OSGeo Journal

The Journal of the Open Source Geospatial Foundation Volume 1 / May 2007

In This Volume

Real World Implementations of Open Source software

Introducing Mapbender, deegree, openModeller ...

Understanding Spatial Relationships

Examining the Web Processing Server (WPS) Specification

Package Interaction - GRASS-GMT, Tikiwiki, PyWPS, GRASS-R ...

Software Updates

News, and more...

OSGeo
Your Open Source Compass

Integration Studies

Producing Press-Ready Maps with GRASS and GMT

by Dylan Beaudette

Overview

The production of high quality, printable maps is an important component of most geographic analysis. While GRASS has not traditionally been suited for the production of printed materials, commands like `ps.map` and more recently a helpful wrapper `G-ps.map`, make it possible to output high quality Postscript maps. However, the rather limited syntax and capabilities of the `ps.map` approach often result in a map that must be finalized in a DTP (desktop publishing) application such as Inkscape, Scribus, or The Gimp. The map composer functionality found in QGIS may be a good alternative for map production in the near future, but the current version is lacking in terms of flexibility, stability, and the ability to export data in formats such as EPS or PDF. With such limited options for the production of high quality printed materials when using an entirely open source workflow, numerous people have cobbled together the necessary glue required to interface GRASS to the various programs included with the Generic Mapping Tools package. The GMT project is best described by its authors as :

> GMT is an open source collection of 60 tools for manipulating geographic and Cartesian data sets (including filtering, trend fitting, gridding, projecting, etc.) and producing Encapsulated PostScript File (EPS) illustrations ranging from simple x-y plots via contour maps to artificially illuminated surfaces and 3-D perspective views. GMT supports 30 map projections and transformations and comes with support data such as coastlines, rivers, and political boundaries. GMT is developed and maintained by Paul Wessel and Walter H. F. Smith with help from a global set of volunteers, and is supported by the National Science Foundation. It is released under the GNU General Public License.

This article describes the combined efforts of several people (David Finlayson, Hamish Bowman, Brent Wood, myself, and others) to create a better means of interfacing GRASS to GMT. Several permutations

of a `*.out.gmt` style command have been created, using both Python and Bash scripting, to perform the required export, configuration, and execution of GMT commands. After reading this article you should have a general idea of how to compose simple maps in GMT, along with some template scripts for using data exported from GRASS. A follow-up article will present Spearfish-specific (the standard sample GRASS dataset) examples of map creation with GMT.

GMT

The Generic Mapping Tools applications can be installed on just about any platform using either prepackaged binaries or by compiling from source code. The GMT tools can also be installed from your favorite distribution's package management system (e.g. aptitude), however this method is not recommended as the package is not likely to be current. If you are on a UNIX-like operating system compiling from source is a relatively simple process, and ensures that you have a current release. Assuming that you have a working development toolkit (gcc, make, etc.) navigate to the 'Download' page [1], and follow the directions. GMT is distributed with a comprehensive manual, map-making tutorials, and basemap data at several scales. Once you have a functional copy of GMT installed on your machine, and have downloaded the example data [2] you can follow along with the examples contained in this article. This archive also contains a couple scripts which elaborate on several of the examples included within this article. The script `template.sh` can be used as a starting point for an automated approach to exportng and plotting GRASS raster and vector data with GMT tools. Note that this script is a starting point and far from complete.

GMT commands are run from the shell (with an intricate set of flags, switches, and other arguments) and send the resulting Postscript fragments to standard output. The default behavior of all GMT applications can be adjusted with the `gmtset` command. This command is usually used prior to any commands which produce output, so that key elements such as paper size, font spacing, etc. are established. The first plotting command run (usually `psbasemap`) creates the initial output file with the standard ">" operator , while each subsequent call adds to this file with the double-">" append operator. The `-K` flag is used with all GMT commands prior to the last command, so that Postscript output file is not prematurely finalized. We will use some sample data collected from a Mapserver application to illustrate various aspects of GMT use, along with some ideas on how to couple GRASS and GMT.

Sample Application

```
# define some global settings:
gmtset ANNOT_FONT_PRIMARY Times-Roman \
HEADER_FONT_SIZE 16 \
ANNOT_FONT_SIZE_PRIMARY 12 \
LABEL_FONT_SIZE 14 \
BASEMAP_TYPE plain \
PLOT_DEGREE_FORMAT DF \
PAPER_MEDIA letter+

# sample map centered on the western USA
pscoast -JB-116/36/30/42/7i \
-R-125/-108/31/44 -B5 \
-Gwhite -W0.5p \
-A250 -Dh -Na -Xc -Yc -P -K > query_centers.eps

# plot mapxy points:
psxy sample_data/mapxy_locations.latlong -J -R \
-Sc0.075c -W1/1/200/1 -G1/200/1 -O -K >> query_centers.eps

# plot ka-map points
psxy sample_data/ka-map_locations.latlong -J -R \
-Sc0.075c -W1/255/1/1 -G255/1/1 -O >> query_centers.eps
```

The example above creates a plot of the western USA, with red and green dots symbolizing location-specific activity on a website. The map frame is created with the `psbasemap` command, coastline and political boundaries are created with `pscoast`, and points are created with the `psxy` command. Point locations are stored as simple, longitude-latitude pairs in text format, while the coastline is drawn from the GMT built-in data set. The output can be seen in Figure 1.

Full explanations of the various command line options can be found on the manual page of each command. In addition, a more complete example GMT session performed outside of GRASS can be found on the author's website [3]. In the above example, the projection and region parameters were manually defined by the user. Additional formatting elements (such as the `-P` flag for portrait layout) must be manually planned and defined on the command line. Thus the automatic generation of an output map depends, in part, on a dynamic approach to defining the options passed to GMT commands. A scripting language, capable of extracting GRASS region infor-

[1] http://gmt.soest.hawaii.edu/gmt/gmt_download.html
[2] http://169.237.35.250/~dylan/GRASS/newsletter/sample_data.tar.gz
[3] http://casoilresource.lawr.ucdavis.edu/drupal/node/102

mation from a running GRASS session, is an ideal approach to automating this task.

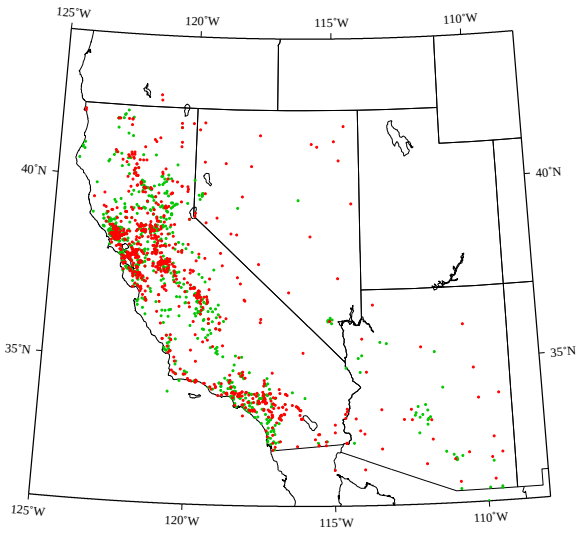

Figure 1: Query locations, symbolized by active mapping application. Green dots represent Landview-based Mapserver application, red dots represent Ka-map-based Mapserver application.

Coupling GRASS and GMT

The general approach to interfacing GRASS and GMT can be summarized as follows:

- collect information from the user on:
  - input data
  - formatting options
  - output options
- collect geographic parameters from the current GRASS region
- calculate key layout parameters
  - projection information
  - scale restrictions
  - tick intervals
- export GRASS raster data to GMT .grd format
- export GRASS vector data to GMT ASCII format
- run GMT commands with above settings

Collecting Information from the User

The GRASS script parser provides a convenient approach to the creation of customized GRASS commands, without the need to understand C programming. A carefully crafted script can be used to pass details about a GRASS dataset along with information on the current region settings to GMT commands, using the familiar syntax and GUI structure of a GRASS module. With these options saved to local variables it is possible to alter default settings such as paper size, font size, and text orientation by running the `gmtset` program with our local variables substituted by the script interpreter [4].

Collecting Geographic Information

Extracting key information from the current region settings (for later use in constructing parameters for GMT) can be accomplished in a number of ways. One such example involves `g.region` and unix text processing tools, thanks to Bruce Raup for the elegant `eval` suggestion. Note that all of the following code snippets require that you are in an active GRASS session.

```
# need to be in an active GRASS session for this to work.
#
# region extents: used with -R flag
# resolution: used with the -I flag
#
# elegant approach to extracting extent and
# region resolution as suggested by
# Bruce Raup - National Snow and Ice Data Center
#
region_assignments=`g.region -g`
eval $region_assignments

# setup flags for GMT commands
region="-R$w/$e/$s/$n"
inc="-I$ewres/$nsres"

# extract extent values for calculating map aspect ratio
extent_assignments=`g.region -ge`
eval $extent_assignments

# preserve aspect ratio from UTM E,N coordinates:
aspect_ratio=`echo $ns_extent $ew_extent
| awk '{printf("%f", $1 / $2)}'`

# calculate the map length based on the
# original aspect ratio
map_length=`echo $map_width $aspect_ratio
| awk '{printf("%f", $1 * $2)}'`
```

[4] More information on the GRASS script parser can be found at http://grass.itc.it/gdp/html_grass63/g.parser.html

```
# calculate the y location (in paper units)
# for the field sheet title
field_sheet_title_y=`echo $map_length
| awk '{printf("%f", $1 + 0.33)}'`

#compile the projection string, with
# width/length variables
#linear projection, width inches / length inches
projection_string="X${map_width}i/${map_length}i"
```

A more robust approach would involve the GRASS-Python bindings to directly extract region parameters. Once the region data has been extracted and saved to local variables, we can perform calculations required to derive other parameters such as map scale, map aspect ratio, map size on paper, etc. The region settings are used to create the *plotting region* flag (`-R$w/$e/$s/$n`), often used only in the first plotting function. All subsequent plotting functions can inherit these values by specifying blank `-J` and `-R` flags along with the `-O` (overlay) flag.

Map Projections in GMT

The traditional approach to using the GMT tools involves plotting data, encoded in longitude-latitude pairs, in a projection specified with the `-J` flag. Coordinates are transformed as the output Postscript is being generated. As noted in the first code snippet above, an Albers Equal Area projection centered at (116°W, 36°N), having standard parallels at 30°N and 42°N, and which is 7 inches wide is encoded as `-JB-116/36/30/42/7i`. See the GMT manual for a full listing supported projections and their parameters.

When working within GRASS, data is usually (but not always) in some cartesian projection and thus the special *linear* projection mode in GMT should be used. Once a suitable map width and length (in this case using inches) have been established the linear projection parameters are encoded as `-JX${map_width}i/${map_length}i`. Note that imperial or metric units can be used within GMT and switching between them only invloves adding a 'c' for centimenters or 'i' for inches after the actual number.

Exporting and Plotting GRASS Raster Data

Exporting GRASS raster data to a GMT-compatible format can be accomplished with a combination of `r.out.bin` and `xyz2grd` (included in the GMT distribution). Proper use of `xyz2grd` requires knowledge about input raster data: i.e. floating-point or integer, etc. For a CELL raster exported with `r.out.bin`, the flag `-ZTLh` would be used to instruct `xyz2grd` that the input data stream is scanline oriented short integer data. For FCELL or DCELL maps the corresponding flags would be `-ZTLf` or `-ZTLd` respectively. The `-F` flag instructs `xyz2grd` to use pixel-registration of grid cells, the same registration used for GRASS rasters. A simple case statement can be used within our example script to automatically create the call to `xyz2grd`. An optional flag in our example script can be used to toggle this step to facilitate tweaking other parameters without the time consuming process of exporting raster data. The author has found that setting the region to an integer resolution results in the most reliable operationof `xyz2grd`.

```
# get the map type of a given raster
MAP_TYPE=`r.info -t "$output_raster" | cut -f2 -d'='`

# export the raster based on its type,
# region settings from above example
# and resolution settings above example
case "$MAP_TYPE" in
CELL)
r.out.bin input=$output_raster output=- null=-9999 \
| xyz2grd -G$output_raster.grd \
$region $inc -ZTLh -F -N-9999 ;;
FCELL)
r.out.bin input=$output_raster output=- null=-9999 \
| xyz2grd -G$output_raster.grd \
$region $inc -ZTLf -F -N-9999 ;;
DCELL)
r.out.bin input=$output_raster output=- null=-9999 \
| xyz2grd -G$output_raster.grd \
$region $inc -ZTLd -F -N-9999 ;;
esac
```

Once raster data has been exported to .grd format it is nearly ready for use with GMT image plotting commands such as `psimage`. This command can either use a single grid file, along with a color palette (.cpt) file, or color-separate red, green, and blue channels [5].

Exporting GRASS Color Table Data

Currently there are no *simple* methods available for converting GRASS color table information into a suitable color palette file for GMT. The existing implementations are able to closely approximate the .cpt format, however the author has not been able to successfully use these automatically generated files. A working conversion script (preferable written in a

[5]The specifications for GMT color palette files can be found at http://www.soest.hawaii.edu/GMT/gmt/doc/html/GMT_Docs/node57.html

[6]Further testing and adaption of David Finlayson's r.out.gmt.py script may be the quickest approach.

high level scripting language such as Python) will be posted to the GRASS wiki as soon as it is completed [6].

GMT provides several tools for manually creating a color palette file based on a user-supplied data range, or the data range as read from a GMT grid file. While this is not an ideal approach, it can be useful when one of the standard GMT color palettes is appropriate. In the special case where the exported image is grayscale, the standard "gray.cpt" color palette can be used to create a new cpt file with the `makecpt` program.

```
# make a suitable color palette for a grayscale
# CELL raster, the grid values are integers
# ranging from 0-255
makecpt -Cgray -T0/255/1 -V > doqq.cpt
```

When working with color channels that have already been separated (e.g. landsat or other multispectral imagery) it is possible to use `psimage` without a color palette file, instead specifying red, green, and blue input grid files. This approach is most convenient when working with data that has been collected in separate bands. For single band data, color separates can be created with `r.mapcalc`.

```
# get color-seperates ready for GMT

r.mapcalc "some_raster.red = r#some_raster"
r.mapcalc "some_raster.blue = b#some_raster"
r.mapcalc "some_raster.green = g#some_raster"
```

Exporting and Plotting GRASS Vector Data

The GMT vector format is based on a very simple, vertex-based ASCII format, designed primarily to store geometry data. However, with some scripting it is possible to encode simple attribute data within this format for the production of thematic maps. An alternative approach involves exporting vector data that has been pre-filtered with a GRASS tool such as `v.extract`, into several files where each represents a specific class. Using the first approach (encoding attributes within the GMT vector file) the symbology for each point, line, or polygon is encoded within the file, whereas the symbology is set with flags on the command line for the second approach.

Point Data

Exporting point data is the simplest GRASS → GMT vector conversion. In most cases, where a set of points is to be plotted in GMT using a fixed symbology, the output of `v.out.ascii` or `v.out.ascii.db` can be directly plotted with the GMT command `psxy`. Note that filtering the output from either command with UNIX text processing tools is a convenient method for converting attributes into symbology or adjusting labelling.

```
# export from GRASS to ASCII format
v.out.ascii in=points fs=" " > points.xy

# plot points with a white outline and blue fill,
# using a circular symbol 0.125 inches in diameter
# appending the resulting Postscript to 'outfile.eps'
psxy points.xy -R -J -M -Sc0.125 -G255/1/1 \
-W1/255/255/255 -O -K >> output.eps
```

Labeling point data can be accomplished by first filtering the output from `v.out.ascii` with a simple `awk` script, then using the output with the `pstext` command. The labeling format used by GMT is documented on the `pstext` manual page, however a simple explanation is given within the sample code below.

```
# export from GRASS to ASCII format with selected
# attributes ('ID' column)
# filter output with awk, re-ordering columns,
# and inserting label properties into the form:
# x_coord, y_coord, font_size, rotation_angle,
# font_number, label_offset, label_text
v.out.ascii.db in=points columns=ID | awk -F"|"
'{print $2, $3, 10, 0, 4, "BL", $4}' \
> points_with_labels.xy

# plot label text from geometry and label
# properties stored in 'points_with_labels.xy'
# label text will be blue and offset by 0.1 cm
# in the horizontal and vertical direction
# specified in column 5 of the input file
pstext points_with_labels.xy -R -J -Dj0.1c/0.1c
-G0/0/255 -K -O >> outfile.eps
```

Line and Polygon Data

There is currently no direct method for converting GRASS line and polygon data into a GMT compatible format. However, through the use of an external application called `shp2gmt` [7], it is possible to create GMT compatible vector files by first exporting GRASS data to shapefile and then converting with `shp2gmt`. The text output from `shp2gmt` contains all of the attribute data stored in the input shapefile

[7] This small utility program originally written by (Frank Warmerdam), was modified by (Mark Fenbers) to include attribute information from the shapefile as well. The code for the modified `shp2gmt` can be found at http://www.arcknowledge.com/gmane.comp.gis.gmt.user/2004-01/msg00121.html

along with vertex coordinates, and can be easily filtered with awk to include symbology parameters. An example GRASS session illustrating this type of operation is listed below.

```
# export from GRASS to shapefile format
v.out.ogr -e in=lines dsn=. olayer=lines
v.out.ogr -e in=polys dsn=. olayer=polys

# convert from shapefile format to GMT format
shp2gmt lines.shp > lines.xy
shp2gmt polys.shp > polys.xy

# optionally filter GMT vector file to include symbology
awk '
{
# all multi-record line segments with the
# attribute 'some_value'
# will be plotted with a red pen
if ($0 ~ /some_value/) printf "> -Wred\n"
# ... add more lookups for the other classes
# for lines that do not match, just print them
# verbatim (i.e. the vertex data)
else print
}
' lines.xy > lines-thematic.xy

# plot lines, with symbology set within the GMT vector file
psxy lines-thematic.xy -R -J -O -K >> outfile.eps
```

Labeling line or polygon features follows the same general approach as labelling point features, through the use of pstext. For simple maps, an almost automatic approach to labelling polygon data involves exporting polygon centroids with v.out.ascii.db, as listed above. Labelling line data requires more user interaction, as each label needs a coordinate and angle associated with it. With a little work it is possible to use the file created by v.label to produce a GMT-compatible set of label placement instructions. An example awk script, used within a GRASS session, is presented below.

```
# create the GRASS label file:
# rotating the labels to match the line segments
v.label -a map=trails column=trail_name labels=trails.lab

# convert the GRASS labels format to GMT format
awk '
BEGIN{FS="\n" ; RS="\n\n"}
{
split($1, e, ": ")
split($2, n, ": ")
split($15, r, ": ")
split($16, l, ": ")
# if there is no rotation, we need to add a 0-rotation
if(NF == 16) {rotation = r[2] ; label = l[2]}
else {rotation = 0 ; label = r[2]}
# create the GMT labeling instructions
print e[2], n[2], 10, rotation, 4, label
}
' DATABASE/LOCATION/MAPSET/paint/labels/trails.lab
> labels.xy

# plot with pstext
pstext labels.xy -R -J -Dj0.1c/0.1c -G0/0/255 -K -O
>> outfile.eps
```

While the above approach may work for simple maps, user intervention is usually required for more complex maps. A more robust approach for labeling point, line, and polygon features with advanced capabilities such as label collision detection is needed for a fully automated approach.

Miscellaneous Map Elements

Additional elements such as a context map (psbasemap with psxy), scalebar (psbasemap), or legend (pslegend) can be added after primary map components have been added to the output file. It is always a good idea to check that the last GMT command run does not use the -K flag, to insure that the output file is finalized. Through careful use of the -O, -X, and -Y flags it is possible to re-define the plotting region for each call to a GMT program. This approach works well for creating a mini-sized context map, within another finished map. Extending the first example, below is a more complete script illustrating the use of psbasemap to produce map elements such as a scale bar and north arrow, along with the creation of a mini-context map within a map.

```
# define some global settings:
gmtset ANNOT_FONT_PRIMARY Times-Roman \
HEADER_FONT_SIZE 16 \
ANNOT_FONT_SIZE_PRIMARY 12 \
LABEL_FONT_SIZE 14 \
BASEMAP_TYPE plain \
PLOT_DEGREE_FORMAT DF \
PAPER_MEDIA letter+

# sample map centered on the western USA
pscoast -JB-116/36/30/42/7i \
-R-125/-108/31/44 -B5 \
-Gwhite -W0.5p \
-A250 -Dh -Na -Xc -Yc -P -K
> query_centers_2.eps

# plot mapxy points:
psxy sample_data/mapxy_locations.latlong -J -R \
-Sc0.075c -W1/1/200/1 -G1/200/1 -O -K
>> query_centers_2.eps

# plot ka-map points
psxy sample_data/ka-map_locations.latlong -J -R \
-Sc0.075c -W1/255/1/1 -G255/1/1 -O  -K
>> query_centers_2.eps

# add scalebar
psbasemap -J -R -Lf-119/32.5/32/150k:"Kilometers": \
```

```
-O -K -P -V >> query_centers_2.eps

#make a context map
pscoast -JB-116/36/30/42/7i \
-R-130/-100/25/50 \
-X0.25i -Y1.75i \
-Gwhite -W0.5p \
-A250 -Di -Na -P -K >> query_centers_2.eps

# add the region box
psxy context_box.xy -M -JB -R  -W3/1/1/0 -P -O
>> query_centers_2.eps
```

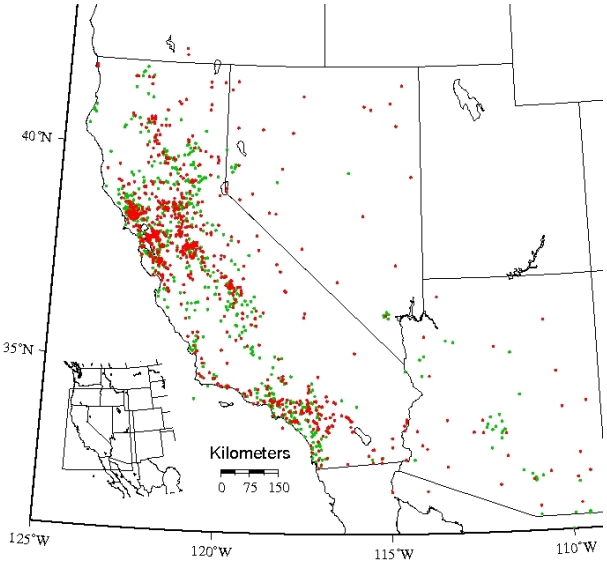

Figure 2: Addition of a context map and scalebar.

Current Limitations

The current unix shell approach to interfacing GRASS and GMT has several serious disadvantages: lack of portability to non-unix systems, lack of sophisticated mathematical operators, and a dependancy on several helper tools such as `awk`, `sed`, `grep`, etc. These disadvantages are particularly annoying when trying to automatically construct map elements such as tick intervals or scale-based decorations. The examples in this article use manually-defined tick intervals, however when creating maps from GRASS data (which is usually in a projected coordinate system) it is time consuming to manually set the annotated and non-annotated tick intervals for each map. See the `template.sh` script included in the sample data archive for ideas. Again, tighter integration with GRASS via the Python bindings would solve many of the above problems. Until a fully-Python implementation is complete, estimation of automatic tic intervals can be simplified with the following perl script.

```
#!/usr/bin/perl -w

# the first argument is the maximum extent in map units
# divide by a resonable number of annotations per edge
$x = $ARGV[0] / 10;

print round_up($x) , "\n";

# round to a resonable scale
# found at:
# http://www.perlmonks.org/?node_id=599865

sub round_up {
  my $n = shift;
  my $scale = 10**int(log($n)/log(10));
  $n = 9 if $scale == 1; #magic for single digits
  if ($n > $scale) {
    $n = int($n/$scale+1)*$scale;
  }
  $n;
}
```

Conclusions

This article summarizes the current condition of converting GRASS data into GMT format for the production of high quality, Postscript maps. Although several templates and examples exist to automate this process, for complex map production considerable modifications by the user are required. Given sufficient progress on projects like the GRASS GUI and Python bindings, it may be possible to create a more complete system for streamlining the GRASS → GMT workflow. There is currently an effort lead by Brent Wood to overhaul the GMT vector format which would simplify the process of thematic mapping with GMT. In addition Brent is working on including write-support for the GMT vector format into OGR (GDAL). Once completed, vector export from GRASS to GMT could be as simple as `v.out.ogr format=GMT`. Proposed work by Wolf Bergenheim, on a general label collision detection and correction algorithm for `v.label`, would be an excellent solution to complex labeling tasks in both GRASS and GMT. Tune in next time for an in-depth example of creating a complex map using GRASS data from the Spearfish sample data.

Dylan Beaudette
University of California at Davis
http://casoilresource.lawr.ucdavis.edu
debeaudette AT ucdavis.edu

Acknowledgements
Various reviewers & the GRASS News Project

Newsletter online: http://www.osgeo.org/journal

OSGeo Homepage: http://www.osgeo.org

Mail contact through OSGeo, PO Box 4844, Williams Lake, British Columbia, Canada, V2G 2V8

ISSN 1994-1897