Sometimes these numbers have real meanings, as in an elevation model in which the cell's value is its elevation. Some are more arbitrary codes representing qualitative categories, such as landcover. The following tables provide guides to units and codes used in the water resource assessment tools. The GRASS tool "r.reclass" allows speedy recoding of map layers to these codes.

Input layers and data

Elevation

Values in the digital elevation model represent the altitude above mean sea level expressed in meters.

Rainfall

Rainfall maps should be rainfall for a single day, measured in hundredths of an inch. Thus a one and a quarter inch rainfall would be represented as 125. The GRASS command "s.surf.idw" can be used to interpolate a rainfall map between observation stations, however, s.surf.idw does not understand the effects of terrain on rainfall and will make significant mistakes in hilly country and for thunderstorms passing through a widely scattered set of observation points. A design storm may be preferable for assessment purposes.

Soil, texture

Values are a code for the dominant texture.

```
1 = clay soils
2 = silt soils
3 = sandy soils
4 = peat
5 = water
```

Soil, hydrologic soil group

Hydrologic soil group is primarily dependent on the soil texture. However depth to bed rock and depth to water table may strongly influence a soil's classification.

```
1 = hydrologic soil group A
2 = hydrologic soil group B
3 = hydrologic soil group C
4 = hydrologic soil group D
5 = water
```

Soil, erodibility K factor

K factors are a decimal value greater than zero and typically less than .5. Formally they represent the soil loss rate per unit of per erosion index unit for a specified test plot. (Agriculture Handbook Number 537 "Predicting Rainfall Erosion Losses", USDA 1978) Information about local soils should be available from your local Soil Conservation Service office. Because GRASS uses only integer values, multiply K factors by 100. Thus a K factor of .37 would be coded as 37. Use a code of 100 for water.

Landcover

The landcover map must be coded so the water resource assessment tools can recognize the landcovers indicated. The following table illustrates the 15 landcover category codes expected. When attempting to make the best fit of available data to this encoding scheme, remember to think of the areas both in terms of their ability to slow runoff and their contributions to water quality. Some creative lies may be useful for special

purposes. For instance, a development built to the performance standard that peak discharge is not to exceed a field in good condition could be coded as an old field for hydrologic analysis and built up for pollution analysis.

```

1 = corn
2 = rye
3 = oats
4 = soybeans
5 = hay
6 = grass
7 = old field (grass)
8 = old field (shrub)
9 = pasture
10 = forest
11 = wetlands
12 = fens
13 = water
14 = built up
15 = barren

```

Output Map layers

Many maps can be produced with the water resource assessment tools. The following sections outline how these maps are prepared, the units of measure and codes employed along with some notes as to how they should and should not be used. More complete information is available in the model documentation. The source code is also provided and contains a great deal of internal documentation. Output maps are presented in the approximate order in which they would be produced.

The idealized elevation file is a digital elevation file. Cell values represent elevation in meters. This elevation model will differ from the input DEM provided by the user, in that depressions in the data are filled in so that water landing anywhere on the idealized surface can flow to the edge. Although useful for analyzing terrain, and employed for slope measurements, the idealized elevation model is probably further removed from reality than the original data upon which it is based. The idealized elevation data may not be resampled to a different cell resolution and retain its desired "idealized" characteristics.

Drainage Direction

Cells in the drainage direction map are coded to indicate the neighboring cell into which they drain. The eight nearest neighbors are represented by the integers 1 through 8 as depicted in figure X3.

```

4 3 2
5 1
6 7 8

```

figure X3.

Drainage direction is similar to aspect. However they are not exactly the same thing. In an aspect map, the cells at the bottom of a V shaped valley may face each other without draining into each other. This map is produced by searching the border of the window for the lowest cell and draining all cells in that drainage. The algorithm moves up through the drainage one unit of elevation at a time until all adjacent cells of that elevation are drained, trying to assign the most direct drainage direction. Then the next lowest undrained border cell is found and that watershed is drained until the entire map is drained. The routine sometimes has difficulty at the top of watersheds if the next valley does not clearly drop away.

Drainage Accumulation

Values in the drainage accumulation map represent the number of cells, including the cell with the value, that drain through the cell. Thus the minimum drainage accumulation is one, while the maximum is the number of cells in the largest watershed in the study area. The drainage accumulation map is used to find the watershed area for each cell in the study area and to guide a search up the watershed to find the length of the longest stream. The drainage accumulation map can be used to define the stream networks by reclassifying the cells below a minimum accumulated drainage to zero. The remainder represents the streams. Drainage accumulation is drainage area represented in cells. Multiplying a drainage accumulation by the area of a cell yields the drainage area above that cell.

Runoff

The values in the runoff map represent the depth of water expected to run off that cell, from the input rainfall, based on the soil and landcover maps specified. Values could range between zero and the amount of rain that fell on that cell. Runoff, like rainfall, is measured in hundredths of inches. A value of 75 represents .75 inches of runoff. Runoff values can be multiplied by the area of each cell to give a volume of runoff. However, care should be taken when converting units because cell sizes in GRASS are defined in meters. Upgradient values in the runoff map are summed during the peak discharge calculations and used in the contaminant source area and routing calculations.

Peak Discharge

Peak discharge is the maximum rate at which water passes through a cell measured in cubic feet per second. These estimates are based on an empirical relationship which describes the entire upstream portion of the watershed. If the entire upstream portion of the watershed is not within the current window, the model assumes the upstream edge is the top of the watershed. This is wrong and estimates of peak discharge in streams which enter the window are meaningless as are routings of contaminants based on peak discharge. The calculations are retained so that local contributions from side streams and slopes may be modeled. These local effects may be of substantial importance to those involved in the local area regardless of distant conditions.

Contaminant Source Areas

Sediment, nitrogen and phosphorus nutrients and COD can be simulated. The algorithms used to predict contaminant mobilization are dependent on the contaminant and the landcover class. In non urban areas sediment production is based on the universal soil loss equation. Nitrogen and phosphorous are modeled both in solution and associated with sediment in rural areas and only as a function of sediment in urban areas. COD is assumed to be soluble and is calculated by loading factors.

Sediment loadings are expressed as kilograms per hectare. Nutrients and COD loadings are expressed as grams per hectare.

These loadings are a function of the rainfall, the soil, slope and landcover. They do not represent the natural scatter found in natural events. Results may be used as a relative indicator of contaminant generation. As such these maps may highlight those areas in a watershed that would benefit most from conservation efforts. Subtracting two source area maps generated for different scenarios would help locate the areas of greatest change.

Routing of Contaminants

Contaminants are routed down stream assuming total conservation unless a best management practice map for the particular contaminant is used. If a best management practice map is used contaminants flux across a BMP is reduced by the percentage indicated.

The output maps indicate the total mass of contaminant passing through a cell. These maps can be used as indicators of the contaminants delivered to any point down stream. Similar analysis for varying land cover regimes could be subtracted with "[r.mapcalc](#)" to find areas of greatest positive and negative change resulting from the changes in scenarios. Again these numbers represent relative indicators.

Interface Program

The Water resources assessment tool is run through an interface program invoked from the grass environment at any GRASS prompt by typing `wrat`. Work is organized in projects which should represent the investigation of a geographic area. The user interface is a series of menus which guide the user through the program. A project file is created which keeps track of the options used and input and output map names. The project file also creates a database window based on the active window at the time a project is started. The active window is returned to this active window for subsequent work sessions on the project.

First Menu

At this menu the user can:

- 1 start a new project
- 2 start a project based on an existing project
- 3 work on an existing project
- 4 remove project files
- 5 exit

The user is prompted for needed information such as the name of the project to work on with the option of listing project files if needed. Once a project is selected the second menu is offered. The User may do:

- 1 Terrain Analysis
- 2 Runoff Analysis
- 3 Contaminant Analysis
- 4 return to main menu

Choices 1, 2 or 3 lead to menus for each of those analysis.

In the Terrain Analysis the options are:

- 1 Create an idealized elevation model
- 2 Create a drainage direction map
- 3 Create a drainage accumulation map
- 4 Create a slope map
- 5 Return to the previous menu

GRASS Raster Commands

In each case except option 5 the user is prompted for needed information. where appropriate the likely choices are supplied if not insisted upon. The drainage direction map must be based on the idealized elevation map. The drainage accumulation map needs a drainage direction map which should be the output map from option 2. In this way the user is guided along.

In the **runoff analysis section** there are only three options:

- 1 Create a runoff map
- 2 Simulate peak discharge
- 3 Return to the previous menu

In this section the options are not as straight forward and the user is coached along. The user must provide the names of several input maps and can obtain a listing by entering list instead of a map name where requested. The user must also choose between a design storm and rainfall map. The most difficult information requested of the user is the antecedent moisture condition. This is a number between 1 and 3 inclusively which describes the amount of moisture already in the soil which effects the amount of rain which can be absorbed during the current storm. 1 represents extremely dry conditions and 3 extremely wet conditions. The concept comes from the Soil Conservation Service curve number method of predicting runoff. The basic guidance from the SCS is offered with the request and a value of 2 neither wet or dry is the default. Peak discharge is quite sensitive to the antecedent moisture condition so some care should be taken here.

The **contaminant section** has only three options as well:

- 1 Model Contaminant source areas
- 2 Rout contaminants through the watershed
- 3 Return to the previous menu

When modeling contaminant source areas the user may model just sediment or sediment and any combination of nitrogen, phosphorus and COD. Because nutrients are associated with sediment sediment predictions are required for nutrient modeling. Most input for this section is straight forward except the number of days since the last significant rain. The amount of dirt available to wash off of urban areas accumulates between rainstorms. Rain storms wash urban areas clean. a default of 7 days is suggested because this puts urban and rural areas on roughly even footings for comparison. To complicate the issue street sweeping also removes contaminants from road ways and so if the study area has a regular street cleaning program that should be reflected by entering fewer dry days. Several minor rain storms, especially short hard rains will accomplish the same cleaning as a long large rain. SWMM assumes that a half inch of rain removes half of the available dirt and the next half inch removes half of what is left and so on. A worst case for urban areas is a rain after an extended dry period. However the goal is to compare watersheds or source areas. Agricultural areas contribute the most pollution with an antecedent moisture condition of 3 since that is when the most runoff and erosion takes place.

Summary

The tools presented here run within the GRASS GIS. A flow chart of input and analysis (figure X1) can be used as a guide for the order of analysis. The system of units and means for encoding qualitative information are supplied for both input and output map layers. Careful preparation of input maps is strongly encouraged since this effort will likely save time and trouble later in the analysis.

The ultimate utility of analyses performed with these water resource assessment tools is dependent on the interpretation of the user. Always keep in mind that these results are based on empirical relationships which

GRASS Raster Commands

represent expectations over a long term average. For this reason numerical results should be used only for comparison of areas and scenarios. Employing the entire set of tools from the tool kit may be inappropriate, and or redundant information.

You can't hurt the model by running it! So feel free to experiment!

AUTHOR Brian R. Brodeur Cook College Remote Sensing Center
<bbrodeur_dep_boston@state.ma.us>

Last changed: \$Date: 2002/01/25 05:45:34 \$