New lidar processing functionality in GRASS GIS 7.1

webinar for U.S. Fish and Wildlife Service Remote Sensing Technical Group

Vaclav Petras (Vashek)

Center for Geospatial Analytics

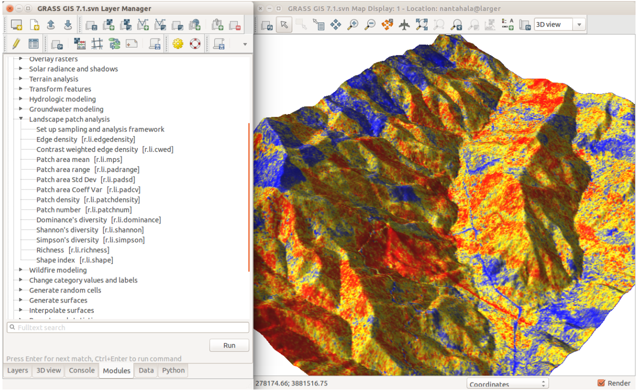
NC STATE UNIVERSITY

February 22, 2016

GRASS GIS

- all in one
- functionality split to modules
- unified access to all modules from GUI, command line, Python and R
- GUI independent but integrated

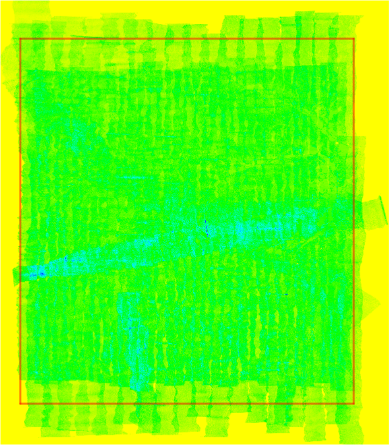
GRASS GIS



Typical lidar workflow

Explore the data with *r.in.lidar*

- import extent based on input data (`-e` flag)
- resolution set to high number (coarse, `resolution=10`)
- count number of points per cell (`method=n`)
- try different class and return filters (`class_filter`, `return_filter`)
- decide on computational region extent and resolution (`g.region`)



r.in.lidar, 578 million points in 90 files to 1882 × 1651 cells using 50MiB in 2 min

Typical lidar workflow

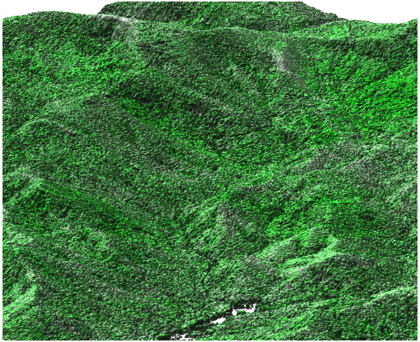
Analyze with *r.in.lidar*

- statistics of point counts, height and intensity
- using different resolutions

Create surface with *v.in.lidar* and interpolation

- few/sparse points, smaller extent
- interpolate smooth surface without NULLs

Continue analysis using standard GRASS GIS tools



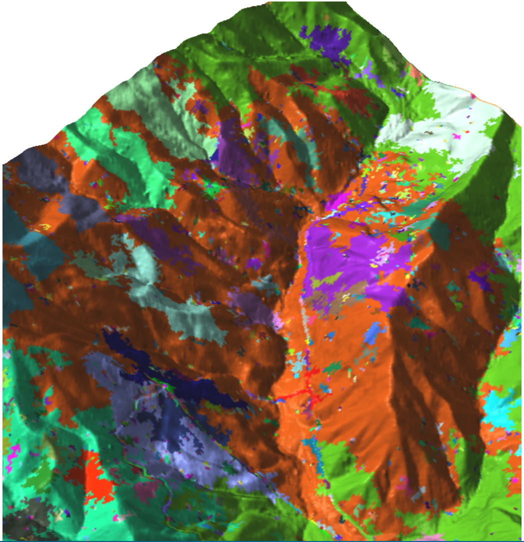
r.in.lidar and r.neighbors, range smoothed maximum surface, 551 million points from 90 files to 8076 x 7223 cells using 480MiB in 4 min

Openly accessible study materials by NCSU OSGeoREL

- GRASS GIS, Python and more
- Geospatial Modeling and Analysis `ncsu-osgeorel.github.io/geospatial-modeling-course`
- UAV/lidar Data Analytics course `ncsu-osgeorel.github.io/uav-lidar-analytics-course`

r.in.lidar

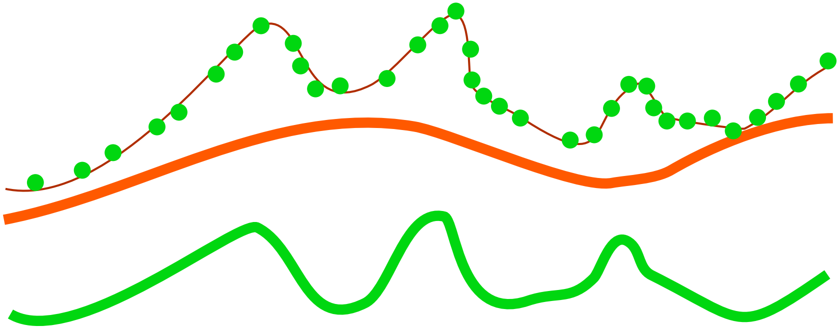
- one file or multiple files as input
- produces raster from points
  - n, min, max, sum
  - mean, range, skewness, ...
- limits import by
  - range of Z
  - return, class
- subsequent raster-based processing
- use intensity instead of the Z
- base raster



i.segment on count of ground points and count of non-ground points

r.in.lidar: compute height above a given raster during binning

r.in.lidar – compute height above given surface (`base_raster`)



The resolutions of binning and ground raster can differ.

r.in.lidar: base_raster workflow

`r.in.lidar method=mean class=2`

- ☐← 29 million points from 90 files
- ☐→ ground with NULLs (8×8km, 1m)

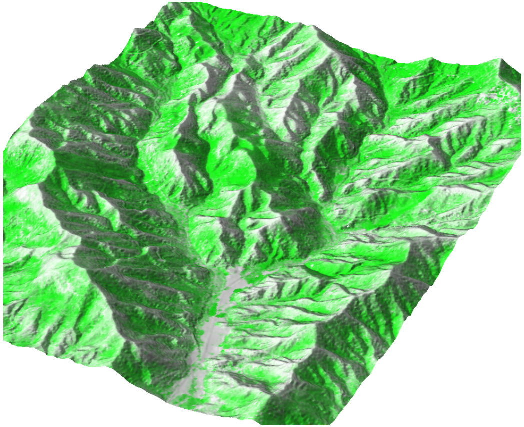
`r.neighbors size=25`

- ☐← ground with NULLs
- ☐→ continuous ground

`r.in.lidar method=max resolution=10 class_filter=1`

- ☐← 522 million points from 90 files
- ☐← continuous ground
- ☐→ max veg height per 10m cell

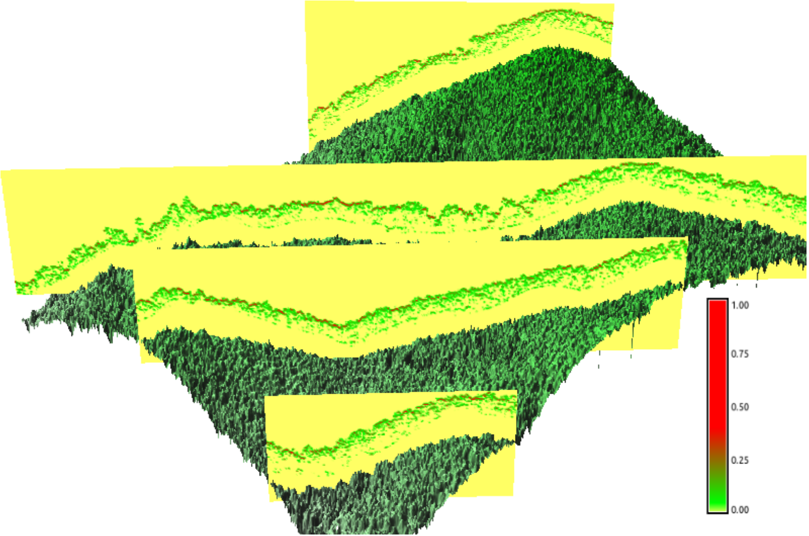
9 min, 480 MiB, Ubuntu, 2.7 GHz, SSD
578 million points in 90 files



r3.in.lidar

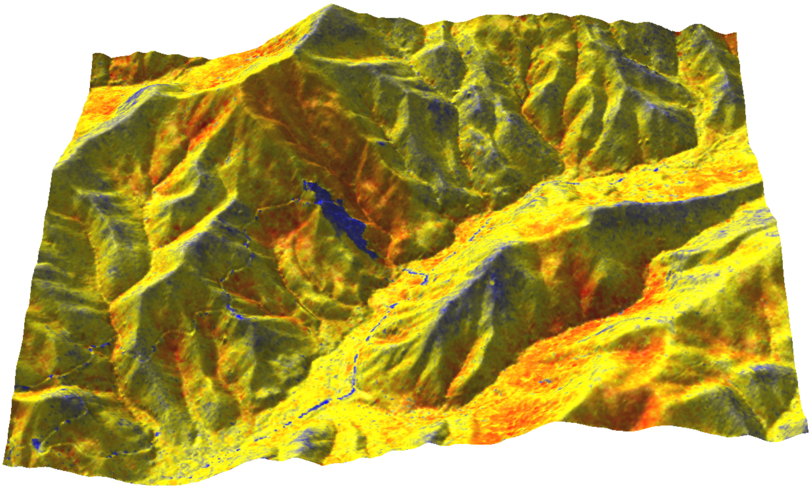
- generally similar to *r.in.lidar*
- 3D raster
- proportional count
  - count per 3D cell relative to the count per vertical column
- intensity can be used instead of count

under development



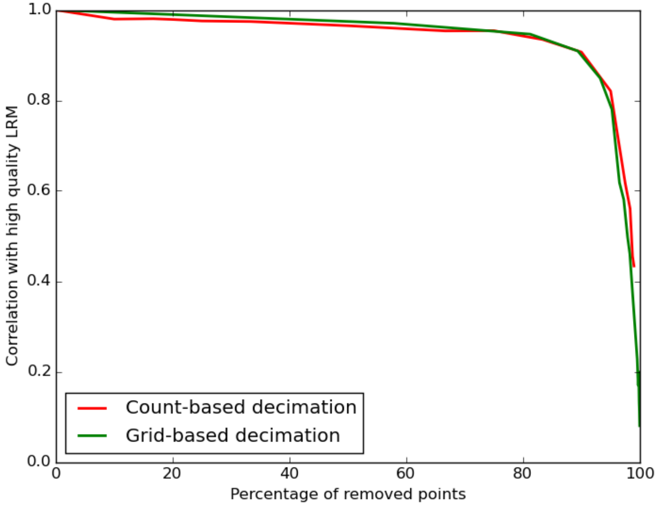
v.in.lidar

- filtering same as in *r.in.lidar*
- 3D vector
- used together with
  - *r.in.lidar*
  - interpolation modules (*v.surf.rst*, *v.surf.bspline*, *v.surf.idw*)
- decimation
- polygons as a mask



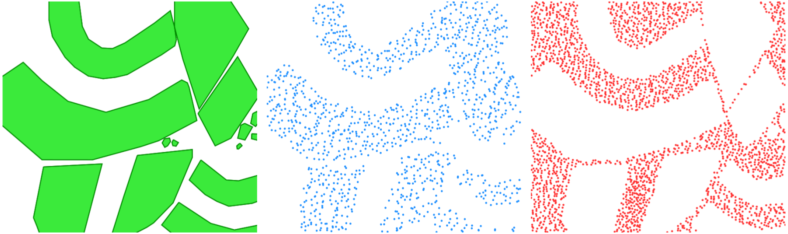
range from *r.in.lidar* on ground obtained from *v.in.lidar* followed by *v.surf.rst*

Comparison of count-based and grid-based decimation



Crop the point cloud by polygon

v.in.lidar – limit the import to selected areas (2D)



`areas` `v.in.lidar mask=` `v.in.lidar -i mask=`

Speed optimization

r.in.lidar

- choose computation region extent and resolution ahead
- have enough memory to avoid using `percent` option

v.in.lidar

- `-r` limit import to computation region extent
- `-t` do not create attribute table
- `-b` do not build topology (applicable to other modules as well)
- `-c` store only coordinates, no categories or IDs

Memory requirements

r.in.lidar

- depend on the side of output and type of analysis
- can be reduced by `percent` option
- `ERROR: G_malloc: unable to allocate ... bytes of memory`
- on Linux available memory for process is RAM + SWAP partition

v.in.lidar

- low when not building topology
- `export GRASS_VECTOR_LOWMEM=1`

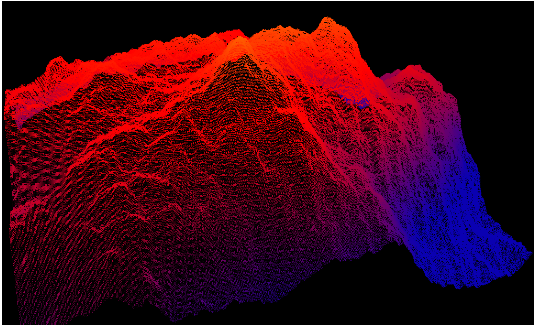
Limits

- vector features with topology: count limit is about 2 billion features per vector map
- points without topology: count theoretically limited only by the 64bit architecture (may be 16 exbibytes per file but depends on the file system)
- more limits for 32bit versions for operations which require memory
- number of open files limitation (system limit)
  - often 1024, change using *ulimit* on Linux
- *r.watershed* 90,000 x 100,000 (9 billion cells) in 77.2 hours (2.93GHz)
- *v.surf.rst*: minutes = 1.5e-11 * points^2 (test: 13 min for 1 million points and 201880 cells)
- read the documentation
- write to grass-user mailing list or GRASS GIS Tracker in case you hit limits

v.out.lidar

exports points in a vector map as lidar points

- visualization (plas.io, CloudCompare)
- further processing (PDAL, libLAS, CloudCompare, . . .)
- testing workflows with generated data

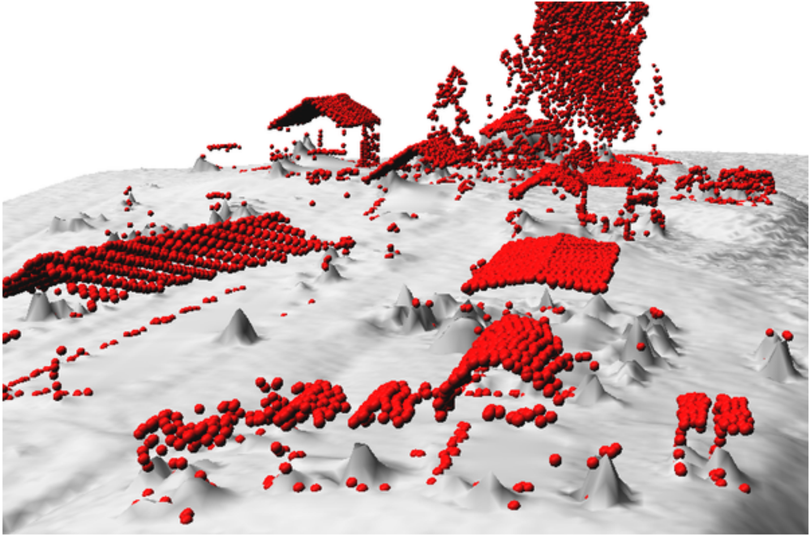


r.surf.fractal output in plas.io

More smaller tools under development.

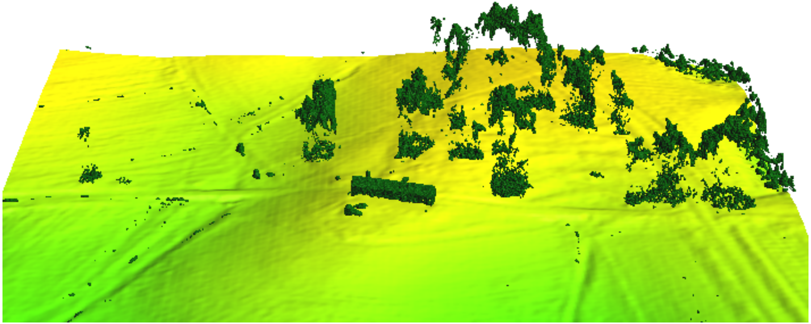
v.lidar.edgedetection suite

- *v.lidar.edgedetection*
- *v.lidar.growing*
- *v.lidar.correction*
- uses return information



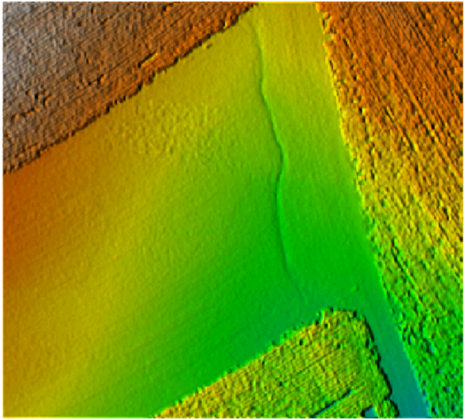
v.lidar.mcc

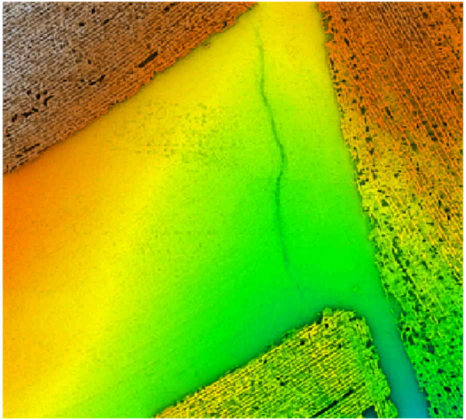
- ground and non-ground
- multiscale curvature based classification algorithm



r.skyview

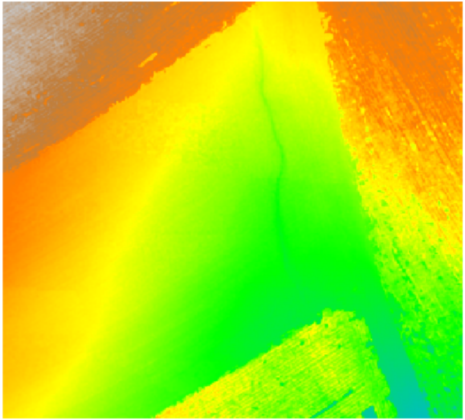
- sky-view factor

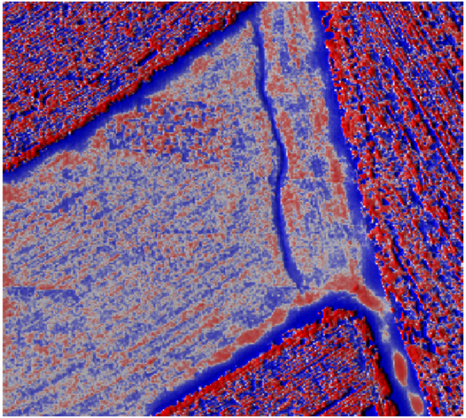




r.local.relief

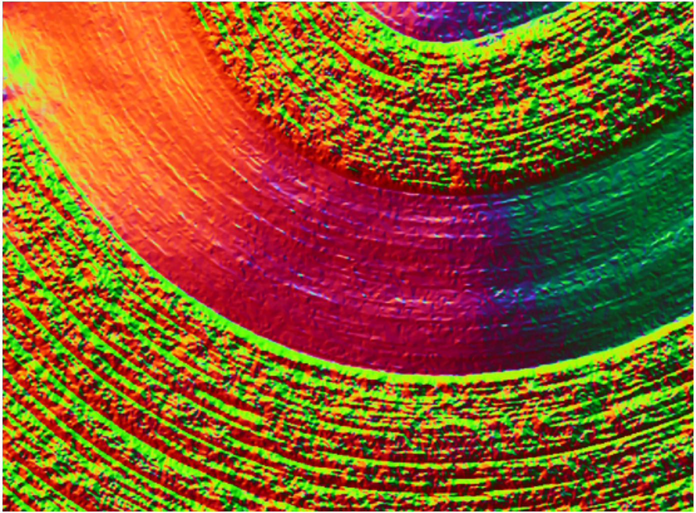
- local relief model





r.shaded.pca

- relief shades from various directions
- combined into RGB composition



Integration with PDAL

experimental

- *v.in.pdal*
- reprojection during import
- ground filter (alternative to *v.lidar.edgedetection* or *v.lidar.mcc*)
- compute height as a difference from ground

planned

- *r.in.pdal*, *r3.in.pdal*, *v.out.pdal*, *r.out.pdal*, ...
- more PDAL filters

GRASS GIS lidar tools roadmap

- now: basic tools available in GRASS GIS 7.0
  - 7.0.3 released this January with 64bit support for MS Windows
- now: presented functionality available for testing in development version of GRASS GIS
  - daily build for MS Windows and Ubuntu
  - self-compiled version (simple for Fedora, CentOS, ... possible on Mac OS)
- summer: 3D raster, PDAL, 2D display, smooth reprojections
- fall/winter: backport of stable functionality to 7.0 or release of 7.1

Acknowledgements

Software: GRASS GIS

Presented functionality are work done by Vaclav Petras, Markus Metz, and the GRASS development team. Thanks to users for feedback and testing, especially to Doug Newcomb, Markus Neteler, and William Hargrove.

Dataset: Nantahala NF, NC: Forest Leaf Structure, Terrain and Hydrophysiology

Lidar data acquisition and processing completed by the National Center for Airborne Laser Mapping (NCALM). NCALM funding provided by NSF's Division of Earth Sciences, Instrumentation and Facilities Program. EAR-1043051.
`http://dx.doi.org/10.5069/G9HT2M76`

Summary

- rasterize early
- make use of existing methods for raster and vector processing
- 3D rasters, PDAL integration
- the plan for next 30 years driven by users – grass-user mailing list

Get GRASS GIS 7.1 development version at

`grass.osgeo.org/download`