White Paper: The GRASS 5 Geophysics Framework

Authors:

Benjamin Ducke, University of Bamberg; Brandenburg Heritage Management (Germany)

Purpose

This white paper outlines a software architecture that supports high-level analysis and visualisation of geophysical data using command line and GUI tools within the GRASS GIS environment. Its aim is to provide a flexible framework in which every imaginable processing of geophysical signals/images can be carried out in a unified and convenient way. Once this framework is in place, adding new geophysical functionality to GRASS GIS will be a matter of simply implementing the necessary arithmetics using mapcalc syntax, C code or shell scripts that make use of the functionality already present in the GIS. Authors of new geophysical modules will not have to be concerned with user interface design, data storage or GRASS integration in general. Users of existing functionality will benefit from a hazzle-free way of analysing and visualising any number of measurement grids without cluttering their working location with unneeded raster maps.

Terminology and Basic Concepts

In the context of this document, the following are key concepts:

Grid = a GRASS floating point raster map produced by rasterising a file with raw ASCII measurement data. Geophysical field work usually results in the creation of a number of grids covering a site of interest. The number of data points on the X and Y axes, i.e. the resolution of the grid raster, is defined by the number of measurements taken in both directions. Each grid has a reference that assigns it to a unique position in one and only one *grid layout* (see below). From a technical point of view, a grid is just a GRASS floating point raster map that resides in a special folder outside the standard 'fcell' directory. Thus, it is not visible by default which prevents grids from cluttering up the user's workspace when using 'g.list' or GUI tools to browse GRASS raster maps.

Grid Layout = an ASCII file that describes an orthogonal raster of equal-sized, adjacent cells. A layout has:

a unique name

dimensions (number of rows and columns; specifies the number of cells)

grid dimensions (in working location's units)

geographic coordinates of its four corners (in the working location's coordinate system)

references to grids (at most one per cell)

This defines the topological structure of a geophysical analysis. Each cell can accommodate exactly one grid or remain empty. All grids referenced to the same layout must have identical resolutions. This rigid structure means additional work in planning and carrying out the data acquisition in the field but greatly simplifies and speeds up the data analysis within any software environment.

Composite grid = a composite grid is a regular GRASS FP raster map. It is produced by patching all the grids (or a subset of them) in a grid layout together and geo-referencing/rectifying the result, using the layout's corner coordinates. The user can create a composite grid on demand via the 'r.composite' module.

Filter = a GRASS stand-alone module that works on all or a subset of the grids in a layout and applies some sort of algorithm to the them (e.g. high pass, low pass filtering, destriping, deconvolution). Any GRASS module can act as a filter within this framework, provided that it handles some standard filter arguments (see below). As opposes to other GRASS modules, a filter module does not run immediately when called by the user. Rather, it addds the command line by which it was called to the filter list (see below) and does the actual work only when a special flag is set (which should only be done by the 'r.composite' modul).

Filter arguments = a set of parameters that are passed to a filter on the command line or via a GUI to control its operation. As opposed to other GRASS modules, every program that qualifies as a filter in this framework must support a set of unified arguments. A wrapper function can be used to modify other GRASS modules to adhere to this standard, which includes:

the name of a grid layout to operate on (specified via 'grid=<grid name>')

an optional specification for a subset of grids to operate on (grid range, see below)

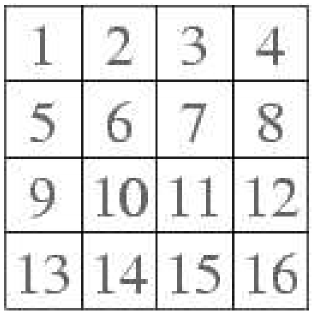
any additional arguments needed to control the operation

an 'output=<map name>' argument that should only be set by 'r.composite' and signals the filter to actually execute its raster manipulation code (see above)

Filter list = an XML file that contains a sequence of filters to be applied to all or a subset of the grids in a grid layout. There exists exactly one filter list for each grid layout, both having the same name but residing in separate directories (see section on “File Structure”). Filters mentioned in the filter list are applied in the sequence they appear in XML file. In addition, the filter list stores arguments for each filter. A GUI program allows manipulation of the filter-list while ensuring XML file correctness.

*Grid coordinat*e = a single integer number that uniquely identifies a grid in a grid layout. The smallest grid coordinate is “1”, the largest is “grid rows multiplied by grid columns”. The numbering starts at the top left cell/grid in the layout.

Grid range = a specification of grids to work on, as used in applying filters. In syntax, this is similar to specifying pages to print in a word processor, e.g.: “1, 3-6, 8” to select grids 1,3,4,5,6 and 8. This type of specification is very easy to parse and redundancies can be ignored.



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Illustration 1A grid layout of dimensions 4x4 with grid coordinates

Outline of Workflow

Copy raw ASCII data from serial line or file into the 'grid_ascii' directory under the GRASS location or any other place in the user's file system.

1. Create a new grid layout in the GRASS working location using 'm.grid.layout.create'.
2. Convert the ASCII data into grids using 'r.in.grid'.
3. Add filters.
4. Produce Composite grid using 'r.composite' and display result.
5. Modify filters and color ramp with GUI tools.
6. Go back to 3. until results seem OK.

File Structure

All GRASS files reside in a database directory somewhere in the file system. The database directory contains subdirectories, called locations, and these in turn have mapsets, that often correspond with user accounts on a Unix/Linux workstation.
The mapsets contain the actual data, i.e. raster maps, site lists, vector files etc. Each type of data resides in a directory of its own. In GRASS terminology, they are also called 'elements'. GRASS allows us to create any number of custom elements. For the geophysical framework, we need three additional elements that store
A file listing of a GRASS mapset might look like:

```
cellhd    dig        dig_cats    fcell      grid_flist    site_lists
cats      cell_misc  dig_ascii   dig_plus   grid_layout   WIND
cell      colr       dig_att     grid       hist          windows
```

The 'grid_layout' element contains ASCII files with layout descriptions (see Section on “Terminology”). For each grid layout <name> there also exists a file with the same name in the 'grid_flist' element. This is an XML file containing a list of filters and their respective arguments. In the 'grid' element, we store the individual grid rasters belonging to each grid layout. The naming scheme is: <name>.<number>. Thus, a grid layout called 'MyGrid' with 2 rows and 2 cols would have the following grids associated with it:

MyGrid.1, MyGrid.2, MyGrid.3, MyGrid.4

(note that the sequence is always complete. The 'm.grid.layout.create' module creates a NULL-filled raster file for each cell initially. This simplifies grid processing, especially for shell scripts.)

Creating Geophysical Modules

Support is offered for writing filter modules in C as well as any scripting language. When writing a filter module, programmers should:

1. provide two modes:
   A. only the module name and its arguments are written to the filter list (default operation). Shell programmers can use 'm.write.filter' to add content to the XML filter list.
   B. the actual processing is done and the result map stored in a raster map specified by the output=<map name> argument.
2. be aware, that grid raster maps are not available through regular GRASS C API or GRASS modules, as they reside in a separate directory:
   A. C programmers should use special API calls to open raster files in the 'grid' directory (see section 'Description of C API').
   B. Shell script programmers should use r.gettile to copy a tile raster
3. always provide an 'output=<file name>' parameter. Otherwise, 'r.composite' will not be able to interact correctly with the filter module.

Overview of GRASS Modules

Managing Grid Layouts

m.grid.layout.create grid=<grid layout name> cols=<int> rows=<int> xres=<int> yres=<int>,

upperleft=<Easting|Northing> upperright=...

Create a new grid layout: User needs to specify tile size, number of tiles in X and Y direction and coordinates of the grid's four corners in the working location's geographic projection. This creates a new ASCII file in the 'grid_layout' element (directory, see section on 'File Structure'). It also creates an empty XML-filter list file with the same name in 'grid_flist' and 'cols x rows' NULL-filled raster maps of 'xres x yres' pixels in 'grid', one for each grid in the layout, numbered sequentially and starting at '1'.

m.grid.layout.edit

Make changes to an already existing grid layout.

Managing Grids

r.in.grid input=<file> grid=<grid layout name> coor=<int> [NULL=<val>, ...]

Converts an ASCII file containing measurements to a grid raster: User has to specify the ASCII file to read in (simply a name, if the file is in the 'grid_ascii' directory, fully qualified path otherwise), the name of the grid layout and the grid coordinate to attach it to[1]. Also, the user has to specify the sequence of the measurements in the file (zig zag, parallels, starting point, direction) and value of the NULL coding. All other information can be read from the grid layout. This creates a grid raster file with the resolution of the measurements (*not* that of the current working region) names it <grid layout name>.<int> and places it into the the 'grid' element (directory, see section on 'File Structure').

Using Filters

r.filter.* grid=<grid layout name> [range=<grid range>] [<further arguments>] [output=<map>]

Attaches a filter name, standard filter arguments and all <further arguments> to the filter list that relates to <grid layout name>: By default, this is all that gets done when the user calls 'r.filter'. Filter modules will only process data when the 'output=' parameter is specified. In this case, the results will be save to the GRASS raster <map>. See separate subsection on “Filter modules” for an overview of functionality.

r.grid.process grid=<grid layout name> range=<grid range> output=<map>

For testing and debugging of filter modules: Applies all filters specified for <grid layout name> to one grid, writes the resulting raster map to <map> in the 'fcell' directory.

m.filters grid=<grid layout name>

Produces a GUI to let the user manipulate filter settings for a grid layout.

Creating Composite Grids

r.composite grid=<grid layout name>

Creates a GRASS raster map by applying filters and using r.patch to stitch all grids in a layout together. Also uses GRASS imaging modules to rectify and geo-reference the result in the current working location. Since GRASS automatically resamples raster maps to the current location's resolution settings, the original grids will get stretched or shrunken accordingly to fit. As an alternative, the user may specify a more advanced algorithm to resample the grids to match the working location's resolution.

Prior to calling 'r.patch', all filter modules in the filter list relating to <grid layout name> are run (by passing the “output=” option to all modules on the list) and the results stored in temporary

[1] A simple serial line device driver could also be implemented in this module, to read data directly from the instrument.

raster files. These temporary files are then patched together to create the result map. The original grid rasters remain unchanged.

GRASS Modules with Enhancements

The following standard GRASS modules will be enhanced to support the new 'grid_layout' and 'grid' elements.

g.remove

Add support for removing single grids or grid layouts (including all referenced grids and the filter list) from disk.

g.rename

Add support for renaming grid layouts (and automatically all associated grids and filter lists).

g.copy

Add support for copying grid layouts (and automatically all associated grids and filter lists).

g.list

Add support to list grid layouts in a one-item-per-line fashion, with additional information about dimensions (filenames of individual grids can be easily deduced from this output):

```
grid_one  4x4 grids (100 x 20 measurements)
grid_two  5x3 grids (100 x 20 measurements)
...
```

Scripting interface

r.grid.get grid=<grid layout name> coor=<int>

Copies a grid at position <int> from <grid layout name> to the regular GRASS 'fcell' directory, thereby making it visible to all GRASS moduls. This is done without applying any filters.

m.add.filter grid=<grid layout name> filter=<module name> args=<arg1, arg2, ... argN>

Add an entry to a grid layout's filter list.

Available filter modules

The following filter modules will be part of the geophysical framework:

r.filter.destripe

r.filter.upward

r.filter.downward

r.filter.highpass

r.filter.lowpass

Description of C API

C functions needed to ease filter programming:

char *GP_find_cell (char *name, char *mapset)

Finds grid '*name' in the current mapset search path (if *mapset is an empty string) or in the specified mapset. Returns mapset where the grid raster resides.

int GP_open_grid_old (char *name, char *mapset)

In analogy to G_open_cell_old (), this opens a grid raster file in the 'grid' element and returns a file handle for use by regular GRASS 5 raster API calls.

int GP_get_cellhd (char *name, char *mapset, Cell_head *cellhd)

Read raster header of grid into Cell_head structure (contains original measurements resolution settings!).

GP_add_filter (char *grid_layout_name, char *filter_cmd, char **arguments)

Add a new entry to the filter list for '*grid_layout' (using libxml2).