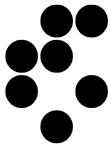
Institut "Jožef Stefan", Ljubljana, Slovenija
Department of Communication Systems (E-6)



GRASS-RaPlaT
Radio Planning Tool for GRASS

User Manual
V1.0a

Igor Ozimek, Andrej Hrovat, Andrej Vilhar, Tomaž Javornik

Ljubljana, September 2013

Contents

Figures

Tables

1. GRASS-RaPlaT overview

GRASS GIS [1,2], shortly GRASS (*Geographic Resources Analysis Support System*), is a free Geographic Information System (GIS) software used for geospatial data management and analysis, image processing, graphics/maps production, spatial modeling, and visualization. It is available as prebuilt packets for various Linux distributions, MS Windows and (Mac) OS X, as well as in source code.

RaPlaT (*Radio Planning Tool for GRASS*) [3,4,5] is an add-on for GRASS for radio signal coverage calculation. It uses the GRASS' support for geographic environment (terrain relief) and other GRASS functionalities (displaying, etc.) important for radio coverage computations and display.

RaPlaT comprises a set of C modules (small programs written in C, specifically for the GRASS environment) and Python scripts. They belong to the following groups:

1. A group of path loss model modules each calculating radio signal path loss according to a specific radio signal propagation model. The obtained raster diagram, which tells the path loss in [dB] in each point of the terrain surface, corresponds to a hypothetic isotropic transmission antenna with 0 dB gain. This group currently comprises the following modules:
    - *r.fspl* - Free Space Path Loss model,
    - *r.hata* - Okumura-Hata model,
    - *r.cost231* - COST 231 model,
    - *r.hataDEM* - modified Hata - Okumura-Hata DEM model,
    - *r.waik* - Walfish-Ikegami propagation model.
2. Module *r.sector*, which takes the isotropic path loss results calculated by a path loss model module, and modifies it according to the selected antenna characteristics (radiation pattern and gain), its position and orientation.
3. Module *r.MaxPower*, which calculates the received power in each raster point of the terrain surface for one or more transmission antennas (e.g. for a cellular communication network, like GSM or UMTS). In case of multiple transmission antennas, it calculates the maximum received power from any transmitter in each receive point (raster point on the terrain relief map), and can also build a data table (using e.g. MySQL or PostgreSQL) comprising the relevant data of a chosen number of strongest received signals in each receive point.
4. Script *r.radcov*, which helps the user by automatically calling the above modules. The user only uses *r.radcov* and does not need to deal directly with individual modules listed above.
5. Some other auxiliary modules (C-modules and Python scripts).

Most of the path loss model modules need only DEM (*Digital Elevation Map*, i.e. a raster map describing the terrain profile) for their computations. Some modules (currently only *r.hataDEM*) need also a so-called *clutter map*, which describes the signal fading at each raster point due to the land use or type of vegetation (e.g. buildings, roads, forest, grass, rivers, lakes, etc.). Unfortunately, geographical maps are generally not publicly and freely available. When using RaPlaT in a professional environment (e.g. by a mobile network operator) commercial DEM and clutter maps are normally available. For non-commercial use there are other options. GRASS GIS web page itself provides some demo locations in US with DEM and land-use maps, which can be used for demo purposes. Another option are publicly

available SRTM (*Shuttle Radar Topography Mission*) maps with global Earth coverage, which were generated by NASA and are based on radar measurements performed during a Space Shuttle mission in February 2000, [6,7].

The RaPlaT C modules are currently distributed as source code for Linux environment only. They have been tested under Ubuntu 10.04 and 12.04 with GRASS versions 6.4.0 and 6.4.3RC2. Before use, the modules must be compiled using standard Linux tools and the GRASS development environment. Standard precompiled GRASS Linux packages do not include the development support, so GRASS must be installed from its source code distribution. For someone familiar with Linux this is not a big problem, but to make RaPlaT more accessible for users with less Linux experience, we created a virtual machine with Ubuntu 12.04 and all the required components preinstalled. The virtual machine was prepared with WMware Player, which is free for non-commercial use and can run on Linux and MS Windows operating systems [8].

2. RaPlaT in details

The main overall structure of the RaPlaT tools is depicted in Fig 1. It consists of a number of path loss model modules (implementing different radio propagation models), the *r.sector* and *r.MaxPower* modules, and the *r.radcov* Python script which ties everything together. Input and output data are depicted in Fig. 1 as differently colored parallelograms - textual input and output files in orange, GRASS raster files in blue, and databases in yellow.

The user can call individual modules, however he/she would normally only call *r.radcov*, which in turn calls other modules as necessary.

The user defines the parameters of one or more radio transmitters together with the chosen path los models in a cell list file, which is a simple data table in the CSV (*Comma-Separated Values*) format [9,10]. The list of all available antenna types is given in another CSV format file, which references the actual antenna data files written in the standard MSI text format [18]. The *r.radcov* script first executes the required path loss model modules for the given set of transmitters (as specified in the cell list file), continues with calling *r.sector* for all the transmission antennas and finishes with calling *r.MaxPower* for calculation of the overall radio signal coverage.

RaPLaT path loss model modules and *r.sector* require a DEM map (DEM - *Digital Elevation Map*), which describes the terrain relief. Some path loss models (*r.hataDEM* from the above modules) additionally need a clutter map, which describes the signal loss due to the land-use (buildings, forests, lakes, etc.)

Fig. 1 depicts an additional module, *r.clutconvert*, which is used for creation of clutter maps (describing land-use-dependent signal loss) from general land-use GRASS raster maps.

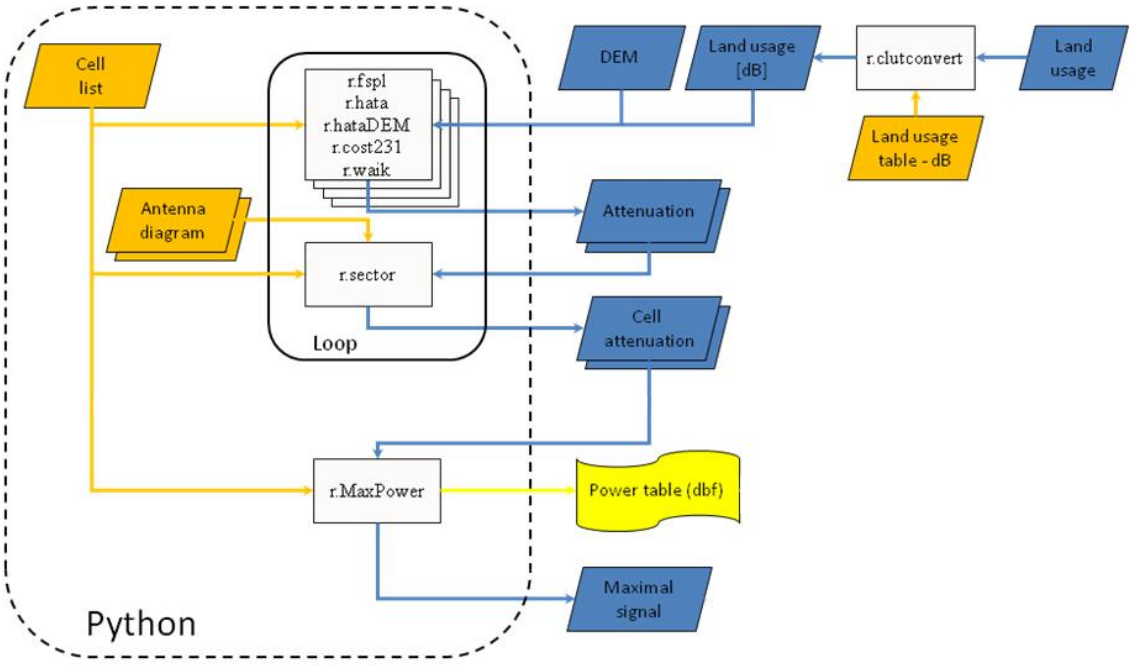


Fig. 1: GRASS-RaPlaT block diagram

GRASS modules generally work in the so called *current region*, which defines the geographic region extents and resolution (resampling of input maps, if necessary, is done automatically). The *r.radcov* script lets the user set the computation region independently of the current region (a temporal current region is established for the execution of the called modules). RaPLaT reduces the execution times of the path loss models and *r.sector* modules by additionally limiting computation to a circle with a given radius around each antenna. The points outside this area are assigned the *null* value (no signal received). The *r.radcov* script allows setting the radius independently for each antenna in the input cell list table (see below), or globally with the *radius_ovr* command line parameter.

Radio coverage computation requires a GRASS *location* cartographic projection with distances expressed in meters (e.g. the Gauss-Krueger coordinate system for Slovenia in our case). It cannot work correctly in a *location* with so called Longitude/Latitude pseudo projection, where locations and distances are expressed in angular degrees.

2.1. Run a complete radio coverage computation - r.radcov

A radio coverage computation could be accomplished by calling individual modules (described in details later in this document): isotropic path loss model modules, *r.sector* and *r.MaxPower*. Such use would be quite awkward and demanding, hence we created *r.radcov*, a Python script, which ties everything together and calls individual modules as necessary. The script gets the necessary information for radio coverage calculations from two tables written in the CSV text format, and from the *r.radcov* command line parameters.

RaPlaT can be used to calculate coverage by radio signals from multiple transmitters, as is the case with cellular networks (e.g. GSM radio network). The user describes the whole configuration in a cell list file ("cell list" here is actually a list of installed antennas with related data, as will be explained shortly). The file in the CSV format and can be created with OpenOffice Spreadsheet (but also with MS Excel in the MS Windows environment - *r.radcov* understands the peculiarities of the MS Excel CSV format, including its European version; the RaPlaT tool itself is currently supported only on Linux).

The cell list file is specified with the *csv_file* command line parameter and contains a table, an example of which is shown in Table 1 (three transmit antennas on a single location).

Table 1: An example of the cell list table

cellName	antennaID	beamDirection	electricalTiltAngle	mechanicalAntennaTilt	heightAGL	antennaType	positionEast	positionNorth	power	radius	model	P1	P2	P11
IJS-A	1	30	0	0	20	COS-21	460697	99918	30	10	hata	urban		
IJS-B	2	135	0	0	20	COS-21	460697	99918	30	10	hata	urban		
IJS-C	3	270	0	0	20	COS-21	460697	99918	30	10	hata	urban		

The corresponding CSV file (as generated by OpenOffice Spreadsheet) would be:

```
"cellName","antennaID","beamDirection","electricalTiltAngle","mechanicalAntennaTilt","heightAG
L","antennaType","positionEast","positionNorth","power","radius","model","P1","P2","P3","P4","
P5","P6","P7","P8","P9","P10","P11"
"IJS-A",1,30,0,0,20,"COS-21",460697,99918,30,10,"hata","urban",,,,,,,,,,
"IJS-B",2,135,0,0,20,"COS-21",460697,99918,30,10,"hata","urban",,,,,,,,,,
```

```
"IJS-C",3,270,0,0,20,"COS-21",460697,99918,30,10,"hata","urban",,,,,,,,,
```

The first line contains the header (in the example above, it is split into three lines to fit on the page, but should actually be a single line). Each following line contains data for one transmission antenna. The *r.radcov* script parses this table according to a special data structure defined by a Python script variable, *cellTableDescrib*, which will not be explained here (but is at least to a certain degree self-explanatory). The data columns, their types and value constraints as defined by this structure are shown in Table 2.

Table 2: Description of the cell list table columns

Name	Type	Allowed values	Description
cellName	name	(see description)	Cell name (characters 'A'..'Z', 'a'..'z', numbers, '_', '-')
antennaID	id	1..999999	Antenna identification number
beamDirection	i	0..360	Antenna horizontal direction (0: northwards; positive: clockwise)
electricalTiltAngle	i	0..10	Antenna electrical vertical tilt (downwards)
mechanicalAntennaTilt	i	-90..+90	Antenna vertical direction (positive: downwards)
heightAGL	f	0.0..300.0	Antenna height above the terrain
antennaType	antype	unconstrained	Antenna type
positionEast	i	unconstrained	Antenna position – E-W in [m]
positionNorth	i	unconstrained	Antenna position – N-S in [m]
power	f	0.0..140.0	Transmission power in [dBm] (1mW..10kW)
radius	f	0.0..1000.0	Max. distance of the receiver in [km]
model	s	'hata', 'cost231', 'hataDEM', 'waik', 'fspl', 'itm'	Radio signal path loss model
Parameters P1..P11 for the Hata model			
P1	s	'urban', 'suburban', 'open'	Area type for the Hata model
P2 .. P11	-	*(not used)*	
Parameters P1..P11 for the Cost231 model			
P1	s	'metropolitan', 'medium_cities'	Area type for the Cost231 model
P2 .. P11	-	*(not used)*	
Parameters P1..P11 for the hataDEM model			
P1	f	Unconstrained	Parameter A0 for the hataDEM model
P2	f	Unconstrained	Parameter A1 for the hataDEM model
P3	f	Unconstrained	Parameter A2 for the hataDEM model
P4	f	Unconstrained	Parameter A3 for the hataDEM model
P5 .. P11	-	*(not used)*	
Parameters P1..P11 for the Walfisch-Ikegami (waik) model			
P1	f	20..60	Parameter W0 (*Free space loss correction*)
P2	i	30..70	Parameter W1 (*Reduced base antenna height correction*)
P3	i	5..35	Parameter W2 (*Range correction*)
P4	i	3..15	Parameter W3 (*Street width correction*)
P5	i	3..25	Parameter W4 (*Frequency correction*)
P6	i	10..30	Parameter W5 (*Building height correction*)
P7	i	10..25	Parameter W6 (*Street width [m]*)
P8	i	20..50	Parameter W7 (*Distance between buildings [m]*)
P9	i	0..300	Parameter W8 (*Building height [m]*)
P10	i	0..180	Parameter PHI_Street (*Street orientation [deg]*)
P11	s	'metropolitan', 'medium_cities'	Area type
Parameters P1..P11 for the fspl (»free space«) model			
P1 .. P11	-	*(not used)*	

Type means:

- *name* : character string (see description in the table). Used for the cell names.
- *id* : integer value, similar to type *i* (see below) but values must be unique (the same value may not repeat). Used for antenna identification numbers in multi antenna systems (cellular networks).
- *antype* : character string, allowed characters: letters, numbers, ' ', '-', '/' and '.' in any order. Used to define for the antenna types.
- *i* : integer value with min and max bounds. If the value of both min and max bounds is 0, the value is unbounded. The decimal point/comma is not allowed.
- *f* : floating point value with min and max bounds. If the value of both min and max bounds is 0.0, the value is unbounded. The value can be written in the cell list file without the decimal point/comma.
- *s* : a word (character string) form a set of allowed words. The columns to the right can depend on this word (as defined by the *cellTableDescrib* variable; e.g. the P*n* columns depend on the word (model name) in the *model* column).
- - : arbitrary contents.

Empty lines are ignored. The character # as the first character of the cell name or the first character in a line has a special meaning: it marks the cell as a comment only, effectively disabling the cell. This is useful for simple and quick enabling/disabling of individual cells.

An example of radio coverage map (received signal strengths in [dBm]) for a system with three antennas on a single location is shown in Fig. 2. The antennas used here were not a real product but a mathematically created cosine type with half-power (-3 dB) beam width of about 30° and gain 0 dBd.

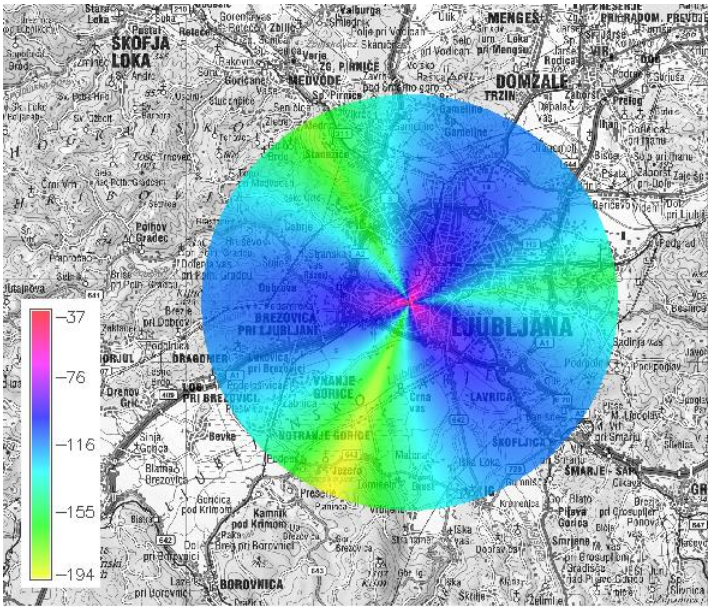


Fig. 2: Coverage by three antennas on one location (*r.hata*, 900 MHz)

- Usage:

```
r.radcov [-rpc1] csv_file=string antmap_file=string dem_map=string
[clutter_map=string] [region=string] frequency=value cellnum=value out_map=string
db_driver=string database=string out_table=string [dbperf=value] [procnum=value]
[model_ovr=value] [radius_ovr=value] [rx_threshold=value] [--overwrite] [--verbose]
[--quiet]
```

- Flags:

```
-r          Reuse results from existing intermediate model/sector files
-p          (purge) Delete all unused sector radio coverage files
-c          (check) Test run without actually performing radio coverage
              Computation
-1          Rx (dBm) values in output map replaced by 1.0 when above
              rx_threshold
--o         Allow output files to overwrite existing files
--v         Verbose module output
--q         Quiet module output
```

- Parameters:

```
csv_file     Radio cell/sector table in CSV format
             default: sector_table.csv
antmap_file  Antennas map file
             default: $GISBASE/etc/radio_coverage/antenna_diagrams/antennamap.csv
dem_map      DEM file for radio coverage simulation
             default: dem_map@PERMANENT
clutter_map  Clutter map file (required for hataDEM model)
             default: clutter_map@PERMANENT
region       Computation region (dem,current or region,rast,n,e,s,w,res - see
               g.region)
             default: current
frequency    Radio frequency (MHz)
             default: 900
cellnum      Number of succesive path loss values to be written in the table
             default: 5
out_map      Simulated radio coverage - raster (output)
             default: out_raster
db_driver    Database driver
             options: none,dbf,mysql,pg
             default: none
database     Database name
             default: $GISDBASE/$LOCATION_NAME/$MAPSET/dbf
out_table    Simulated radio coverage - db table (output)
             default: out_db
dbperf       Database insert performance (rows/INSERT; 99: special fast mode via
               CSV)
             options: 1-99
             default: 20
procnum      Number of parallel processes (-1: automatic, 0: non-parallel)
             default: -1
model_ovr    Model override (with parameters)
radius_ovr   Radius override (km)
rx_threshold Minimum received power (dBm) for radio signal coverage
```

- Example (does not create a data table):

```
r.radcov csv_file=cell_list_ijs.csv dem_map=dem_slovenia_25 out_map=ijs_abc
frequency=900 --o
```

2.1.1. Antenna types table

The list of available antenna types with corresponding parameters is given in a CSV format file. The *r.radcov* script reads the antenna type for each cell from the cell list table (CSV file, described above) and then uses the antenna types table to find the corresponding MSI file describing the antenna's characteristics. (The MSI format is described later in the *r.sector* chapter.)

The antenna types table file is specified with the *r.radcov*'s *antmap_file* command line parameter. The default path is *$GISBASE/etc/radio_coverage/antenna_diagrams/ antennamap.csv*, where the GISBASE environment variable is set by GRASS and contains

the path to its program directory. An example of the antenna types table is shown in Table 3 (only one antenna is listed).

Table 3: An example of the antenna types table

antennaType	Frequency	frequencyLower	frequencyUpper	EDT	MSIfilename	technology
COS-21	1500	800	2200	0	COS_21	none

The CSV file must be in standard format (the modified European MS Excel semicolon-separated-values format is not supported). The CSV file corresponding to Table 3 would be (generated by OpenOffice Spreadsheet):

```
"antennaType","frequency","frequencyLower","frequencyUpper","EDT","MSIfilename","technology"
"COS-21",1500,800,2200,0,"COS_21","none"
```

The first line contains the header; each following line contains the following data:

- *antennaType* - antenna type name, allowed character are letters, numbers, ' ', '-', '/' and '.' in any order,
- *frequency* - nominal frequency of the antenna in [MHz],
- *frequencyLower* - the lower frequency bound of the antenna in [MHz],
- *frequencyUpper* - the upper frequency bound of the antenna in [MHz],
- *EDT* - electrical tilt of the antenna in [°] (downwards), a non-negative integer value,
- *MSIfilename* - the name of the MSI file without the *.MSI* or *.msi* extension that describes the antenna characteristics for this particular combination of antenna type & frequency & electrical tilt,
- *technology* - used to describe the type of the radio communication technology the antennas is made for (e.g. GSM 900, GSM 1800, UMTS 2100); an arbitrary comment, not used for processing.

Empty lines are ignored. The character # as the first character of the antenna type or the first character in a line has a special meaning: it marks the line as a comment only.

An antenna type can support multiple frequency bands (nominal frequencies) and electrical tilts, with each combination having different characteristics (described by the corresponding MSI files). Hence, the same antenna type can appear in the table multiple times. The *r.radcov* searches the table for rows with the required antenna type, electrical tilt and with the frequency range (defined by the lower and upper frequency bound) that includes the simulation radio frequency set by the *frequency* command line parameter of *r.radcov*. If multiple table rows fulfill these requirements, *r.radcov* takes the one with the antenna nominal frequency closest to the simulation radio frequency.

The MSI files must be located in the same directory with the antenna types table file or in any of its optional subdirectories. Subdirectories can be used for logically grouping MSI files and have no other meaning. The *r.radcov* automatically searches the whole directory subtree for the MSI files. The MSI filenames must be unique even if located in different subdirectories.

2.1.2. Computation region management

In general, GRASS modules (including the RaPlaT modules) perform computations in the so called current region, which can be set with the GRASS command *g.region*. A region is a rectangle defined by its geographic borders and resolution. The *r.radcov* script has a

command line parameter called *region* that allows user to specify a computational region for the radio coverage computation, and temporarily sets it as the current region during the computation. If signals can be received in this region that emanate from outside transmitters, the computational region is automatically enlarged to include those transmitters.

The *region* parameter allows setting the computation region in a few different ways:

- *region=current* - the existing current region is used as the computation region (this is the default setting),
- *region=dem* - the region of the DEM map (its extents and resolution) is used as the computation region,
- *region=region:saved_region_name* (or *region=region=saved_region_name*) - a prevoiously saved region is used as the computation region (the GRASS *g.region* command can be used to save a current region),
- *region=rast:raster_map* (or *region=rast=raster_map*) - a GRASS raster map region is used as the computation region (*region=dem* can be regarded as a special shorthand form of *region=rast:…*),
- *region=n:_,e:_,s:_,w:_,res:_* (or *'region=n=_ e=_ s=_ w=_ res=_'*), with _ standing for numerical values, sets the computation region by explicitly defining its extents and resolution; instead of all five values, any subset of them can be set (with the rest of them retaining the existing values).

The last three ways of computation region definition (i.e. with exception of *current* and *dem*) can be combined - e.g. the computation region can be defined with a raster map and then the resolution modified. The mechanism is the same as in the case of the GRASS *g.region* command because this command is actually used for region setting (after replacing ‘:’ with ‘=’ and ‘,’ with ‘ ’), so see *g.region* help for more detailed information.

After the computation region is defined with the above procedure, and before it is actually set and used, *r.radcov* performs some refinements:

- unifies north-south and east-west resolution by taking the latter (east-west) for both directions,
- rounds the resolution to the integer value (in [m]),
- rounds the region extents so that the values of pixel center coordinates are multiples of the resolution value, and that this new region does exceed the extents of the original one.

2.1.3. Parallel execution support

GRASS modules are generally single-thread processes that execute on a single processor core. This is also true for RaPLaT modules. However, under certain limitations it is possible to execute multiple modules in parallel on a multi-core processor [11,12]. To speed-up coverage computation for multiple-antenna communication networks, *r.radcov* is capable of calling and executing modules in parallel. The number of concurrently executing modules (path loss model modules in the first step, and *r.sector* in the second step) is defined by the value of the *procnum* command line parameter. By default (or setting *procnum*=-1, automatic mode) the number of concurrently executing modules equals the number of existing processor cores in the system. A positive value explicitly defines the number of modules to be executed in parallel. Even if it is set to 1, the underlying parallel scheduling and stdout/stderr buffering mechanisms are still active. The parallel mode of execution (and related supporting mechanisms) can be switched off completely by setting *procum*=0.

2.1.4. Reuse and Purge

During the execution of *r.radcov*, intermediate GRASS maps are created by the path loss model modules and by *r.sector*. They are not deleted automatically and can optionally be reused for another similar coverage computation (these can be considered as a kind of caching of the intermediate results for future computations). By avoiding unnecessary and possibly lengthy re-computations of model path loss and *r.sector* intermediate results, the overall time required for a radio coverage computation can be reduced considerably.

The names of intermediate maps are generated automatically and contain important information that *r.radcov* needs to reuse them automatically in another radio coverage computation. The names of the maps generated by the path loss model modules are built according to the following pattern:

```
_model_P1_..._Pmax_positionEast_positionNorth_heightAGL_radius_frequency
```

e.g.:

```
_hata_urban_460697_99918_20_10_900
```

The names of the maps generated by *r.sector* are based on the above pattern (*model* in the pattern below) extended with additional *r.sector* related information:

```
cellName-antennaID(model)_beamDirection_electricalTiltAngle_mechanicalAntennaTilt_antennaType
```

e.g.:

```
IJS-A-1_hata_urban_460697_99918_20_10_900_30_0_0_COS-21
```

By default, *r.radcov* ignores any existing intermediate maps and calculates everything anew. The user can request reusing existing maps by specifying flag *-r* ("reuse"), however this must be done with great caution. The intermediate map names do not contain information about the DEM and clutter maps and the computation region that were used during their creation. Hence, the user must keep in mind that the cached maps may only be used if the maps and the computation region did not change.

The number of intermediate maps can become quite large, making a mess out of the user's mapset. When not needed any more, the user can request *r.radcov* to delete them by specifying flag *-p* (purge, all maps with the names following the above patterns will be deleted).

2.1.5. Database support

The *r.radcov* script supports writing the output data table, which is performed by *r.MaxPower*. The related command line parameters *cellnum*, *db_driver*, *database*, *out_table* and *dbperf* are equivalent to those of *r.MaxPower* and are described there. By default (if *db_driver* is not defined or is set to *none*) no data table is created. The *r.radcov* script does not check for the actually installed and GRASS-supported database management systems. Instead, it has a fixed list of them: *none*, *dbf* (GRASS' own built-in database), *mysql* (MySQL) and *pg* (PostgreSQL). Of course, MySQL and PostgresSQL can only be used if they are installed, and the GRASS' support for them is also installed.

2.1.6. Other parameters

There are some more *r.radcov* command line parameters and flags that will be described here briefly.

The *radius_ovr* and *model_ovr* parameters override per-cell settings in the previously described cell list table. While settings there are individual for each transmit antenna (cell), this two parameters set the radius and model with parameters globally, i.e. for all antennas. The *model_ovr* parameter expects a comma-delimited list (without spaces) consisting of the model name and its parameters as described in Table 1 (e.g. *model_ovr=hata,urban*).

The parameter *rx_treshold* is equivalent to that of the *r.MaxPower* module and is used directly by that module. When specified, the received signal is ignored in those raster points where its received strength (in [dBm]) falls below the threshold value.

The flag *-1* (number one) modifies the output coverage map. Normally, the map contains the received strengths of the strongest received signal in each raster point, in [dBm]. When this flag is specified (usually together with *rx_treshold*) the map contains only binary information about the signal coverage, i.e.1.0 or 0.0 if a point is covered or not covered, respectively, by a signal (above the threshold, if one is specified).

The flag *-c* ("check") causes *r.radcov* to not call and execute modules. Instead, it only prints all the commands (module calls) that would be executed, and the contents of the related input cell list file generated by *r.radcov* for *r.MaxPower*. (The file itself is a temporary file and is automatically deleted when *r.radcov* completes its execution.)

2.2. Radio propagation models (isotropic antenna)

RaPLaT containes a number of modules that calculate radio signal path loss according to various path loss models. The result is a raster GRASS map with each point having the value of the signal fading in [dBm] in that point relative to the transmitter (no particular antenna is assumed yet, the situation corresponds to the isotropic radiation diagram with 0 dB gain).

2.2.1. r.fspl

The *r.fspl* module calculates the radio signal loss according to the free space model (FSPL – *Free Space Path Loss*), according to the equation (1), [13].

$$FSPL = 32.4 + 20\log R[km] + 20\log f[MHz] \quad (1)$$

where:

$FSPL$: loss in dB

R : distance between the transmitter and the receiver

f : transmission frequency in MHz

The model takes into account LOS (*Line of Sight*, i.e. the visibility between the transmitter and the receiver), but in general this is a very simplified theoretical model that works fine in empty space but does not give accurate results in real terrestrial propagation environments where the signal loss deviates from the free space "squared distance" law (it generally increases with distance with a higher exponent than 2).

An example of a path loss map obtained with *r.fspl* is shown in Fig. 3. (The transmitter is placed at the IJS location in Ljubljana, computation is limited to 10 km around the transmitter, the actual command used is listed in the Example below. The same holds for the other models described in the next chapters.)

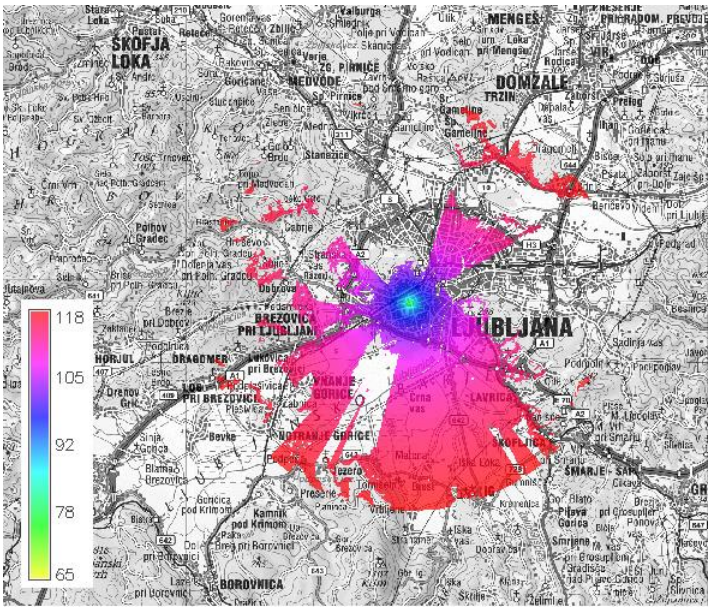


Fig. 3: Path loss at 2 GHz computed with *r.fspl*

- Usage:

```
r.fspl [-q] inputDEM=name output=name coordinate=x,y [ant_height=value]
frequency=value [radius=value] [--overwrite] [--verbose] [--quiet]
```

- Flags:

```
 -q          Quiet
--o          Allow output files to overwrite existing files
--v          Verbose module output
--q          Quiet module output
```

- Parameters:

```
inputDEM     Name of input raster map
output       Name for output raster map
coordinate   Base station coordinates
ant_height   Height of the antenna (m)
             default: 10
frequency    Frequency (MHz)
radius       Radius of calculation (km)
             default: 10
```

- Example:

```
r.fspl inputDEM=dem_slovenia_25@PERMANENT output=fspl_ijs_25
coordinate=460697,99918 ant_height=20 frequency=2000 radius=10 --o
```

2.2.2. r.hata

The *r.hata* module implements the Okumura-Hata radio propagation empirical model [14]. It is one of the most widely used models for radio coverage estimation and is based on the empirically estimated rules (measured propagation data). It is valid for:

- carrier frequency: 150 - 1500 MHz,
- distance between transmitter and receiver: 1 - 20 km,
- effective BS (transmitter) antenna height: 30 - 200 m,
- effective MS (receiver) antenna height: 1 - 10 m.

It contains three sub-models, for urban, suburban and open geographic areas, as defined by the following equations:

$$L_U = 69.55 + 26.16 \log f[MHz] - 13.82 \log h[m] - C_H + (44.9 - 6.55 \log h[m]) \log R[km] \quad (2)$$

$$L_{SU} = L_U - 2\left(\log \frac{f[MHz]}{28}\right)^2 - 5.4 \quad (3)$$

$$L_O = L_U - 4.78(\log f[MHz])^2 + 18.33 \log f[MHz] - 40.94 \quad (4)$$

$$C_H = 0.8 + (1.1 \log f[MHz] - 0.7) h_M[m] - 1.56 \log f[MHz] \quad (5)$$

where:

L_U, L_{SU}, L_O : loss in dB for urban, suburban and open environments, respectively

h : difference between the transmitter and receiver antenna heights

h_M : receiver antenna height above the ground

C_H : correction factor related to the receiver antenna height

R : distance between the transmitter and the receiver

f : transmission frequency in MHz

The rate of signal loss with the distance depends on the antenna height. For a very high antenna, it approximates the loss in empty space (the "squared distance" law, 20 dB/decade). The model ignores terrain configuration (relief, LOS), which is its main drawback, and the loss due to land use (clutter map). The model can give useful results if there are no major obstacles between the receiver and the transmitter.

An example of a path loss map obtained with *r.hata* is shown in Fig. 4.

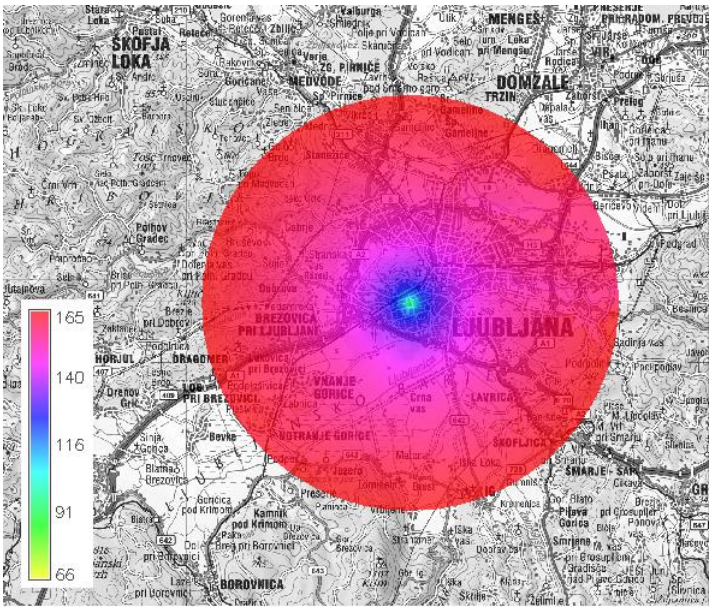


Fig. 4: Path loss at 900 MHz computed with *r.hata*

- Usage:

```
r.hata [-q] inputDEM=name output=name coordinate=x,y [ant_height=value]
[radius=value] [area_type=string] frequency=value [--overwrite] [--verbose] [--
quiet]
```

- Flags:

```
 -q          Quiet
--o          Allow output files to overwrite existing files
--v          Verbose module output
--q          Quiet module output
```

- Parameters:

```
inputDEM     Name of input raster map
output       Name for output raster map
coordinate   Base station coordinates
ant_height   Height of the antenna (m)
             default: 10
radius       Radius of calculation (km)
             default: 10
area_type    Type of area
             options: urban,suburban,open
             default: urban
frequency    Frequency (MHz)
```

- Example:

```
r.hata inputDEM=dem_slovenia_25@PERMANENT output=hata_ijs_25
coordinate=460697,99918 ant_height=20 frequency=900 radius=10 --o
```

2.2.3. r.cost231

The *r.cost231* module implements the COST231 empirical model, which extends the Okumura-Hata model to the 1500-2000 MHz band [15]. It is valid for:

- carrier frequency: 1500 - 2000 MHz,
- distance between transmitter and receiver: 1 - 20 km,
- effective BS (transmitter) antenna height: 30 - 200 m,
- effective MS (receiver) antenna height: 1 - 10 m.

The model is based on the Hata model for the suburban areas:

$$L[dB] = 46.33 + 33.9 \log f[MHz] - 13.82 \log h[m] - a(h_r) + \\ + (44.9 - 6.55 \log h[m]) \log d[km] + C \tag{6}$$

where:

C : =0 for medium-sized cities and suburban areas, =3 for large cities' centers

h : difference between the transmitter and receiver antenna heights

h_r : receiver antenna height above the ground

d : horizontal distance between the transmitter and the receiver

f : transmission frequency in MHz

The height correction factor $a(h_r)$ is given by:

$$a(h_r) = (1.1 \log f[MHz] - 0.7) h_r[m] - (1.56 \log f[MHz] - 0.8) \tag{7}$$

The model is adjusted for higher transmission frequencies. It is mostly suitable for medium-sized and large cities and assumes the transmit (base station) antenna to be positioned above the surrounding buildings. The model only partially takes into account the terrain configuration (the effective height h in the equation (6)) and ignores the signal loss behind large obstacles.

An example of a path loss map obtained with *r.cost231* is shown in Fig. 5.

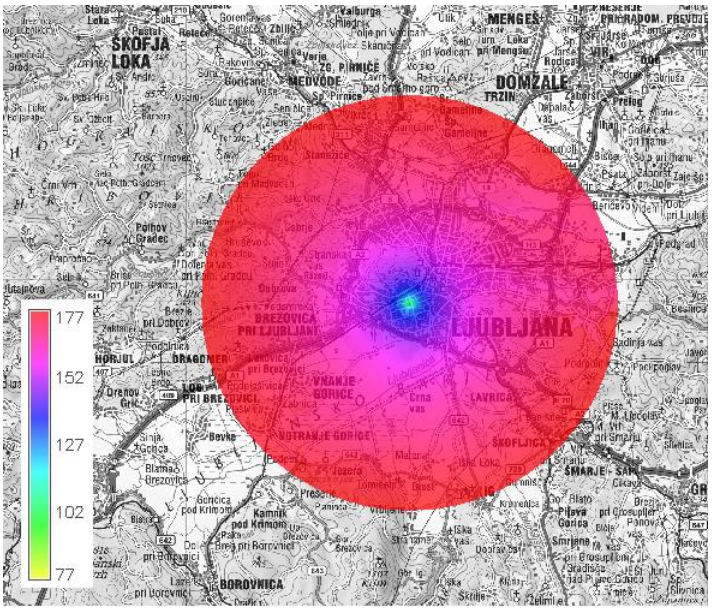


Fig. 5: Path loss at 2 GHz computed with *r.cost231*

- Usage:

```
r.cost231 [-q] inputDEM=name output=name coordinate=x,y [ant_height=value]
[radius=value] [area_type=string] frequency=value [--overwrite] [--verbose] [--
quiet]
```

- Flags:

```
 -q          Quiet
--o          Allow output files to overwrite existing files
--v          Verbose module output
--q          Quiet module output
```

- Parameters:

```
inputDEM     Name of input raster map
output       Name for output raster map
coordinate   Base station coordinates
ant_height   Height of the antenna (m)
             default: 10
radius       Radius of calculation (km)
             default: 10
area_type    Type of area
             options: medium_cities,metropolitan
             default: medium_cities
frequency    Frequency (MHz
```

- Example:

```
r.cost231 inputDEM=dem_slovenia_25@PERMANENT output=cost231_ijs_25
coordinate=460697,99918 ant_height=20 frequency=2000 radius=10 --o
```

2.2.4. r.hataDEM

The *r.hataDEM* module implements a modified/extended Okumura-Hata model. The radio signal loss depends on the transmission radio frequency, the distance between the transmitter and the receiver, the height of the transmit and receive antennas, and also on the terrain profile, land use and earth surface curvature. The model is valid for:

- carrier frequency: 10 MHz - 2 GHz,

- distance between transmitter and receiver: 200 m - 100 km,
- effective BS (transmitter) antenna height: 20 - 200 m,
- effective MS (receiver) antenna height: 1 - 5 m.

The basic concept of the model is shown in Fig. 6.

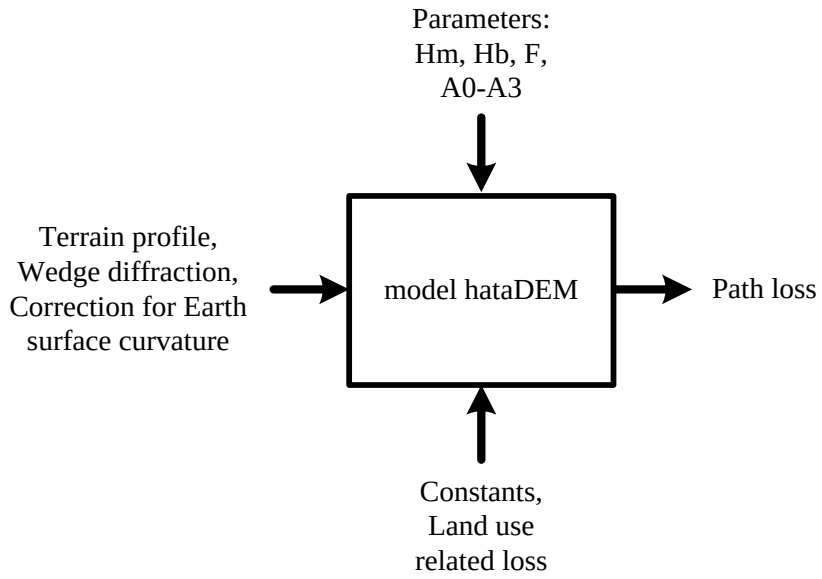


Fig. 6: Basic concept of the hataDEM model

The general path loss equation of the model is:

$$L[dB] = HOA[dB] + mk + \sqrt{(\alpha KDFR)^2 + (JDFR)^2} \tag{8}$$

where:

HOA : Okumura-Hata equation for "open" areas
mk : land-use related signal loss at the receiver location in [dB]
KDFR : contribution of wedge diffraction in [dB]
α : parameter related to KDFR
JDFR : diffraction loss due to the Earth surface curvature in [dB]

The Okumura-Hata path loss as defined by this model is:

$$\begin{aligned} HOA[dB] = {} & A0 + A1 \cdot \log d[km] + A2 \cdot \log Heff[m] + A3 \cdot \log d[km] \cdot \log Heff[m] - \\ & - 3.2[\log(11.75 \cdot Hm[m])]^2 + 44.49 \cdot \log f[MHz] - 4.78 \cdot (\log f[MHz])^2 \end{aligned} \tag{9}$$

where:

A0-A3 : model tuning parameter
Heff : difference between the transmitter and receiver antenna heights
Hm : receiver antenna height above the ground
d : horizontal distance between the transmitter and the receiver
f : transmission frequency in MHz

We implemented the single-wedge loss as:

$$L_{ke}[dB] = -20 \cdot \log \frac{1}{\pi \cdot \nu \cdot \sqrt{2}} \tag{10}$$

$$\nu = h \cdot \sqrt{\frac{2(d_1 + d_2)}{\lambda \cdot d_1 \cdot d_2}} \tag{11}$$

where:

h : height of the wedge above the direct line between transmitter and receiver

d_1, d_2 : the distances of the mobile and base stations from the wedge

Since *r.hataDEM* was originally intended for calculation on smaller geographic areas (cellular networks with transmitter-receiver distances of up to 35 km), it ignores the effect of the Earth surface curvature. Additionally, we fixed the value of parameter α to α=1.

An example of a path loss map obtained with *r.hataDEM* is shown in Fig. 7.

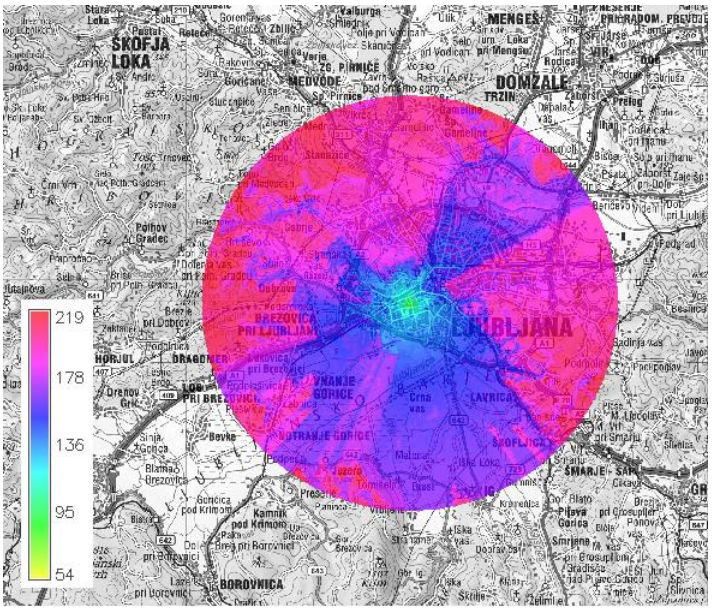


Fig. 7: Path loss at 2 GHz computed with *r.hataDEM*

- Usage:

```
r.hataDEM [-q] inputDEM=name clutter=name output=name A0=value A1=value A2=value
A3=value coordinate=x,y [ant_height=value] [radius=value] frequency=value [--
overwrite] [--verbose] [--quiet]
```

- Flgas:

```
 -q          Quiet
--o          Allow output files to overwrite existing files
--v          Verbose module output
--q          Quiet module output
```

- Parameters:

```
inputDEM     Name of input raster map
clutter      Clutter raster map with path loss coefficients
output       Name for output raster map
A0           Parameter A0
A1           Parameter A1
A2           Parameter A2
```

```
A3            Parameter A3
coordinate    Base station coordinates
ant_height    Height of the antenna (m)
              default: 10
radius        Radius of calculation (km)
              default: 10
frequency     Frequency (MHz)
```

- Example:

```
r.hataDEM inputDEM=dem_slovenia_25@PERMANENT
clutter=clut_slovenia_25_loss@PERMANENT output=hataDEM_ijs_25
coordinate=460697,99918 ant_height=20 frequency=2000 radius=10 A0=42 A1=42 A2=-12
A3=0.1 --o
```

2.2.5. r.waik

The *r.waik* module implements the Walfisch-Ikegami semi-deterministic model for path loss computation in microcells. It was developed in the framework of the COST231 project [15] and is based on the Walfisch-Bertoni [16] and Ikegami [17] models. The model computes path loss in two different ways, based on LOS (*Line of Sight*). It is valid for (the receiver and transmitter antenna heights constraints are different for LOS and NLOS cases and are described later):

- carrier frequency: 800 - 2000 MHz,
- distance between transmitter and receiver: 20 m - 5 km,
- receiver and transmitter height constraints are different for LOS and NLOS cases - see below.

In the LOS case (transmitter-receiver visibility), the loss within the street canyon is defined as:

$$L[dB] = 42.64 + 26 \log d[km] + 20 \log f[MHz], \quad d[km] \geq 0{,}02 \tag{12}$$

The first constant corresponds to the empty space loss at the distance of 20 m. The transmitter antenna height must be at least 30 m, and there should be no obstacles in the first Fresnel zone. The signal loss is exponential with the distance, the exponent value is 2,6.

In the NLOS case (no direct visibility between the transmitter and the receiver), the model uses the following parameters:

- transmitter height: h_t (4 m to 50 m),
- receiver height: h_r (1 m to 3 m),
- buildings height: h_{roof} (3 m × number of floors plus 3 m for gabled roofs and 0 m for flat roofs),
- the transmitter antenna height above the roof height: $\Delta h_t = h_t - W8$,
- the receiver antenna height below the roof height: $\Delta h_r = W8 - h_r$,
- spacing between buildings: b (if no data is available, the recommended value is between 20 m and 50 m),
- street width: w (if no data is available, the recommended value is $b/2$),
- incident angle of radio rays: ϕ (if no data is available, the recommended value is 90°).

The path loss is:

$$L[\mathrm{dB}] = \begin{cases} L_0 + L_{rts} + L_{msd}, & L_{rts} + L_{msd} \geq 0 \\ L_0, & L_{rts} + L_{msd} < 0 \end{cases} \tag{13}$$

It consists of three components:

- the free space loss L_0,
- the rooftop-to-street diffraction loss L_{rts},
- the multiple screen diffraction loss L_{msd}.

The free space loss is:

$$L_0 = W0 + 20\log(d) + 20\log(f) \tag{14}$$

The rooftop-to-street diffraction loss is:

$$L_{rts} = -8.2 - W3\log(W6) + W4\log(f) + W5\log(\Delta h_r) + L_{11} \tag{15}$$

where the orientation-related loss is:

$$L_{11} = \begin{cases} -10 + 0.354\,\phi, & 0 \leq \phi < 35° \\ 2.5 + 0.075\,(\phi - 35°), & 35° \leq \phi 55° \\ 4.5 - 0.11\,(\phi - 55°), & 55° \leq \phi \leq 90° \end{cases} \tag{16}$$

The multiple screen diffraction loss is:

$$L_{msd} = L_{21} + k_a + k_d\log(d) + k_f\log(f) - 9\log(W7) \tag{17}$$

where the shadowing gain is:

$$L_{21} = \begin{cases} -18\log(1 + \Delta h_t) & h_t > h_{roof} \\ 0 & h_t \leq h_{roof} \end{cases} \tag{18}$$

The k_a and k_d parameters depend on the path length d and the transmitter height above the roofs:

$$k_a = \begin{cases} W1, & h_t \geq h_{roof} \\ W1 - 0.8\,(h_t - h_{roof}), & h_t < h_{roof} \wedge d \geq 0.5\ m \\ W1 - 0.4\,d(h_t - h_{roof}), & h_t < h_{roof} \wedge d < 0.5\ m \end{cases} \tag{19}$$

$$k_d = \begin{cases} W2, & h_t \geq h_{roof} \\ W2 - \dfrac{15\,(h_t - h_{roof})}{h_{roof}}, & h_t < h_{roof} \end{cases} \tag{20}$$

Parameter k_a represents the increase of path loss when the transmitter is located below the roof levels, while parameters k_d and k_f represent the path loss due to distance and frequency. The latter is defined by

$$k_f = -4 + k_{f1}\left(\frac{f}{925} - 1\right) \tag{21}$$

The value of k_{f1} is 1,5 for city centers, and 0,7 elsewhere.

Parameters W0-W8 should be set according to the recommended values given in Table 4.

Table 4: Parameters and their values for the Walfisch-Ikegami model

Parameter	Description	Value range	Default value
W0	Free space loss correction	20 - 60	32.5
W1	Reduced base antenna height correction	30 - 70	54
W2	Range correction	5 - 35	10
W3	Street width correction	3 - 15	10
W4	Frequency correction	3 - 25	10
W5	Building height correction	10 - 30	20
W6	Width of roads [m]	(rec. W7 / 2)	15
W7	Building separation [m]	(rec. 20 - 50)	30
W8	Height of buildings [m]	-	12

The COST231-Walfish-Ikegami provides good path loss estimates if the transmission antenna is located above the roof level. If it is located near the ground level, the estimates are bad because the model does not take into account the waveguide effect of the large city street canyons.

An example of a path loss map obtained with *r.waik* is shown in Fig. 8.

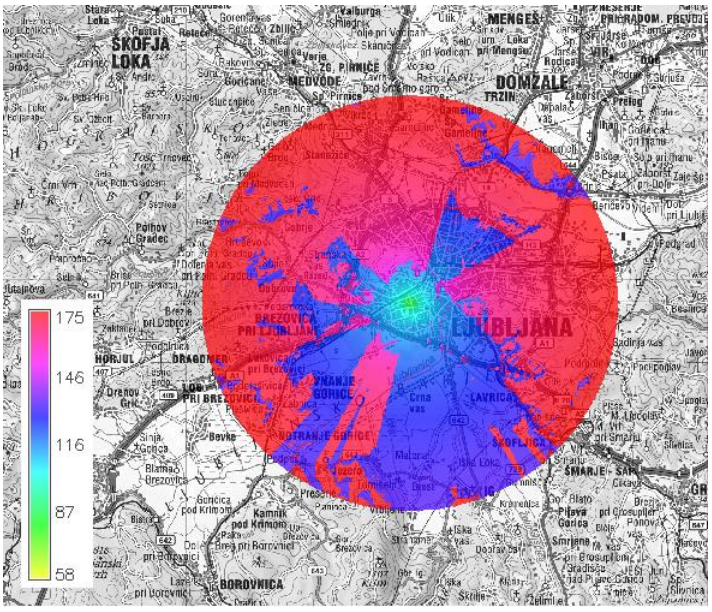


Fig. 8: Path loss at 2 GHz computed with *r.waik*

- Usage:

```
r.waik [-q] [--overwrite] [--verbose] [--quite] inputDEM=name output=name
coordinate=x,y [ant_height=value] frequency= value [radius= value]
[free_space_loss_correction= value] [bs_correction= value] [range_correction=value]
[street_width_correction=value] [frequency_correction=value]
[building_height_correction=value] [street_width=value]
[distance_between_buildings=value] [building_height=value] [PHI_Street=value]
[area_type=string]
```

- Flags:

```
 -q          Quiet
--o          Allow output files to overwrite existing files
--v          Verbose module output
--q          Quiet module output
```

- Parameters:

```
inputDEM                    Name of input raster map
output                      Name for output raster map
coordinate                  Base station coordinates
ant_height                  Height of the antennas (m)
                            default: 10
frequency                   Frequency (MHz)
radius                      Radius of calculation (km)
                            default: 10
free_space_loss_correction  Free space loss correction
                            default: 32.5
bs_correction               Reduced base antenna height correction
                            Default: 54
range_correction            Range correction
                            default: 10
street_width_correction     Street width correction
                            default: 10
frequency_correction        Frequency correction
                            default: 10
building_height_correction  Building height correction
                            default: 20
street_width                Widths of roads (m)
                            default: 15
distance_between_buildings  Building separation
                            default: 30
building_height             Heights of buildings (m)
                            default: 12
PHI_Street                  Street orientation angle (deg)
                            default: 90
area_type                   Type of area
                            options: metropolitan,medium_cities
                            default: medium_cities
```

- Example:

```
r.waik inputDEM=dem_slovenia_25@PERMANENT output=waik_ijs_25
coordinate=460697,99918 ant_height=20 frequency=2000 radius=10 --o
```

2.3. Add transmission antenna - r.sector

The path loss model modules described so far compute path loss for the case of isotropic transmission (a hypothetical isotropic antenna with 0 dB gain), without considering actual transmission antenna characteristics, position and orientation. The next step in to apply the antenna radiation pattern, which is the task of the *r.sector* module. The antenna radiation pattern and other data must be given in the MSI Planet Antenna File Format [18]. This is a text format with the following structure:

```
NAME <name>
MAKE <make>
FREQUENCY <frequency>
H_WIDTH <h_width>
V_WIDTH <v_width>
FRONT_TO_BACK <front_to_back>
GAIN <gain>
TILT <tilt>
POLARIZATION <polarisation>
COMMENT <comment>
```

```
HORIZONTAL 360
0 <0H>
.
.
359 <359H>
VERTICAL 360
0 <0V>
.
.
359 <359V>
```

The variables are:

```
NAME            Name of the antenna
MAKE            Name of the manufacturer
FREQUENCY       Frequency in MHz
H_WIDTH         Opening angle in the horizontal plane between the -3 dB points
V_WIDTH         Opening angle in the vertical plane between the -3 dB points
FRONT_TO_BACK   Front to back ratio in dB
GAIN            Antenna gain in dBd when in dBi this must be specified
TILT            Electrical tilt of the main beam in degrees
POLARIZATION    Horizontal, vertical, +45 or -45
COMMENT         Comment
0H..359H        Horizontal gain data points per horizontal angle relative to
                maximum gain being zero. Any value below zero is assumed to be
                negative. Minus sign is not used with these values
0V..359V        Horizontal gain data points per horizontal angle relative to
                maximum gain being zero. Any value below zero is assumed to be
                negative. Minus sign is not used with these values
```

In practice, MSI files usually use only a subset of the parameters listed above, e.g. NAME, FREQUENCY, GAIN in dBd (default) or dBi, TILT, COMMENT, and of course the HORIZONAL 360 and VERTICAL 360 sections. Besides, the TILT parameters does not necessarily specify the actual electrical tilt value to which the radiation pattern corresponds, but is often assigned no value or the keyword ‘ELECTRICAL’ (for antennas not having or having electrical tilt option, respectively).

An actual MSI file could look like this (this particular file does not describe a real physical antenna but a mathematically generated one with the cosN radiation pattern):

```
NAME COSN21
FREQUENCY 2140
GAIN 19 dBd
TILT ELECTRICAL
COMMENT simple cos^4 antenna diagram
HORIZONTAL 360
0 0.0000
1 0.0139
2 0.0556
.
.
357 0.1251
358 0.0556
359 0.0139
VERTICAL 360
0 0.0000
1 0.0139
2 0.0556
.
.
357 0.1251
358 0.0556
359 0.0139
```

The *r.sector* module only reads and uses the GAIN parameter value expressed in dBd or dBi (i.e. relative to a dipole or isotropic antenna, respectively; dBd is default, x [dBd] = (x + 2,15) [dBi]), and the pattern definition specified in the HORIZONTAL and VERTICAL sections. It calculates the 3-D radiation pattern based on the antenna's given horizontal and vertical patterns, its gain, and its physical position and direction. It then generates an output path loss raster map by applying this pattern to the isotropic path loss raster map previously computed by a path loss model module.

Fig. 9 shows an example of a path loss map calculated by *r.sector*, using the path loss map previously computed by *r.hata* (shown in Fig. 4) and the just mentioned artificial cosN-type antenna. The antenna is directed 30° eastwards (north is the reference, positive values correspond to the clockwise rotation).

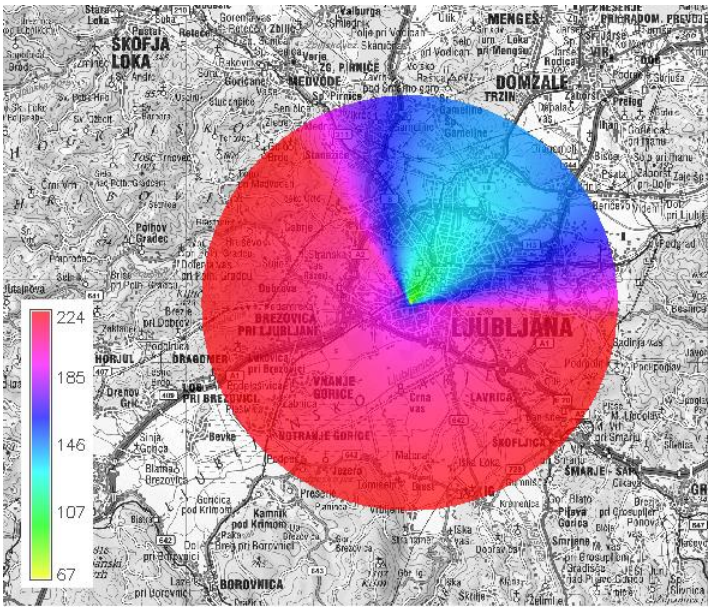


Fig. 9: Path loss computed by *r.sector*, based on *r.hata* path loss (Fig. 4)

- Usage:

```
r.sector [-q] pathloss_raster=name inputDEM=name output=name ant_data_file=string
beam_direction=value mech_tilt=value height_agl=value east=value north=value
[radius=value] [--overwrite] [--verbose] [--quiet]
```

- Flags:

```
 -q          Quiet
--o          Allow output files to overwrite existing files
--v          Verbose module output
--q          Quiet module output
```

- Parameters:

```
pathloss_raster    Omni antenna path loss raster
inputDEM           Elevation model - required for transmitter height determination
output             Name for output raster map
ant_data_file      Antenna data file
beam_direction     Beam direction (deg)
mech_tilt          Mechanical antenna tilt (deg)
height_agl         Above ground level height (m)
east               Easting coordinate
north              Northing coordinate
radius             Radius of calculation (km)
                   default: 10
```

- Example:

```
r.sector pathloss_raster=hata_ijs_25 inputDEM=dem_slovenia_25
output=sector_hata_ijs_25 ant_data_file=/usr/local/src/grass-6.4.3RC2/dist.i686-pc-
linux-gnu/etc/radio_coverage/antenna_diagrams/_demo_/COS_21.MSI beam_direction=30
mech_tilt=0 height_agl=20 radius=10 east=460697 north=99918 --o
```

2.4. Calculate complete coverage - r.MaxPower

The *r.MaxPower* module calculates the received radio signal strength(s) from one or more transmitters (transmit antennas). It does this by taking the path loss raster map(s) produced by *r.sector* and applying the corresponding transmission power(s). It obtains the list of all input path loss raster maps with corresponding transmission powers (for all the transmission antennas, here also called *cells*) in a CSV format text file. The module produces a raster map file containing the received strength of the strongest received signal for each raster point. If a *rx_treshold* parameter value is specified, the signals with lower received strengths are ignored. If flag -1 (number one) is additionally specified, only a simple coverage area map is generated (value 1 for the received signal above the threshold, 0 otherwise) instead of the received signal strength map.

In addition to the raster map, *r.MaxPower* can generate a data table (using standard databases supported by GRASS, like MySQL or PostgreSQL and also the GRASS' own built-in DBF) containing data about a certain number (user selectable, parameter *cell_num*) of the strongest received signals in each raster point, suitable for further processing by other non-GRASS tools. The data table generation is activated by specifying the *driver* parameter with a value other than *none* (*dbf* for GRASS' DBF, *mysql* for MySQL, *pg* for PostgreSQL).

- Usage:

```
r.MaxPower [-q1] cell_input=string [rx_threshold=value] output=name table=string
driver=string database=string cell_num=value [dbperf=value] [--overwrite] [--
verbose] [--quiet]
```

- Flags:

```
 -q          Quiet
 -1          Rx (dBm) values in output map replaced by 1.0 when above
               rx_threshold
--o          Allow output files to overwrite existing files
--v          Verbose module output
--q          Quiet module output
```

- Parameters:

```
cell_input      Cells data table
rx_threshold    Minimum received power [dBm] for radio signal coverage
                default: -999
output          Name for output raster map
table           Table name
driver          Driver name
                options: mysql,ogr,pg,dbf,sqlite,none
                default: none
database        Database name
                default: $GISDBASE/$LOCATION_NAME/$MAPSET/dbf/
cell_num        Number of successive path loss values to be written in the table
dbperf          Database insert performance(rows/INSERT; 99: special fast mode
                  via CSV)
                options: 1-99
                default: 20
```

- Examples (the first one does not create a data table, but dummy values for the *table* and *cell_num* parameters must be specified anyway; the second one creates a DBF data table named *ijs_abc* in the default GRASS *dbf* folder within the user's mapset):

```
r.MaxPower cell_input=cell_list output=ijs_abc table=ijs_abc cell_num=5 --o

r.MaxPower cell_input=cell_list output=ijs_abc table=ijs_abc driver=dbf dbperf=1
cell_num=5 --o
```

2.4.1. The input cell list file

The cell list file ("Cells data table") is specified by the *cell_input* parameter. It is a text file in a CSV-like (actually "Semicolon-Separated Values") format, where each line contains data for a transmission antenna (here also called *cell*):

<cell_name>;<antenna_index>;<sector-raster-map_name>;<transmit-power>;<model-with-parameters>

There can be a single cell, or a number of them (e.g. in the case of a cellular radio network). No header line is used, and no empty lines are allowed (including at the end of file).

The only important columns are <*sector-raster-map_name*> and <*transmit-power*>. Other columns are only informal and could be empty, they are written to the data table along with the calculated received powers. Their purpose is:

- <*cell_name*> : an (arbitrary) cell name that helps the user identify the cell identity/location,
- <*antenna_index*> : antenna index for uniquely identifying each antenna (cell) in the system; there can be more than one antenna in a cell (e.g. two antennas might be connected in parallel to a single transmission signal via a power splitter to obtain a required transmission pattern),
- <*model-with-parameters*> : contains information about the radio propagation model used, and its parameters (independently for each antenna).

Following is an example of a cell list file (no empty lines at the beginning/end):

```
IJS-A;1;IJS-A-1_hata_urban_460697_99918_20_10_900_30_0_0_COS-21;30;hata;urban
IJS-B;2;IJS-B-2_hata_urban_460697_99918_20_10_900_135_0_0_COS-21;30;hata;urban
IJS-C;3;IJS-C-3_hata_urban_460697_99918_20_10_900_270_0_0_COS-21;30;hata;urban
```

2.4.2. The output data table

The optionally generated data table contains one row of data for each raster point of the output coverage raster map. Raster points with no coverage (no received radio signal from any transmitter) are not included in the map, nevertheless the table can be quite large and needs considerable time for creation. The row format of the table is shown in Table 5.

Table 5: Output data table format

Column name	x	y	Resolution	cell1	id1	Pr1	model1	…	cellN	idN	PrN	modelN	E_c/N_0
format	int 6	int 6	int 4	varchar 32	int 6	real 6	varchar 128	…	varchar 32	int 6	real 6	varchar 128	real 6

Description of columns:

1. **x**: x coordinate of geographic location (a map raster point), in [m] (*integer* - 4 bytes, default print length is 6).
2. **y**: y coordinate of geographic location (a map raster point), in [m] (*integer* - 4 bytes, default print length is 6).
3. **resolution**: resolution of the computed coverage map, in [m] (*integer* - 4 bytes, default print length is 4).
4. **cell*i***: cell name (*varchar*, max 32 chars (bytes)).
5. **id*i***: antenna index (*integer* - 4 bytes, default print length is 6).
6. **Pr*i***: calculated received signal power in [dBm], received in this geographic location (output map raster point) from *cell*i (*real* - 4 bytes floating point, default print length is 6). Valid values are > -999.0. The value of -999.0 has a special meaning - no signal received (replacing -∞).
7. **model*i***: Path loss model name with parameters, used for this *cell*i (*varchar*, max 128 chars (bytes)).
8. $\mathbf{E_c/N_0}$: the received power od the strongest signal divided by the sum of the received powers of all signals, in [dB] (*real* - 4 bytes floating point, default print length is 6). Valid values are > -999.0. The value of -999.0 has a special meaning - no signal received. (This data is useful for single frequency band systems such as those using CDMA, e.g. UMTS.)

Columns 4, 5, 6 and 7 repeat *N*-times, as defined by the command line parameter *cell_num* but not exceeding the number of all cells specified in the input cell list file.

The command line parameters related to the output data table are:

- *driver* : defines the database management system used (called also simply the database). The default value *none* means that no database is used and no data table is generated (other parameters are ignored, but dummy values for the *table* and *cell_num* parameters must be specified anyway). Contrary to *r.radcov* (which does not check for actually installed and supported databases but have a fixed list of them), *r.MaxPower* uses GRASS's information about available databases (and lists them when called with flag *-help*). Value *dbf* selects GRASS' own built-in database, which GRASS uses for its own purposes. It is a relatively simple database with limited functionality and efficiency and is not really recommended for use with RaPlaT except for simple cases. Before writing to the disk it creates the whole data table in the main memory (which quickly runs out for large tables) and does not support fast writing modes (hence it can be very slow for large tables, see also the *dbperf* parameter). GRASS includes support for a number of external databases. When building GRASS from source, the GRASS support must be activated for each database to be used, and that database must already be installed. The two recommendable high-performance databases that are specifically supported and tested with *r.MaxPower* are MySQL (parameter value *msyql*) and PostgreSQL (parameter value *pg*).
- *database* : defines the name of existing database, where the output data table will be created. The default value represents the location of the GRASS' built-in DBF database (which is always available, but is not recommendable for larger radio coverage projects; the three environment variables represent GRASS working environment - the basic GRASS database folder, the user-selected GRASS *location* and the user's GRASS *mapset*. The output data table is stored as a dbf-format file named *$GISDBASE/$LOCATION_NAME/$MAPSET/dbf/<table_name>.dbf*. In case

of MySQL or PostgreSQL, the (empty) database must first be created with their own tools, with an arbitrary name (e.g., a reasonable database name could be *grass*).

- *table* : defines the output table name for a particular radio coverage computation.
- *cell_num* : defines the number of the strongest received signals in each point of the output coverage raster that are stored in the data table (see Table 5).
- *dbperf* : stands for "database performance" and selects faster modes of writing tables. For *dbperf* values between 2 and 98, the so called multiple-row inserts are used, where instead of a single data row, a group of data rows (2 to 98) is inserted with each SQL INSERT statement. For MySQL and PostgreSQL, a reasonable value is 20, which is the default. In our case it speeds up the data table creation around 2,5x for PostgreSQL and 3,8x for MySQL (relative to the basic single-row insert mode). For the GRASS' built-in DBF database, *dbperf* value should be set to 1, (the basic non-accelerated mode, because GRASS DBF does not support multiple-row inserts). The value of 99 selects a special very fast mode which is supported only for MySQL and PostgreSQL. It doesn't write data rows directly into the output data table but instead creates an intermediate CSV text file. This is at the end converted to the data table in a single step, using non-standard database-specific commands (for MySQL: *LOAD DATA LOCAL INFILE 'file.csv' INTO TABLE <table> FIELDS TERMINATED BY ',' ENCLOSED BY '"';* for PostgreSQL: *COPY <table> FROM 'file.csv' CSV QUOTE '"';*). In our case, by using this mode, speeds-up of data table creation of around 20x were achieved.

2.5. Prepare clutter map - r.clutconvert

In addition to DEM, some radio signal propagation models (*hataDEM* in our case) need also information about the signal loss at the receiver location related to the land use (e.g. urban, agricultural, forest areas, etc.) This information is given in the form of a raster map called *clutter* map, where the value of each raster point specifies the received signal loss in [dB] in that point due the land use. Land use is often given as a raster map with values specifying the use (e.g. 1 - irrigated agriculture, 2 - rangeland, 3 - coniferous forest, 4 - deciduous forest, 5 - mixed forest, 6 - disturbed). The *r.clutconvert* module converts such land use map to the corresponding clutter map. The conversion from land use types (integer numbers) to signal loss values in [dB] is specified by a table defined in a text file. The numbers representing land use depend on the particular land use map, and the radio signal loss values depend on the radio frequency. Hence, even if there is only one land use map, there will probably be a number of these conversion table files, each for a particular frequency band. The one to be used by *r.clutconvert* is specified with its *Path_loss_values* command line parameter).

The conversion table file consists of a number of lines. Lines can be either comment or data lines. A comment line starts with '#' and is ignored. A data line specifies conversion from a land use value (an integer number) to the corresponding radio signal loss in [dB] (generally a floating point value), with ':' as separator. The following is an example of this file (here, the first line is split into three lines to fit on the page, but is actually a single line):

```
#Terrene type loss factors - hataDEM model (1 - irrigated agriculture, 2 -
rangeland, 3 - coniferous forest, 4 - deciduous forest, 5 - mixed forest, 6 -
disturbed)
1:15.5
2:15
3:25
4:20
```

```
5:22
6:21
```

Fig. 10 shows the official (commercial) Slovenian land use map for the Ljubljana region (it includes 12 categories). The corresponding clutter map generated by *r.clutconvert* is shown in Fig. 11.

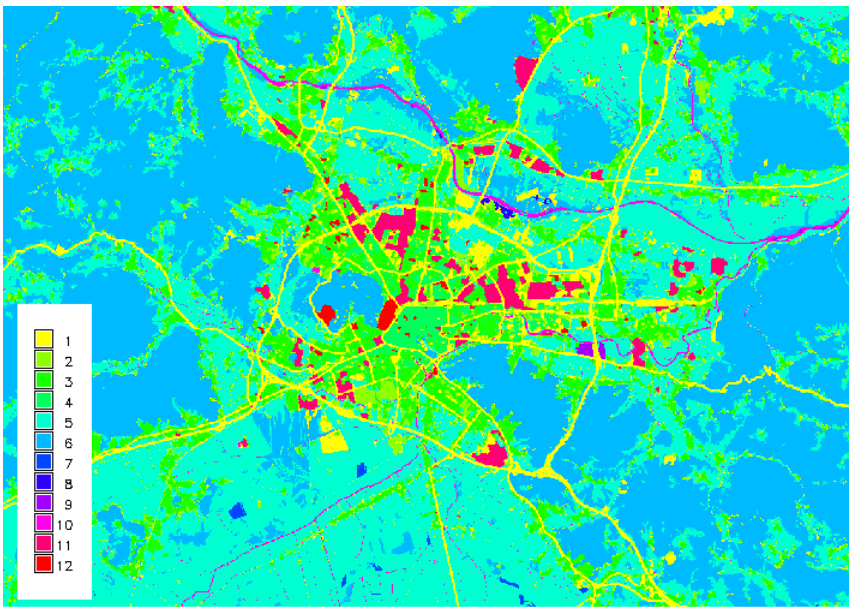


Fig. 10: Official land use map for the Ljubljana region

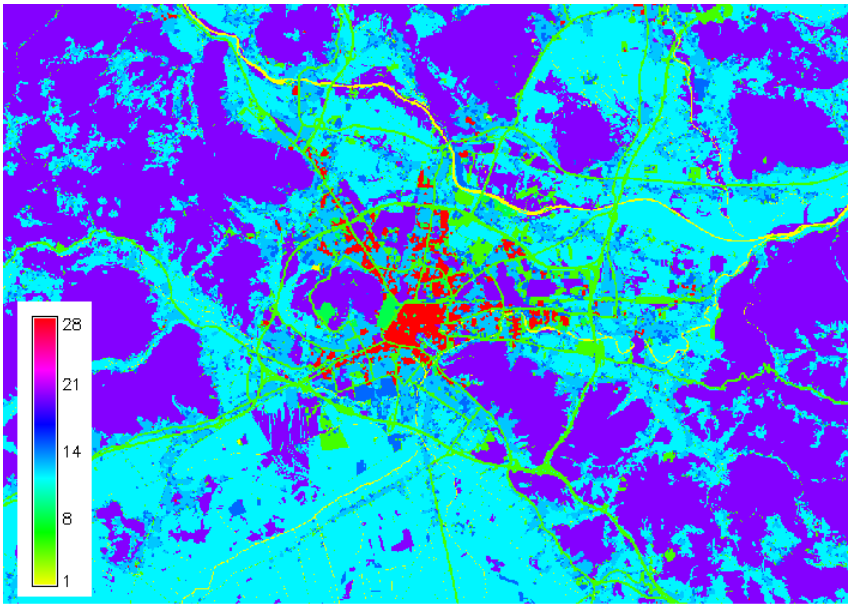


Fig. 11: The corresponding clutter map generated by *r.clutconvert*

- Usage:

```
r.clutconvert input=name Path_loss_values=name output=name [--overwrite] [--
verbose] [--quiet]
```

- Flags:

```
--o          Allow output files to overwrite existing files
--v          Verbose module output
--q          Quiet module output
```

- Parameters:

```
input                Name of input raster map
Path_loss_values     Path loss factors for land use
output               Name for output raster map
```

- Example:

```
r.clutconvert input=clut_category Path_loss_values=/home/user1/convtable
output=clut_dBloss --o
```

3. SRTM maps

SRTM (*Shuttle Radar Topography Mission*) map are Digital Elevation Maps (DEMs) obtained by a special radar system onboard Space Shuttle Endeavour during its 11-day mission in February 2000 [6,7]. The maps are publicly available [7]. Two alternate sources of SRTM based maps are [19] and [20]. Another interesting project producing global DEMs is ASTER [21], however those maps are currently less reliable.

SRTM maps are an alternative source (free for non-commercial use) to produce a GRASS DEM for a particular geographic region that can be used with RaPlaT for radio coverage computation. The suitable version of maps is located at http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/ [7]. It contains six subdirectories for different parts of the words: *Africa*, *Australia*, *Eurasia*, *Islands*, *North_America* and *South_America*.

Each SRTM3 map covers a geographic region of 1° (degree) longitude and latitude with resolution of 3" (arc seconds). This would result in the raster dimensions 1200x1200 (1200 = 1°/3" = 3600/3). In fact, the raster dimension is 1201x1201, with region boundaries extending 1,5 arc second (half the pixel size) beyond the 1 degree bounds in all for directions (E, W, N, S). (This way, the raster pixels on the region borders have their centers exactly at the 1° bounds, and theses bordering pixels overlap with the bordering pixels of the four neighboring regions.) The resolution of 3" corresponds to roughly 90 m on the equator, and correspondingly less in the E-W (longitude) dimension at other latitudes.

Maps are available as zipped files named according to the geographic coordinates (latitude, longitude, in degrees). For northern and eastern locations, the names have the form *NyyExxx.hgt* (with added .zip for zipped files), where *yy* and *xxx* represent the latitude and longitude (in degrees, non-negative values) of the lower left (S, W) corner of the 1 degree region. E.g., the file N45E014.hgt (which contains a part of Slovenia, including its capital Ljubljana), covers the region between 45°-0.5" and 46°+0.5" North (latitude) and between 14°-0.5" and 15°+0.5" East (longitude). For the southern and western location, the N and E in the file names are replaced by S and W, respectively.

The filename extension *.hgt* does not denote any special file format but simply stands for the word "height". The file contains raw data with no special formatting. The data are 1442401 (1201x1201) values of elevations above the sea level in [m], each written as a two-byte (16-bit) signed integer. Voids (raster points without a valid height data) are indicated by the special value -32768. The integers are written in the "big-endian" form. The term big/little-endian denotes the order of bytes which a particular processor type uses to write multi-byte integer values into memory bytes (little-endian systems store the least significant byte at the lowest memory byte address, while big-endian systems use the opposite order). The Intel x86 processor family uses the little-endian form, hence in order to read the integer values in *.hgt* files correctly on an Intel-based computer architecture (independently of the operating system used), the two bytes must be swapped.

Compared to commercial DEM's, SRTM maps have some limitations. Their resolution is not very high (3" or 90 m at the equator) and they contain void areas without valid measurements. The radar measurements were performed with 30 m x 30 m spatial sampling with the following minimal required accuracies quoted at the 90% level [22]:

- ≤ 16 m absolute vertical height accuracy,
- ≤ 10 m relative vertical height accuracy,
- ≤ 20 m absolute horizontal circular accuracy.

The actual accuracy of the generated maps is discussed in [23] and other documents, e.g. [24,25].

Manually selecting and downloading individual one-degree SRTM maps and combining them into a larger DEM would be cumbersome and error-prone, so we created scripts to automate these tasks. The user only specifies the final region extents and the corresponding SRTM repository (folder with files) and the rest is done automatically.

3.1. SRTM maps and GRASS locations/projections

Positions on the earth surface are defined globally in angular degrees relative to the equator and the prime meridian (going through Greenwich). Why these coordinates are natural, the third dimension, the height in meters (usually above the sea level), is a bit more complicated. It depends on the Earth form, which is not exactly a sphere but is modeled as a spheroid (ellipsoid of revolution). The current standard defining its parameters is WGS 84 (WGS - Word Geodetic System).

Local geographic maps are made by projecting a part of the earth surface to a plane using different cartographic projections (e.g. the Mercator projection with its parameters suitably defined for a particular location). Positions on these maps can be expressed as distances in meters relative to a suitably chosen reference point. Different countries generally use different cartographic projections.

In GRASS, a basic property of each GRASS *location* (which in turn includes users' GRASS *mapsets*) is its cartographic projection, which is defined at the time of the *location* creation. The positions on maps belonging to such (projected) *locations* are expressed in meters relative to the corresponding reference point. An example of such a GRASS *location* projection that we use for Slovenia is (as printed by *g.proj*):

```
GRASS 6.4.0 (Slovenia):~ > g.proj -p
-PROJ_INFO------------------------------------------------
name       : Transverse Mercator
proj       : tmerc
ellps      : bessel
lat_0      : 0
lon_0      : 15
k          : 0.9999
x_0        : 500000
y_0        : -5000000
towgs84    : 426.9206,142.6186,460.0871,4.90806,4.488093,-12.423166,17.1128
no_defs    : defined
-PROJ_UNITS-----------------------------------------------
unit       : metre
units      : metres
meters     : 1
GRASS 6.4.0 (Slovenia):~ >
```

A GRASS *location* can also be created without a cartographic projection by choosing the so called Latitude/Longitude pseudo-projection. The positions on the corresponding map are then expressed in angular degrees. The projection properties of such *location* are (as printed by *g.proj*):

```
GRASS 6.4.0 (LongLat):~ > g.proj -p
-PROJ_INFO------------------------------------------------
name       : Lat/Lon
proj       : ll
datum      : wgs84
ellps      : wgs84
no_defs    : defined
```

```
-PROJ_UNITS---------------------------------------------
unit       : degree
units      : degrees
meters     : 1.0
GRASS 6.4.0 (LongLat):~/GRASS-GIS >
```

Radio coverage calculations depend on distances in meters and hence require a location with a cartographic projection. SRTM maps, on the other hand, represent small one angular degree patches using unprojected coordinates in angular degrees. Hence, a Longitude/Latitude GRASS *location* must be created to handle and process these maps. After a GRASS DEM map is created from a set of SRT maps, it can be projected to the final GRASS *location* (with cartographic projection) using the GRASS *r.proj* command. All projections of coordinates and whole GRASS raster maps between two coordinate systems are performed by the PROJ.4 library and the GDAL part of the related GDAL/OGR library.

In the next subchapter we will explain the tools and procedures for:

- downloading the necessary SRTM maps,
- joining them into a GRASS map in a Latitude/Longitude pseudo-projection GRASS *location*,
- projecting the GRASS map to the final cartographic *location*,
- optionally removing void regions by interpolation,
- creating a dummy clutter map (in the absence of a real clutter map; required for *r.haraDEM*).

At the end, an example is given in which a SRTM-based DEM map is generated for Slovenia and compared with the official (commercial) DEM map.

3.2. Download SRTM maps - m.getSRTMmaps

To make downloading the necessary SRTM maps easier, we have developed a script, *m.getSRTMmaps*. It downloads and extracts (unzips) the SRTM maps required for the chosen region. The region is rectangular and defined by its bounds expressed in angular degrees (parameters *n*, *e*, *s* and *w*).

In principle, *m.getSRTMmaps* does not depend on GRASS and would not need any special GRASS support for it to work. However, for a consistent user experience, the script makes use of the GRASS command line parsing mechanism and hence works only within the GRASS environment (i.e. in a Terminal window where GRASS has been started). The GRASS *location* and its projection properties are not important, but since the following steps would normally be creating the longitude/latitude DEM map with the *m.SRTMtoGRASS* script (described later), it would probably be used within a longitude/latitude GRASS *location*.

The *m.getSRTMmaps* script downloads and extracts (unzips) maps to a destination directory defined with the parameter *mapsdir*. If *mapsdir* is not set, the current working directory is used. During downloading, each required zipped map is downloaded only if neither the zipped nor the extracted map with the same name exists yet in the destination directory. During extracting, each required map is extracted only if no map with the same name exists yet in the destination directory. In other words, existing file are kept unchanged and the corresponding download/extract operations are skipped with informative message printed on the screen.

By default, *m.getSRTMmaps* seeks the required maps in the *Eurasia* subdirectory (http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/Eurasia/). For other geographic regions, the

user must explicitly specify URL with the *urldir* parameter, substituting *Eurasia* with *Africa*, *Australia*, *Islands*, *North_America* or *South_America*.

- Usage:

```
m.getSRTMmaps [-p] n=value s=value e=value w=value [urldir=string] [mapsdir=string]
[--verbose] [--quiet
```

- Flags:

```
-p          The list of required maps will be printed only (no downloading
            /extracting)
--v         Verbose module output
--q         Quiet module output
```

- Parameters:

```
n           Value for the northern edge [degrees]
s           Value for the southern edge [degrees]
e           Value for the eastern edge [degrees]
w           Value for the western edge [degrees]
urldir      URL of the required SRTM maps
            default: http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/Eurasia/
mapsdir     Local directory for SRTM maps
```

- Example:

```
m.getSRTMmaps n=47.0 s=45.3 e=16.7 w=13.3
```

The example above downloads and extracts SRTM maps for the Slovenian geographic region.

3.3. Convert SRTM maps to a GRASS DEM - m.SRTMtoGRASS

The *m.SRTMtoGRASS* script first converts the required (previously downloaded and extracted) SRTM maps to GRASS maps (using the GRASS *r.in.bin* command). The GRASS map names are generated by adding prefix *tmp_* to the original SRTM map names (e.g. the SRTM map *N45E013.hgt* is converted to GRASS map *tmp_N45E013*.

Next, the *m.SRTMtoGRASS* sticks (merges) just converted GRASS maps to the final DEM map (using the GRASS *r.patch* command).

This script requires a Longitude/Latitude GRASS *location* to work and does not run in cartographic (projected) GRASS *locations*. Parameters *n*, *s*, *e*, *w* and *mapsdir* are equivalent to the equally named parameters of *m.getSRTMmaps*.

- Usage:

```
m.SRTMtoGRASS [-p] n=value s=value e=value w=value [mapsdir=string] out_DEM=string
[--overwrite] [--verbose] [--quiet]
```

- Flags:

```
-p          The list of required maps will be printed only (no downloading
            /extracting)
--o         Allow output files to overwrite existing files
--v         Verbose module output
--q         Quiet module output
```

- Parameters:

```
n           Value for the northern edge [degrees]
```

```
s            Value for the southern edge [degrees]
e            Value for the eastern edge [degrees]
w            Value for the western edge [degrees]
mapsdir      Local directory for SRTM maps
out_DEM      Output DEM
```

- Example:

```
m.SRTMtoGRASS n=47.0 s=45.3 e=16.7 w=13.3 out_DEM=outdem --o
```

The example above merges the SRTM maps covering the Slovenian region (downloaded and extracted in the previous step) to the final DEM map. As a side effect, partial DEM maps corresponding to the individual SRTM maps are created.

3.4. Project GRASS DEM to the final cartographic location

With previous steps we created a DEM map in a Longitude/Latitude GRASS *location*. For normal work, we usually use GRASS *locations* with cartographic projections. This is especially true for radio coverage computations, which require distances expressed in meters. Hence, the just created DEM map must be projected from the original Longitude/Latitude GRASS *location* to the final projected *location*. This can be done with the GRASS *r.proj* command, which maps a raster map from a non-current GRASS *location* to the current one.

In the following example, the LongLat GRASS *location* is where the original SRTM-based GRASS DEM was created. Cubic interpolation is used for mapping from the original SRTM original (around 90 m in N-S direction) to the new resolution of 100 m. The command is run within the user's final cartographic GRASS *location* and projects the raster map *outdem* from the *mapset* named *user1* in the Longitude/Latitude *location* named *LongLat* to the currently used (final) *mapset* and *location*.

```
r.proj input=outdem location=LongLat mapset=user1 output=outdem resolution=100
method=cubic --o
```

3.5. Removing voids

As has already been mentioned and will be illustrated in the next chapter for the case of Slovenia, SRTM maps (and hence the resulting DEM) can contain void areas. GRASS contains a module, *r.fillnulls*, that can fill void regions with interpolated values:

```
r.fillnulls input=outdem output=outdem_filled
```

This can be done in the final (projected) GRASS *location*. Of course, the interpolated values can differ considerably from the correct ones, especially if the void regions are large.

3.6. What about land use / clutter maps?

Certain models (e.g. hataDEM) require a clutter map (land-use-related loss maps) in addition to DEMs. In order to use such models, we must provide a clutter map. It can be generated from a land use map with the previously described *r.clutconvert* GRASS/RaPlaT module. In absence of a real land use map, a fake clutter map can be prepared that defines a suitable constant land-use-related loss over the whole region covered by DEM. It can be generated by the GRASS *r.mapcalc* command. Since it works in the *current* GRASS region, we must first set the region to that of DEM to produce a region-wise compatible clutter map.

The following commands would produce a clutter map named *outclut* with 10 dB loss over the whole DEM region defined by the raster map named *outdem*:

```
g.region rast=outdem
r.mapcalc outclut=10.0
```

3.7. An example: SRTM-based DEM for Slovenia

With the tools and procedures described in the next chapters, we will generate DEM for Slovenia and compared it to the official (commercial) DEM. We will use the following GRASS *region* definition for Slovenia (as printed by *g.region*):

```
GRASS 6.4.0 (Slovenia):~ > g.region -p
projection: 99 (Transverse Mercator)
zone:       0
datum:      towgs84=426.9206,142.6186,460.0871,4.90806,4.488093,-12.423166,17.1128
ellipsoid:  bessel
north:      195000
south:      25000
west:       370000
east:       630000
nsres:      100
ewres:      100
rows:       1700
cols:       2600
cells:      4420000
```

The official DEM is shown in Figs. 12 and 13 with Fig. 12 showing also the geographic map (Google Map) of the region for reference.

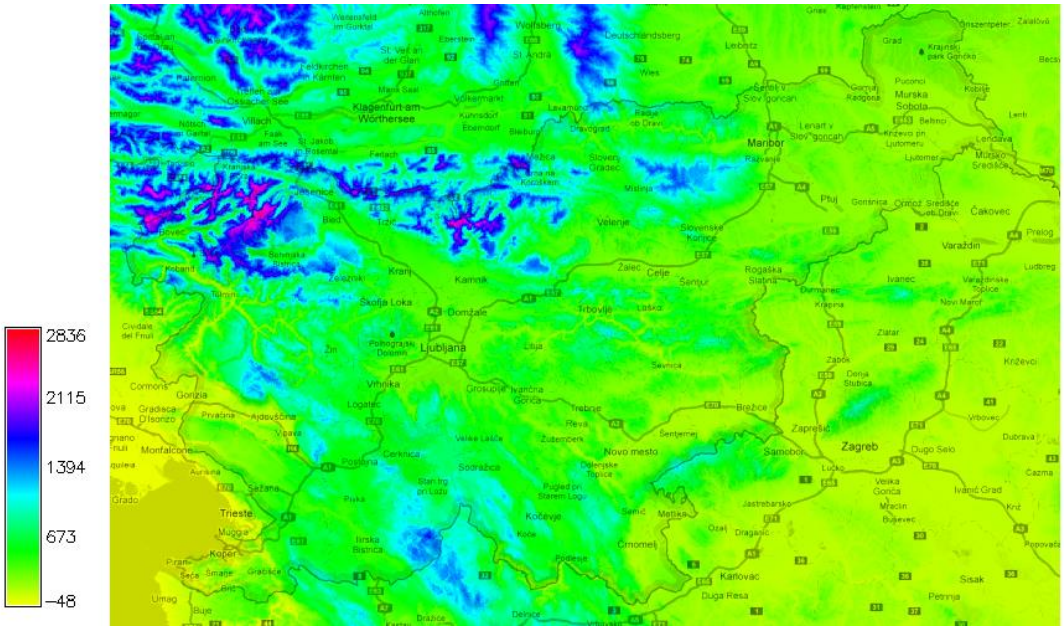


Fig. 12: Commercial Slovenian DEM + Google Map

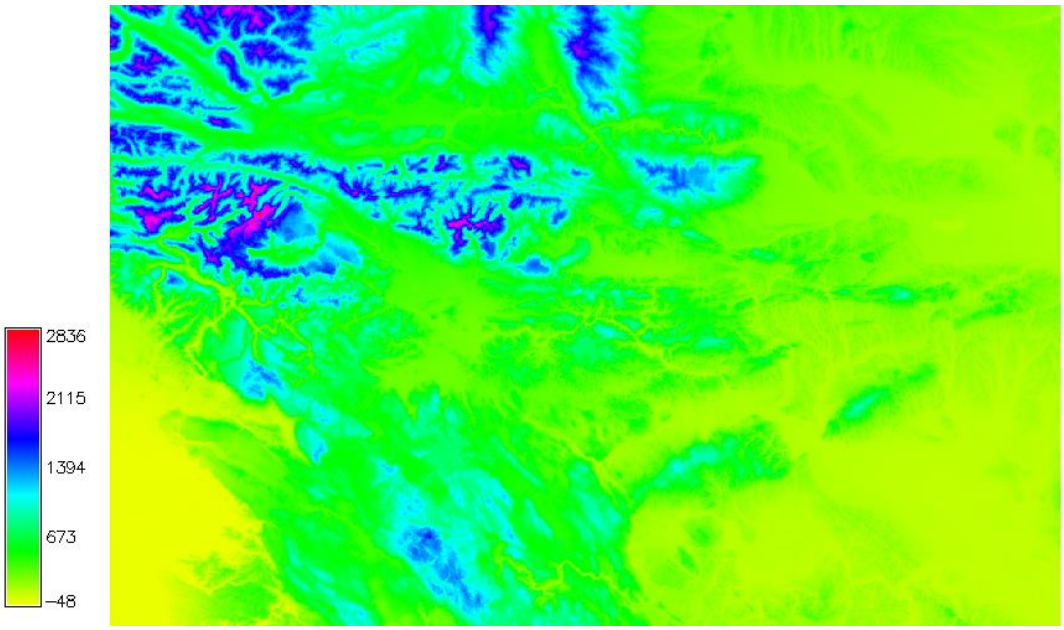


Fig. 13: Official (commercial) Slovenian DEM

Fig. 14 shows DEM created from the SRTM maps for the same region and with the same color scale and color legend. The only obvious differences are significant voids (white areas) mostly in the northwest mountainous region.

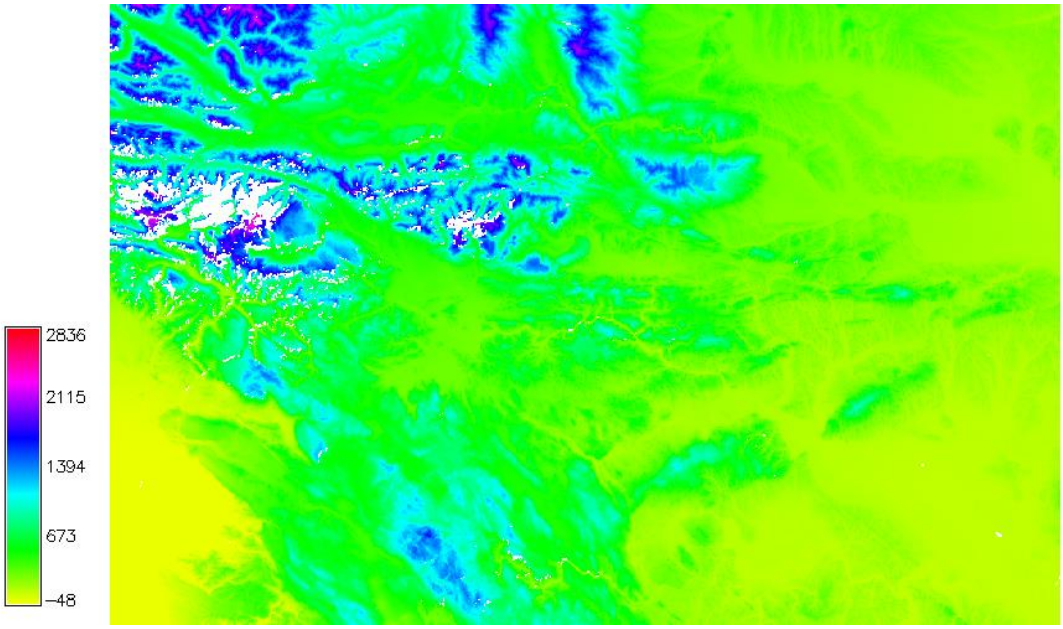


Fig. 14: Slovenian DEM created from SRTM maps

Fig. 15 shows height differences between the official and SRTM-based DEMs. White areas represent the void areas of the SRTM-based DEM. They are accompanied with some small areas of large height errors (reddish/greenish colored) reaching -436 to +366 m. The prevalent color, however, is blue, which corresponds to relatively small errors.

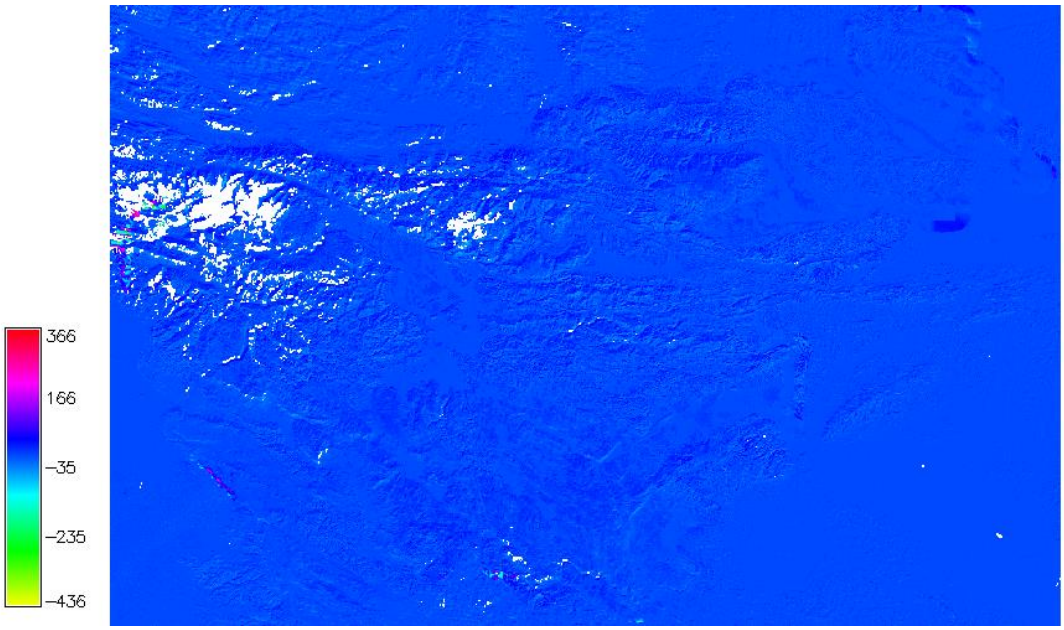


Fig. 15: Differences between the comercial and SRTM-based Slovenian DEMs

Figs. 16 to 18 give some more insight into the SRTM-based DEM height error magnitude. Figs. 16 and 17 show height errors along two diagonals: from lower-left corner to upper-right corner (Fig. 16) and form upper-left corner to lower-right corner (Fig. 17). We can see that errors of 20 m and even twice that much are not uncommon.

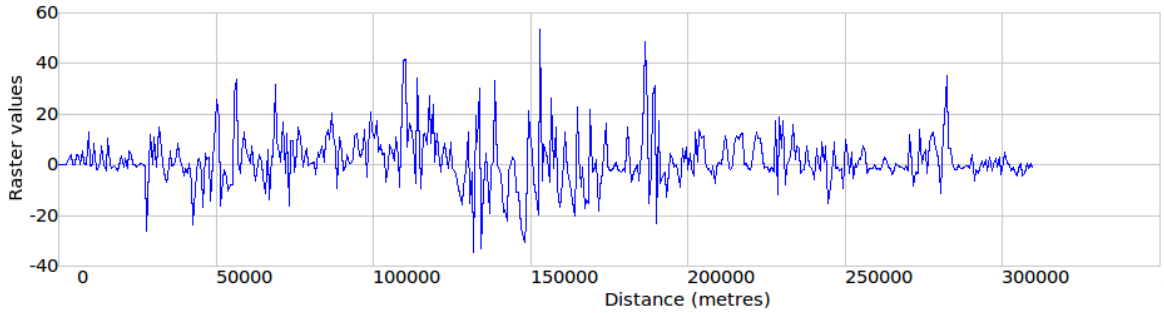


Fig. 16: Profile difference - lower-left corner to upper-right corner diagonal

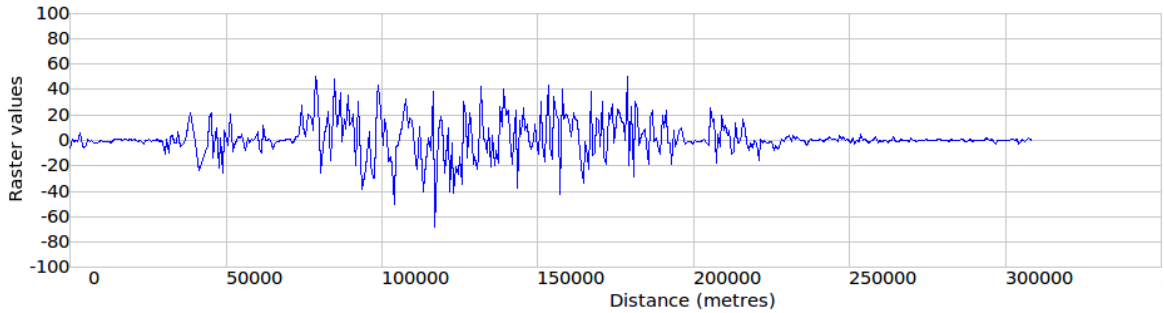


Fig. 17: Profile difference - upper-left corner to lower-right corner diagonal

Fig 18 contains a histogram of height difference in Fig. 15. It shows that the mean height error is approximatelly 0 and that the majority of errors are below about 30 m.

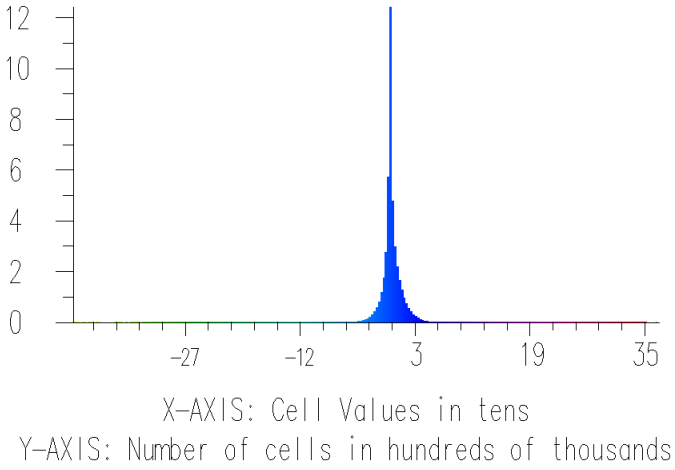


Fig. 18: Height difference histogram

This height error is somewhat larger that could be expected from the SRTM accuracy specifications. A closer look reveals that high error values are largely related to steep slopes where height errors are caused not only by direct height measurements errors but also by horizontal position errors. E.g. for a very steep terrain of 45°, a horizontal position error of 20 m (which corresponds to only 20% of the pixel size for a 100 m map raster) could result in a vertical position error of up to 20 m (depending on the direction of the horizontal position error relative to the slope direction).

The void areas in the SRTM-based map in Fig. 14 can be removed by interpolation with the GRASS GIS *r.fillnulls* command. The result is shown in Fig. 19. Voids are gone, but while this can work well for small areas, large height errors can be introduced for large void areas, as can be seen in Fig. 20 (some mountain peaks are completely lost with the interpolated heights up to 1259 m below the actual peak height).

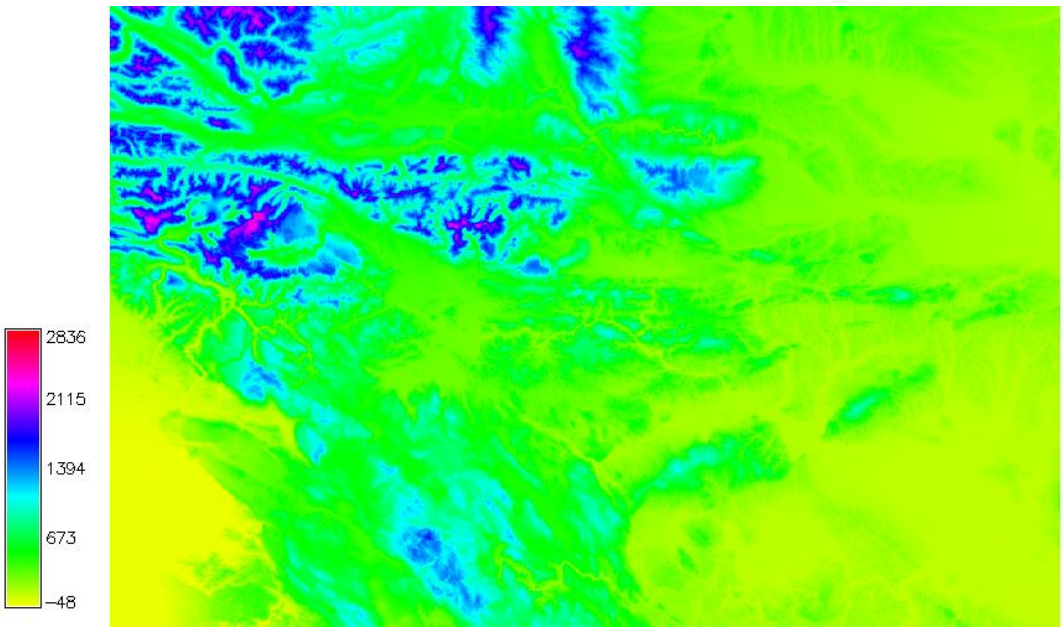


Fig. 19: Slovenian DEM created from SRTM maps with voids filled (*r.fillnulls*)

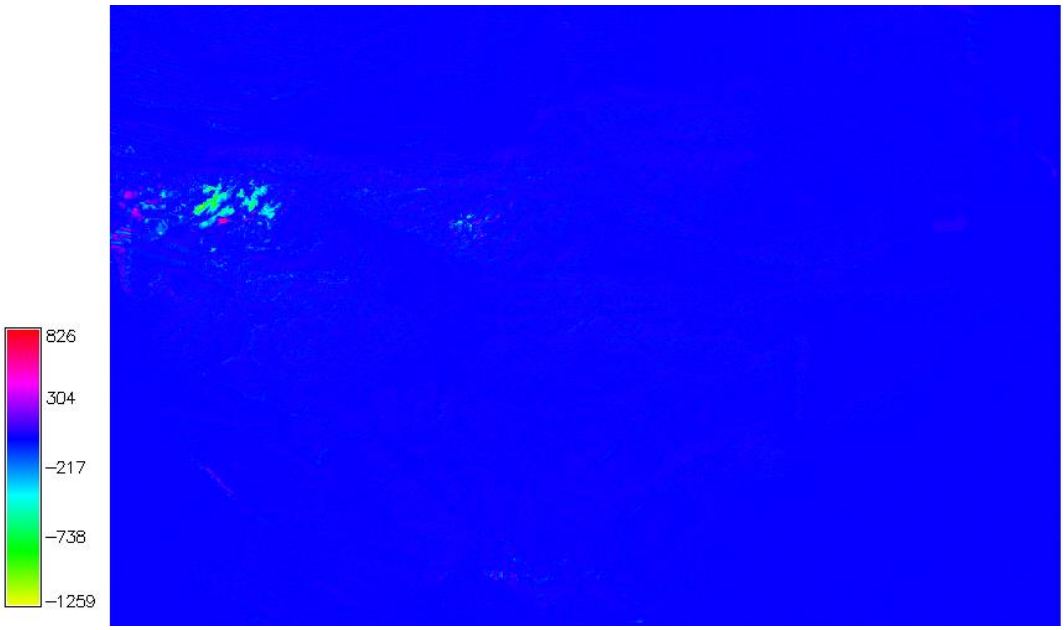


Fig. 20: Differences between the comercial and the SRTM-based voids-filled Slovenian DEMs

Hence, filling (large) voids with interpolation is problematic, and could cause completely false radio coverage results in case of putting a transmitter on a such location.

4. Virtual machine - GRASS & RaPlaT preinstalled

To make the use of GRASS RaPlaT easier, without the need to first install everything from scratch, we prepared a VM (*Virtual Machine*, a "virtual appliance") with Ubuntu OS and GRASS/RaPlaT tools installed. It was prepared with WMware Player, which is a simple VM hypervisor meant for personal desktop use. It is free for non-commercial use and can run on Linux and MS Windows operating systems [8]. The original VM would run natively in VMware Player and could easily be converted with VMware tools for other VMware's VM hypervisor (e.g. for ESXi, which is used in multiple-user server-based environments). However, for distribution, we converted the RaPlaT VM to the OVF format (*Open Virtualization Format*) which is compatible also with other vendors' VM hypervisors, e.g. with VirtualBox. The distribution contains three file:

- **.ovf* file : It contains description of the virtual machine (number of CPUs, memory size, and many other things) in XML text format.
- **.vmdk* (*Virtual Machine Disk*) file : It contains the VM disk image (in compressed form). This file format was originally developed by VMware but is now an open format.
- **.mf* (*Manifest File*) - This file is optional. It contains the SHA1 digest of the two previously described files, and - if present - serves to check their integrity.

The VM disk size is 40 GB, the memory size is 1 GB, and only on processor (one processor core) is used. The configured memory size and the number of CPU's (or CPU cores) are very modest so that the machine could be run on a host computer with limited resources. This settings can be easily changed and it is recommended to set a larger memory size and more CPU's (CPU cores), especially if one wants to use the *r.radcov*'s parallel execution support.

The RaPlaT VM was created with software installation and configuration performed according to the installation procedures described in details in the next chapter. The following main software was installed:

- Ubuntu 12.04 LTS (Precise Pangolin) desktop 32-bit.
- MySQL 5.5.
- PostgreSQL 9.1.
- GRASS 6.4.3RC2, a development installation from the source distribution, with support for MySQL and PostgreSQL. A number of other packages (libraries), either required or optionally used by GRASS, were also installed. This GRASS version is officially a "Release Candidate" version, but marked as stable and recommended. (It has some bugs corrected, which makes it a better choice compared to 6.4.2.)
- RaPlaT GRASS add-on.

The installed versions of software components are new, supported and recommended. Unfortunately they (or their combination) exhibit some minor bugs, which are supposed to be corrected in future versions. The two most obvious are:

- When exiting GRASS GUI (*Graphic User Interface*), i.e. closing the GRAS GIS Layer Manager window, a message is displayed multiple times in the GRASS Terminal (text) window (*"(python:*nnnn*): LIBDBUSMENU-GLIB-WARNING **: Trying to remove a child that doesn't believe we're its parent."*). There are no other related problems and the message can safely be ignored. (It is a known bug caused indirectly by an external standard library used by GRASS.)

- A GUI mouse problem in GRASS in connection with the new Ubuntu graphical desktop GUI, *Unity*. The problem appears in certain situations, e.g. when the user in the GRAS GIS Layer Manager window tries to add a map to the display and wants to use the slider to browse the maps. It fails since pressing the left button (to hold and move the slider) unexpectedly cancels the operation (a possible workaround is to first press the right button followed by ESC to close the resulting pop-up menu, and then the left button; or to browse maps by using the mouse scroll wheel).

4.1. Preconfigured Ubuntu user accounts

The following Ubuntu user accounts are preconfigured:

- *user1* - this is the primary (administrative) account with *sudo* rights. It is also the owner of the main GRASS database directories and the *PERMANENT* subdirectories, hence this user is also the GRASS administrator.
- *grassuser* - this is an ordinary system account, but it is also the owner of the corresponding GRASS *mapset* subdirectory(-s). This account can serve as a model for an ordinary GRASS user.

The initial passwords for both accounts are equal to the usernames. **It is strongly recommended to change both as soon as possible**, especially when (and before) enabling the remote access over the network.

Both user accounts are members of the Ubuntu *grass* group, specially created for GRASS users. A user must be a member of the *grass* group to have access to the GRASS database containing GRASS *locations* with users's *mapsets*.

4.2. About GRASS installation

The whole GRASS installation procedure would create two sets of binaries. The first one is generated during the compilation (*make*) within the GRASS source directory (created by decompressing the GRASS source distribution). In our case, the GRASS source directory is */usr/local/src/grass-6.4.3RC2*, the binaries are located in the *dist.i686-pc-linux-gnu* subdirectory, and the command (shell script *grass64*) that is used to start the GRASS application is located in *bin.i686-pc-linux-gnu*. To simplify calling of this script, we created a symbolic link to it, *grass64-d* (*d* for "development installation"), in */usr/local/bin*. Hence, GRASS is started simply by typing *grass64-d* in a Terminal window.

After starting, GRASS creates an environment variable, *GISBASE*, which contains the complete path to *dist.i686-pc-linux-gnu*. This directory contains the whole GRASS binary distribution with everything needed to actually run GRASS. This location within the GRASS source directory is suitable for development purposes including the development and compilation of additional modules (like those from the RaPlaT source distribution). The place for additional modules is in the *doc* subdirectory of the GRASS source directory. Simple examples of such modules are already included there as part of the GRASS source distribution.

The second set of binaries would be created from *dist.i686-pc-linux-gnu* (*make install*) in the normal system location */usr/lib/grass64*, and the GRASS start-up script *grass64* in */usr/bin*. This second set of binaries in the final system location is not really necessary to use GRASS, and was not created in our RaPlaT VM.

4.3. About GRASS database

GRASS uses a DBF-type database for storing its geographic maps and related information. It has the form of a directory tree, with the first level of subdirectories containing GRASS *locations*, and the second level containing GRASS user's *mapsets* (including the special *PERMANENT mapsets*, owned by the GRASS administrator). A suitable system location for this database is in */var/local* (complete database path */var/local/grassdata*). The VM database contains two demo locations:

- *Slovenia*, with a cartographic projection suitable for this geographic region. It contains *mapsets PERMANENT* (containing a SRTM-based DEM for Slovenia), *user1* and *grassuser*.
- *LongLat*, with Longitude/Latitude pseudo projection, which can be used for processing SRTM maps (as described earlier). It contains only one *mapset*, *PERMANENT*, for use by *user1*.

If GRASS DBF database is used also for output data tables optionally created by RapPlaT, they are normally stored in the *dbf* subdirectory within the current user's *mapset*. (Since each table is an independent DBF-format file, it could also be stored in another location, e.g. the user's home directory).

4.4. About MySQL

The MySQL database in the RaPLaT VM already contains a pre-created database named *grass*, which can be used to store data tables created by RaPlaT. Both preconfigured local users, *user1* and *grassuser*, have full access ("grant all") on this database.

The following commands can be used (in a Terminal window) to list the existing MySQL databases and data tables (the pre-created *grass* database in initially empty).

To login to MySQL and simultaneously connect the *grass* database:

```
mysql grass
```

This is a shortcut for first performing login to MySQL and then connecting the *grass* database (SQL keywords are usually written in capitals although they are not case-sensitive):

```
mysql
CONNECT grass;
```

To list the available databases (must be logged-in to MySQL):

```
SHOW DATABASES;
```

To list the available data tables of the connected database:

```
SHOW TABLES;
```

To exit MySQL:

```
\q
```

For administrative tasks (e.g. creating new MySQL users), the *root* MySQL login is required (no database if connected at login in this example, the user will be asked for the password of the MySQL *root* account):

```
mysql -u root -p
```

The password for *root* it is normally set during MySQL installation. In case of the RaPlaT VM, it is the same as for the Ubuntu *user1* account, i.e. *user1*. Again, **it is strongly recommended to change it**, since any user with an Ubuntu account could potentially login to MySQL as *root*.

The MySQL users are stored in the database *mysql* in the table *user*. The following commands can be used to list the existing users (*root* login is required, the first command lists only the user names, while the second one prints all columns of the table):

```
SELECT user FROM mysql.user;
SELECT * FROM mysql.user;
```

The MySQL configuration file *my.cnf* is in the */etc/mysql* directory. The main database directory is */var/lib/mysql*, where each database has its own subdirectory for its tables (the database *grass* is stored in the directory */var/lib/mysql/grass*).

The MySQL server can be restarted, stopped or started with the following commands:

```
sudo service mysql restart
sudo service mysql stop
sudo service mysql start
```

Besides the original MySQL user manual [26], short and useful quick start guides can be found in the web, e.g. [27].

4.5. About PostgreSQL

The PostgreSQL database in the RaPLaT VM already contains a pre-created database named *grass*, which can be used to store data tables created by RaPlaT. Both preconfigured local users, *user1* and *grassuser*, have full access on this database.

The following commands can be used (in a Terminal window) to list the existing PostgreSQL databases and data tables (the pre-created *grass* database in initially empty).

To login to PostgreSQL with the database *grass*:

```
psql grass
```

To list the available databases (must be logged-in to PostgreSQL):

```
\l
```

To list the available data tables of the connected database:

```
\dt
```

To list all users:

```
SELECT * FROM pg_user;
```

To exit PostgreSQL:

```
\q
```

For administrative tasks, the user should run *psql* as the special Ubuntu user *postgres* using *sudo* (no *postgress* password is set and required, but only users with *sudo* permission can run it, e.g. *user1*):

```
sudo -u postgres psql
```

The PostgreSQL configuration files are in the */etc/postgresql/9.1/main* directory. The main database directory is */var/lib/postgresql/9.1/main*.

The PosgreSQL server can be restarted, stopped or started with the following commands:

```
sudo service postgresql restart
sudo service postgresql stop
sudo service postgresql start
```

Besides the original PostgreSQL user manual [28], short and useful quick start guides can be found in the web, e.g. [29].

5. New GRASS and RaPlaT installation

RaPlat C modules are currently distributed as source code for Linux environment only and must be compiled before use. Standard precompiled GRASS Linux packages do not include the development support needed for compilation of third-party modules, so GRASS must be installed from its source code distribution. The required installation and configuration procedure are described in this chapter. They were used to prepare the RaPlaT virtual machine and would be used for installation on a standalone physical machine (with Ubuntu 12.04 pre-installed). However, this description should be taken only as guidelines and your mileage may vary (slightly) due to the constantly changing software updates and versions.

Installation of GRASS RaPlaT and related software described in this chapter was done on a freshly installed and updated Ubuntu 12.04.1 desktop 32-bit operating system. In the example below it is assumed, that the username of the primary user account (the one with administration rights, i.e. *sudo* permissions) is *user1*.

5.1. Install MySQL & PostgreSQL

First we install the database management systems (shortly databases) that we want to use with GRASS, so that GRASS *make* can find them and build the GRASS binaries properly. In our case these are MySQL and PostgreSQL. We can install both databases in the Ubuntu standard way with *Ubuntu Software Center*. (For MySQL, search for "mysql" ("mysql-server" and "mysql-client"); for PostgreSQL, search for "postgresql".)

During MySQL installation you are asked for the MySQL *root* user password. Note that this has nothing to do with the Ubuntu *root* user account (which is by default disabled for login, has no password defined, and *sudo* is used instead for administrative tasks). Any Ubuntu user that knows the MySQL *root* password can gain administrative MySQL access.

PostgreSQL has a different approach to gain administrative access. Its administrative permissions are assigned to a special Ubuntu user account, *postgres*, which is created during its installation. This account is by default disabled for login and no password is defined, but it can be used with *sudo*. Hence, no PostgreSQL password is required by default for the administrative access, but it is available only to the users with Ubuntu administrative rights (those with *sudo* permissions).

Originally, MySQL and PostgreSQL have only command line interfaces. If you prefer GUI-based management, you can additionally install MySQL Workbench (search for "mysql-workbench") for MySQL and pgAdmin III for PostgreSQL. To connect to the databases you use their internal administrative accounts *root* (for MySQL) and *postgres* (for PostgreSQL, but you must first define its internal *postgres* password).

5.2. Install GRASS and related software packages

First update Ubuntu 12.04 with the latest updates (*Ubuntu Update Manager*, or commands *sudo apt-get update* and *sudo apt-get upgrade*). Installation of GRASS from its source is performed according to [30] in the following steps:

1. Install the required software packages ("dependencies" - libraries and other supporting software). Use the following command (it is a bit different from what is suggested in [30] for Ubuntu 12.04):

```
sudo apt-get install \
build-essential \
make flex bison gcc libgcc1 g++ cmake ccache \
python python-dev python-qt4 python-qt4-dev \
python-opengl \
python-wxversion python-wxtools python-wxgtk2.8 \
python-dateutil libgsl0-dev python-numpy \
wx2.8-headers wx-common libwxgtk2.8-dev libwxgtk2.8-dbg \
libwxbase2.8-dev  libwxbase2.8-dbg \
libncurses5-dev \
zlib1g-dev gettext \
libtiff-dev libpng12-dev \
tcl8.5-dev tk8.5-dev \
libcairo2 libcairo2-dev \
sqlite3 libsqlite3-dev \
libpq-dev \
libreadline6 libreadline6-dev libfreetype6-dev \
txt2tags \
libfftw3-3 libfftw3-dev \
libqt4-core libqt4-dbg libqt4-dev libqt4-gui libqt4-sql libqt4-qt3support \
lsb-qt4 qt4-designer qt4-dev-tools qt4-doc qt4-qtconfig \
libapt-pkg-perl resolvconf \
libjasper-dev \
ruby \
subversion \
ffmpeg ffmpeg2theora \
libffmpegthumbnailer-dev \
libavcodec-dev \
libxmu-dev \
libavformat-dev libswscale-dev \
checkinstall \
libglu1-mesa-dev libxmu-dev
```

   And for the MySQL support:

```
sudo apt-get install libmysqlclient-dev
```

2. Install pre-compiled packages for PROJ.4, GEOS and GDAL (of course this can also be done with the graphical *Ubuntu Software Center* instead of the following commands):

```
sudo apt-get install libproj-dev
sudo apt-get install libgeos-dev
sudo apt-get install libgdal-dev
```

3. Install GRASS.

   From [31] download the GRASS source, in our case the latest version 6.4.3RC2. Extract the downloaded file (*grass-6.4.3RC2.tar.gz*) to */usr/local/src*, take ownership of the resulting directory *grass-6.4.3RC* and make it the current director (*cd*):

```
cd /usr/local/src
sudo tar -xvzf /home/user1/Downloads/grass-6.4.3RC2.tar.gz
sudo chown -R user1:user1 grass-6.4.3RC2
cd /usr/local/src/grass-6.4.3RC2
```

Run *configure* with suitable configure parameters for the GRASS build process (type *configure --help* to print help on usage and options; in the following example, support for PostgreSQL is enabled and some non-default locations are specified):

```
CFLAGS="-O3" LDFLAGS="-s" ./configure \
--enable-largefile=yes \
--with-readline \
--with-nls \
--with-cxx \
--with-proj-share=/usr/share/proj \
--with-gdal \
--with-geos=/usr/bin/geos-config \
--with-python=yes \
--with-wxwidgets \
--with-cairo \
--with-tcltk-includes="/usr/include/tcl8.5/" \
--with-opengl-libs=/usr/include/GL \
--with-ffmpeg=yes --with-ffmpeg-includes="/usr/include/libavcodec
/usr/include/libavformat /usr/include/libswscale" \
--with-freetype=yes --with-freetype-includes="/usr/include/freetype2/" \
--with-postgres=yes \
--with-postgresql=yes --with-postgres-includes="/usr/include/postgresql" \
--with-sqlite=yes \
--with-mysql=yes --with-mysql-includes="/usr/include/mysql" \
--with-odbc=yes
```

(Note that the line *--with-ffmpeg* above is split into two lines to fit on the page, but it should be a single line.)

The warning "*configure: warning: libmysqld not found*" (if printed) can be ignored.

If the building process was successful, the configuration result is printed, in our case:

```
GRASS is now configured for:  i686-pc-linux-gnu

  Source directory:            /usr/local/src/grass-6.4.3RC2
  Build directory:             /usr/local/src/grass-6.4.3RC2
  Installation directory:      ${prefix}/grass-6.4.3RC2
  Startup script in directory: ${exec_prefix}/bin
  C compiler:                  gcc -O3
  C++ compiler:                c++ -g -O2
  Building shared libraries:   yes
  64bit support:               no
  OpenGL platform:             X11

  MacOSX application:         no
  MacOSX architectures:
  MacOSX SDK:

  Tcl/Tk NVIZ:                yes

  BLAS support:               no
  C++ support:                yes
  Cairo support:              yes
  DWG support:                no
  FFMPEG support:             yes
  FFTW support:               yes
  FreeType support:           yes
  GDAL support:               yes
  GEOS support:               yes
  GLw support:                no
  LAPACK support:             no
  Large File support (LFS):   yes
  Motif support:              no
  MySQL support:              yes
  NLS support:                yes
  ODBC support:               yes
```

```
OGR support:                yes
OpenGL support:             yes
PNG support:                yes
PostgreSQL support:         yes
Python support:             yes
Readline support:           yes
SQLite support:             yes
Tcl/Tk support:             yes
wxWidgets support:          yes
TIFF support:               yes
X11 support:                yes
```

Now we can proceed with compiling GRASS (using the configuration from the previous step):

```
make -j2
```

If the process is successful, GRASS binary distribution subdirectory *dist.i686-pc-linux-gnu* (or similar) is created in */usr/local/src/grass-6.4.3RC2*. This distribution is fully functional without any further installation steps. This location is also the location where third party modules (in our case the RaPlaT modules) can be compiled (as described later). GRASS can be started by running the shell script *grass64* in the *bin.i686-pc-linux-gnu* subdirectory. To make this script callable without explicitly specifying its location, we create a symbolic link, *grass64-d*, in */usr/local/bin*:

```
sudo ln -s /usr/local/src/grass-6.4.3RC2/bin.i686-pc-linux-gnu/grass64
/usr/local/bin/grass64-d
```

(Note that the line above is split into two lines to fit on the page, but it should be a single line.)

From now on, GRASS can be started simply by typing *grass64-d* in a Terminal window (of course, some other name can be used instead).

There is an optional additional step in installing GRASS, which we will not perform. We could install GRASS to the standard Ubuntu system location for binaries with *sudo make install*, or we could create a standard binary installation package with *sudo checkinstall*. In any case, the previously created binaries created within */usr/local/src/grass-6.4.3RC2* directory tree would remain unchanged.

Finally, we can update links to the most recent shared libraries with:

```
sudo ldconfig
```

Before starting GRASS for the first time, its DBF database directory must be set up (see the next chapters).

5.3. Install RaPlaT modules

The RaPLaT modules (C modules and Python scripts) are distributed as source code, each in its own directory. These directories should be copied into *doc/raster* (for C modules) and *doc/python* (for Python scripts) subdirectories of the */usr/local/src/grass-6.4.3RC2* directory. We then *cd* to each of the modules directories and execute *make*. For C modules, this compiles the modules and generates the binary code in the *bin* subdirectory within *dist.i686-pc-linux-gnu* (the proper place for GRASS binaries), and html help files in the *docs/html*

subdirectory. For Python scripts, *make* copies scripts to the appropriate place, i.e. the *scripts* subdirectory within the */usr/local/src/grass-6.4.3RC2*. (The *make* process for the Python scripts is not really necessary since they are already executable and we can just copy them manually wherever we want). From now on, the RaPlaT modules and scripts can be called like any GRASS module or script within the GRASS environment (i.e. in the Terminal windows where GRASS has been started and is running).

To simplify the installation, a shell script, *RaPlaT_make*, has been prepared that builds all the modules in a single step. The detailed description is in the RaPlaT *Quick Start Guide*. The RaPlaT distribution is accompanied with two additional archive files, one with a demo antenna setup and another with RaPlaT demo commands and CSV files. Please consult RaPlaT *Quick Start Guide* for their description and installation. The manual also contains a short description of the demo GRASS database installation, which can be used instead of the description in the following subchapter.

5.4. Set up GRASS database and users

When GRASS is started for the first time, it asks for the location of its database with *locations* and *mapsets*. Hence, before starting GRASS, we have to set up its database. But before doing that, we will create an additional Ubuntu group (*grass*, all GRASS users should become its members) and an additional user account (*grassuser*, a generic non-privileged GRASS user). The following example assumes that the primary user account is *user1* (with Ubuntu administrative permissions, i.e. *sudo* permissions), and it will also be used for the GRASS database administration (it will be the owner of the database directory tree, except for the personal *mapsets* subtrees, which are owned by individual non-privileged GRASS users). Both, *user1* and *grassuser* will be members of the *grass* group.

```
sudo addgroup grass
sudo adduser user1 grass
sudo adduser grassuser
sudo adduser grassuser grass
```

The GRASS DBF database has a form of a directory tree with the first-level subdirectories representing *locations*, and the second-level directories representing user's *mapsets* within a *location*. A special *mapset*, *PERMANENT*, is present in each *location* and is readable by all GRASS users and contains the *location*'s shared maps. Before starting GRASS, only the main database directory must exist. A suitable location for it would be */var/local* (e.g. */var/local/grassdata*). After starting GRASS, the administrative GRASS user (the one with write access in this directory) can than create a new *location* (using the *Location wizard* button, the special *PERMANENT mapset* is created in the *location*), and all users can create their *mapsets* in a *location* (using the *Create mapset* button, provided that the write permission on the *location* directory is set for the GRASS users group, *grass* in our demo case).

Instead of starting with an empty GRASS database, it might be helpful to set-up the demo GRASS database distributed with RaPlat. This database includes two *locations*:

- *Slovenia*, which can be used for radio coverage computations. It includes *mapsets* for two users, *user1* (the GRASS administrator) and *grassuser* (a normal non-privileged GRASS user).
- *LongLat*, which can be used to convert and join SRTM maps to a GRASS DEM maps. It has only the PERMANENT *mapset* for use with *user1*.

The GRASS administrator (*user1* in our case) can create the database directory */var/local/grassdata* and extract the RaPlaT demo *locations* with the following commands:

```
sudo mkdir /var/local/grassdata
cd /var/local/grassdata
sudo tar -xvzf /home/user1/Downloads/RaPlaT_demolocs.tar.gz
```

The next step is to set the file and directory ownerships and permissions properly. The whole database directory tree ownership is first given to the user *user1* and group *grass*, except for the individual user's *mapset* where ownerships are given to the corresponding users (only the *grassuser*'s *mapset* in our case):

```
sudo chown -R user1:grass /var/local/grassdata
sudo chown -R grassuser /var/local/grassdata/Slovenia/grassuser
```

(The above commands differ a bit from those in the *Quick Start Guide* but the result is the same.)

If we want, we can apply some additional settings on the database directory tree (chances are that some of these settings are already set by default). We can ensure that only the users (owners of *mapsets*) have *write* access on their *mapsets*:

```
sudo chmod -R u+w,go-w /var/local/grassdata
```

We can allow non-privileged users to freely create their own *mapsets* in *locations* by setting the group *write* permission on *location* directories:

```
sudo chmod g+w /var/local/grassdata/*
```

It is convenient to set the *setgid* bit on directories (the *ls* command displays it in place of group *x* permission as *s* when *x* is set, and *S* otherwise). This way, the newly created files and subdirectories will inherit the existing directory group ID (*grass* in our case) instead of getting the primary group ID of the user creating them. This can be done with the following command:

```
sudo find /var/local/grassdata -type d -exec chmod g+s {} \;
```

Be aware that by setting a group ID (*grass* in our case) other that the primary user group (*user1*, *grassuser*), members of this group (*grass*) may gain unwanted write access to the other users' *mapsets*, hence it is wise to review the group *write* permissions on directories and files within the database directory tree.

5.5. Set up MySQL / PostgreSQL for use with GRASS-RaPlaT

RaPLaT module *r.MaxPower* (called by *r.radcov*) optionally creates an output data table. If we want to use MySQL or PostgreSQL to store it, the following must be done first:

- a database (or more of them) must be created to store these tables,
- GRASS users must be granted access to this database.

The following examples show how this was done for the RaPLaT demo VM. More information about MySQL and PostgreSQL usage can be found in [26,27] and [28,29], respectively.

In the examples below, the traditional command-line used interface is used. Various GUI-based MySQL and PostgreSQL management tools can be used instead, e.g. MySQL Workbench and pgAdmin III for MySQL and PostgreSQL, respectively.

5.5.1. Set up MySQL

We can create a MySQL database with the following command:

```
mysqladmin -u root create grass -p
```

The parameter *-u root* requires execution with the MySQL *root* account (to obtain administration rights) and the user is asked for its password. This password is usually defined during the MySQL installation and is independent from the Ubuntu *root* password (which is by default not defined and login is disabled).

The following command sequence can be used to create MySQL users and grant them the necessary permissions on the *grass* database. The first command runs the MySQL client, performing login to the MySQL administrative *root* account. The second command lists existing databases and is used to verify that the *grass* database does exist. The following four commands create two MySQL local users, *user1* and *grassuser*, and grant them all permissions on the data tables in the *grass* database. The last command ends the MySQL client session. (Command prompts are shown to differentiate between the Ubuntu shell and PostgreSQL client commands.)

```
user1@ubuntu:~$ mysql -u root -p
mysql> SHOW DATABASES;
mysql> CREATE USER 'user1'@'localhost';
mysql> CREATE USER 'grassuser'@'localhost';
mysql> GRANT ALL ON grass.* TO 'user1'@'localhost';
mysql> GRANT ALL ON grass.* TO 'grassuser'@'localhost';
mysql> \q
```

The MySQL users are stored in the table *user* of the database *mysql*. The following commands can be used to list the existing users (the first command lists only the usernames):

```
SELECT user FROM mysql.user;
SELECT * FROM mysql.user;
```

Besides being able to create data tables with RaPlaT (*r.radcov*, *r.MaxPower*) in the *grass* database, the new MySQL users (*user1*, *grassuser*) can now also use the MySQL client *mysql* to work with the *grass* data tables. E.g. to get the list of all data tables in the *grass* database, *user1* would execute the following commands:

```
user1@ubuntu:~$ mysql grass
mysql> SHOW TABLES;
mysql> \q
```

The default main database directory is */var/lib/mysql*, where each database has its own subdirectory for its tables (the database *grass* is stored in the directory */var/lib/mysql/grass*). The MySQL configuration file *my.cnf* is in the */etc/mysql* directory. If necessary, the MySQL server can be restarted, stopped or started with the following commands:

```
sudo service mysql restart
sudo service mysql stop
sudo service mysql start
```

5.5.2. Set up PostgreSQL

PostgreSQL administration is done using a special Ubuntu user account, *postgres*, which is created during the PostgreSQL installation. By default, no password is defined for this account and direct login is disabled. Instead, *sudo* is used, and all users having Ubuntu *sudo* permission have administrative PostgreSQL access.

A user can become a *postgress* user with any of the following four commands:

```
sudo -u postgres -s
sudo su postgres

sudo -u postgres -i
sudo su - postgres
```

The last two commands also perform login, and set the current directory to the *postgres* home directory, which is the main data directory for PostgreSQL databases (*/var/lib/postgresql* by default).

Instead of explicitly becoming the *postgres* user (until *exit* is executed), only a per-command user change is used in the following examples. The *grass* database can be created with the following command:

```
sudo -u postgres createdb grass
```

The two users, *user1* and *grassuser*, are created with the following commands:

```
user1@ubuntu:~$ sudo -u postgres createuser
Enter name of role to add: user1
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles? (y/n) n

user1@ubuntu:~$ sudo -u postgres createuser
Enter name of role to add: grassuser
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles? (y/n) n
```

In the described setup we created no PostgreSQL *schema*, so all users implicitly access the *public* schema and are granted database access by default. Such setup is suitable for one or a few cooperating users in a database.

Besides being able to create data tables with RaPlaT (*r.radcov*, *r.MaxPower*) in the *grass* database, the new PostgreSQL users (*user1*, *grassuser*) can now also use the PostgreSQL client *psql* to work with the *grass* data tables. To connect to the *grass* database, get the list of all databases, data tables, users, and then exit, *user1* (or *grassuser*) would execute the following commands (command prompts are shown two differentiate between the Ubuntu shell and PostgreSQL client commands):

```
user1@ubuntu:~$ psql grass
grass=> \l
grass=> \dt
grass=> SELECT * FROM "pg_user";
grass=> \q
```

The PostgreSQL configuration files are in the */etc/postgresql/9.1/main* (for version 9.1). The default main database directory is */var/lib/postgresql/9.1/main*. If necessary, the PostgreSQL server can be restarted, stopped or started with the following commands:

```
sudo service postgresql restart
sudo service postgresql stop
sudo service postgresql start
```

6. References

[1] GRASS GIS, Wikipedia, http://en.wikipedia.org/wiki/GRASS_GIS

[2] GRASS GIS, home page, http://grass.osgeo.org/

[3] GRASS-RaPlaT, The Radio Planning Tool for GRASS GIS system, Home page, http://www-e6.ijs.si/index.php/en/software/grass-raplat

[4] Andrej Hrovat, Igor Ozimek, Andrej Vilhar, Tine Celcer, Iztok Saje, Tomaž Javornik, Radio coverage calculations of terrestrial wireless networks using an open-source GRASS system. WSEAS Transactions on Communications, 2010, vol. 9, no. 10, pp. 646-657.

[5] Igor Ozimek, Andrej Hrovat, Andrej Vilhar, Tine Celcer, Iztok Saje, Tomaž Javornik, GRASS-RaPlaT - an open-source tool for radio coverage calculations, V: Joint Workshop on Wireless Communications, 1-2 March 2011, Paris, France, JNCW 2011. [S. l.]: IEEE, France section, 2011, 6 pp.

[6] Shuttle Radar Topography Mission, Wikipedia, http://en.wikipedia.org/wiki/Shuttle_Radar_Topography_Mission

[7] Shuttle Radar Topography Mission, NASA - Jet Propulsion Laboratory, http://www2.jpl.nasa.gov/srtm/

[8] VMware Player, Home page, http://www.vmware.com/products/player

[9] Comma-separated values, Wikipedia, http://en.wikipedia.org/wiki/Comma-separated_values

[10] Common Format and MIME Type for Comma-Separated Values (CSV) Files, RFC 4180, October 2005, http://www.rfc-editor.org/rfc/rfc4180.txt

[11] GRASS-Wiki, Category:Parallelization, http://grasswiki.osgeo.org/wiki/Category:Parallelization

[12] GRASS-Wiki, Parallel GRASS jobs, http://grasswiki.osgeo.org/wiki/Parallel_GRASS_jobs

[13] S. R. Saunders, Antennas and Propagation for Wireless communication systems.

[14] M. Hata, Empirical formula for propagation loss in Land Mobile radio services, IEEE Transactions on Vehicular Technology, Vol. 29, no. 3, avgust 1980.

[15] D. J. Cichon, T. Kurner, Propagation prediction models, COST 231 Final Rep., http://www.lx.it.pt/cost231/

[16] J. Walfisch, H. L. Bertoni, A Theoretical Model of UHF Propagation in Urban Environments, IEEE Trans. Antennas Propagat., Vol. 36, pp. 1788–1796, December 1988.

[17] J. Walfisch, H. L. Bertoni, A Theoretical Model of UHF Propagation in Urban Environments, IEEE Trans. Antennas Propagat., Vol. 36, pp. 1788–1796, December 1988.

[18] MSI Planet Antenna File Format, http://radiomobile.pe1mew.nl/?The_program:Definitions:MSI

[19] Digital elevation data, http://www.viewfinderpanoramas.org/dem3.html

[20] CGIAR-CSI, SRTM 90m Digital Elevation Database v4.1, http://www.cgiar-csi.org/data/srtm-90m-digital-elevation-database-v4-1

[21] ASTER Global Digital Elevation Map Announcement, , NASA - Jet Propulsion Laboratory, http://asterweb.jpl.nasa.gov/gdem.asp

[22] Shuttle Radar Topography Mission, SRTM mission statistics, NASA - Jet Propulsion Laboratory, http://www2.jpl.nasa.gov/srtm/statistics.html

[23] Rodriguez, E., C.S. Morris, J.E. Belz, E.C. Chapin, J.M. Martin, W.Daffer, S. Hensley, 2005, An assessment of the SRTM topographic products, Technical Report JPL D-31639, Jet Propulsion Laboratory, Pasadena, California, 143 pp., http://www2.jpl.nasa.gov/srtm/SRTM_D31639.pdf

[24] Ricardo Passini, Karsten Jacobsen, Accuracy analysis of SRTM height models, http://www.ipi.uni-hannover.de/uploads/tx_tkpublikationen/RP_KJ_07_SRTM.pdf

[25] Antonios Mouratidis, Pierre Briole, Kostas Katsambalos, SRTM 300 DEM (versions 1, 2, 3, 4) validation by means of extensive kinematic GPS measurements: a case study from North Greece, International Journal of Remote Sensing, Vol. 31, No. 23, 10 December 2010, 6205–6222, http://users.auth.gr/kvek/mouratidis_et_al_2010_ijrs.pdf

[26] MySQL 5.5 Reference Manual (Including MySQL Cluster NDB 7.2 Reference Guide), http://dev.mysql.com/doc/index.html, http://dev.mysql.com/doc/refman/5.5/en/index.html, http://downloads.mysql.com/docs/refman-5.5-en.a4.pdf

[27] MySQL/MySQL Quick Start Guide, http://www.coderguide.com/Guides:MySQL/MySQL_Quick_Start_Guide

[28] PostgreSQL 9.1.9 Documentation, The PostgreSQL Global Development Group, http://www.postgresql.org/docs/manuals/, http://www.postgresql.org/docs/9.1/interactive/index.html, http://www.postgresql.org/docs/9.1/static/index.html, http://www.postgresql.org/files/documentation/pdf/9.1/postgresql-9.1-A4.pdf

[29] PostgreSQL QuickStart/Reference Commands..., http://www.linuxweblog.com/postgresql-reference

[30] GRASS-Wiki, Compile and Install Ubuntu, http://grasswiki.osgeo.org/wiki/Compile_and_Install_Ubuntu

[31] GRASS GIS, Download, http://grass.osgeo.org/download/, http://grass.osgeo.org/download/software/#g64x