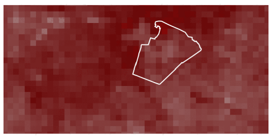
Table of contents

TGRASS: temporal data processing with GRASS GIS

FOSS4G Europe 2017 workshop

Veronica Andreo, Luca Delucchi and Markus Neteler



Course outline:

- Introduction to GRASS GIS
- Introduction to GRASS Temporal Framework
- Hands-on to raster time series processing

Software requirements:

- GRASS GIS 7.2: download | OSGeo-live
- GRASS GIS Add-ons: r.modis.download, r.modis.import and v.strds.stats
- pyModis library

Sample data:

- Main dataset: North Carolina basic location (50 MB)
  - extra "mapset" `modis_lst`: monthly land surface temperature (LST) from MODIS sensor (MOD11B3.006) for North Carolina (2015-2016)

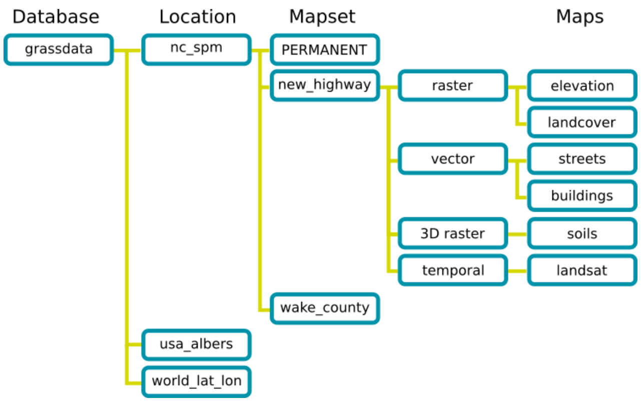
GRASS GIS introduction

Working with GRASS GIS is not much different from any other GIS. Just a few commonly used terms need to be introduced first.

GRASS DATABASE, LOCATION and MAPSET

The **GRASS DATABASE** (also called "GISDBASE") is an existing directory which contains all GRASS GIS projects. These projects are organized in subdirectories called LOCATIONs.
A **LOCATION** is defined by its coordinate system, map projection and geographical boundaries.
MAPSETs are subdirectories within Locations. In a **MAPSET** you can organize GIS maps thematically, geographically, by project or however you prefer.

Table of contents

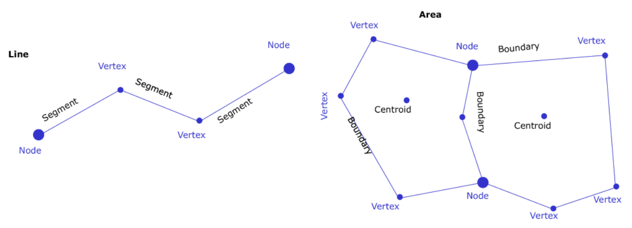


GRASS DATABASE, LOCATIONs and MAPSETs

Raster and vector maps

GRASS GIS is able to read most GIS data formats directly (mainly done through the GDAL library). GRASS GIS has its own internal formats to manage raster and vector data, so your data have to be imported or linked into a GRASS LOCATION/MAPSET.

The **GRASS GIS vector format is topological**, this means that adjacent geographic components in a single vector map are related to each other. For example, in a non-topological GIS if two areas share a common border that border would be digitized twice and also stored in a duplicate manner. In a topological GIS such as GRASS GIS, this border exists only once and it is shared between these two areas. The topological representation of vector data helps to produce and maintain vector maps with clean geometry. Moreover, it enables certain analyses that can not be conducted with non-topological or spaghetti data.



All vector data types in GRASS GIS

Modules

GRASS GIS is composed of more than 450 modules to perform any kind of GIS analysis. It is possible to analize raster, raster 3D, imagery and vector maps along with their alphanumerical attributes.
The wealth of modules is organized by their first name in order to easily find the desired functionality. The graphical user interface offers a tree view as well as a search engine.

GRASS GIS module families

Prefix	Function class	Type of command	Example
g.*	general	general data management	g.rename: renames map
d.*	display	graphical output	d.rast: display raster map, d.vect: display vector map
r.*	raster	raster processing	r.mapcalc: map algebra, r.univar: univariate statistics
v.*	vector	vector processing	v.clean: topological cleaning
i.*	imagery	imagery	i.pca: Principal Components

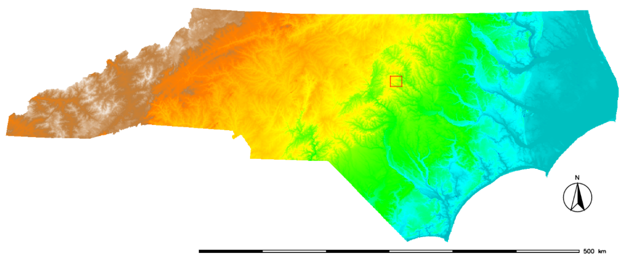
		processing	Analysis on imagery group
r3.*	voxel	3D raster processing	r3.stats: voxel statistics
db.*	database	database management	db.select: select value(s) from table
ps.*	postscript	map creation in PostScript format	ps.map: PostScript map creation
t.*	temporal	space-time datasets	t.rast.aggregate: raster time series aggregation

Table of contents

It is also possible to install further modules, called **Add-ons**, from a centralized GRASS GIS Add-on repository at OSGeo or from github using the command *g.extension*.

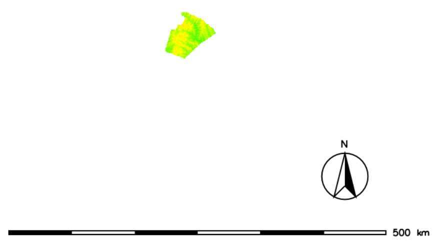
Region and Mask

The **computational (or current) region** is the *actual setting of the region boundaries and the actual raster resolution*.
The raster input maps are automatically (on the fly) cropped/padded and rescaled to match the current region, the output maps have their bounds and resolution equal to those of the current computational region, while vector maps are always considered completely.



The red box shows the computational region currently set

MASK: if a mask is set, raster modules will operate only on data falling inside the masked area(s), i.e. any data falling outside of the mask are treated as if their pixel value were NULL. To set the mask you can use r.mask or create a raster called `MASK` (this raster map name is reserved for this purpose).



By setting the MASK (here based on a ZIP code) only the raster data inside the masked area are used for further analysis

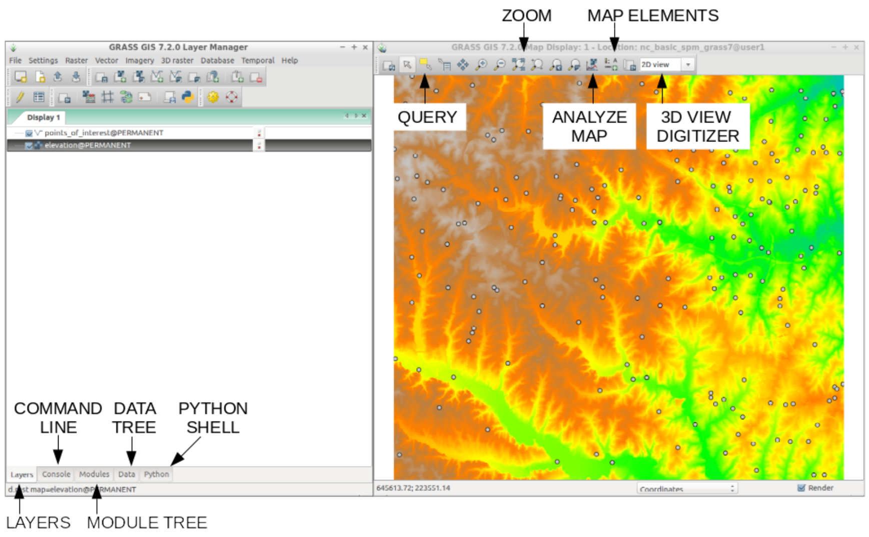
Interfaces

GRASS GIS offers different interfaces for the interaction between user and software. Let's see them...

Graphical User Interface

Table of contents

The GUI is the simpler way to approach GRASS GIS. The GRASS GIS GUI is composed of two elements, the `Layer Manager` where you can find all the GRASS GIS modules and manage your data and, the `Map Display` where you can navigate, print and query your maps. The GUI also comes with a Python shell for rapid prototyping.



GRASS GIS Graphical User Interface (GUI)

Otherwise you can also use GRASS GIS inside QGIS by either using the `GRASS GIS plugin` or through `Processing`.

Command line

The command line is the traditional and, probably, the most powerful way to use GRASS GIS, used daily by many GRASS GIS power users worldwide.

Python

Python is a powerful and simple programming language, and you can use it to:

- interface with the functionalities offered by GRASS GIS
- create your own workflows chaining several GRASS GIS modules
- create new add-ons by using GRASS GIS modules along with a wide number of Python libraries.

Here's an example of Python code to run GRASS GIS modules:

Python

```python
!/usr/bin/env python

# simple example for pyGRASS usage: raster processing via modules approach

from grass.pygrass.modules.shortcuts import general as g
from grass.pygrass.modules.shortcuts import raster as r

g.message("Filter elevation map by a threshold...")

# set computational region
input = 'elevation'
g.region(rast=input)

# hardcoded:
# r.mapcalc('elev_100m = if(elevation > 100, elevation, null())', overwrite =

# with variables
output = 'elev_100m'
thresh = 100.0
r.mapcalc("%s = if(%s > %d, %s, null())" % (output, input, thresh, input), over
r.colors(map=output, color="elevation")
```

WPS - OGC Web Processing Service

It is possible to run GRASS GIS modules through the web using the Web Processing Service (WPS is an OGC standard). The Free and Open Source software ZOO-Project and PyWPS allow the user to run GRASS GIS commands in a simple way.

Temporal GRASS GIS introduction

GRASS GIS is the first Open Source GIS that incorporated capabilities to manage, analyze, process and visualize spatio-temporal data, as well as the temporal relationships among time series.

Importantly, the **TGRASS** concept is based on metadata and does not duplicate any datasets. It follows a *snapshot* approach, i.e.: add time stamps to existent maps, to integrate time dimension to the classical spatial GIS. In GRASS GIS words, all pixels in a raster map and all elements in a vector map, share the same time stamp. A collection of time stamped maps (snapshots) of the same variable are called **space-time datasets (STDS)** in TGRASS. Each map in a STDS might have a different spatial and temporal extent.

TGRASS uses an SQL database to store the temporal and spatial extension of STDS, as well as the topological relationships among maps and among STDS.

Terminology overview

Temporal database

A temporal database is a mapset-specific database in which all time stamped maps are registered, i.e.: their spatial and temporal extents, unique ID and map type metadata are stored. This allows users to perform complex SQL queries using the spatio-temporal extent and metadata information for map selection (See Temporal data analysis section).

Space-time datasets

Space-time datasets are called differently according to the map type they are formed of:

- Space time raster datasets (**STRDS**) collections of time stamped raster maps.
- Space time 3D raster datasets (**STR3DS**) collections of time stamped 3D raster maps.
- Space time vector datasets (**STVDS**) collections of time stamped vector maps.

Spatio-temporal modules

- **t.***: General modules to handle STDS of all types
- **t.rast.***: Modules that specifically process STRDS
- **t.rast3d.***: Modules that specifically process STR3DS
- **t.vect.***: Modules that specifically process STVDS

Absolute time vs relative time

Two time definitions are used in TGRASS:

Absolute time	Relative time
Gregorian calendar (ISO 8601 time format notation)	An integer and a time unit (year, month, day, hour, minute, seconds)

Table of contents

Example: 2013-10-15 13:00:00	Example: 4 years, 90 days, 1 second

Time intervals vs time instances

GRASS GIS supports both time intervals and time instances as map time stamps:

Time intervals	Time instances
Defined by start time and end time: [start, end)	Defined by start time
Support for gaps, allow overlapping, might contain time instances, might be irregularly spaced in time	Punctual event

Granularity

The granularity is the largest common divider granule of time intervals and gaps between intervals or instances from all time stamped maps that are collected in a STDS. It is represented as a number of seconds, minutes, hours, days, weeks, months or years.

Workflow overview

Now, how do we work with time series in GRASS GIS? Where do we start? Well, assuming we already have our maps in the GRASSDBASE, the first step is to set the connection to the temporal database. This is something we will need to do only once per mapset. After that, we create the STDSs, i.e.: the containers for the time series, and finally we assign time stamps to maps and register them in the STDS. After these basic steps, the following will depend on our specific objectives.

1. Set and connect the temporal database (mapset specific): t.connect
2. Create the STDS (raster, vector or raster 3D): t.create
3. Assign time stamps to maps and register them in the STDS: t.register
4. Check basic info, integrity and validity of STDS: t.list, t.info, t.topology
5. Edit, update, unregister, remove maps and/or STDS: t.rename, t.remove, t.support, t.unregister
6. List maps, make selections, get univariate statistics: t.rast.list, t.vect.list, t.select, t.rast.extract, t.vect.extract, t.rast.univar, t.vect.univar
7. Spatio-temporal processing (a bit raster biased):
   - data aggregation: t.rast.series, t.rast.aggregate, t.rast.aggregate.ds
   - data accumulation: t.rast.accumulate, t.rast.accumulate,
   - gap-filling and smoothing: t.rast.gapfill, t.rast.neighbors
   - spatio-temporal algebra: t.rast.mapcalc, t.rast.algebra, t.vect.algebra
8. Visualization: g.gui.timeline, g.gui.tplot, g.gui.animation, g.gui.mapswipe

Hands-on to raster time series processing

Getting the MODIS satellite sensor data

MODIS is a payload scientific instrument on board the NASA Terra and the Aqua satellites with 36 spectral bands. Data are available for download upon user registration.

Create the *SETTING* file with the following content, e.g. in the directory `$HOME/gisdata/`:

```
your_NASA_user
your_NASA_password
```

The *r.modis.download* directly downloads from the MODIS data server into a local folder previously created (for example: `lst_modis`):

Standard Bash Python

```
r.modis.download -g settings=$HOME/gisdata/SETTING product=lst_terra_monthly_5
r.modis.import -w files=$HOME/lst_monthly/listfileMOD11B3.006.txt spectral="( 
g.list type=raster pattern="MOD11B3*"
```

We will now visualize one of the MODIS LST images and add different map decorations. We first change the color palette from grey to viridis and set the computational region to the state of North Carolina (using `boundary_state` vector map).

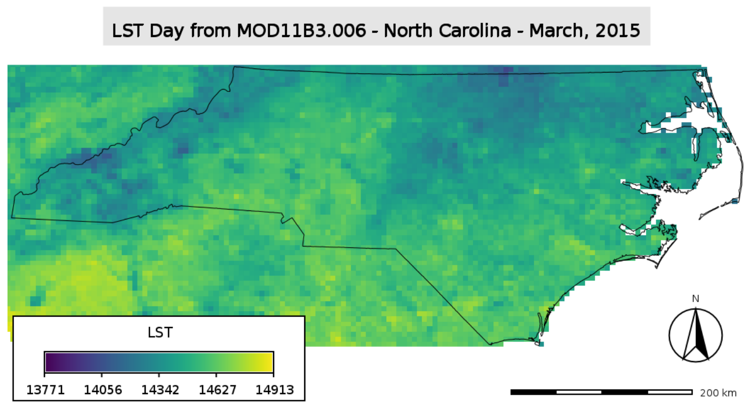
Standard Bash Python

```
r.colors MOD11B3.A2015060.h11v05.single_LST_Day_6km color=viridis
g.region -p vector=boundary_state
```

We can open a monitor and run the commands from the command line or do everything in the main GUI and copy the commands for future reference or replication.

Standard GUI

```
# Open a monitor
d.mon wx0

# Display raster map
d.rast map=MOD11B3.A2015060.h11v05.single_LST_Day_6km

# Display vector map
d.vect map=boundary_state type=boundary

# Add raster legend
d.legend -t -s -b raster=MOD11B3.A2015060.h11v05.single_LST_Day_6km \
 title=LST title_fontsize=20 font=sans fontsize=18

# Add scale bar
d.barscale length=200 units=kilometers segment=4 fontsize=14

# Add North arrow
d.northarrow style=1b text_color=black

# Add text
d.text -b text="LST Day from MOD11B3.006 - North Carolina - March, 2015" \
 color=black bgcolor=229:229:229 align=cc font=sans size=8
```



MODIS LST map with decorations (scaled Kelvin pixel values)

Create temporal raster dataset (STRDS)

To create a space-time raster data set (STRDS) implies to create an SQLite table in the temporal database, i.e.: a container table, that will hold our raster time series and will allow us to easily handle huge amounts of maps by only using the STRDS as input in temporal commands.

Table of contents

If this is the first time you use the temporal framework, you need to create and set the connection to the temporal database by means of *t.connect*. As the temporal database is mapset specific, you'll need to repeat this step in each mapset in which you'll have STDSs.

Once the connection is set, you can create the empty STRDS, i.e.: the empty table, in which you'll put (aka: register) all your time series maps afterwards. For the creation of any STDS, we need to specify which **type of maps** (raster, raster3d or vector) the STDS will contain and which **type of time** (absolute or relative) the maps represent.

Standard | Bash | Python

```
t.connect -d
t.create type=strds temporaltype=absolute output=LST_Day_monthly title="Monthly
t.list type=strds
t.info input=LST_Day_monthly
```

Once the STRDS is created, we assign time stamps to maps and add them to the STRDS, i.e.: we *register* maps in the STRDS. To register maps in a STDS, we need to pass the empty STDS as input and the list of maps to be registered. There are different ways to register maps in STDS. For more options, you can check the *t.register* manual and the related wiki page.

Standard | Bash | Python

```
t.register input=LST_Day_monthly file=$HOME/lst_monthly/monthly_lst_to_registe
```

Alternatively, we could have registered our raster maps using the `maps`, `start` and `increment` options along with the `i` flag for interval creation. The command in that case would look as follows:

Standard

```
t.register -i input=LST_Day_monthly \
 maps=`g.list type=raster pattern=MOD11B3*LST_Day* separator=comma` \
 start="2015-01-01" increment="1 months"
```

Let's check now the basic info again to see how it looks like and list the raster maps in our `LST_Day_monthly` STRDS:

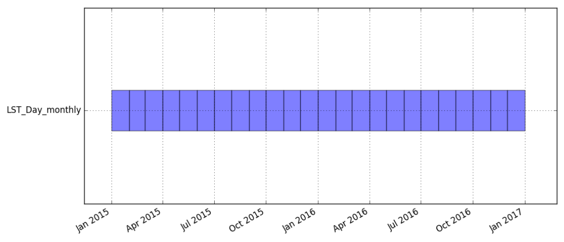
Standard | Bash | Python

```
t.info LST_Day_monthly
t.rast.list LST_Day_monthly
```

Let's see our STRDS graphically. We will use the *g.gui.timeline* tool.

Standard | GUI

```
g.gui.timeline inputs=LST_Day_monthly
```



Timeline plot for LST_Day_monthly time series

Temporal data analysis

Table of contents

One basic but very important function when handling hundreds or thousands of maps is the listing function, i.e.: we usually need to list maps that meet a certain condition. For example, we need maps which start month is June, maps with minimum values lower than 100, and so on. The GRASS GIS Temporal framework has different commands for that task: *t.list* for listing STDS and maps registered in the temporal database, *t.rast.list* for maps in raster time series and, *t.vect.list* for maps in vector time series. All these commands allow us to list STDSs and/or maps according to different criteria. Let's see some examples with our `LST_Day_monthly` STRDS:

Standard

```
# Maps with minimum value lower than or equal to 14000
t.rast.list input=LST_Day_monthly order=min columns=name,start_time,min where='

name|start_time|min
MOD11B3.A2015032.h11v05.single_LST_Day_6km|2015-02-01 00:00:00|12950.0
MOD11B3.A2016032.h11v05.single_LST_Day_6km|2016-02-01 00:00:00|12964.0
MOD11B3.A2015001.h11v05.single_LST_Day_6km|2015-01-01 00:00:00|13022.0

# Maps with maximum value higher than 14000
t.rast.list input=LST_Day_monthly order=max columns=name,start_time,max where='

name|start_time|max
MOD11B3.A2016001.h11v05.single_LST_Day_6km|2016-01-01 00:00:00|14360.0
MOD11B3.A2015001.h11v05.single_LST_Day_6km|2015-01-01 00:00:00|14396.0
MOD11B3.A2015032.h11v05.single_LST_Day_6km|2015-02-01 00:00:00|14522.0

# Maps between two given dates
t.rast.list input=LST_Day_monthly columns=name,start_time where="start_time >=

name|start_time
MOD11B3.A2015121.h11v05.single_LST_Day_6km|2015-05-01 00:00:00
MOD11B3.A2015152.h11v05.single_LST_Day_6km|2015-06-01 00:00:00
MOD11B3.A2015182.h11v05.single_LST_Day_6km|2015-07-01 00:00:00
MOD11B3.A2015213.h11v05.single_LST_Day_6km|2015-08-01 00:00:00

# Maps from January
t.rast.list input=LST_Day_monthly columns=name,start_time where="strftime('%m'

name|start_time
MOD11B3.A2015001.h11v05.single_LST_Day_6km|2015-01-01 00:00:00
MOD11B3.A2016001.h11v05.single_LST_Day_6km|2016-01-01 00:00:00

# Maps from June, 1st
t.rast.list input=LST_Day_monthly columns=name,start_time where="strftime('%m-

name|start_time
MOD11B3.A2015152.h11v05.single_LST_Day_6km|2015-06-01 00:00:00
MOD11B3.A2016153.h11v05.single_LST_Day_6km|2016-06-01 00:00:00
```

To explore a bit more our time series, we will obtain univariate statistics for the maps in the STRDS. There's a dedicated module for that: *t.rast.univar*. There's also the possibility to obtain extended statistics such as first quartile, median value, third quartile and percentile 90 by setting the `e` flag. Let's see:

Standard | Bash | Python

```
t.rast.univar input=LST_Day_monthly

id|start|end|mean|min|max|mean_of_abs|stddev|variance|coeff_var|sum|null_cells
MOD11B3.A2015001.h11v05.single_LST_Day_6km@modis_lst|2015-01-01 00:00:00|2015-(
MOD11B3.A2015032.h11v05.single_LST_Day_6km@modis_lst|2015-02-01 00:00:00|2015-(
MOD11B3.A2015060.h11v05.single_LST_Day_6km@modis_lst|2015-03-01 00:00:00|2015-(
MOD11B3.A2015091.h11v05.single_LST_Day_6km@modis_lst|2015-04-01 00:00:00|2015-(

t.rast.univar -e input=LST_Day_monthly

id|start|end|mean|min|max|mean_of_abs|stddev|variance|coeff_var|sum|null_cells
MOD11B3.A2015001.h11v05.single_LST_Day_6km@modis_lst|2015-01-01 00:00:00|2015-(
MOD11B3.A2015032.h11v05.single_LST_Day_6km@modis_lst|2015-02-01 00:00:00|2015-(
MOD11B3.A2015060.h11v05.single_LST_Day_6km@modis_lst|2015-03-01 00:00:00|2015-(
MOD11B3.A2015091.h11v05.single_LST_Day_6km@modis_lst|2015-04-01 00:00:00|2015-(

t.rast.univar input=LST_Day_monthly separator=comma output=stats_LST_Day_monthl
```

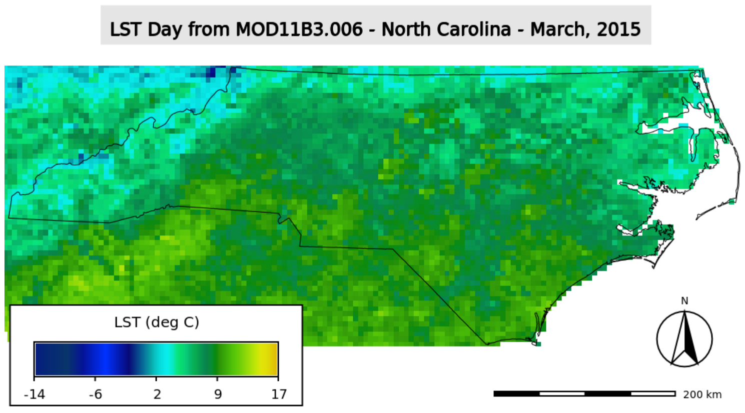
However, those statistics are a bit difficult to directly interpret since values are in °K*50. Let's re-scale the data to °C and run the previous command again. Then, we will set a proper color palette for the STRDS and display one map. The latter, you already know how to do.

Table of contents

To transform all the maps in our `LST_Day_monthly` time series into °C we will use the *t.rast.algebra* module. This module allows to perform a very wide variety of operations in the temporal and spatial domains, as well as much of the more *"classical"* operations already available in *r.mapcalc*.

Standard | Bash | Python

```
t.rast.algebra basename=LST_Day_monthly_celsius expression="LST_Day_monthly_cel
t.rast.univar input=LST_Day_monthly_celsius
t.rast.colors input=LST_Day_monthly_celsius color=celsius
```



MODIS LST re-scaled to degrees Celsius

Alternatively, we could have used *t.rast.mapcalc* to perform the previous transformation. The command would look like this:

Standard

```
t.rast.mapcalc input=LST_Day_monthly output=LST_Day_monthly_celsius basename=LS
```

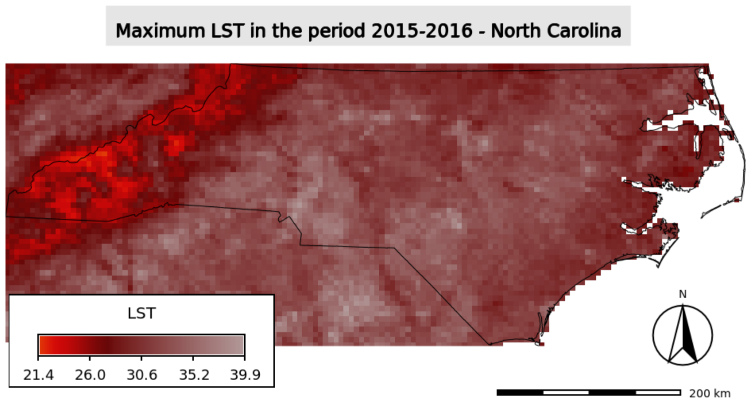
Temporal aggregations

There are basically two dedicated modules to perform temporal aggregations in GRASS GIS. The first one that we will use is *t.rast.series*. This module is a wrapper for *r.series* and allows us to aggregate our STRDS or parts of it with different methods. We will use it now to obtain the absolute maximum LST in the past two years.

Standard | Bash | Python

```
t.rast.series input=LST_Day_monthly_celsius output=LST_Day_max method=maximum
r.colors map=LST_Day_max color=celsius
d.mon wx0
d.rast LST_Day_max
d.vect map=boundary_state type=boundary
d.legend -t -s -b raster=LST_Day_max title=LST title_fontsize=20 font=sans font
d.barscale length=200 units=kilometers segment=4 fontsize=14
d.northarrow style=1b text_color=black
d.text -b text="Maximum LST in the period 2015-2016 - North Carolina" color=bla
```

Table of contents



Maximum LST in the period 2015-2016 in North Carolina

Now, by means of the spatio-temporal algebra, we will get the month in which the absolute maximum LST occurred. For that, we will first compare our `LST_Day_monthly_celsius` STRDS with the map of absolute maximum LST `LST_Day_max` that we just obtained before. If they coincide, we keep the month for that pixel, otherwise it will be NULL. Then, we aggregate the resulting `month_max_lst` STRDS with *t.rast.series* `method=maximum` and we get the map with the pixelwise month in which the absolute maximum LST has occurred in the past two years. Finally, we remove the intermediate STRDS, since we are only interested in the aggregated map.

Standard | >_ Bash | Python

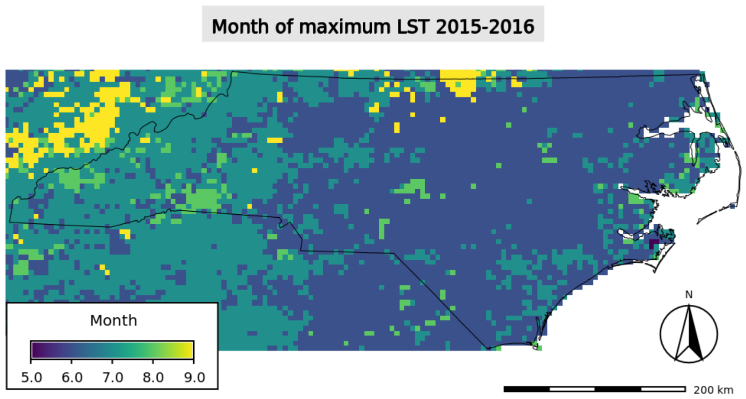
```
t.rast.mapcalc -n inputs=LST_Day_monthly_celsius output=month_max_lst expressic
t.rast.series input=month_max_lst method=maximum output=max_lst_date
t.remove -rf inputs=month_max_lst
```

Note that the flags `"-rf"` force (immediate) removal of both the STRDS (i.e.: the container table) and the maps registered in it.

Finally, we display the resulting map:

Standard

```
d.mon wx0
d.rast max_lst_date
d.vect map=boundary_state type=boundary
d.legend -t -s -b raster=max_lst_date title=LST title_fontsize=20 font=sans foı
d.barscale length=200 units=kilometers segment=4 fontsize=14
d.northarrow style=1b text_color=black
d.text -b text="Month of maximum LST 2015-2016" color=black bgcolor=229:229:22
```



Month of maximum LST for the period 2015-2016

The other module that allows us to perform temporal aggregations is *t.rast.aggregate*. With this module we are able to aggregate raster maps in our STRDS with different granularities. Note that this module also has the option `where` that allow us to set specific dates for the aggregation. We will use this module to get 3-month and 6-month average LST.

Standard | Bash | Python

```
t.rast.aggregate input=LST_Day_monthly_celsius output=LST_Day_mean_3month baser
t.info LST_Day_mean_3month
t.rast.list LST_Day_mean_3month

t.rast.aggregate input=LST_Day_monthly_celsius output=LST_Day_mean_6month baser
t.info LST_Day_mean_6month
t.rast.list LST_Day_mean_6month
```

Now, we will extract the 3-month average LST for points in a vector map. We will use the vector map `points_of_interest` that is already available in the PERMANENT mapset of NC Location. There are different commands that allow us to perform this task, but we'll use *t.rast.what*. This module samples a STRDS at specific vector point coordinates and writes the output to stdout using different layouts. You might explore the options available in the parameter `layout` to see the different ways to write the output.

Standard | Bash | Python

```
t.rast.what points=points_of_interest strds=LST_Day_mean_6month output=trastwha
```

Even though the most common is to extract raster data to points, we may also be interested in getting spatially aggregated time series data for polygons. GRASS GIS has an add-on for this: *v.strds.stats*. It calculates zonal statistics of each raster in a STRDS and writes the output to the attribute table of a new polygon vector map. We first need to install the add-on through *g.extension*. Then, we will extract the average, minimum and maximum monthly LST for the different geologic types in NC.

Standard | Bash | Python

```
g.extension v.strds.stats
v.strds.stats input=geology strds=LST_Day_monthly_celsius output=geology_aggr_l
v.db.select map=geology_aggr_lst file=ts_polygons.csv
```

Temporal data visualization

Aside from *g.gui.timeline* that we saw before, GRASS GIS offers different options to visualize time series data. The first one is *g.gui.mapswipe* that allows us to compare two maps from different dates for example by swiping a visibility bar. The module can be called from command line by specifying the two maps to compare or executed from the GUI as shown below in the respective tabs.

Standard | GUI

```
g.gui.mapswipe first=LST_Day_monthly_celsius_6 second=LST_Day_monthly_celsius_1
```

Another tool to visualize time series in more dynamic way and that allows us to see changes in space an time simultaneously is *g.gui.animation*. It also allows to animate vector time series, as well as lists of raster or vector maps not specifically registered as time series. When executed from command line, options are more limited than when the module is run from the GUI, where it is possible to add different decorations and customize several options.

Standard | GUI

```
g.gui.animation strds=LST_Day_monthly_celsius@modis_lst
```

Finally, if you want to plot the time series of our variable of interest, be it in a STRDS or a STVDS, for a specific point of your study region, GRASS GIS has *g.gui.tplot*.

Basically, you need to set the strds or stvds and a pair of X,Y coordinates. The latter can be typed directly, copied from the map display and pasted or directly chosen from the display. See instructions below.

Standard GUI

```
g.gui.tplot strds=LST_Day_monthly_celsius@modis_lst coordinates=419604.018913,
```

Other (very) useful links

- GRASS intro workshop: https://ncsu-osgeorel.github.io/grass-intro-workshop/
- Unleash the power of GRASS GIS: https://grasswiki.osgeo.org/wiki/Unleash_the_power_of_GRASS_GIS_at_US-IALE_2017
- Temporal data processing wiki: https://grasswiki.osgeo.org/wiki/Temporal_data_processing
- GRASS GIS temporal workshop: http://ncsu-geoforall-lab.github.io/grass-temporal-workshop/
- GRASS GIS and R for time series processing: https://grasswiki.osgeo.org/wiki/Temporal_data_processing/GRASS_R_raster_time_series_processing

References

- Gebbert, S., Pebesma, E. (2014). *A temporal GIS for field based environmental modeling*. Environmental Modelling & Software, 53, 1–12. DOI
- Gebbert, S., Pebesma, E. (2017). *The GRASS GIS temporal framework*. International Journal of Geographical Information Science 31, 1273-1292. DOI
- Neteler, M., Bowman, M.H., Landa, M. and Metz, M. (2012): *GRASS GIS: a multi-purpose Open Source GIS*. Environmental Modelling & Software, 31: 124-130 DOI
- Neteler, M., Mitasova, H. (2008): *Open Source GIS: A GRASS GIS Approach*. Third edition. ed. Springer, New York. Book site

Last changed: 2017-07-09 20:12

GRASS GIS manual main index | Temporal modules index | Topics index | Keywords Index | Full index

TGRASS: temporal data processing with GRASS GIS Workshop at FOSS4G Europe 2017 by Veronica Andreo, Luca Delucchi and Markus Neteler is licensed under a **Creative Commons Attribution-ShareAlike 4.0 International License** - Thanks to **Vaclav Petras** for the style.