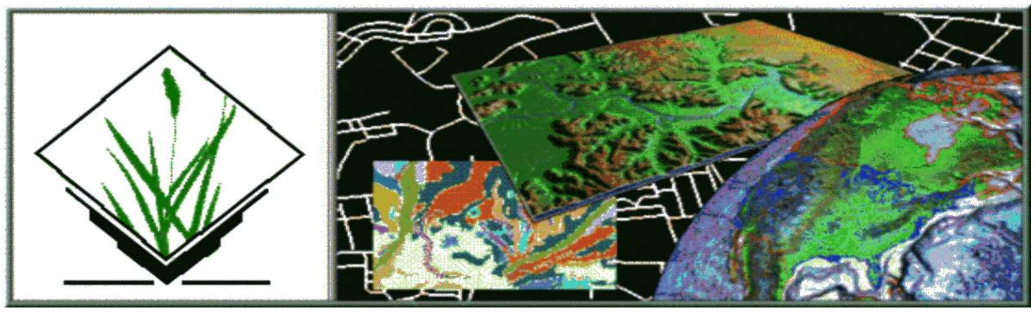


GRASS-News

Geographic **R**esources **A**nalysis **S**upport **S**ystem Volume 3, June 2005

Editorial

by Martin Wegmann

Dear *GRASS* user,

welcome to the third volume of *GRASSNews* which features a broad spectrum of articles. **Preprocessing of SRTM data** and its **further use in GRASS** is the topic of the first two articles. Followed by GRASS- R articles describing the **new GRASS 6 - R interface** and the **use of R for raster manipulation**.

Moreover *r.infer* is presented, a tool for **knowledge management**, this shall be the beginning of a series featuring different approaches on knowledge management in GRASS.

A very promising **preview of QGIS 0.7** including the new capabilities to interact with GRASS and the presentation of the *GRASS extension manager* (GEM) in the **News section** shows the two parallel approaches on the GRASS user inferface.

These articles outline pretty well some capabilities of GRASS but far more functions could be presented, personally I would like to see the actual use of GRASS functions in different projects and a more detailed presentation of GRASS visualisation potentials.

Looking forward to *N*^{o} 4, with kind regards

Martin Wegmann

Martin Wegmann
DRL - German Aerospace Centre @
Remote Sensing and Biodiversity Unit
Dept. of Geography, University of Würzburg, Germany
BIOTA-Project
http://www.biota-africa.org
http://www.biogis.org
wegmann AT biozentrum.uni-wuerzburg.de

Contents of this volume:

SRTM and VMAP0 data in OGR and GRASS

by Markus Neteler

Abstract

Years ago, hunting for free geospatial data was rather challenging. Today elevation data sets with an almost global coverage at high resolution are available from the Shuttle Radar Topography Mission (SRTM data set). Also a global set of vector maps at 1:1 million scale is available. Combined, both data sets provide a base cartography for most parts of the world. The technical issues for SRTM raster data are void filling, peak elimination and coastline extraction. To use the VMAP0 vector data, the user must deal with its unusual data format. In this article the preparational steps for using these data sets are described.

SRTM elevation data

The Shuttle Radar Topography Mission (SRTM) is based on single-pass radar interferometry, for which two radar antennas were mounted onto the Space Shuttle to simultaneously take two images from slightly different locations. One antenna was inboard, the other at the end of a 60 meter mast. The sent radar signal was reflected back and received by both the main and outboard antennas.

SRTM data are distributed at different resolutions depending on the part of the world in which you are interested. Within the U.S.A. these data are delivered at nominally 30 meters horizontal resolution (pixel size). For the rest of the world they are distributed at 90 meters resolution. The vertical precision is estimated as around 10 meters mean error. The horizontal datum is the World Geodetic System 1984 (WGS84). The vertical datum is mean sea level as defined by the Earth Gravitational Model geoid (EGM96 geoid model (1996)). Users commonly ignore the original vertical datum and use SRTM data as referenced to the WGS84 ellipsoid as geoid models are mostly unavailable in GIS. A test for the Trentino (Northern Italy) province has shown that the deviations between the EGM96 geoid undulation and the WGS84 ellipsoid are for this area within the submeter range which is beyond the vertical error of the SRTM data.

SRTM data acquisition

There are (at least) two possibilities to acquire SRTM raw data tiles:

1. Original tiles at 0.00083333 deg (3 arc-seconds, around 90m) resolution from NASA FTP site (NASA SRTM Website, 2004). Interestingly, these files irregularly disappear or are shifted to another FTP site. The file format is "hgt" with zip compression which corresponds to a BIL (band interleave by line) file without header file. Instead of having a header, the spatial reference is coded into the file name. SRTM file name coordinates refer to the *center of the lower left* pixel. To generate a BIL header, this coordinate pair has to be transformed to the *center of the upper left* pixel. While GRASS also refers to cell centers, the GDAL library (`http://www.gdal.org/`) which is used to read the raw SRTM data refers to pixel *corners*. The tile size is one degree by one degree.

2. Probably more convenient are the larger SRTM tiles from (Global Land Cover Facility (GLCF; GLCF SRTM Website (2004)) which have been patched to WRS2 size in oder to match LANDSAT scene positions and sizes. Here the format is GeoTIFF.

SRTM data preparation and import into GRASS

First of all, a latitude-longitude location with WGS84 ellipsoid and geodetic datum is needed. If you don't have it, such a location can be easily generated from either the startup screen (use the EPSG code button in GRASS 6 and enter "4326" as code number along with a new location name) or from within an existing location (such as the Spearfish sample location) with following command:

```
g.proj -c proj4='+init=epsg:4326' location=latlong
```

The import of the SRTM data into this location then depends on the data source:

HGT files from NASA: Use the `r.in.srtm` script to import a SRTM `.hgt.zip` file. The script takes care of the relevant referencing and also automatically applies a color table.

GeoTIFF file from GLCF: Use `r.in.gdal` to import a SRTM GeoTIFF file. Additionally, the NULL (No Data) value has to be assigned afterward with `r.null`, otherwise the map won't be properly visible. Finally, you can assign a nice color table with `r.colors` (e.g., `rules=srtm`).

Note, that you can easily zoom to a couple of adjacent tiles with `g.region` as it accepts multiple raster files.

Void filling

Part of the post-processing is the filling of "no data" holes in many of the SRTM data tiles. These voids appear in regions with rugged terrain due to the SAR (SAR, Side Aperture Radar) data acquisition technique which was used to generate the SRTM data. Higher mountains shadow the radar signal which leads to holes in the DEM resulting from the interferometry. Another reason for voids are water bodies with poor reflectance of the RADAR signal.

The `r.fillnulls` script usually does an acceptable job at filling these holes. It extracts a ring of values around the holes, then it interpolates the values across the holes using the regularized splines with tension (RST) interpolation method (see the `v.surf.rst` manual for details). Then the closed holes are patched into the original data. The underlying idea is to leave as many values as possible untouched during the void filling. An example is shown in Fig. 1 and Fig. 2.

An alternative is `r.resamp.rst` which we use below to resample the SRTM DEM to higher resolution.

Peaks elimination

Besides holes, peaks and outliers also appear in the SRTM data. They are often artifacts of the interferometry processing. If you intend to use the SRTM data just for visualization or rendering, the use of some simple filters may be sufficient. But to make a hydrologically sound elevation model, more complex steps must be performed.

While outlier detection and removal techniques are virtually unlimited, we propose here just some simple examples. It is probably convenient to reproject the data set(s) to a metric projection first (`r.proj` within the target location; using `v.in.region` along with `v.proj` can be helpful to "find" the area of interest):

using r.neighbors: We can zoom to a SRTM tile and then locally calculate mean and standard deviation for a given moving window. Then we verify if a pixel deviates too much from twice the 3x3 standard deviation and, if so, replace it with the 3x3 mean value:

```
# Example: 3*90m = 270m box, small peaks
MAP=N46E011
g.region rast=$MAP -p
r.neighbors $MAP out=$MAP.mean \
  meth=average size=3
r.neighbors $MAP out=$MAP.stddev \
  meth=stddev size=3
r.mapcalc "$MAP.filt=if(abs($MAP - \
  $MAP.mean) > 2 * $MAP.stddev, \
  $MAP.mean, $MAP)"
r.colors $MAP.filt rast=$MAP
```

The differences can be calculated by subtracting the filtered from the original map and visualized in NVIZ for graphical inspection.

using r.resamp.rst: this module can be used to resample the SRTM data to higher resolution, e.g. to match a pan-sharpened LANDSAT-7 scene (pan-sharpen with `i.fusion.brovey`). It uses the regularized splines with tension (RST) interpolation method. Peaks will be smoothed and also data voids will be closed:

```
r.resamp.rst $MAP elev=$MAP.1425m \
  ns=14.25 ew=14.25 tension=100
g.region rast=$MAP.1425m -p
nviz $MAP.1425m
```

You can experiment with the tension parameter to minimize the "stair" artifacts in the resulting map. Further details to optimize the interpolation are given in Cebecauer et al. (2002).

Coastline extraction

The coastlines are not well indicated in SRTM data because of the limited Radar backscatter over water. To some extent this also applies to lakes and snow/ice.

There are at least two free vector map products which can be used to solve this problem: the Global, Self-consistent, Hierarchical, High-resolution Shoreline database (GSHHS; GSHHS vector data set (2004)) and the VMAP0 vector data (see next section).

Currently original GSHHS data cannot be imported into GRASS 6 (only into GRASS 5 with `v.in.gshhs`) but by using `v.convert` it can be converted from an existing GRASS 5 location. There is also a SHAPE file version of GSHHS available (GSHHS SHAPE format data set, 2004), which can be imported by using `v.in.ogr` but it was generated from an older GSHHS version. With `v.to.rast` a raster MASK can be generated at an appropriate resolution and applied with `r.mapcalc` to sharpen the SRTM coastlines. The GSHHS data as well as the VMAP0 data also contain larger lakes.

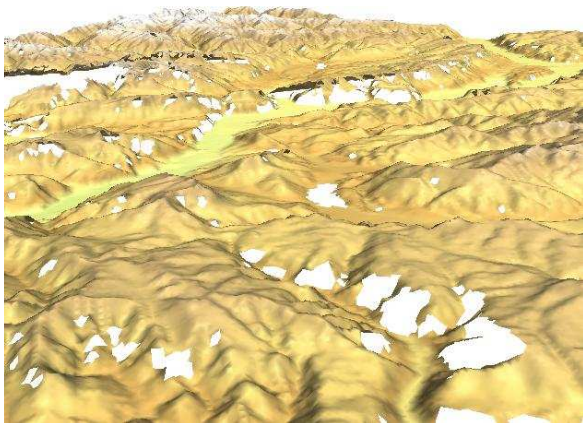


Figure 1: Original SRTM data (Trento/Italy region)

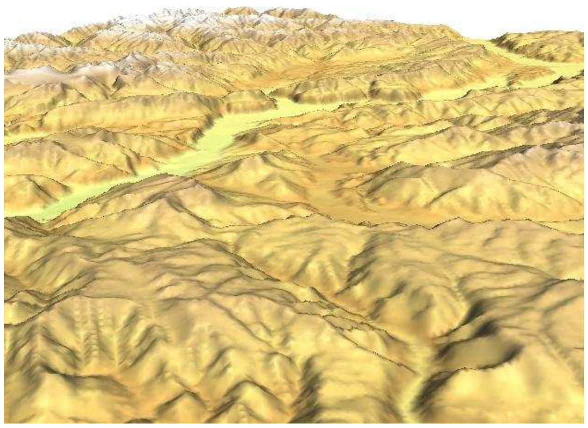


Figure 2: SRTM data filled with `r.fillnulls` (Trento/Italy region)

The VMAP0 vector maps

The Vector Map (VMAP) Level 0 data set developed by the NGA (National Geospatial Agency, formerly NIMA) was developed in the 1990s on top of the Digital Chart of the World (DCW). It it currently available in the fifth revision (VMAP-R5) from 2000. The VMAP0 vector data are produced at a map scale of 1:1,000,000. The data set consists of vector geometry, vector attributes, and further textual data. VMAP0 can be acquired from NGA either on four CDROMs or online as four big files (VPF/VMAP0 data set, 2000). It is divided into 10 themes consisting of 50-70 maps: boundaries, data quality, elevation, hydrography, industry, physiography, population, transportation, utilities, and vegetation. The world coverage is divided into four libraries based on geographic area:

- North America (NOAMER)
- Europe and North Asia (EURNASIA)
- South America, Africa and Antarctic (SOAMAFR)
- South Asia and Australia (SASAUS)

The map datum is North_American_Datum_1983 on a GRS80 ellipsoid (EPSG code 4269). In most cases a reprojection will be needed. VMAP0 country codes in the attribute tables can be expanded from DCW Data Dictionary (1993), p. 131, and WorldFactBook (2005).

Data preparation and import

The data access is provided through the OGR library (`http://www.gdal.org/ogr/`), which supports various vector formats. It must be compiled with the OGDI driver (`http://ogdi.sf.net`) to enable OGR to read the VMAP0 format. If you are not lucky to find a precompiled OGDI driver for your computer platform, you will have to compile the driver yourself. To compile the OGDI driver carefully read the README file included with the source code.

Some special operations are required to make the original VMAP0 file structure accessible to OGDI/OGR. The commands are indicated in Fig. 3. The syntax to access a VMAP0 layer is somewhat unusual (in general: `layer@theme(*)_type`), so their names must be entered carefully.

Only then OGR (`ogrinfo`, `ogr2ogr`, `v.in.ogr`, UMN Mapserver, QGIS) will be able to read these data. Note that `/vrf` has to be added to the path when accessing VMAP0 data via OGR. The conversion of VMAP0 layers with OGR tools is shown in Fig. 4. With `ogr2ogr` it is also possible to join full country names or other attributes to the existing attribute tables as well as extracting only vectors of interest.

The resulting SHAPE files can be imported into a GRASS 6 Latitude-Longitude/WGS84 location. To avoid filtering away tiny polygons, we redefine the `min_area` parameter to a smaller value:

```
MYAREA=sasaus
v.in.ogr polbnda_${MYAREA}_wgs84.shp \
   out=polbnda_${MYAREA}_wgs84 min_area=3.5e-09
g.region vect=polbnda_${MYAREA}_wgs84 -p
d.mon x0
d.vect -c polbnda_${MYAREA}_wgs84
```

Direct import of VMAP0 maps into GRASS

Instead of converting the maps to SHAPE or another format beforehand, you can also directly import original (but preprocessed for file names) VMAP0 data

```
# check OGR installation for OGDI driver:
ogrinfo --formats

# define path to VMAP0:
MYPATH=/path/to/vmap0

cd ${MYPATH}
echo "Removing trailing dot ..."
find -name '*.' > /tmp/vm_list1
cat /tmp/vm_list1 | sed 's+.$++g' > /tmp/vm_list2
paste /tmp/vm_list1 /tmp/vm_list2 | sed 's+^+mv +g' > rename.sh
# run the new script:
sh rename.sh
rm -f /tmp/vm_list1 /tmp/vm_list2 rename.sh

echo "Changing upper case directories to lower case names..."
find -type d | egrep "[[:upper:]]" > /tmp/vm_list1
cat /tmp/vm_list1 | tr "[[:upper:]]" "[[:lower:]]" > /tmp/vm_list2
paste /tmp/vm_list1 /tmp/vm_list2 | sed 's+^+mv +g' > rename.sh
# run the new script:
sh rename.sh
rm -f /tmp/vm_list1 /tmp/vm_list2 rename.sh
```

Figure 3: Shell code to fix the original VMAP0 file/directory names for OGDI/OGR usage.

```
# define VMAP0 map names and path:
MYAREA=sasaus
V0AREA=v0sas
MYPATH=/path/to/vmap0

# note: no trailing slash in next line:
MYDISK="/vrf/${MYPATH}/${V0AREA}/vmaplv0/${MYAREA}"

# info on political boundaries (polygon)
ogrinfo -ro -summary gltp:${MYDISK} 'polbnda@bnd(*)_area'

# convert/reproject political boundaries (polygon)
ogr2ogr -t_srs '+init=epsg:4326' \
  polbnda_${MYAREA}_wgs84.shp gltp:${MYDISK} 'polbnda@bnd(*)_area'
ogrinfo -summary polbnda_${MYAREA}_wgs84.shp polbnda_${MYAREA}_wgs84

# convert/reproject coastlines (line)
ogr2ogr -t_srs '+init=epsg:4326' \
  coastl_${MYAREA}_wgs84.shp gltp:${MYDISK} 'coastl@bnd(*)_line'
ogrinfo -summary coastl_${MYAREA}_wgs84.shp coastl_${MYAREA}_wgs84

# convert/reproject cities (polygon)
ogr2ogr -t_srs '+init=epsg:4326'  \
  builtupa_${MYAREA}_wgs84.shp gltp:${MYDISK} 'builtupa@pop(*)_area'
ogrinfo -summary builtupa_${MYAREA}_wgs84.shp builtupa_${MYAREA}_wgs84

# look at the maps with QGIS (http://www.qgis.org)
qgis *.shp
```

Figure 4: Shell code to convert VMAP0 maps to SHAPE file format with OGR tools.

into GRASS 6. This requires a Latitude-Longitude location with NAD83 geodetic datum which we can easily generate using the right EPSG code. To create a NAD83 location in GRASS 6, start GRASS and hit the "Create Location from EPSG" button. For the new location name we enter "vmap0nad83", as EPSG code we enter number "4269". If you never have used GRASS before, the database field must be filled as well, otherwise it will be predefined. Then click "OK" to create the location.

A terminal window will open and ask for "Datum Transformation Parameters". To list the available options, enter "list" (use space bar to scroll down, q to quit). We select "6" - "Used in Default nad83 region". Then you will be notified that GRASS closes itself after having created the NAD83 location. We start the software again and select the new "vmap0nad83" location.

After entering, we can verify the projection with `g.proj`:

```
g.proj -w
GEOGCS["grs80",
    DATUM["North_American_Datum_1983",
        SPHEROID["grs80",6378137,298.257222101],
        TOWGS84[0.000,0.000,0.000]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433]]
```

Now we can import the original VMAP0 maps directly:

```
MYAREA=sasaus
VOAREA=v0sas
MYPATH=/path/to/vmap0
MYDISK="/vrf/${MYPATH}/${VOAREA}/vmaplv0/${MYAREA}"

# List all layers:
v.in.ogr -l dsn="gltp:${MYDISK}" out=dummy

# Import of political boundaries
# at subnational level:
v.in.ogr dsn="gltp:${MYDISK}" \
  out=${MYAREA}_polbnda \
  layer='polbnda@bnd(*)_area'

# Show column names:
v.info -c ${MYAREA}_polbnda

# Display imported map:
g.region vect=${MYAREA}_polbnda -p
d.mon x0
d.vect -c ${MYAREA}_polbnda
```

To reproject the map(s) to another projection (e.g., Latitude-Longitude/WGS84) a separate location is needed. Within that location run `v.proj` to reproject the VMAP0 map(s) into this location.

Final remarks

The SRTM and VMAP0 data close an important gap in the availability of worldwide spatial data. This is of particular interest for countries where either spatial data are lacking or unaccessible due to political or economical constraints.

Acknowledgments

The author is grateful to Frank Warmerdam for his continuous support and availability to resolve OGDI/VMAP0 issues.

Bibliography

Cebecauer, T., J. Hofierka, and M. Suri, 2002
Processing digital terrain models by regularized spline with tension: tuning interpolation parameters for different input datasets. In B. Benciolini, M. Ciolli, and P. Zatelli, editors, *Proc. of the Open Source Free Software GIS – GRASS users conference 2002, Trento, Italy, 11-13 September 2002*, September 2002.
http://citeseer.ist.psu.edu/cebecauer02processing.html

EGM96 geoid model, 1996
http://earth-info.nga.mil/GandG/wgsegm/egm96.html

GSHHS vector data set, global shorelines, 9/2004
http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html

GSHHS SHAPE format data set, global shorelines in SHAPE format, 1/2004
http://www.ngdc.noaa.gov/mgg/shorelines/data/gshhs/gshhs_shp/

GLCF SRTM WRS2 tiles server, 2004
http://glcf.umiacs.umd.edu/data/srtm/

NASA SRTM original tiles server, 2004
ftp://e0mss21u.ecs.nasa.gov/srtm/

VPF/VMAP0 data set, Vector map Level 0 data set, 2000
http://earth-info.nga.mil/JavaGlobeStart.html or http://geoengine.nima.mil/geospatial/SW_TOOLS/NIMAMUSE/webinter/rast_roam.html

DCW Data Dictionary, 1993
http://www.lib.ncsu.edu/stacks/gis/dcwdoc.pdf

World Fact Book: Appendix D - Cross-Reference List of Country Data Codes, 2005
http://www.cia.gov/cia/publications/factbook/appendix/appendix-d.html

Markus Neteler
ITC-irst - SSI - MPBA
Via Sommarive, 18
38050 Povo (Trento)
Italy
http://mpa.itc.it
neteler AT itc it

Mapping freely available high resolution global elevation and vector data in GRASS

The volcanoes of Tongariro National Park

by M. Hamish Bowman

Introduction

In 1992 the US government's Defense Mapping Agency (DMA) released the Digital Chart of the World (DCW) which contained the most complete vector representations of the world's coastlines, roads, and so on available in the public domain. The DMA was later folded into the National Imagery and Mapping Agency (NIMA) which in turn is these days part of the National Geospatial-Intelligence Agency (NGA). In 1995 NIMA released an updated version of the DCW and renamed it Vector Smart Map level 0 (VMap0).

In February 2000 as part of a joint collaboration between NASA, NIMA, and the German and Italian space agencies, the Space Shuttle Endeavour carried out the Shuttle Radar Topography Mission (SRTM) which mapped the Earth's surface in unprecedented detail. In late 2004 the complete raw data from this mission was released to the general public on NASA's website. This dataset provides a quite remarkable leap in availability of topographic information for many remote parts of the world.

In this article we will see how these two datasets may be easily loaded and plotted in GRASS GIS 6. For example purposes, the area covered will be around Mount Ruapehu, a semi-active volcano in New Zealand's North Island; perhaps familiar to many as "Mount Doom" in the recent Lord of the Rings movies. Mount Ruapehu is located in the heart of Tongariro National Park[1], notable as the fourth National Park established worldwide and a tapu (sacred) place of the Māori people. UNESCO[2] has conferred rare dual World Heritage status on the Park in both cultural and environmental listings.

All GIS commands are given for GRASS 6.0 and are also available through the GRASS GUI menu system, although not listed here. More detailed instructions on the import and cleaning of these datasets can be found in the **preceding article** in this issue of *GRASSNews*. Further information about working with maps on a planetary scale can be found in a companion article by the author in *GRASSNews* volume 1[3].

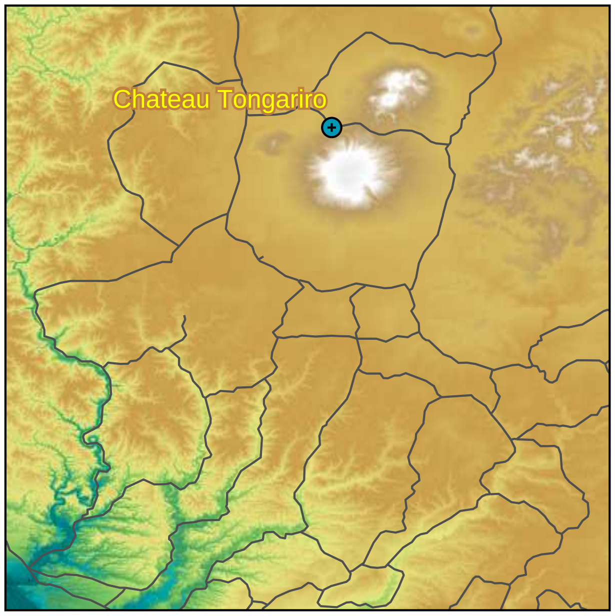


Figure 1: View of the Tongariro National Park SRTM tile overlain by VMap0 road data. Plotted with ps.map.

Obtaining the data

SRTM information is available from the Shuttle Radar Topography Mission homepage at NASA's JPL website:
`http://www2.jpl.nasa.gov/srtm/`
Data coverage for the United States is available at 1-arcsec (approx. 30m) resolution and the rest of the world up to 60° latitude is available at 3-arcsec (approx. 90m) resolution. The r.in.srtm module distributed with GRASS 6.0 is designed for the 3-arcsec data. For loading 1-arcsec data you will need to obtain the r.in.srtm script from the development version of GRASS. The raw data is distributed without restriction and may be downloaded from the following NASA FTP site:
`ftp://e0mss21u.ecs.nasa.gov/srtm/`
The data files are divided into separate directories on the FTP site by continental region and exist as compressed 1 degree square HGT files. Tongariro National Park is covered by the S40E175 (40° South latitude, 175° East longitude) tile in the "Islands" region:

[1] `http://www.doc.govt.nz/Explore/001~National-Parks/Tongariro-National-Park/index.asp`
[2] `http://whc.unesco.org`
[3] `http://grass.itc.it/newsletter/GRASSNews_vol1.pdf`

ftp://e0mss21u.ecs.nasa.gov/srtm/Islands/S40E175.hgt.zip
(1.7mb download)
The Mapability.com website provides an easy interface for downloading compressed versions of the 1:1 million scale VMap0 data from NIMA. The same site details copyright restrictions on the dataset, which is ostensibly released into the public domain.
http://www.mapability.com/info/vmap0_index.html
The VMap0 data is also split up into a number of continental regions; Australasia which we are interested in for this article is contained within the "sasaus" region.
http://geoengine.nima.mil/ftpdir/archive/vpf_data/v0sas.tar.gz
(240mb download)
The VMap0 dataset must be uncompressed after download and its directory structure sanitized as detailed in the **preceding article** in this issue of *GRASS-News*. The SRTM data is accessed by GRASS in its compressed form.

Importing into GRASS

Both SRTM and VMap0 data are distributed in Latitude-Longitude coordinates, although they use different map datums. If we blithely assume that the error from incorrect datum settings is smaller than the inherent error due to the resolution of the VMap0 data, we can load everything into a Lat-Lon location using the WGS84 datum (EPSG code #4326). If you wish to use the data for more than just visualization purposes and perform the import using the correct map datums, please see the instructions in the **preceding article** in this issue of *GRASSNews*. Users wishing to create high resolution maps of the United States may wish to use NOAA's Online Coastline Extractor[4] or the U.S. Census Bureau's high resolution TIGER[5] maps instead of the VMap0 data, in addition to using the higher resolution SRTM-1" elevation data.

SRTM

The r.in.srtm module is used to import SRTM data. As the raw data file may contain some holes and other artifacts we note this in the output map name.

```
GRASS > r.in.srtm input=S40E175 \
    output=S40E175.raw
```

To correct any holes in our new raster map, we clean the data with the r.fillnulls module after adjusting the region settings to match the bounds of the imported map.

```
GRASS > g.region rast=S40E175.raw
GRASS > r.fillnulls input=S40E175.raw \
    output=S40E175
```

To save disk space we can now remove the raw version of the map.

```
GRASS > g.remove rast=S40E175.raw
```

The SRTM data can now be displayed in the GRASS display monitor with the following commands:

```
GRASS > d.mon x0
GRASS > d.rast S40E175
```

If you don't mind getting your hands dirty, you can make the coastline appear a bit crisper without modifying the underlying data by editing the map's color table. It can be found in the `$MAPSET/colr/` directory; with a text editor change the two 0 elevation rules to 10. (elevation is the left most number of each grouping, the others are red, green, and blue intensities)

VMap0

Loading VMap0 data into GRASS is achieved with the v.in.ogr module. The OGR[6] library must be compiled with support for the OGDI[7] driver. Detailed instructions on setting up OGR with OGDI support is listed in the **preceding article** in this issue of *GRASS-News*. You can check that the OGDI driver is active by running the following command at the shell prompt and making sure that OGDI is listed.

```
$ ogrinfo --formats
```

To load the data, we first set up the directory path required for the OGDI driver by adding "`gltp:/vrf`" to the directory structure. For example if the data is located in `/var/local/topodata/`, as a shortcut we can setup a shell variable containing this data path in the format that the OGDI driver requires:

```
GRASS > VRF="gltp:/vrf/var/local/topodata/\
    vmap0/v0sas/vmaplv0/sasaus"
```

We can then check for available layers with v.in.ogr's `-l` flag. To make the output easier to read, we trade commas for newlines with the UNIX "`tr`" command.

```
GRASS > v.in.ogr -l dsn="$VRF" output=dummy \
    | tr ',' '\n'
```

Data layers may also be listed with the ogrinfo program.

```
$ ogrinfo -ro "$VRF"
```

[4] http://www.ngdc.noaa.gov/mgg/shorelines/shorelines.html
[5] http://www.census.gov/geo/www/tiger/index.html
[6] http://www.gdal.org/ogr/
[7] http://ogdi.sourceforge.net

For this example we will load in maps of the coastline, roads, and built-up areas. These are contained in the coastl@bnd, roadl@trans, and builtupa@pop map layers respectively. A full explanation of the of various data layers may be found at the following website:
http://www.terragear.org/docs/vmap0/coverage.html
In addition to restricting input to a few specific data layers, to save space we will restrict the spatial coverage to the area of New Zealand during import by using v.in.ogr's spatial option. Here we set up another shell variable shortcut specifying the western, southern, eastern, and northern bounds as detailed in the v.in.ogr help page.

```
GRASS > AREA="160,-50,180,-30"
```

Now we are ready to import the data:

```
# coastline:
GRASS > v.in.ogr dsn="$VRF" spatial=$AREA \
    layer='coastl@bnd(*)_line' \
    output=coastl_vmap0

# roads:
GRASS > v.in.ogr dsn="$VRF" spatial=$AREA \
    layer='roadl@trans(*)_line' \
    output=roadl_vmap0

# built-up areas:
GRASS > v.in.ogr dsn="$VRF" spatial=$AREA \
    layer='builtupa@pop(*)_area' \
    output=builtupa_vmap0
```

The vector data can be displayed in the GRASS monitor with the d.vect command.

```
GRASS > d.vect roadl_vmap0
```

Point data

In addition to formal datasets we can import map data from a local map or GPS position with the v.in.ascii module. For example, the classic Chateau Tongariro, built in 1929, is perched at the base of the mountain at approximately 39.203°S 175.540°E.

```
GRASS > echo "175.540|-39.203|Chateau Tongariro" \
     | v.in.ascii output=chateau
```

Additionally, by using v.in.ascii in "standard" mode, polygon and line features such as the ski field lift may be imported and displayed.

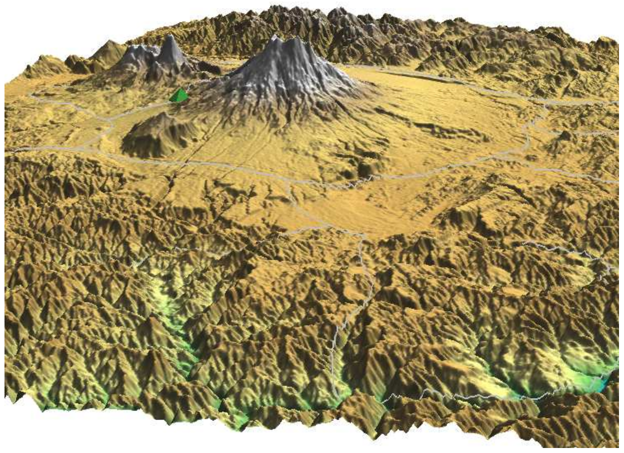


Figure 2: 3D view of Mount Ruapehu and Mount Ngauruhoe created with NVIZ.

Displaying the maps

After starting a display monitor with d.mon, the maps can be viewed using the respective display modules, d.rast for raster maps and d.vect for vector maps (including site points) as shown in figure 1. If you have been doing other work in the mapset you may have to reset the region bounds first with g.region.

```
GRASS > d.mon x0
GRASS > g.region rast=S40E175
```

Now we display the maps.

```
GRASS > d.rast S40E175
GRASS > d.vect coastl_vmap0 color=blue
GRASS > d.vect roadl_vmap0 color=black
GRASS > d.vect builtupa_vmap0 type=area \
    color=none fcolor=200:200:200
GRASS > d.vect chateau color=red icon=basic/point
```

We can use d.vect to add some labels too.

```
GRASS > d.vect builtupa_vmap0 display=attr \
    attrcol=nam
GRASS > d.vect chateau display=attr \
    attrcol=str_1 xref=right lcolor=yellow
```

You can use d.legend to add a colorbar legend and d.text to draw text on the display. The d.font.freetype module may be used to set a nice font before drawing the labels.

```
GRASS > d.legend map=S40E175 at=50,95,2,5 \
    range=0,2750 label=6
GRASS > echo "meters" | d.text at=15,76 \
    color=black size=3
```

To create a 3D view, we need to scale the map into consistent map units in all dimensions. You can either reproject the SRTM data into a meter-grid projection to match the elevation units or scale the elevation data to match the horizontal units, here degrees. It is probably best to reproject the data with the r.proj module, but for simplicity's sake in this article we will stay in a latitude-longitude location and rescale the z-axis. We use r.mapcalc to create the scaled map.

```
GRASS > r.mapcalc \
    'S40E175.scaled=S40E175 / (60 * 1852.0)'
```

Next we need to set some new color rules to match our scaling. With some hand calculations from the "srtm" color rules and a little experimentation we set the following rules for our scaled map:

```
GRASS > r.colors S40E175.scaled color=rules << EOF
    0% black
    3% aqua
    3%  57:151:105
    0.0009  117:194:93
    0.0018  230:230:128
    0.0045  202:158:75
    0.0090  214:187:98
    0.0120  185:154:100
    0.0160 white
    100% white
EOF
```

Now we can use NVIZ to visualize the data in 3D as shown in figure 2.

```
GRASS > nviz elev=S40E175.scaled vector=roadl_vmap0
```

A fly-by movie can be constructed with the d.nviz module. See the help page for more information.

If you would like to create a shaded relief image, you can process the unscaled SRTM map with the r.shaded.relief module:

```
GRASS > r.shaded.relief map=S40E175 \
    shadedmap=S40E175.shade units=meters
```

Hardcopy output

You can save the display window to a graphics file with the d.out.png script or use ps.map to create a PostScript file for printing. An example ps.map command list follows:

```
GRASS > g.region res=0:00:03
GRASS > ps.map output=ruapehu.ps << EOF
paper a4
   end
raster S40E175
vpoints chateau
   symbol basic/cross1
   size 4
   end
vpoints chateau
   fcolor orange
   symbol basic/circle
   size 10
   end
vlines roadl_vmap0
   end
text 10% -7% Tongariro National Park
# size is in map units (degrees)
   size 0.07
   ref center left
   end
mapinfo
   where 5.0 9.5
   end
end
EOF
```

In Conclusion

GRASS provides a powerful and versatile platform for loading, displaying, and analyzing many forms of cadastral, remote-sensing, and ground determined (e.g. GPS) datasets from disparate sources in a common geographic framework. With the advent in the last six months of world-wide high resolution SRTM elevation data, combined with other datasets freely available to the public such as the VMap0 vectors, the opportunities for new analysis and mapping of previously unsurveyed areas are fresh and many. When combined with Free and open software the barriers to entry are lowered to include anyone with a computer and Internet connection. With the extraordinary detail of the new datasets and their widespread availability, we can expect an unprecedented opening of new frontiers in geographic analysis which had previously been impossible. These are exciting times for both professional and part-time geographers from all disciplines. Have fun!

Acknowledgments

Thanks is due to Markus Neteler on the GRASS side and Frank Warmerdam on the GDAL, OGR, and ODGI side for getting VMap0 and SRTM data to load successfully into GRASS. The fact that it is working today is a direct result of their hard work over the past several years. We must also thank the good folks who produced this data and keep it available on-line, as well as all those who have donated their energy and time to the GRASS GIS development effort.

M. Hamish Bowman
Department of Marine Science
University of Otago
Dunedin, New Zealand
`hbowman (at) albers.otago.ac.nz`

Interfacing GRASS 6 and R

Status and development directions

by Roger Bivand

(*The following commands work on Linux/Unix only; other platforms are under active development - progress will be reported on the STATGRASS mailing list*[8] *and the R-spatial SourceForge project homepage*[9]*. Please visit these sites as well if you are interested in contributing to the development (Ed.).*)

Introduction

As Wegmann and Lennert (2005) show in the 2004 GRASS user survey, users of GRASS have found the interface between GRASS 5 and the R data analysis programming language and environment of at least some value. The R interface was mentioned by 119 respondents (40 %) in (Wegmann and Lennert, 2005, p. 15, Fig. 57), so that, despite the command line interface and syntax complications of R, the interface has served some purposes (see for example Grohmann (2004)). The interface as documented first in Bivand (2000), and fully in Neteler and Mitasova (2004), works with GRASS releases up to and including GRASS 5.4.0. This note is intended to show which design choices may be made when interfacing GRASS 6 and R, and how much progress has been made so far. The basic website for the new interface is shared by other R spatial packages, and is hosted at SourceForge[9].

The GRASS 5 interface to R is an R contributed package, and is available from CRAN, the Comprehensive R Archive Network as described by (Neteler and Mitasova, 2004, section 13.2). It ships with a snapshot of a subset of source files from the core GRASS libraries. Some of these files have been modified to suit the changed setting of a continuously open interface, and cannot readily be merged back into the main code base. When work on the GRASS 5 interface began, both computer speed, memory, and disc capacity were much greater problems than at present, and it was important to move from running GRASS through the R `system()` function to accessing GRASS data directly.

Things have changed, and a fresh start seems attractive for the interface between R and GRASS 6. The new package, named **spgrass6**, is being hosted on the R-spatial sourceforge site, and can be accessed using new mechanisms for managing package repositories in R 2.1 and later. The package will also use as many other contributed R packages as seems sensible, for example **sp** for spatial data object classes, and **spGDAL** as a wrapper for functions in the **rgdal** package.

Installing the interface package

While the GRASS 5 interface is released on CRAN, the GRASS 6 interface will, at least for the foreseeable future, be available from the sourceforge site. The site will contain details of other contributed packages that may be needed, both those released on CRAN, and those on R-spatial. The interface package at present depends on two packages, and these (**sp**, **rgdal**) should be installed first from CRAN, before **spgrass6** is installed from R-spatial (assumining that the workstation is online):

```
> install.packages(c("sp", "rgdal"),
+     dependencies = TRUE)
> rS <- "http://r-spatial.sourceforge.net/R"
> install.packages("spgrass6", repos = rS)
> install.packages("spGDAL", repos = rS)
```

This method of installation works since R 2.1, released 19 April 2005. At present only source packages are available from R-spatial, but Windows binary packages will follow. Because OS X is now so similar to Linux and other Unix versions, no binaries will be distributed; installation from source may depend on external libraries, such as GDAL and PROJ.4, but since GRASS 6 itself mandates these, this dependence is not seen as a major problem.

The sp package

The **sp** package contains defintions of classes for spatial objects, and permits much of the interface to be simplified. In particular, using these classes means that no special display functions are required. The classes provide "slots" for projection and other region or window defining structural data, and to a certain extent take over the role of the `grassmeta` object from the GRASS 5 interface.

The package is being developed by a team of authors and maintainers of R contributed packages for spatial data analysis, including Edzer Pebesma, Paulo J. Ribeiro, Barry Rowlingson, Virgilio Gómez Rubio, and Roger Bivand. We are very grateful for any suggestions, bug reports, and other ideas to help to improve our work.

It currently provides a foundation `Spatial` class, with a bounding box and a projection.

[8] http://grass.itc.it/statsgrass/index.php
[9] http://r-spatial.sourceforge.net

These slots are inherited by all the other classes. SpatialPoints is a Spatial object with coordinates, and SpatialPointsDataFrame is a SpatialPoints object with data in a data frame slot, one row of data per point coordinate; SpatialPoints must have at least 2 dimensions. Data frames are the main workhorse of computing in R, they are very much like data base tables, where the number of rows (records) must be the same for each column, but the columns (fields) can vary by type.

Raster data are handled in the SpatialPixels and SpatialGrid objects, and their associated *DataFrame objects. The difference between the two is that while a SpatialGrid object has to be a full, rectangular grid, a SpatialPixels object can be incomplete. Both have a grid slot with a GridTopology object defining the grid itself.

Vector data of more complex types than point coordinates are handled as lines or rings, in hierarchies of objects. The top-level objects are SpatialLines and SpatialRings, and can be bound to object attributes in data frames, one data frame row per member of the SpatialLines or SpatialRings objects. The SpatialLines and SpatialRings objects are built of lists of SLines and SRings objects, which themselves are lists of atomic SLine and SRing objects. No topology checking is done in **sp** on these objects. There are wrapper functions for importing shapefiles (using the **maptools** package) and e00/ArcInfo v.7 binary coverages (using the **RArcInfo** package), and for exporting shapefiles (using **maptools**).

Using graphics with sp objects

Display functions are provided for **sp** classes using both base and lattice graphics. The motivation for this is to make it possible for interface and data import/export packages, and data analysis packages, to concentrate on their core functions. This provision means that, while the legacy interface to GRASS 5 needed to provide plotting functionality, the new interface can rely on it being provided by **sp** if GRASS data is held in R in **sp** classes. The **sp** display facilities are under active development, with help from the developers of lattice/trellis graphics in R. Lattice graphics are analytic graphics more than presentation graphics, using panelled displays to explore the way different variables condition relationships. A typical example would be to explore anisotropy in variogram displays by panels showing different directions; much of this has been accomplished in Edzer Pebesma's **gstat** package (Pebesma, 2004).

Using the spgrass6 package with raster data

Provided that the **spgrass6** package has been installed following the packages it depends upon, **sp** and **rgdal**, the interface is used more or less as before. R is started from within a GRASS session from the command line, and the **spgrass6** loaded with its dependencies:

```
> library(spgrass6)
Loading required package: rgdal
Loading required package: abind
Loading required package: pixmap
Loading required package: sp
```

The sample location used here is Spearfish, with the default region settings:

```
> system("g.region -p")
projection: 1 (UTM)
zone:       13
datum:      nad27
ellipsoid:  clark66
north:      4928010
south:      4913700
west:       589980
east:       609000
nsres:      30
ewres:      30
rows:       477
cols:       634
```

At the present stage of the new interface, raster data transfer is done layer by layer, and uses temporary ASCII files in Arc ASCII grid format. The following command reads the soil pH values into a SpatialGridDataFrame object, treating the values returned as floating point (double in R):

```
> soilsph <- readFLOAT6sp("soils.ph")
> summary(soilsph)
Object of class SpatialGridDataFrame
Coordinates:
              min     max
coords.x1  589995  608985
coords.x2 4913715 4927995
Is projected: TRUE
proj4string : [+proj=utm]
 proj4string : [+zone=13]
 proj4string : [+a=6378206.4]
 proj4string : [+rf=294.9786982]
 proj4string : [+no_defs]
 proj4string : [+nadgrids=conus]
Number of points: 2
Grid attributes:
  cellcentre.offset cellsize cells.dim
1            589995       30       634
2           4913715       30       477
Data attributes:
    soils.ph
 Min.   :   0.00
 1st Qu.:   3.00
 Median :   4.00
 Mean   :   3.29
 3rd Qu.:   4.00
 Max.   :   5.00
 NA's   :9927.00
```

Development will move towards binary transfer, and to using GDAL. As this example shows, however, GDAL using the GRASS plugin accesses and transfers the data using its region and resolution settings, not the current region:

```
> library(spGDAL)
> gmeta6 <- gmeta6()
> fname <- paste(gmeta6$GISDBASE, gmeta6$LOCATION_NAME,
+     "PERMANENT", "cellhd", "soils.ph",
+     sep = "/")
> soilsph_GDAL <- read.GDAL(fname)
```

```
> summary(soilsph_GDAL)
```

```
Object of class SpatialGridDataFrame
Coordinates:
      min     max
x  590010  608990
y 4914010 4927990
Is projected: NA
proj4string : [NA]
Number of points: 2
Grid attributes:
  cellcentre.offset cellsize cells.dim
x            590010       20       950
y           4914010       20       700
Data attributes:
     band1
 Min.   :   0.000
 1st Qu.:   3.000
 Median :   4.000
 Mean   :   3.291
 3rd Qu.:   4.000
 Max.   :   5.000
 NA's   :6717.000
```

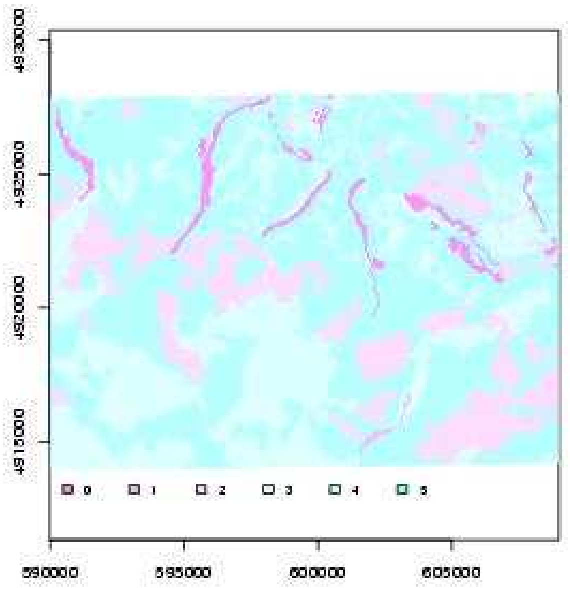


Figure 1: Soil pH values for Spearfish

We can display the data using an image function for the data object class; the result is shown in Figure 1:

```
> table(soilsph$soils.ph)
```

```
    0      1      2      3      4      5
 7823    698  45672  84102 153051   1145
```

```
> image(soilsph, "soils.ph", col = rev(cm.colors(6)))
> legend(c(590000, 605000), c(4912570,
+     4913850), legend = 0:5, fill = rev(cm.colors(6)),
+     cex = 0.8, bty = "n", horiz = TRUE)
```

A feature of the legacy interface was the ability to move category labels to R; this is at present emulated by matching the labels reported by GRASS using `r.stats -l` to the integer values present in the raster data. This is implemented in the `readCELL6sp` function, when the `cat=` argument is set to `TRUE`. As before, category layers are represented in R as `factor` columns:

```
> dem <- readFLOAT6sp("elevation.dem")
> lcov <- readCELL6sp("landcover.30m",
+     cat = TRUE)
```

```
> summary(lcov)
```

```
Object of class SpatialGridDataFrame
Coordinates:
              min     max
coords.x1  589995  608985
coords.x2 4913715 4927995
Is projected: TRUE
proj4string : [+proj=utm]
 proj4string : [+zone=13]
 proj4string : [+a=6378206.4]
 proj4string : [+rf=294.9786982]
 proj4string : [+no_defs]
 proj4string : [+nadgrids=conus]
Number of points: 2
Grid attributes:
  cellcentre.offset cellsize cells.dim
1            589995       30       634
2           4913715       30       477
Data attributes:
                      landcover.30m
 Evergreen Forest            :135375
 Grasslands/Herbaceous       : 67396
 Pasture/Hay                 : 56263
 Small Grains                :  9023
 Emergent Herbaceous Wetlands:  5673
 (Other)                     : 18587
 NA's                        : 10101
```

Checking the imported data against the original counts, we can see that the numbers seem to agree.

```
> system("r.stats -lc landcover.30m")
```

```
11 Open Water 30
21 Low Intensity Residential 3370
22 High Intensity Residential 239
23 Commercial/Industrial/Transportation 1444
31 Bare Rock/Sand/Clay 26
32 Quarries/Strip Mines/Gravel Pits 1531
41 Deciduous Forest 5568
42 Evergreen Forest 135375
43 Mixed Forest 379
51 Shrubland 1394
71 Grasslands/Herbaceous 67396
81 Pasture/Hay 56263
82 Row Crops 1058
83 Small Grains 9023
85 Urban/Recreational Grasses 260
91 Woody Wetlands 3288
92 Emergent Herbaceous Wetlands 5673
* no data 10101
```

Figure 2 shows how much of the functionality of the legacy raster interface has been maintained, by relating elevation and landcover categories in the same way as Neteler and Mitasova (2004) relate soil types and elevation (pp. 339–340):

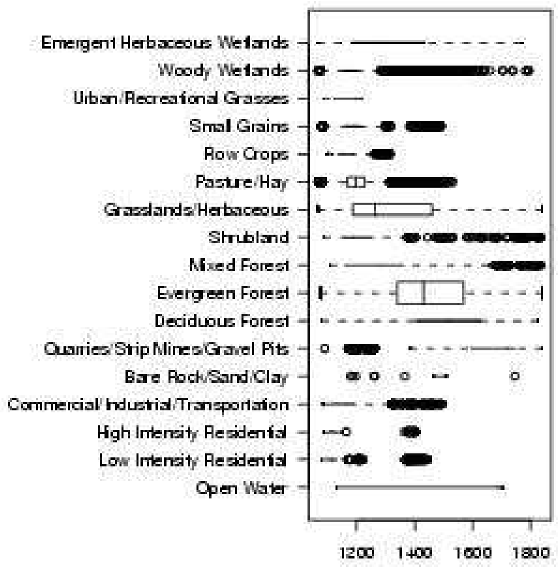


Figure 2: Boxplot of elevation by landcover types in the Spearfish region; the box widths represent the relative areal contributions of the landcover categories.

```
> lcovareas <- table(lcov$landcover.30m) *
+     ((gmeta6$ewres * gmeta6$nsres)/10000)
> boxplot(dem$elevation.dem ~ lcov$landcover.30m,
+     medlwd = 1, horizontal = TRUE,
+     las = 1, width = lcovareas)
```

A function `writeRast6sp` for moving numeric raster layers to GRASS from R has also been provided; this uses `r.in.gdal` and again the Arc ASCII grid format, so care is needed to differentiate between integer and floating point rasters if the first thousand data items appear to be integer.

Using the spgrass6 package with vector data

Following suggestions by Miha Staut and others, some progress has been made on interfacing GRASS 6 vector data. The package provides some skeleton functions for reading and writing point data using ASCII tables, but here we will look at using `v.out.ogr` and the `read.shape` function in the **maptools** package. To move the bugsites points layer to R, we first export it as a shapefile, next read it into R, and finally insert the imported values into a `SpatialPointsDataFrame` object:

```
> tDir <- tempdir()
> system(paste("v.out.ogr bugsites dsn=",
+     tDir, " olayer=bugsites type=point",
+     sep = ""))
> library(maptools)
> res <- read.shape(paste(tDir, "bugsites.shp",
+     sep = "/"))
> crds <- matrix(Map2points(res), ncol = 2)
> p4s <- CRS(gmeta6$proj4)
> bugsDF <- SpatialPointsDataFrame(coords = crds,
+     proj4string = p4s, data = res$att.data)

> summary(bugsDF)

Object of class SpatialPointsDataFrame
Coordinates:
              min     max
coords.x1  590232  608471
coords.x2 4914096 4920512
Is projected: TRUE
proj4string : [+proj=utm]
 proj4string : [+zone=13]
 proj4string : [+a=6378206.4]
 proj4string : [+rf=294.9786982]
 proj4string : [+no_defs]
 proj4string : [+nadgrids=conus]
Number of points: 90
Data attributes:
      cat                  str1
 Min.   : 1.00   Beetle site:90
 1st Qu.:23.25
 Median :45.50
 Mean   :45.50
 3rd Qu.:67.75
 Max.   :90.00
```

As can be seen, these commands could be encapsulated into a function, and the `*.prj` file used to secure the correct projection data — these are steps that will occur as the **spgrass6** package develops. Going a little further, a line layer showing stream centerlines can be moved to R, and converted to a `SpatialLinesDataFrame` object:

```
> tDir <- tempdir()
> system(paste("v.out.ogr streams dsn=",
+     tDir, " olayer=streams type=line",
+     sep = ""))
> res <- read.shape(paste(tDir, "streams.shp",
+     sep = "/"))
> streams <- shp2SLDF(res, proj4string = p4s)

> summary(streams)

Object of class SpatialLinesDataFrame
Coordinates:
         min       max
r1  589443.1  609526.8
r2 4913935.5 4928059.2
Is projected: TRUE
proj4string : [+proj=utm]
 proj4string : [+zone=13]
 proj4string : [+a=6378206.4]
 proj4string : [+rf=294.9786982]
 proj4string : [+no_defs]
 proj4string : [+nadgrids=conus]
Data attributes:
      cat         label
 Min.   :1.000   NA's:104
 1st Qu.:2.000
 Median :2.000
 Mean   :1.808
 3rd Qu.:2.000
 Max.   :3.000
```

Having moved several layers into R, we can display them together by placing the streams and bugsites vector layers on the elevation raster layer:

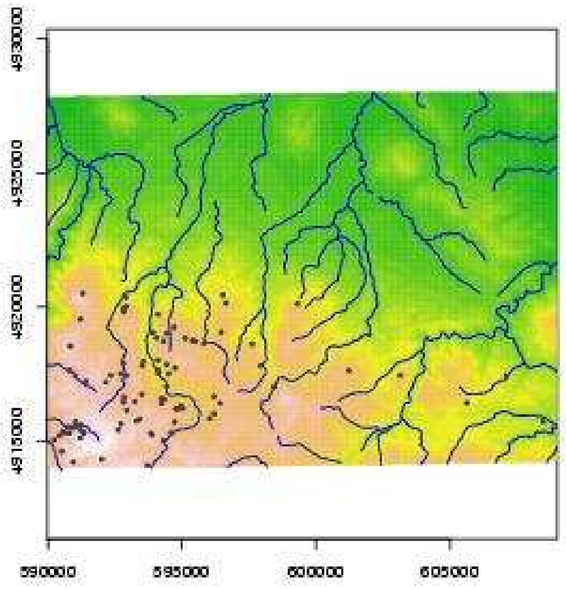


Figure 3: Spearfish elevation map with stream centerlines and bugsites.

```
> image(dem, "elevation.dem", col = terrain.colors(20))
> plot(streams, col = "blue", add = TRUE)
> points(coordinates(bugsDF), pch = 19,
+     col = "grey25", cex = 0.5)
```

Vector layers may be exported using functions from the **maptools** package, and read in using `v.in.ogr`; here we sample 200 points from the current region as represented by the elevation DEM layer, and move them to GRASS:

```
> dem_smple <- spsample(as(dem, "SpatialGrid"),
+     200, "random")
```

```
> summary(dem_smple)
```

```
Object of class SpatialPoints
Coordinates:
         min       max
xc  590052.8  608961.9
yc 4913776.2 4927931.0
Is projected: TRUE
proj4string : [+proj=utm]
 proj4string : [+zone=13]
 proj4string : [+a=6378206.4]
 proj4string : [+rf=294.9786982]
 proj4string : [+no_defs]
 proj4string : [+nadgrids=conus]
Number of points: 199
```

```
> tDir <- tempdir()
> crds <- coordinates(dem_smple)
> write.pointShape(data.frame(ID = 1:nrow(crds)),
+     paste(tDir, "sp_dem", sep = "/"),
+     crds)
> system(paste("v.in.ogr -o dsn=", tDir,
+     " output=sp_dem layer=sp_dem type=point",
+     sep = ""))
```

```
> system("v.category sp_dem option=report")
```

```
LAYER 1:
type          count          min          max
point           199            1          199
line              0            0            0
boundary          0            0            0
centroid          0            0            0
area              0            0            0
all             199            1          199
```

There remain many open questions concerning vector transfer, including the current region issue raised with using GDAL and raster layers, and how to associate projection data properly between the two systems. But at least using OGR mechanisms, the same level of functionality that GDAL provides or will provide for raster should be feasible.

Conclusion

The use of classes defined in the **sp** package will make the construction of an interface between GRASS 6 and R more robust than the legacy interface. The new package will wrap `system()` calls to GRASS commands in R code to check and control the conversion process. Because the **sp** class objects have or will soon have a range of display methods, and coersion methods to class types used in other packages for analysing spatial data, these do not need to be part of the interface package. Reports on problems and bugs, and suggestions for the interface package may best be raised on the STATGRASS mailing list[10]. Further work will attempt to streamline data transfer using the current region and resolution model, based on shared use of GDAL, OGR, and PROJ.4 in both GRASS and R.

Bibliography

Bivand, R. S., (2000) Using the R statistical data analysis language on GRASS 5.0 GIS data base files. *Computers and Geosciences*, 26, 1043–1052.

Grohmann, C. H., (2004) Morphometric analysis in geographic information systems: applications of free software GRASS and R *Computers and Geosciences*, 30, 1055–1067.

Neteler, M. and Mitasova, H., (2004) *Open Source GIS: A GRASS GIS Approach. Second Edition.* Kluwer Academic Publishers/Springer, Dordrecht.

Pebesma, E.J., (2004) Multivariable geostatistics in S: the gstat package. *Computers and Geosciences*, 30: 683-691.

Wegmann, M. and Lennert, M., (2005) GRASS User Survey 2004 *GRASS-News*, 2, 2–16.

Roger Bivand
Economic Geography Section, Department of Economics,

[10] http://grass.itc.it/statsgrass/index.php

Norwegian School of Economics and Business Administration, Bergen, Norway
http://r-spatial.sourceforge.net
Roger.Bivand AT nhh.no

Use of R tightly coupled to GRASS for correction of single-detector errors in EO1 hyperspectral images

by Guido Lorenz

Update

The method described in this article does not work for GRASS > 5.4. Please find below a solution for GRASS versions after 5.4.

Implementation of image correction procedure in GRASS 6.1cvs, coupled with R version 2.01 (under Mac OSX)

Image correction was possible only under concomitant use of GRASS 6.1cvs and GRASS 5.x, because of two specific errors in GRASS 6.1:

1. "region-settings" problem:
   - R was started from within GRASS6.1cvs and GRASS library loaded
   - GRASS region settings were tried to be read into the R-object "G", resulting in the following error message:
     ```
     G <- gmeta()
     Error in gmeta():
     region for current mapset is invalid
     line 11: <top:        1>
     run "g.region"
     ```
   - running "g.region" as suggested by R, from the GRASS 6.1 menu, gives a correct output without error messages, but does not correct the behaviour of the R command:
     ```
      projection: 0 (x,y)
     zone:        0
     north:       1024
     south:       0
     west:        0
     east:        256
     nsres:       1
     ewres:       1
     rows:        1024
     cols:        256
     ```
   - workaround: starting GRASS 5.x simultaneously, using the same location and running the "g.region" command, corrects the behaviour: executing "G <- gmeta()", from within GRASS 6.1cvs in the formerly started session, is possible and "summary(G)" gives back the correct region settings;
2. "r.recode" problem
   - The r.recode-procedure in GRASS is necessary as a final step after the image correction with R. Nevertheless, when executing "r.recode" from the console in GRASS6.1cvs, a menu of the graphic shell is automatically invoked. Unfortunately, in this menu, not all parameters may be given (recoding rules), and running the command is aborted with an error.
   - workaround: Execution of "r.recode" from within GRASS 5.x (console).

Abstract

The outlined procedure is a simple example of a smooth interaction between the GRASS GIS software and the R statistical environment (Ihaka and Gentleman, 1996), showing data interchange and manipulation of single vector elements out of an image matrix.

Introduction

Hyperspectral satellite images from the EO1-Hyperion platform provide 198 valid bands of 10nm bandwidth, covering a spectral range from 430 to 2400nm (Barry, 2001a). Although GRASS software has not offered specific routines for hyperspectral image analysis until now, GRASS was used as a base platform to prepare these images prior processing in other, more specific software packages.

One of the problems to deal with is the occurrence of detector failures of the Hyperion push-broom sensor, resulting in black vertical streaks of 1-pixel width in certain image bands. This may affect the spectral analysis, if those black pixels are interpreted as real reflectance values.

According to Barry (2001), this problem can be easily corrected by substitution of the erroneous pixel column by one of the arithmetic mean of the two adjacent ones. As this requires addressing of individual column vectors out of the complete image matrix, the R statistical environment seems to be an excellent tool to deal with this task, because of its vector- and object-based nature and the possibility of its tight coupling with GRASS (Bivand and Neteler, 2000).

In the following section, a brief description of a correction procedure is given, showing elemental steps of interaction between the two software packages:

Correction of single-detector errors

If the image is already imported into GRASS and the problematic bands and pixel columns are identified, the correction procedure involves the following steps: GRASS is invoked, selecting the corresponding location of the raw image bands (not georeferenced). General region settings have to be checked and adjusted to the total image coverage (`g.region` module). From within GRASS, at terminal level, R is started and the GRASS library is loaded in order to get some specific commands for tightly working with the GIS-software:

```
\GRASS 5.7. > R
> library(GRASS)
  \GRASS environment variables in:
  /tmp/grass57-glorenz-549/gisrc
```

Then, the GRASS environment settings are retrieved with `gmeta()` and assigned to an object under R. The actual settings are controlled with the `summary()` command and compared with the region information formerly checked under the GRASS shell.

```
> G <- gmeta()
> summary(G)
  Data from GRASS 5.0 LOCATION Hyp317_2002a
  with 256 columns and 856 rows;
  x,y
  The west-east range is: 1, 257,
  and the south-north: 1, 857;
  West-east cell sizes are 1 units,
  and south-north 1 units.
```

A specific image band, known to contain a single-sensor error, is read in with the `rast.get()` command and assigned to an R object. In order to facilitate identification of objects, two-component, lowercase object names are used in the following manner: the first name element consists of a single letter, used up from 'a' in alphabetical order, and combined with a second, descriptive element, separated by a period (e.g. c.matrix, a.vector, . . .). The characteristics of the newly created object are checked using the `summary()`,`mode()` and `dim()` commands. Note, that the image band is imported as a numeric object of type list, with no dimension.

```
> a.rawband <- rast.get(G, rlist="band_name",
  catlabels=FALSE, debug=FALSE, interp=FALSE)
> summary(a.rawband)
                 Length &  Class &  Mode
  band_name & 219136 & -none- & numeric
> mode(a.rawband)
  [1] "list"
> dim(a.rawband)
  NULL
```

Now, this dimensionless object has to be transformed to a matrix corresponding to the image dimensions. This is achieved by first transforming the list into a vector, and then reordering the vector into a matrix with the number of rows (`G$Nrow`) and columns (`G$Ncol`) of the corresponding region settings shown above. Care should be taken in specifying `byrow=T` in the matrix creation step, forcing R to read in the values by first filling rows, not columns.

```
> b.vector <- unlist(a.rawband)
> mode(b.vector)
  [1] "numeric"
> dim(b.vector)
  NULL
> c.matrix <- matrix(b.vector, G$Nrow,
  G$Ncol, byrow=T)
> dim(c.matrix)
  [1] 856 256
```

In order to control the correct importation, the erroneous column vector (in this case, column number 92), corresponding to the single-detector error, is printed out: the whole vector contains zero values. The same procedure may be undertaken for the two adjacent columns (91, 93), in order to verify that the error has a horizontal extent of only one pixel.

```
> c.matrix[,92]
  [1]  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  [21] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  [41] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      ...
```

Now, the values of column 92 are replaced by the arithmetic mean of those of the two adjacent columns, and the result is displayed.

```
> c.matrix[,92] <- (c.matrix[,91] +
  c.matrix[,93])/2
> c.matrix[,92]
  [1] 13312 14208 13824 13056 12416 12160 11904
  [7] 11776 12160 13184 13696 13824 13440 13440
  ...
```

For reconverting the matrix into a GRASS raster file, the matrix needs to be transposed (→ `d.matrix`) because of the reading order conventions of R, then transformed to a vector with the length of the total number of pixels (→ `e.vector`) and converted to integer values (→ `f.vector`).

```
> d.matrix <- t(c.matrix)
> dim(d.matrix)
  [1] 256 856
> mode(d.matrix)
  [1] "numeric"
> e.vector <- matrix(d.matrix, G$Ncells)
> f.vector <- as.integer(e.vector)
> typeof(f.vector)
  [1] "integer"
```

As final step, the corrected image, now represented as a vector object in R, is output to a GRASS raster file, with `rast.put()`. In this command, the GRASS environment (`G`), the raster output file name (`lname`) and the input object name (f.vector) are specified. `DCELL` is to be set to `FALSE` and `interp` to `TRUE`, in order to allow correct interpretation of integer values.

```
> rast.put(G, lname="corrected_band", f.vector,
  title="", cat=FALSE, DCELL=FALSE, breaks=NULL,
  col=NULL, nullcol=NULL, defcol=NULL,
  debug=FALSE, interp=TRUE, check=FALSE)
```

The export from R has to be followed by a `r.recode` procedure on the GRASS side in order to ensure that pixel values are taken as integers. Herein, the original cell values are given in comma-style, whereas the output values are specified as integers only. The GRASS raster file may now be visually checked and processed in further analysis.

A possible drawback in this procedure may be the file size of the image band, as R-objects are loaded completely in the memory space (Bivand and Neteler, 2000). In order to reduce memory requirements to a minimum, a small region of interest, which encompasses only a few columns, may be defined on the GRASS side (`g.region`), prior to read the image into R. After the correction procedure, the new, corrected section is merged with the old image band by means of the `r.mapcalc` command.

Final remarks

The capabilities that the R language offers for handling more or less complex arrays, makes it an interesting tool complementing the GRASS GIS capabilities when individual image elements are to be addressed. The integration of R into the GIS environment by means of the specific GRASS library provides the base for a smooth interaction of both software environments.

Acknowledgements

The author would like to thank for the collaboration and satellite images provided by the National Agency of Space Activities, Argentina, in the program "Morning Constellation: Landsat 7, EO-1, SAC-C and Terra", and for funding by the "Consejo de Investigaciones Científicas y Tecnológicas, Universidad Nacional de Santiago del Estero".

Bibliography

Barry, P. 2001a. Introduction to the Hyperion instrument & data processing. *In:* Hyperion & ALI Data Users Workshop. USGS and NASA, Baltimore, MD, USA. http://eo1.gsfc.nasa.gov/miscPages/ALIworkshop.html.

Barry, P. 2001. SWIR example: Mineral identification (Cuprite, NV). *In:* Hyperion & ALI Data Users Workshop. USGS and NASA, Baltimore, MD, USA. http://eo1.gsfc.nasa.gov/miscPages/ALIworkshop.html.

Bivand, R. and Neteler, M. 2000. Open Source geocomputation: using the R data analysis language integrated with GRASS GIS and PostgreSQL data base systems. *In:* Proceedings of the 5th International Conference on GeoComputation. University of Greenwich, Medway Campus, UK. http://reclus.nhh.no/gc00/gc009.htm.

Ihaka, Ross and Gentleman, Robert. 1996. R: A Language for Data Analysis and Graphics. Journal of Computational and Graphical Statistics 5:299–314.

Guido Lorenz
Facultad de Ciencias Forestales
Universidad Nacional de Santiago del Estero
Av. Belgrano (S) 1912
4200 Santiago del Estero
Argentina
glorenz2000 AT yahoo.com

Knowledge Management and GRASS GIS: r.infer

Peter Löwe

Introduction

This article is the first in a series describing applications of knowledge management techniques and GRASS GIS. Knowledge Management is the field of computer sciences which deals with the modelling of human intelligence, knowledge and problem solving strategies. The term Artificial Intelligence also applies for this field of research. To get started, the module *r.infer* will be examined and the knowledge management terminology will be introduced.

R.infer

r.infer has been part of GRASS GIS for a long time. It is included in GRASS Version 5.4, yet waits to be ported to the GRASS 6.x-releases. The code has not changed much since GRASS 4.0, when most of the documentation was written. To find out what it can used for, let's take a look at the old documentation J. Westervelt (1991):

- "r.infer uses an expert system approach and logic-based syntax to perform analyses similar to those made by the Grass programs r.combine"
- "r.infer is an inference engine which applies expert system type rules to a set of user-specified maps. The results are used to generate a new map in the users current mapset under the name infer"

This vocabulary differs from that commonly associated with GRASS GIS. The following section will help to clarify the meaning of the terms expert system, rules and inference engine.

Introducing Knowledge Modelling

Knowledge modelling explores how human expertise can be made available through computer programs. Since there are many approaches to reach this goal, knowledge modelling provides a variety of different technologies.

One of these are classification problems: Questions like "what is this?" are usually asked to an expert in a certain field, a *knowledge domain*. *Expert Systems* (ES) are software tools which tackle such problems. They consist of a heap of rules (the *knowledge base*), combined with an *inference engine* (IE). The IE can be thought of as a independent device which searches the knowledge base for applyable rules for its current data input. Such rules are perfectly suited to formulate vague or general knowledge like "if it smells good, eat it". If the initial part of such a rule (observation: "smells good") matches the current situation, the second part of the rule is executed, diagnosing "eat it". In analogy to neurons, this is referred to as "the rule fires". In the case of ES interacting with GIS, there is no immediate communication with the user. Instead, the ES queries the spatial database directly.

r.infer applies its current set of rules on GRASS raster data and writes a new raster layer containing the diagnoses obtained by the inference process. This is done for each spatial cell of the current region.

Rules in r.infer

In r.infer, a rule is a composition of logical statements regarding the existence of intervals in integer raster maps. Depending on wether the logical statement is true or not, diagnoses can be established and a corresponding value can be inserted in the resulting raster map. The syntax of knowledge bases consists of three basic commands:

IFMAP provides a statement about the presence of category-values or intervals in raster map layers. IFMAP can be combined with logical AND and NOT to negate or concatenate statements: IFNOTMAP, ANDIFMAP, ANDIFNOTMAP

THENMAPHYP is followed by a diagnosis, consisting of an integer value and a message string (the map category).

THEN This option can be used to add an abstraction layer between observations and diagnoses, referred to as symptoms. If a firing rule has established a symptom, it can be used as input for follow-up rules. Please note that if the last firing rule only provides a symptom, the resulting raster cell will contain the null value equivalent 0, which would also be the case if no rules had applied. r.infer does not support NULL values.

The strategy used to execute rules from the knowledge base by r.infer is very straightforward: It starts at the first line of the knowledge base file and works its way down. Once the inference process has produced a first diagnosis (a THENMAPHYP fires), rule evaluation is terminated for the current raster cell and r.infer starts over for the next cell. This prevents newly gained "insight" from causing any previously evaluated rules to fire (again).

Programming paradigms

So far, r.infer has not done anything that could not be done with a bit of effort using r.mapcalc. However, the difference in the underlying philosophies is significant Openshaw et al. (2000): R.mapcalc provides operations for map algebra. This is based on conventional programming to encode algorithms that produce numerical results from their input data. Maps made up of the numerical results are subsequently converted into decisions by human experts. This is usually done by relying on informal processes. Unlike this, expert systems such as r.infer apply a problem solving method to a database holding knowledge about the *meaning* of spatial data, not just the numerical values it is encoded in. The result is a decision about a problem from the specific knowledge domain. In this way, r.infer attempts to mimic the behaviour of a human expert.

Putting r.infer into use

The simplicity of r.infer results in fast inference runs, depending on the size of the current region. According to the original manuals, it was always intended to be used in conjunction with r.mapcalc, r.weight and r.combine.

r.infer can be used repeatedly: Calling up cascading knowledge bases adds flexibility when modelling domain knowledge. Also, attaching multiple categories to the same integer diagnosis can be used to fine-tune the content of the resulting map. Here is a sample knowledge base for locating camp sites within forests in the SPEARFISH region. The following lines of code define a knowledge base of four rules and must be stored in a text file. Let's assume it is called "camping.rules".

```
! Comment lines start with an exclamation mark.
! Each line of code must contain exactly one keyword
plus parameters.
IFMAP vegcover 3-5
THEN forest
! The categories 3-5 include all kinds of forest
IFMAP slope 0-3
THEN even ground
! We're looking for rather flat terrain
IF forest
ANDIF even ground
THENMAPHYP 1 prime location
IF forest
ANDIFNOTMAP slope 0-3
THENMAPHYP 10 second choice
```

Now the knowledge base can be used with r.infer:

```
GRASS GIS$>$ r.infer file=camping.rules \\
```

Now a new raster map containing the inferenced values (1 or 10 in this case) is created by r.infer. It is by default called *infer*.

Problem solving strategies

The way r.infer works to infer diagnoses from observations and symptoms to infer diagnoses is called *forward chaining*. Other approaches exist, namely *backward-chaining*, *establish-refine* and *hypothesize and test*.

It is also worth considering how an inference engine does its work of solving problems: The r.infer-engine only "knows" about TRUE/FALSE in absolute terms. This is referred to as *certain classification*, which leaves no room for uncertainty or "second best" answers. Geographical knowledge is by default incomplete and vague, so other approaches are also well worth being investigated There are several alternative classification strategies for problem solving. According to F. Puppe (1993) the most common ways of reasoning are *heuristic classification*, *set covering classification*, *functional classification* and *set covering classification*. These strategies will be examined in depth in the follow-up articles.

Categoric IF - THEN r.infer only uses categorical reasoning, which leaves no room for doubt or "second best" answers.

Heuristic A heuristic classification process starts by rating all available diagnoses according to the results of the inference run. In a second run, the highest rated diagnoses are elected to be most valid (depending on the KB settings).

Set-Covering The set-covering approach starts from the diagnoses side and looks for mandatory observations and symptoms to justify each diagnosis. The diagnosis with the most support is chosen.

Case-Based Case based Reasoning (CBR) is useful in complex knowledge domains. The approach is similar to supervised classification approaches in Remote Sensing. Sample cases are kept in a case repository ("case-base") and are compared with the features of the current "case". The best matching sample cases are presented.

Conclusion

Unfortunately, r.infer did not undergo similar upgrading and enhancement other GRASS modules did. Because of this it contains anachronisms such as being limited to integer values and lacking NULL support, requiring pre- and postprocessing by other modules. With some effort r.mapcalc "IF-structures" can be used to mimic most basic inference processes. Despite these shortcomings, the simplicity of the r.infer inference engine makes it a good starting point to demonstrate the basic aspects of expert systems.

Bibliography

S. Openshaw, C. Openshaw (1996) Artificial Intelligence in Geography. *Wiley*.

S. Openshaw et al. (2000) Geocomputation. *Taylor and Francis*.

F. Puppe (1993) Systematic Introduction to Expert Systems. *Springer*.

J. Westervelt (1991) GRASS 4.0 Inference Engine r.infer. http://grass.itc.it/gdp/raster/infer.ps.gz. *CERL*.

Dr.Peter Loewe
Geomancers.net
www.geomancers.net
loewe AT geomancers.net

Quantum GIS

Whats New in Version 0.7

by Gary E. Sherman and Tim Sutton

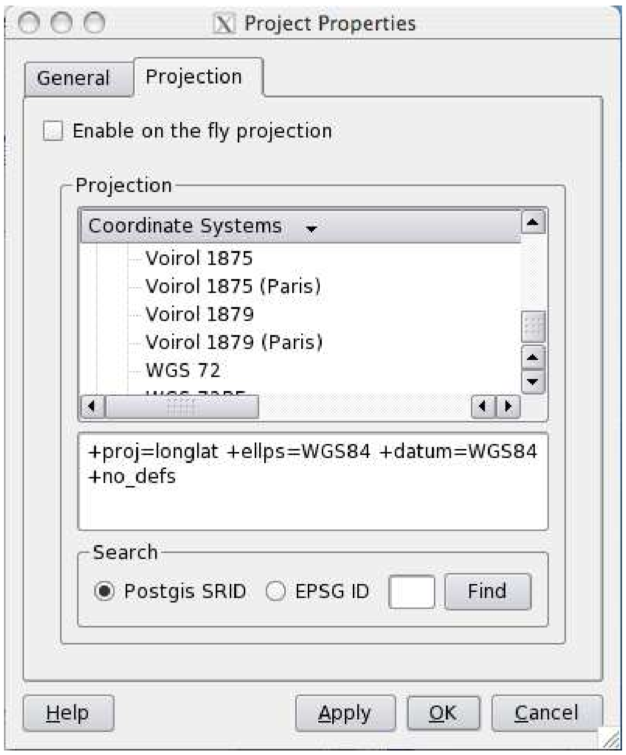
Introduction

Quantum GIS (QGIS) version 0.7 is currently slated for release in the second quarter of 2005. There have been several major enhancements since version 0.6. This article describes some of the new features in 0.7, as well as improvements to existing features.

New Features

Projection Support

At 0.7, QGIS supports on the fly projection (fig. 1) of vector layers from GRASS, OGR, PostGIS, and other data stores. This is a major milestone for QGIS and greatly enhances data integration capabilities. You no longer have to reproject your data using another tool in order to get things to line up properly. QGIS provides support for over 2,500 coordinate systems, largely based on the EPSG defined projections. You can also define custom projections and store them in a user database. This allows you to retain custom projections for use with future QGIS versions.



☐ Enable on the fly projection

Projection
Coordinate Systems
Voirol 1875
Voirol 1875 (Paris)
Voirol 1879
Voirol 1879 (Paris)
WGS 72

```
+proj=longlat +ellps=WGS84 +datum=WGS84
+no_defs
```

Search
● Postgis SRID ○ EPSG ID

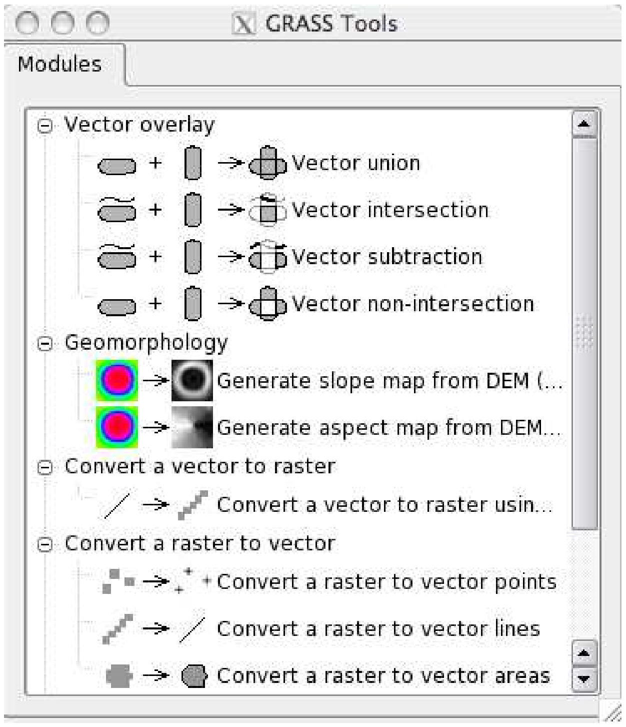
Figure 1: QGIS projection support

GRASS Integration

The GRASS digitizing tools have been been enhanced and now include support for 3 mouse buttons.

The biggest news is the new GRASS toolbox (fig. 2). This allows you to execute GRASS tools from within QGIS and see the outcome. Version 0.7 comes with 12 modules already configured, including:

- Vector union, intersection, subtraction, and non-intersection
- Generate slope or aspect map from DEM
- Convert vector to raster
- Convert raster to vector point, line, or area
- Set raster color table
- Show GRASS environment variables



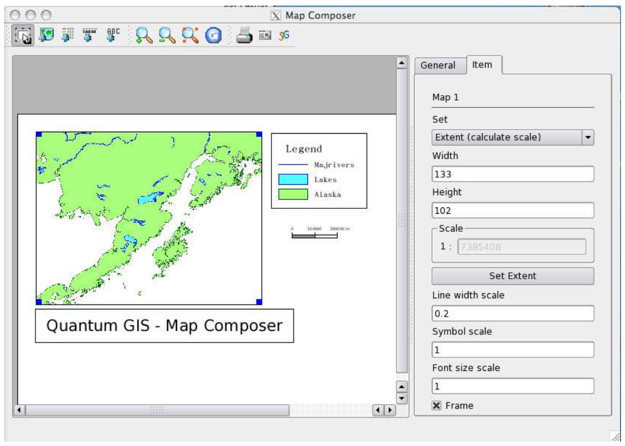
- Vector overlay
  - Vector union
  - Vector intersection
  - Vector subtraction
  - Vector non-intersection
- Geomorphology
  - Generate slope map from DEM (...
  - Generate aspect map from DEM...
- Convert a vector to raster
  - Convert a vector to raster usin...
- Convert a raster to vector
  - Convert a raster to vector points
  - Convert a raster to vector lines
  - Convert a raster to vector areas

Figure 2: GRASS tools in QGIS

Adding a tool (module) to the toolbox is done by simply creating an XML configuration file that defines the tool and options required to execute it. Each module also requires an icon and a record in the menu configuration file.

Map Composer

The map composer (fig. 3) is a new feature that provides improved layout and printing capabilities. The composer allows you to add elements such as the QGIS map canvas, legend, scalebar, and text. You can size and position each item and adjust the properties to create your layout. The result can be printed, exported as an image, or exported to SVG.



Map 1
Set: Extent (calculate scale)
Width: 133
Height: 102
Scale 1 : 7385408
Set Extent
Line width scale: 0.2
Symbol scale: 1
Font size scale: 1
[✓ Frame]

Figure 3: The QGIS map composer

Enhancements

Version 0.7 includes enhancements and changes to existing features in QGIS. While we can't list them all, some of the major changes are:

PostgreSQL/PostGIS - Handling of spatially enabled tables and views in PostgreSQL has been greatly improved. QGIS can now load any table in the database that contains a geometry column, regardless of whether the table has an entry in the *geometry_columns* table. In addition, views that contain a spatial column can now be loaded as well.

Raster graphing tool - It is now possible to produce a histogram for a raster layer.

Raster query - Using the identify tool, you can now get the pixel values from a raster by making it the active layer and clicking on the point of interest.

User preferences - New customizable settings for the digitizing line width, color, and selection color.

New symbols - New symbols for use with point layers are available from the layer properties dialog

Spatial bookmarks - This feature allows you to create and manage bookmarks for an area on the map. Bookmarks are persistent and global; meaning they are available for all projects.

Measure tool - Allows you to measure distances on the map with both segment length and total length displayed as you click

GPX loading - Loading times and memory consumption for large GPX (GPS) files has been drastically reduced.

Digitizing - many enhancements to the digitizing tools have been made, including the ability to capture data straight into PostgreSQL/PostGIS, and improvements to the definition of attribute tables for newly created layers.

Raster Georeferencing - the new Raster Georeferencer plugin can be used to generate a world file for a raster. The plugin allows you to define known control points in the raster coordinate system. Once enough control points are defined, the world file can be generated and the raster properly displayed in QGIS or other GIS applications.

Bug Fixes

There have been many outstanding bugs and issues addressed for the 0.7 release.

Of major importance is the fix for the long-standing bug that caused random lines, fill colors, and polygon mangling on X11. QGIS now has unlimited zoom-in capability with no display irregularities.

Summary

The QGIS team continues to work towards providing an easy to use application for browsing and working with GIS data. If you use QGIS we would love to hear from you. We are particularly interested in how folks are using QGIS in unique and clever ways. The QGIS Community website provides a means for you to contribute your experiences and knowledge. Please visit the site at http://community.qgis.org, register, and contribute by writing a *HowTo*, *User Report*, or *News* item. Thanks for using QGIS.

Gary Sherman
Quantum GIS
http://qgis.org
sherman AT mrcc dot com

Tim Sutton
Quantum GIS
http://qgis.org
tim AT linfiniti dot com

News

GRASS 6 Extensions Manager in development

A brief description of the current state of GEM

The GRASS 6 Extension Manager (GEM) is a small stand-alone program intended to make it easy for users to download, compile and install additional GRASS modules. GEM manages source code, scripts and pre-compiled binaries in a simple file layout called an "extension". Extensions are accompanied by a set of ASCII files that store all relevant information including dependencies on other extensions or specific GRASS versions. Extensions can be stored conveniently in a single compressed archive file, a so-called "extension package" using external programs such as tar and bzip2. An extension (package) can be created easily by copying existing source codes into the right places of a skeleton file layout and filling in some information in simple, commented ASCII files. Makefiles written for GRASS 6 should work with minimal changes as GEM uses a simplified version of the original GRASS make system.

From the user's point of view, using GEM to install a GRASS 6 extension is a greatly simplified process. GEM can hide the details of source code configuration and compilation from the user but will still output meaningful error messages if something goes wrong. GEM can do the following things for the user:

- compile and install extensions stored in a directory or a compressed package (tar.gz, tar.bz2, zip) provided that tar, gzip and friends are available;
- install pre-compiled binaries and scripts for different systems;
- retrieve extension packages from http and ftp sources, provided that wget is available;
- query extensions to display information and license files;
- upgrade extensions;
- remove extensions from the system;
- watch dependencies on other extensions and specific GRASS versions

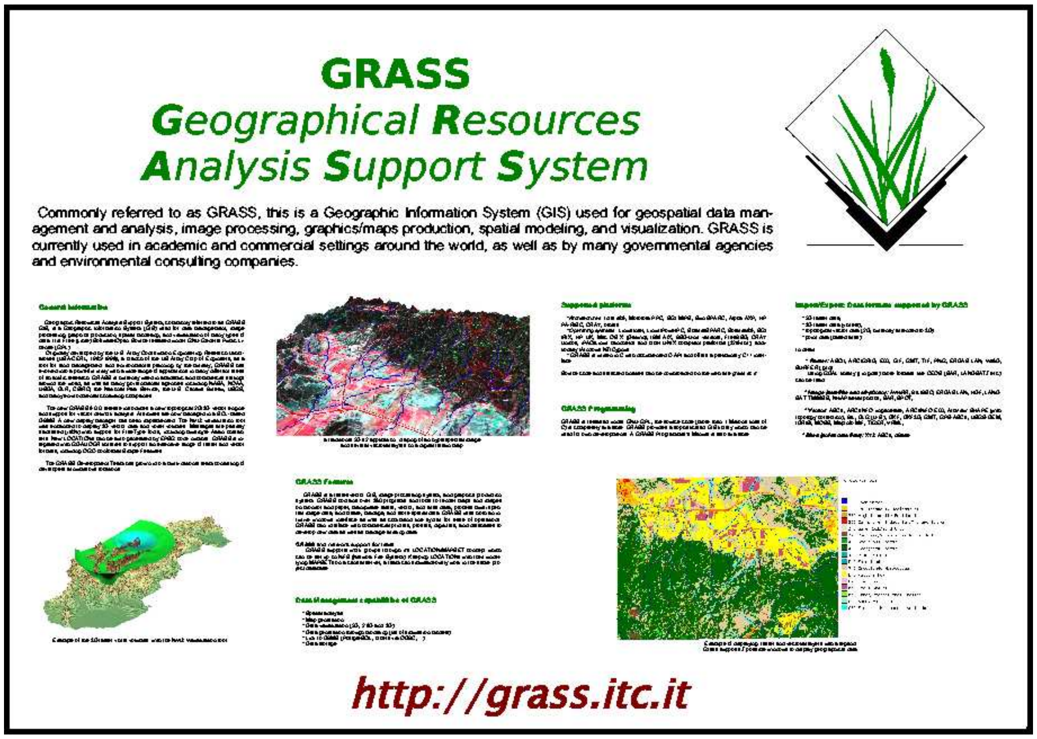

GRASS
Geographical Resources
Analysis Support System
Commonly referred to as GRASS, this is a Geographic Information System (GIS) used for geospatial data management and analysis, image processing, graphics/maps production, spatial modeling, and visualization. GRASS is currently used in academic and commercial settings around the world, as well as by many governmental agencies and environmental consulting companies.
http://grass.itc.it

Figure 1: GRASS poster by Jermoe Martin

A first version will be released as soon as functions to merge extension documentation into the GRASS 6 HTML index and automatic creation of menu entries for GIS Manager and QGIS have been implemented.

Initial tests with GEM have so far been promising. It is hoped that broader testing supported by the GRASS community can commence soon and a fully functional and stable GEM be released before the end of July. GEM will be released as open source software under the GNU GPL.

Benjamin Ducke, M.A.
Archaeoinformation Science
Inst. of Prehistoric and Historic Archaeology
University of Kiel
Johanna-Mestorf-StraSSe 2-6
D 24098 Kiel
Germany
benjamin.ducke AT ufg.uni-kiel.de

GRASS poster

A first GRASS poster has been submitted by Jerome Martin (martin AT et.esiea.fr) and can be downloaded at the GRASS Poster webpage[11]. Further posters presenting GRASS capabilities are welcome and can be submitted via the *GRASSNews* webpage.

[11] http://grass.itc.it/newsletter/postercontest.php

Recent and Upcoming Events

FOSS/GRASS 2004 – Conference Report

12-14 September 2004, Chulalongkorn University, Thailand.

The Free/libre and Open Source Software (FOSS) for Geoinformatics: GIS - GRASS Users Conference was held in Bangkok, Thailand, on the 12-14 September 2004[12]. The conference was organized by the Faculty of Engineering, Chulalongkorn University, Thailand, with support from several other institutions. The FOSS/GRASS 2004 conference was the first-ever international meeting exclusively devoted to the application and development of FOSS for Geoinformatics to be held in Asia. The FOSS/GRASS 2004 conference provided a unique opportunity for sharing knowledge and valuable experiences amongst developers, users and service providers. The FOSS/GRASS 2004 attracted 100 participants belonging to organizations spread over 17 countries (Thailand (38), Japan (21), Indonesia (8), Italy (8), USA (4), Canada (3), Malaysia (3), Spain (3), Germany (2), India (2), Philippines (2), Australia (1), France (1), Hong Kong (1), Iran (1), Poland (1), Switzerland (1)). The Opening Session was jointly chaired by Prof. Direk Lawansiri, Dean, Faculty of Engineering, Chulalongkorn University, Thailand and Dr Suvit Vibulsresth, Director, GeoInformatics and Space Technology Development Agency (GISTDA), Thailand.

The Keynote Session featured four invited speakers who covered various aspects such as capacity-building using FOSS (David Hastings, UN-ESCAP), partnerships between industry and FOSS communities (Dave McIlhagga, DM Solutions Group Inc.), FOSS scenario in Japan (Hideo Nakano, Osaka City University) and an exciting talk of the birth and growth of GRASS (Jim Westervelt, USA-CERL).

In the Technical Sessions that followed, a total of 25 Oral presentation and 16 interactive poster presentations covering almost the entire gamut of FOSS for Geoinformatics were made[13]. The Technical sessions covered themes such as a) GRASS GIS-New Features and Algorithms b) Mathematical methods and techniques c) Natural hazards prediction and management d) Online spatial databases and their applications e) Educational tools f) Landcover analysis and change Detection. A CD-ROM proceedings was also produced.

Figure 1: Dr. Suvit Vibulsresth, Director, GISTDA, Thailand delivering the Opening Speech.

Full papers presented at the conference are available online at FOSS/GRASS 2004 website.

Besides the keynote presentations and technical session, the pre-conference INSTALLFEST attracted an enthusiastic response with more than 30 participants attending. A CD-ROM containing FOSS tools and training material for spatial data analysis (GRASS) and sharing (Mapserver) were provided to the INSTALLFEST participants. Dr. Peter Loewe (University of Wuerzburg, Germany) also demonstrated the GISIX-Live CD for FOSS GIS. The GISIX CD is currently being published for distribution to the INSTALLFEST participants. A post-conference workshop by Dave McIlhagga and Jeff McKenna (DM Solutions Group Inc.) provided an overview of Internet Mapping Technology and also demonstrated the future potential of FOSS for Geoinfomatics. The exhibition and demonstration of the i18n version of GRASS and Mapserver product by the Orkney Inc. Analysis Center, Japan also attracted an enthusiastic response.

Considering the overwhelming success and strong request from the participants for a forum covering all aspects of FOSS for Geoinformatics in 2006, it was proposed to organize Joint Conference in Lausanne, Switzerland in September 2006 with support and participation of FOSS for Geoinformatics communities. We hope that a joint conference in 2006 will be a bigger success and afford an opportunity for closer interaction.

[12] `http://gisws.media.osaka-cu.ac.jp/grass04/`
[13] `http://gisws.media.osaka-cu.ac.jp/grass04/`

Figure 2: An impressive lineup of Keynote Speakers and chairpersons of the keynote (Seated Left to right: Mr. Markus Neteler, Mr. Dave McIlhagga, Prof. Hideo Nakano, Dr. David Hastings, Dr. Jim Westervelt and Prof. Mamoru Shibayama).

Figure 3: Members of the Secretariat Staff smile for a job well done.

We would like to express our sincere thanks to the FOSS/GRASS 2004 International Organizing Committee for their support and the Conference Secretariat at Chulalongkorn University for the excellent arrangements that made it a pleasure to be in Bangkok and enjoy the immense hospitality of the Thai people. We wish that the new friendships that the FOSS/GRASS 2004 conference has so successfully initiated will help in incubating and nourishing new ideas and thereby enrich the FOSS community. May the FOSS be with you ...

Dr. Venkatesh Raghavan
Osaka City University
Japan

Dr. Phisan Santitamnont
Chulalongkorn University
Thailand

GRASS User Meeting of the SouthWest-Germany Workgroup – Meeting Report

3-4 December 2004, Albert-Ludwigs-University, Freiburg, Germany.

The working group of the First Grass User Meeting was initiated and directed by the geographers and members of the GRASS Anwender-Vereinigung e.V. *(http://www.grass-verein.de)* Dr. Sigrid HESS (Mainz), Dipl.-Geogr. Marco LECHNER (Freiburg) and Dr. Peter LÖWE (Würzburg). Participants came from research and education institutions as well from the German GIS industry. This meeting was aimed explicitly at exchanging experiences from the GRASS GIS user community, rather than the developer community. The presentation of Dr. Peter Löwe on the subject of "potentials and chances of FOSS GIS for research and education" led to an animated discussion about the actual gap of teaching at universities and needs of personal power in commerce and industry and the chances of FOSS in this context.

Some ideas for the GRASS community were proposed in relation to the collection of GRASS scripts, similar to ArcScripts with metadata management. Further consensus and group-conjunctive aims were appointed to:

1. the collection and exchange of knowledge and skills;
2. mutual and self-education
3. the bundling of knowledge

A regional user forum, the *"SouthWest Germany GRASS User Forum"*, was established as a future stage to showcase FOSS GIS projects. This forum will be aimed at topics relating to real world applications. Regular meetings were proposed for the future, starting in 2005. The next session will take place at the Department of Physical Geography of the University of Freiburg on 18./19.02.2005.

Dr. Sigrid Hess, Dipl.-Geogr. Marco Lechner and
Dr. Peter Löwe; GAV e.V.
`http://www.grass-verein.de`
`sihess AT uni-mainz.de`
`marco.lechner AT geographie.uni-freiburg.de`
`loewe AT geomancers.net`

GRASS User-panel Southwest Germany

18-19 February 2005, Department of Physical Geography, Albert-Ludwigs-University, Freiburg, Germany.

The second meeting of the GAV Southwest-group was held in Freiburg, Germany on the 18th and 19th of February 2005. It was again kindly hosted by the Chair for Physical Geography at Freiburg University. Presentations concerning practical applications of FOSS GIS were given. The topics ranged from laser scan data, statistical modelling with R, radar-meteorology to web-based applications and integration of GRASS GIS in XliveCD. The intention was to present real-world applications, but also to acknowledge real-world limitations, and shortcomings of FOSS GIS.

Participants came from many different backgrounds, ranging from students, researchers and governmental employees to members of the local GIS-industry.

The popularity of the meeting became evident as the number of participants exceeded the number of pre-registrations.

The speeches will soon be published in written form on the homepage [14].

The next meeting is scheduled for July 15./16. 2005 at the same location.

Dr. Sigrid Hess, Dipl.-Geogr. Marco Lechner and Dr. Peter Löwe; GAV e.V.
`http://www.grass-verein.de`
`sihess AT uni-mainz.de`
`marco.lechner AT geographie.uni-freiburg.de`
`loewe AT geomancers.net`

[14] `http://www.geographie.uni-freiburg.de/ipg/personen/lechner_marco/grass`

The *GRASS newsletter* is a publication of the *GRASS-Project*. The base of this newsletter, the $\LaTeX 2_\varepsilon$& sty source has been kindly provided by the R News editorial board (http://www.r-project.org). All articles are copyrighted by the respective authors. Please use the GRASS newsletter url for submitting articles, more detailed concerning submission instructions can be found on the GRASS homepage (http://grass.itc.it and its mirrors).

GRASS Project Homepage: http://grass.itc.it/
Newsletter online:
http://grass.itc.it/newsletter/index.php

Acknowledgements
Various reviewers & the R News Project

ISSN 1614 - 8746