GRASS GIS: A Useful Tool for the Mountain Cartographer

Pat Dunlavey
Pat Dunlavey Cartographics
40 Oblong Road
Williamstown, MA 01267
Tel: 413-458-9273
Email: pat@pdcarto.com

Presented at:
International Cartographic Association (ICA)
2002 Mountain Cartography Workshop
Timberline Lodge, Mt. Hood, Oregon

INTRODUCTION

The genesis of digital cartography was initially marked by an emphasis on vector and text editing and rendering. The first digital maps were hardly more than pen plots of vector data. Later, as powerful raster editing tools became available, especially Photoshop, digital raster backgrounds became commonplace, backing up vector foreground information. These raster backdrops are used for depicting terrain shading, land cover or other area classification, and aerial or satellite imagery. As Tom Patterson's work with rasterized drainage suggests, the trend for working more and more in the raster domain is such that no cartographic element is excluded from possible treatment as raster data. It is no surprise then that software designed to work with pixels, or raster data, has come to share equal importance with vector software in the cartographer's toolkit. Adobe Photoshop, with its rich layering and compositing abilities, dominates this niche. However, Photoshop lacks some important functionality for working with cartographic data, such as the ability to convert data from one projection to another, or to work effectively with deep-bit data (greater than 8 bits per channel), such as digital elevation models. To cover these gaps, most cartographers have assembled a collection of special-purpose utilities. To make things more confusing, the availability of inexpensive utilities, such as raster reprojection software, is different on various operating systems, such that many cartographers find that they must maintain both Macintosh and Windows systems.

I have found that GRASS <http://grass.itc.it/>, a free open source UNIX GIS software system which can run on various hardware platforms under the Linux OS, on Macintoshes under OSX, and on Windows systems running cygwin <http://cygwin.com/>, provides a very effective complement to Photoshop. GRASS can tackle most raster geodata processing tasks, leaving much less need for additional cartographic raster tools.
(GRASS is primarily a raster GIS. It has modules for importing, displaying and manipulating vector GIS data, but I have not explored them and therefore, I will not be discussing them here.)

Some Background on GRASS

To quote the GRASS Frequently Asked Questions:
"***Geographic Resources Analysis Support System***, commonly referred to as GRASS GIS, is a Geographic Information System (GIS) used for data management, image processing, graphics production, spatial modelling, and visualization of many types of data. It is free software released under GNU General Public License (GPL)." It was originally developed at the United States Army Construction Engineering Research Laboratories (USA-CERL) starting in 1982, as a tool for land management and planning by the US military. It is widely used in academic and government settings, as well as commercially. USA-CERL completed its last full release of GRASS, version 4.1 in 1992, followed by updates and patches through 1995. Since then, an international community of GRASS developers, very similar to the Linux developer community, has functioned very effectively to further enhance GRASS, adding a graphical user interface, and improving floating point functionality, as well as adding over a hundred new modules for applications like erosion and wildfire modeling, three dimensional terrain visualization, and import and export of new file formats. Currently, development is coordinated by Markus Neteler, of Germany.

GRASS, like much software developed under UNIX, is actually a large collection of somewhat independent software programs which are designed to be run from a terminal prompt. For example, to create a new data layer by importing a TIFF image, you might type the following command in the GRASS shell terminal:

```
GRASS:~ > r.in.gdal -o -e
input=/home/pat/incoming/dem2.tif
output=DEM2 title=dem2
```

This command would use the r.in.gdal module to read the TIFF file "dem2.tif" into the current ***Mapset,*** giving it the title "dem2". However, it is not usually necessary to know how to type the terminal command, since GRASS includes an, admittedly somewhat clunky, graphical user interface. To accomplish the same thing under the GRASS GUI, you would select ***Import>Raster maps>Various***, which would display the following dialog:

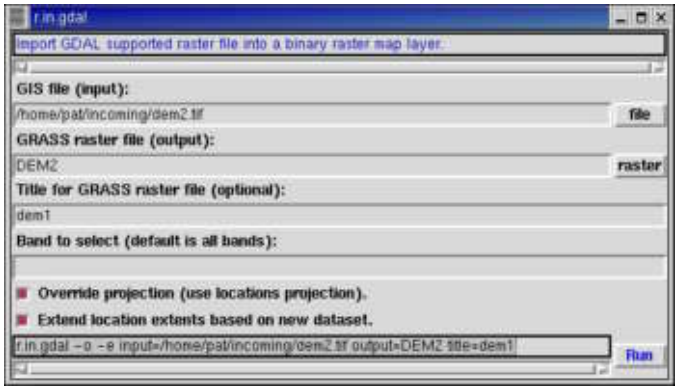


Figure 1

GRASS programs generally work in this way: you type the command followed by required arguments (or construct it

through the GUI), and the module does whatever it does, and then exits.

Some commands are used to display a map: because GRASS does not keep your current work in memory with an automatically refreshing display, you have to explicitly tell it to display a map. But first, you have to create a window, and then select that window for displaying!:

```
GRASS:~ > d.mon start=x0
using default visual which is
TrueColor
ncolors: 16777216
Graphics driver [x0] started
GRASS:~ > d.mon select=x0
GRASS:~ > d.rast map=lc
 100%
GRASS:~ >
```

Here we've created a window x0, selected it for display, and then painted the map called "lc" (land cover) to it, with the following result:

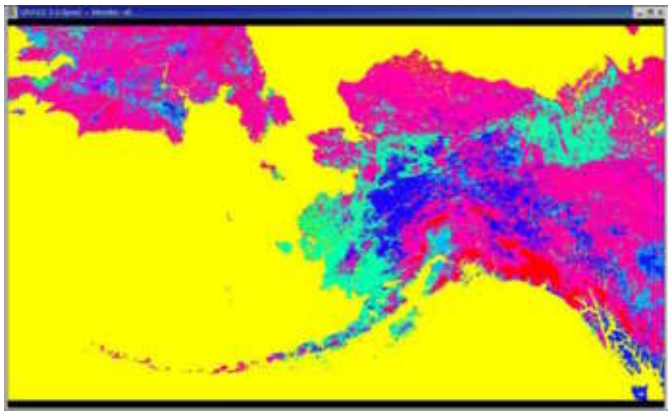


Figure 2

Experienced GRASS and UNIX users use advanced techniques such as piping the results of one command into another on the command line, or scripting a complex series of commands through the UNIX shell. I will show a little later how simple scripting was used to implement some useful functions.

GRASS Modules

GRASS has a very large number of commands, which are broken down as follows:

- Database Commands
- Display Commands
- General Commands
- Grid 3D Commands
- Imagery Commands
- Miscellaneous Commands
- Models: simulation models
- Paint Commands
- Photo Commands
- PostScript Commands
- Raster Commands
- Shell Scripts
- Sites Commands
- Vector Commands

The raster commands alone number 150, and counting, as seen in this table:

r.agnps50.input	r.in.doq	r.null	r.stats
r.agnps50.run	r.in.dted	r.out.agnps	r.sum
r.agnps50.view	r.in.elas	r.out.arc	r.sun
r.answers	r.in.gdal	r.out.ascii	r.sunmask
r.average	r.in.gridatb	r.out.bin	r.support
r.basins.fill	r.in.hdf	r.out.elas	r.surf.area
r.bilinear	r.in.ll	r.out.gridatb	r.surf.contour
r.binfer	r.in.miads	r.out.hdf	r.surf.fractal
r.buffer	r.in.pbm	r.out.mpeg	r.surf.gauss
r.cats	r.in.poly	r.out.pov	r.surf.idw
r.circle	r.in.ppm	r.out.ppm	r.surf.idw2
r.clump	r.in.shape	r.out.rlc	r.surf.random
r.cn	r.in.sunrast	r.out.tga	r.thin
r.coin	r.in.tang	r.out.tiff	r.timestamp
r.colors	r.in.tiff	r.out.xyz	r.to.gnuplot
r.colors.paint	r.in.utm	r.param.scale	r.to.sites
r.combine	r.infer	r.patch	r.topidx
r.composite	r.info	r.plane	r.topmodel
r.compress	r.kappa	r.poly	r.transect
r.contour	r.kineros	r.profile	r.tribs
r.cost	r.lags	r.proj	r.univar
r.covar	r.le.dist	r.quant	r.volume
r.cross	r.le.null	r.recode	r.water.fea
r.describe	r.le.patch	r.random.cells	r.water.outlet
r.digit	r.le.pixel	r.random.surface	r.watershed
r.direct	r.le.rename	r.rational.regression	r.weight
r.distance	r.le.setup	r.random	r.weight2
r.drain	r.le.trace	r.reclass	r.weighted.cn
r.feat.thin	r.line	r.report	r.what
r.fill.dir	r.linear.regression	r.resample	r.wrat
r.fillnulls	r.los	r.rescale	
r.flow	r.mapcalc	r.rescale.eq	
r.flowmd	r.mask	r.rescale.inf	
r.grow	r.mask.points	r.ros	
r.his	r.median	r.runoff	
r.hydro.CASC2D	r.mfilter	r.slope.aspect	
r.in.arc	r.mode	r.spread	
r.in.ascii	r.moran	r.spreadpath	
r.in.bin	r.neighbors	r.stage3	
r.in.dem	r.nntool	r.statistics	

All commands are documented via the ubiquitous UNIX "man" pages.

In the remainder of this paper, I would like to feature the three commands that are highlighted above, ***r.in.gdal,*** ***r.mapcalc,*** and ***r.proj***, and discuss their applicability to tasks commonly encountered in natural terrain cartography.

DATA IMPORT FORMATS SUPPORTED BY GRASS, USING GDAL (GEOSPATIAL DATA ABSTRACTION LIBRARY)

GRASS supports a wide variety of data types, particularly through its ***r.in.gdal*** command, making it an effective bridge between raw GIS data and Photoshop:

- 2D raster data,
- 3D raster data (voxels),
- topological vector data (2D, currently extended to 3D)
- point data (called sites)

In detail:

• Raster: ASCII, ARC/GRID, E00, GIF, GMT, TIF, PNG, ERDAS LAN, Vis5D, SURFER (.grd) ... Using GDAL library (r.in.gdal) more formats like USGS-DEM, CEOS (SAR, LANDSAT7 etc.) can be read	• Image (satellite and air-photo): AVHRR, BIL/BSQ, ERDAS LAN, HDF, LANDSAT TM/MSS, NHAP aerial photos, SAR, SPOT, ...
• Vector: ASCII, ARC/INFO ungenerate, ARC/INFO E00, ArcView SHAPE (with topology correction), BIL, DLG (U.S.), DXF, DXF3D, GMT, GPS-ASCII, IDRISI, MOSS, MapInfo MIF, TIGER, VRML, ...	• Sites (point data lists): XYZ ASCII, dBase

GDAL Raster Formats

Format Name	Creation	Georeferencing
Arc/Info ASCII Grid	No	Yes
Arc/Info Binary Grid	No	Yes
BSB Nautical Chart Format	No	Yes
CEOS (Spot for instance)	No	No
First Generation USGS DOQ	No	Yes
New Labelled USGS DOQ	No	Yes
Military Elevation Data	No	Yes
Eosat Fast Format	No	No
ERMapper Compressed Wavelets (.ecw)	Yes	Yes
ESRI .hdr Labelled	No	Yes
Envisat Image Product	No	No
FITS	Yes	No
Graphics Interchange Format (.gif)	Yes	No
Arc/Info Binary Grid	Yes	Yes
GRASS Rasters	Yes	Yes
TIFF / GeoTIFF	Yes	Yes
Erdas Imagine .hfa	Yes	Yes
Atlantis HKV Image	Yes	Yes
Japanese DEM (.mem)	No	Yes
JPEG JFIF	Yes	Yes
Atlantis MFF	Yes	Yes
OGDI Bridge	No	Yes
PCI .aux Labelled	Yes	No
Portable Network Graphics	Yes	No
Netpbm (.ppm,.pgm)	Yes	No
USGS SDTS DEM	No	Yes
SAR CEOS	No	Yes
USGS ASCII DEM	No	Yes

Table 1

As you can see, with the GDAL import library, GRASS supports the importation of many raster data formats, and if the source data supports georeferencing, then that georeferencing is preserved in GRASS.

Raster data imported into GRASS may be in the form of integer data with 8, 16 or 32-bit precision, or floating point data. Color imagery may be handled either as indexed color (where each value represents a particular RGB color), or as a group of integer maps, each representing a color band, e.g. red, green and blue.

On the output side, GRASS supports several formats that can be read by Photoshop, with TIFF and binary being the most commonly used.

REPROJECTION

For the personal computer-based cartographer, perhaps one of the most troublesome challenges is in dealing with raster data which is not in the same projection as the map being developed. On the Macintosh, GeoCart, at $500, may be prohibitively expensive for this occasional task. On the Windows side, I am not aware of any raster reprojection software currently available for a cost of under $1000. Therefore the ability in GRASS, a totally free program, to handle raster reprojection can easily justify the trouble of learning how to use it.

To reproject data in GRASS, you use the `r.proj` command to copy a map from one ***Location*** to another. For this to make any sense, one needs to understand how GRASS data is organized and therefore a quick digression is called for…

A digression: How GRASS data is organized

Unlike many of the applications that we may be used to in the Windows and Macintosh environments, GRASS does not store a project as a single file. For example, a multi-layered Photoshop file, which could technically be described as an assemblage of individual images together with

parameters defining how the images overlay and behave when overlaid, is nonetheless stored as a single monolithic file which cannot generally be manipulated outside of Photoshop. GRASS, on the other hand, organizes a project into a number of files organized into a file and directory structure on your hard disk. A given GRASS project is identified by its ***Database, Location, and Mapset***. (GRASS has various tools for manipulating these elements of a project, and it's generally best to let GRASS perform these manipulations, rather than to try to create, delete and modify these files directly.)

The ***Database*** is the top level directory, under which one can organize files for a particular project:

Figure 3

For our Alaska map project, we created a GRASS ***Database*** in /home/pat/Grass5Data/Alaska/. (Note that there is another Database here in /home/pat/Grass5Data/Berkshire/, used for a different project.) Inside the ../Alaska directory, are a number of subdirectories (i.e. folders) each of which corresponds to a particular GRASS ***Location***, such as "USAK", which in this case, is the main ***Location*** for our Alaska map project.

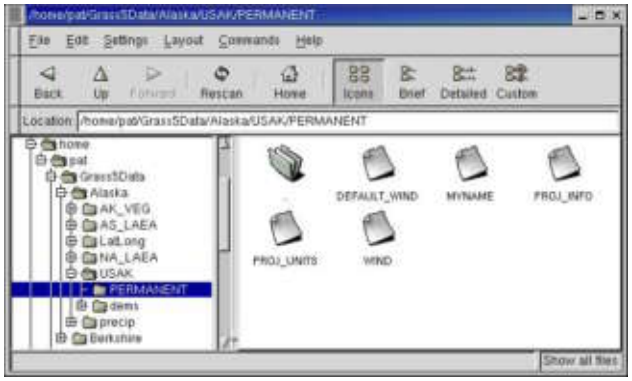


Figure 4

The ***Location*** always contains a subdirectory entitled "PERMANENT", which is actually a special ***Mapset***. It is where information on the projection/coordinate system is stored and how cells in the raster relate to the coordinate system. Therefore, all maps in the Location "USAK" have the same coordinate system, which is defined here, in "PERMANENT".

"PROJ_INFO, a text file, defines projection parameters, e.g.:

```
name: Albers Equal Area
proj: aea
ellps: clark66
a: 6378206.4000000004
es: 0.0067686580
f: 294.9786982000
lat_0: 50.0000000000
lat_1: 55.0000000000
lat_2: 65.0000000000
lon_0: 74.6938150000
x_0: 0.0000000000
y_0: 0.0000000000
```

"WIND", another text file, defines the current region – the rectangle within the coordinate system described by your location's projection which defines the limits of your map:

```
proj:       99
zone:       0
north:      2529149
south:      456509
east:       987151
west:       -2685688
cols:       7346
rows:       4145
e-w resol:  499.97808331
n-s resol:  500.03377563
```

(These files inside the "PERMANENT" Mapset are created during the new ***Location/Mapset*** creation process. More on that in a moment.)

A *Mapset* is roughly analogous to a Photoshop document. It can contain a number of data layers, or maps, which enjoy a kind of awareness of one another. Under this USAK *Location*, we have a *Mapset* entitled "dems", which contains all the destination data layers for our Alaska map project. The .../dems folder contains all the data in its raw form and one does not normally have any reason to poke around inside here.

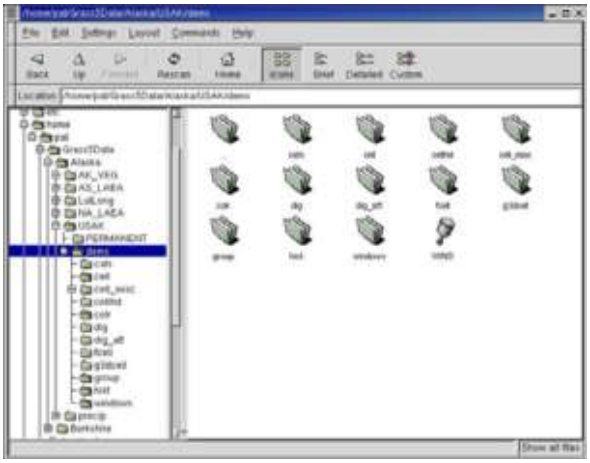


Figure 5

When you launch GRASS, you are presented with the option to use an existing *Database/Location/Mapset*, or to create a new one:

Figure 6

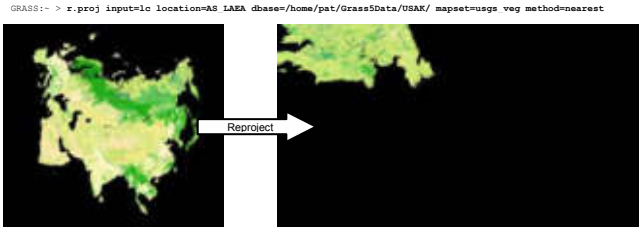
If you elect to create a new Location, GRASS launches a terminal program which leads you through the process of entering the projection and default region information for that location. Once you have gathered the information, the process of defining a new Location takes just a couple minutes.

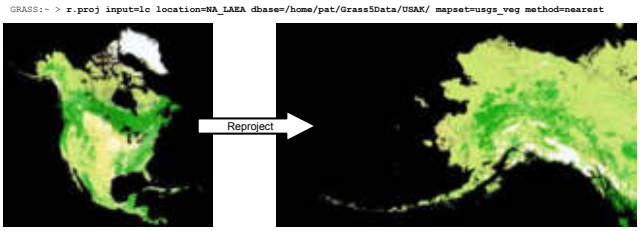
Figure 7

Reprojection continued…

To reproject a map from one projection/coordinate system to another, you need to have created the corresponding source and destination GRASS ***Locations*** and have imported the data to be reprojected into the source ***Location***. That having been done, and after starting a GRASS session with the destination ***Mapset***, `r.proj` is invoked from the command line, to copy the source map to the destination ***Mapset***, with reprojection applied.

For our Alaska project, we used land cover data from the Eros Data Center Global Land Cover Characterization project for Russia and Canada. The GLCC data sets for North America and Asia were each projected to our map's ***Location***…





…and then merged into a single image.

Figure 8

THE MAPCALC MODULE

Perhaps the single most powerful and flexible tool in GRASS is `r.mapcalc`. With mapcalc, one can perform algebraic and logic operations within and between maps. A very simple operation is to merge two maps into a third, using the logic that if a pixel in map A is null, then use the corresponding pixel from map B, otherwise, use the pixel from map A. A command to merge the Asia and North America land cover maps could be…

```
GRASS:~ > r.mapcalc
'lc=if(isnull(nalc),aslc,nalc)'
```

..where "lc" is the destination land cover map name, and "nalc" and "aslc" are the source land cover maps.

In a different example, mapcalc can do smoothing by averaging the values of neighboring pixels…

```
GRASS:~ > r.mapcalc 'smooth=(src[-1,-1]+src[-1,0]+src[-1,1]+src[0,-1]+src[0,0]+src[0,1]+src[1,-1]+src[1,0]+src[1,1])/9'
```

… where the [x,y] notation is used to specify an offset relative to the current pixel.

Or, we could perform a weighted average of two maps…

```
GRASS:~ > r.mapcalc 'wtavg=(src1*6 + src2*4)/10'
```

Using Mapcalc to grow land cover data

We can take this a little further to solve a common problem when trying to combine raster data and vector data, where a water edge in the vector data does not coincide perfectly with the land cover edges suggested in the raster data. In our Alaska project, our land cover raster data did not match well with our vector coastline data, as can be seen here (the black areas fringing the coastlines).

Figure 9

To solve this problem, we decided to grow the non-water pixels into the water pixels. Since this is not a built-in function of GRASS, we used Mapcalc to accomplish it.

The logic is as follows:

> If the pixel is water, then look at the four cardinal neighbor pixels. If a neighbor is not water, then assign the value of that neighbor to the pixel.

This can be coded in the Mapcalc "language" as follows…

```
r.mapcalc 'grown = if(src, src,
   \\
    if(src[0,-1], src[0,-1],          \\
     if( src[0,1],  src[0,1],         \\
      if(src[-1,0], src[-1,0],       \\
       if( src[1,0],  src[1,0]       \\
   )))))'
```

…where "src" is the source map and "grown" is the destination map, and the index value for water is zero.

The result is that every water pixel that is adjacent to a land pixel takes on the value of the land pixel. By looping through this several times, we can grow the land cover into the water by several pixels. Rather than figure out how to type this every time, I wrote a UNIX shell script that automates the process. The script prompts for a source map name, and for the number of growth iterations. The result will be a new map named "grown".

We ran the "grow" script on our Alaska land cover data with ten iterations. This image shows how it eliminated most of the gaps relative to the coast line.

Figure 10

Using Mapcalc to create shaded relief

There are numerous utilities available for creating shaded relief from digital elevation models. However creating relief shading in GRASS has some significant advantages, including the abilities to work with deep-bit DEM data (greater than 8-bits per pixel), and to maintain a georeferenced environment. The method we used to create shaded relief for the Alaska project points out some unique advantages to using GRASS.

Our DEM data was GTOPO30, which is in a geographic (latitude/longitude) projection. Of course we needed to reproject it to our USAK Albers Equal Area projection. However, I had the idea of illuminating the elevation model as though the direction of the sunlight were wrapping around with longitude. All symbology and type on the map is oriented to geographic north, and I felt it would be distracting if the angle of illumination did not likewise maintain an orientation relative to geographic north. Therefore, we chose to render the shaded relief while still in the geographic projection, and then reproject it to our working projection. We discovered when we tried this that the shaded relief looked odd because of differences in scale between the N-S and E-W at high latitudes. We were using a script with Mapcalc (one that is provided in the standard GRASS distribution) to do the shading. Therefore, it was quite easy to open up the script and modify the code so that it would account for the differences in scale between the N-S and E-W axes, as well as for the difference in horizontal and vertical units.

We also wanted to resolution-bump the DEM data, to enhance perception of the the large mountain ranges. This was our process in detail:

- Import GTOPO data to GRASS (figure 11 shows rendering from original DEM)
- Resolution-bump the DEM:

- Create a smoothed copy of the DEM by running it through an r.neighbors filter with 5x5 window (figure 12)
- Add the smoothed DEM and the original DEM together using a weighted average

- Generate shaded relief from the res-bumped DEM (using differential x/y scaling) (figure 13)
- Reproject the shaded relief to our working map projection (figure 14)

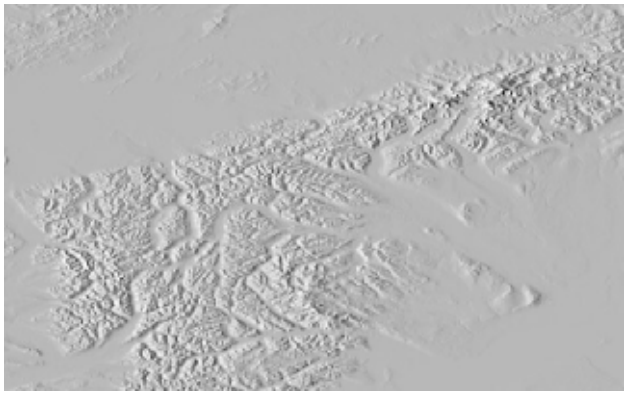


Figure 11 – a shaded relief rendering of the original GTOPO DEM

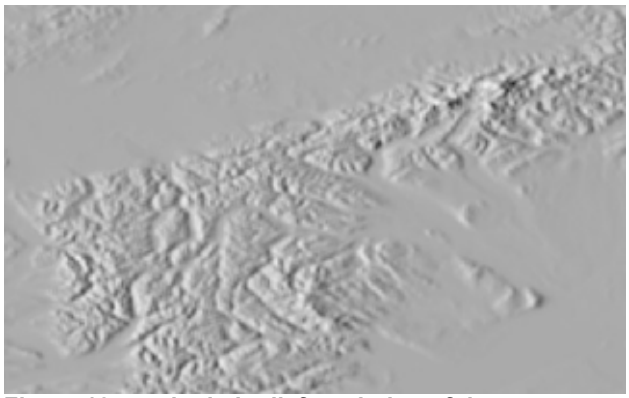


Figure 12 – a shaded relief rendering of the 5x5 smoothed

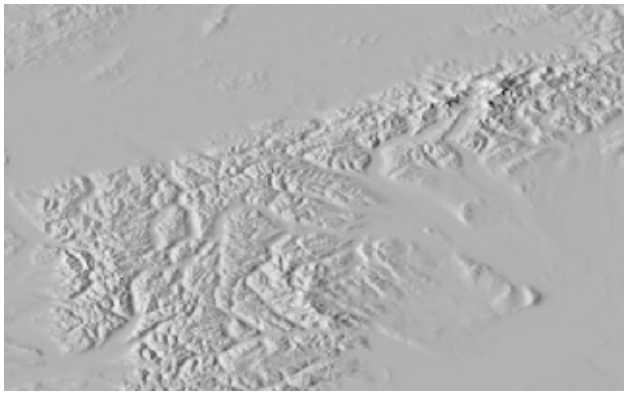


Figure 13 – the resolution-bumped and relief-shaded DEM

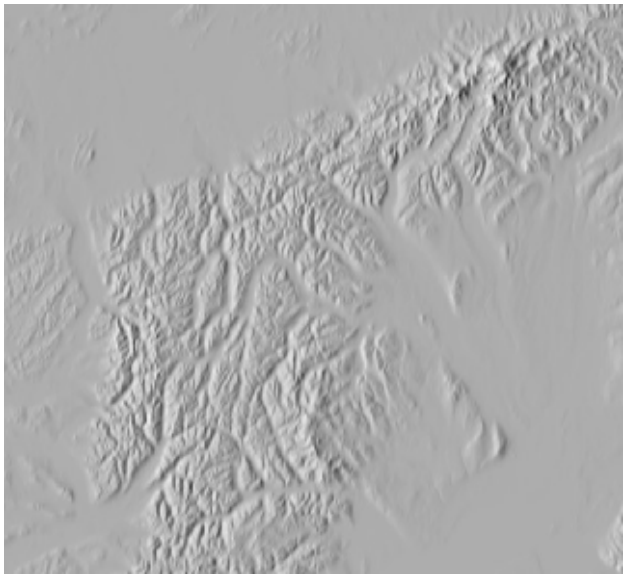
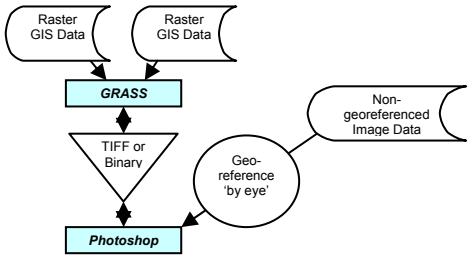


Figure 14 – the final result, after projection

GRASS AND PHOTOSHOP WORKFLOW

I have mentioned the value of doing raster work inside of a georeferenced environment. In GRASS, all data is georeferenced. Assuming you employ a non-arbitrary frame of geographic reference, GRASS can move data from one to another quite easily (see Reprojection above). But how can we make GRASS work together with Photoshop, which does not georeference data explicitly? The answer: carefully!



Moving data from GRASS to Photoshop is quite straightforward (with one significant exception) by employing one of the image formats supported by Photoshop. However, there are times when one will want to create data in Photoshop and then bring it into GRASS. You need to take special precautions to impose and maintain implicit georeference in Photoshop.

Perhaps the simplest method is to start out with a geographic image in GRASS which when exported to Photoshop can be used to register other images “by eye”. This reference image defines the geographic region in Photoshop. (When working in Photoshop, the canvas size should not be changed, and the reference image should not be moved.) With the reference image in the background, new image layers may be pasted in and transformed to align with the reference image.

SOME THINGS THAT ARE NOT SO NICE ABOUT GRASS

- Just getting started is a significant challenge, especially if you are not familiar with UNIX – (that's most of us).
- The user interface is incomplete, and does not shield the user from the command line environment.
- There is no technical support. (though the developer and user community is very helpful.)
- Some modules are more up to date than others (e.g. r.out.tiff cannot export floating point data).
- Several very important modules are not included in the standard distribution (due to licensing issues). The GDAL module, for example, must be obtained separately, and linked into GRASS.

CONCLUSIONS

I think I have demonstrated through these few examples that GRASS can serve as a powerful tool in the cartographer's collection, and can integrate very well with Photoshop to cover the majority of raster data processing needs. I personally have only begun to explore the possibilities. The learning curve is steep, and the inconvenience of moving between operating environments can be daunting. But the price is right. And for the technically oriented cartographer, the possibilities that can be opened by GRASS are limited only by the imagination.

APPENDICES

A: Exporting A Floating Point Grass Raster To Photoshop

The GRASS image export module r.out.tiff segmentation faults (crashes) when exporting floating point raster data. Since reprojection with cubic interpolation, and our shaded relief script both create floating point GRASS raster files, this is an obstacle to exporting to Photoshop. It is not possible to export floating point as binary (the other standard method of getting data into Photoshop) since Photoshop only imports integer-binary data. Our work-around is to use the PPM export format. Photoshop does not understand PPM, but The GIMP, a free Unix image editing program, does. Once the file has been opened in The GIMP, then you can export it as a TIFF file.

B: Grow Pixels Script

```
echo ""
echo Grow into zero cells
g.ask type=old element=cell desc=raster prompt="Enter source file"
unixfile=/tmp/$$
eval `cat /tmp/$$`
rm -f /tmp/$$
if [ ! "$file" ]
then
    exit 0
fi
src="${fullname}"

echo "$src"

gotit=0
while test $gotit -eq 0
do
   echo -n "iterations: "
   read itrs
   if test $itrs -ge 1 -a $itrs -lt 30
   then
      gotit=1
   else
     echo Sorry, iterations must be greater than 0 and less than
30
   fi
done

echo ""
echo Running r.mapcalc, please stand by.
echo Your new map will be named grown.  Please consider renaming.
echo ""

# Note: no space allowed after \\:
 r.mapcalc << EOF
 grown = if(      $src,       $src,               \\
    if($src[0,-1], $src[0,-1],      \\
     if( $src[0,1],  $src[0,1],           \\
      if($src[-1,0], $src[-1,0],      \\
       if( $src[1,0],  $src[1,0]       \\
  )))))
EOF

#Remove nulls created at edges
r.mapcalc 'grown=if(isnull(grown),0,grown)'

   echo "iteration=1"

 i=2
 while [ $i -le $itrs ]
 do
r.mapcalc << EOF
   grown = if(      grown,       grown,            \\
       if(grown[0,-1], grown[0,-1],      \\
        if( grown[0,1],  grown[0,1],       \\
         if(grown[-1,0], grown[-1,0],      \\
             if( grown[1,0],  grown[1,0]       \\
     )))))
EOF

#Remove nulls created at edges
r.mapcalc 'grown=if(isnull(grown),0,grown)'
   echo "iteration=$i"
   let i=$i+1
done

echo ""
echo New map created and named grown.  Consider renaming
```