

# GRASS GIS loves lidar

## FOSS4G NA 2016

Vaclav Petras (Vashek)

Anna Petrasova, Helena Mitasova

Center for Geospatial Analytics

**NC STATE UNIVERSITY**

May 5, 2016



available at

[wenzeslaus.github.io/grass-lidar-talks](https://wenzeslaus.github.io/grass-lidar-talks)

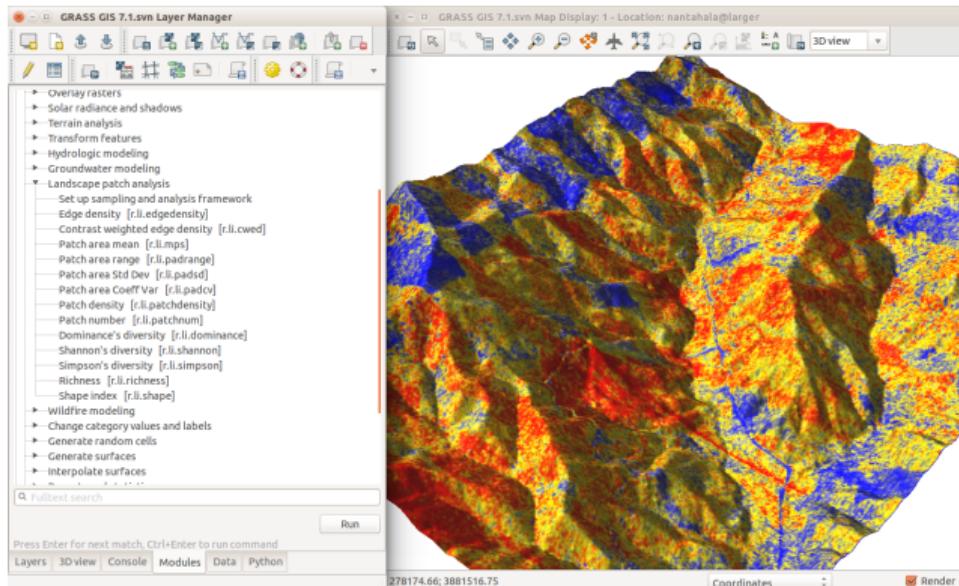
- ▶ longevity
  - ▶ learn now, use forever
  - ▶ over 30 years of development



**GRASS** GIS

# GRASS GIS

- ▶ universal scientific and processing platform
  - ▶ GUI, CLI, Python API
  - ▶ from small laptops to supercomputers
- ▶ lidar processing included
- ▶ data size and type challenges





Welcome to GRASS GIS 7.1.svn (r68305M)

GRASS GIS homepage:

<http://grass.osgeo.org>

This version running through:

Bash Shell (/bin/bash)

Help is available with the command:

g.manual -i

See the licence terms with:

g.version -c

Start the GUI with:

g.gui wxpython

When ready to quit enter:

exit

To run a command as administrator (user "root"), use "sudo <command>".

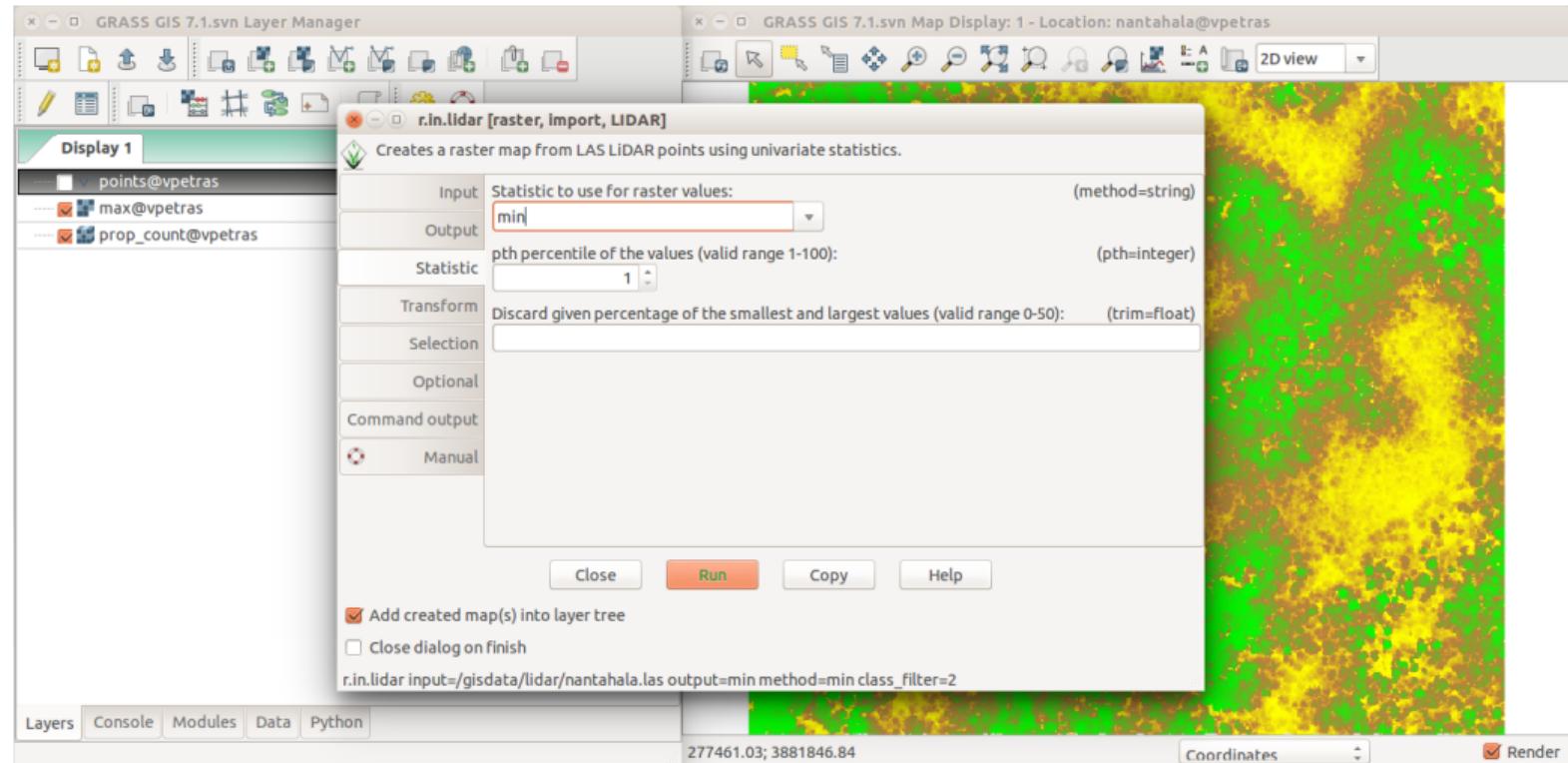
See "man sudo\_root" for details.

GRASS 7.1.svn (nantahala):~/dev/grass/gcc\_trunk > g.region vector=points

GRASS 7.1.svn (nantahala):~/dev/grass/gcc\_trunk > g.region res=5

GRASS 7.1.svn (nantahala):~/dev/grass/gcc\_trunk > r.in.lidar input=/gisdata/lidar/points.las output=mean

# GUI



# Python

The figure shows the GRASS GIS 7.1.svn interface. On the left, the Layer Manager displays three vector layers: 'slope' (grey), 'aspen' (grey), and 'eleva' (blue). The Map Display window on the right shows a 3D perspective view of a hillside, colored in yellow with some red and lime green points representing lidar data. Below the map are controls for 'Coordinates' and a checked 'Render' button. In the center, the GRASS GIS Simple Python Editor window is open, displaying the following Python script:

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

import os
import grass.script as gscript

def main():
    region = gscript.region()

    vectors = []
    for lidar_file in os.listdir('.'):
        if lidar_file.endswith('_spm.las'):
            bbox = gscript.read_command('r.in.lidar', input=lidar_file,
                                         output='foo', flags='g').strip()
            bbox = gscript.parse_key_val(bbox, vsep=' ', val_type=float)
            if (bbox['n'] < region['s'] or bbox['s'] > region['n']
                or bbox['e'] < region['w'] or bbox['w'] > region['e']):
                gscript.info('Skipping tile %s' % lidar_file)
                continue
            name = 'tile_' + lidar_file.rsplit('.', 1)[0]
            vectors.append(name)
            gscript.run_command('v.in.lidar', input=lidar_file, output=name,
                                flags='rt', class_filter=2)
            gscript.run_command('v.patch', input=vectors, output='merged_points',
                                flags='b', overwrite=True)
            gscript.run_command('g.remove', type='vector', name=vectors, flags='f')

if __name__ == "__main__":
    main()
```

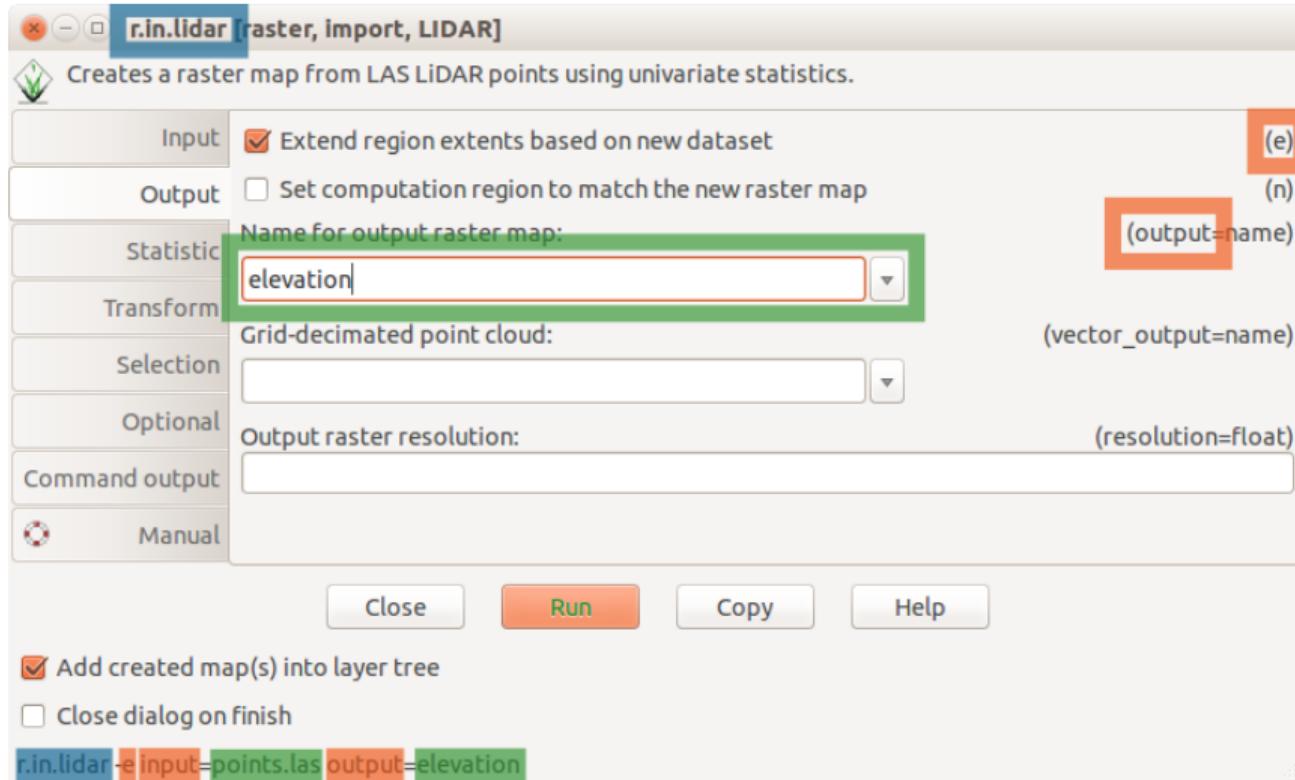
Documentation, Command Line, Shell, Bash:

```
r.in.lidar input=points.las output=elevation -e
```

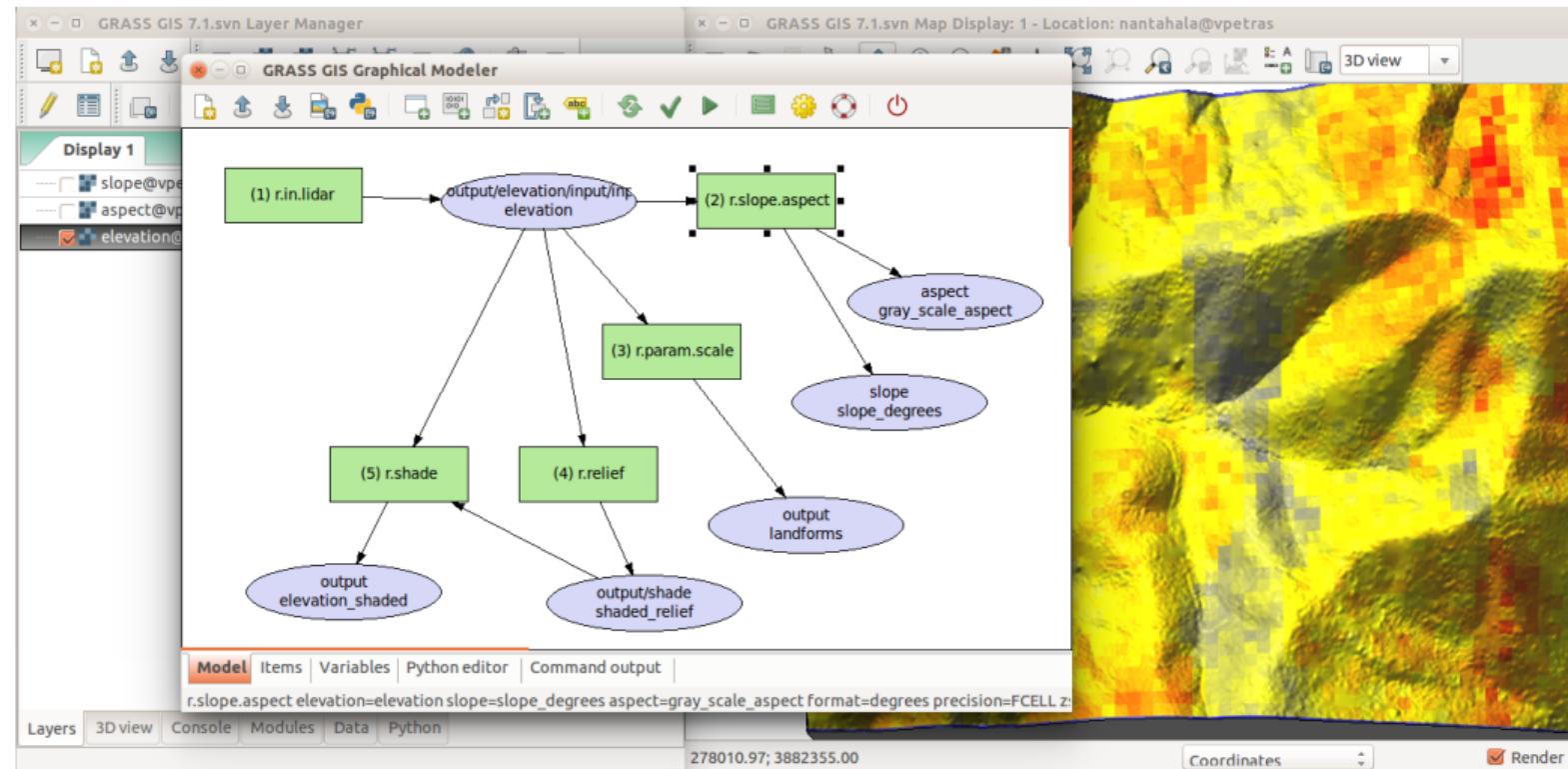
Python:

```
from grass.script import run_command  
run_command('r.in.lidar', input="points.las",  
            output="elevation", flags='e')
```

# Module GUI



# Graphical Modeler



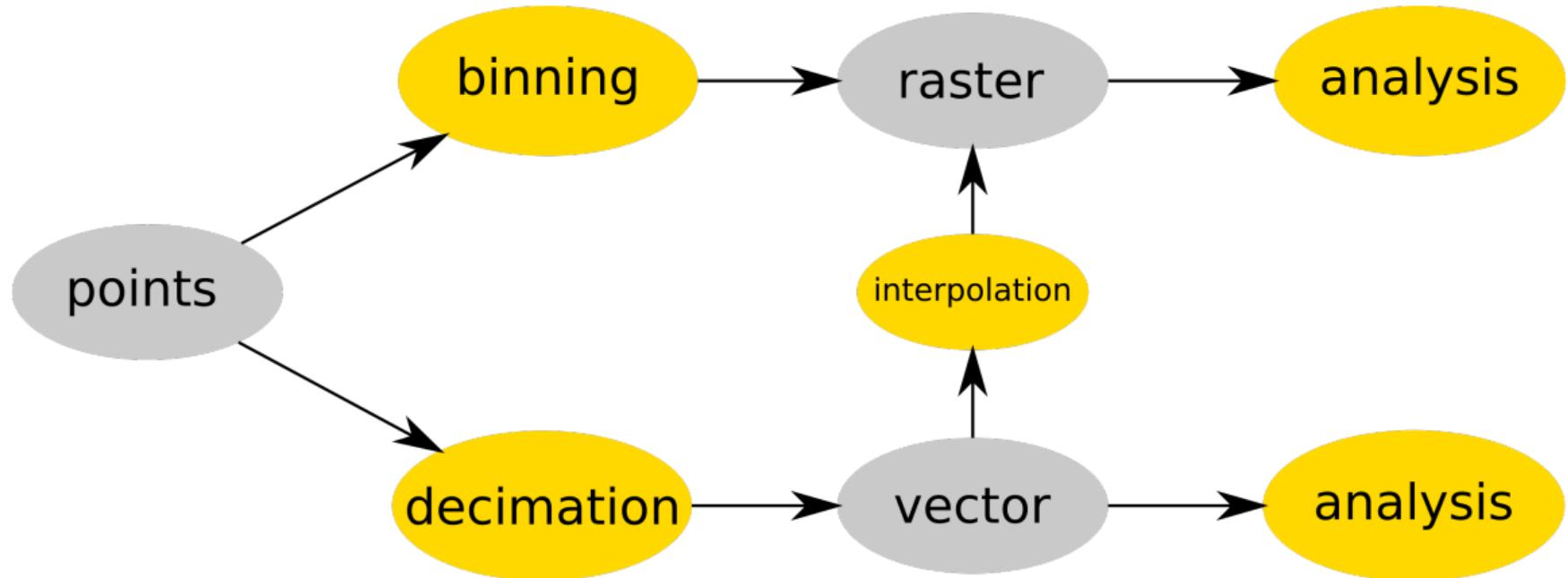
# Points

- ▶ collected by lidar
- ▶ generated by Structure from Motion (SfM) from UAV imagery
- ▶ a lot of points



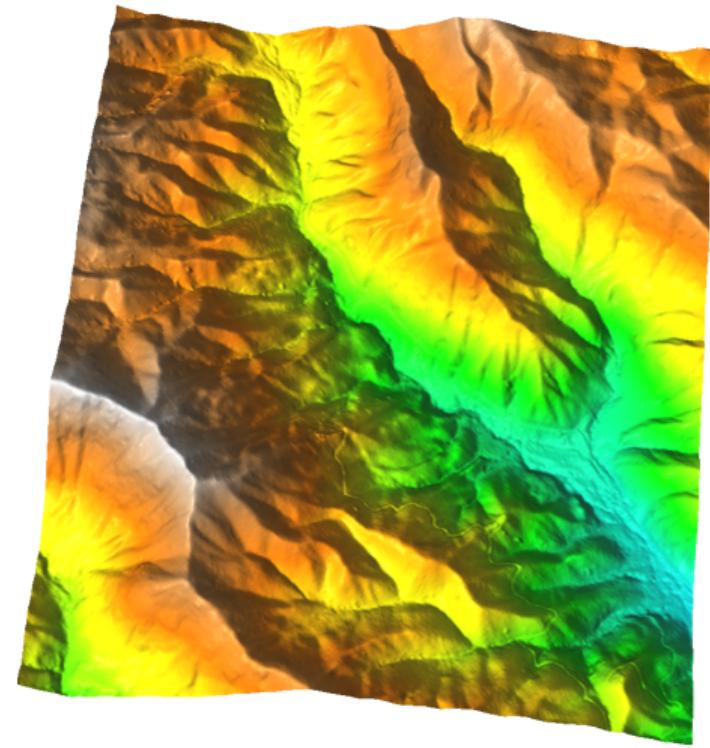
surface interpolated from points and visualized in GRASS GIS

# Workflow overview



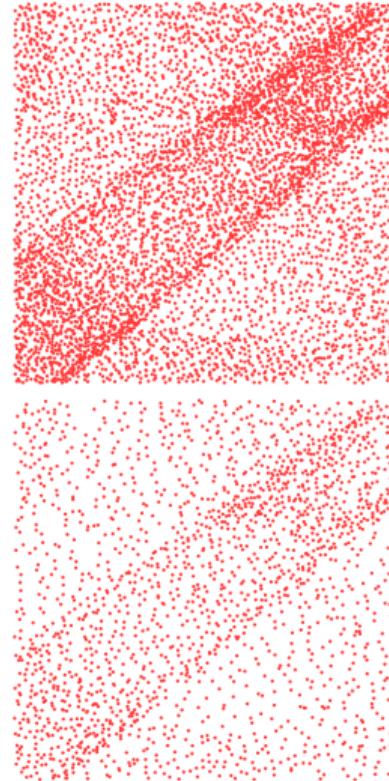
# Surface interpolation

- ▶ *v.surf.idw*
  - ▶ Inverse Distance squared Weighting
- ▶ *v.surf.bspline*
  - ▶ Bicubic or bilinear Spline interpolation with Tykhonov regularization
- ▶ *v.surf.rst*
  - ▶ Regularized Spline with Tension



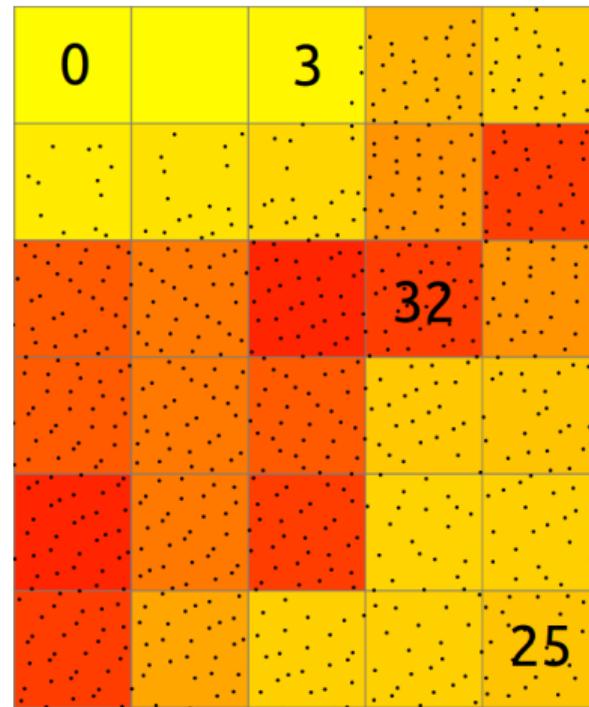
# Import and decimation

- ▶ *v.in.lidar*
  - ▶ libLAS
  - ▶ LAS/LAZ to GRASS GIS native vector
- ▶ decimation  $\approx$  thinning  $\approx$  sampling
  - ▶ count-based decimation (skips points)
  - ▶ grid-based experimental, others needed?
  - ▶ fast count-based as good as more advanced decimations



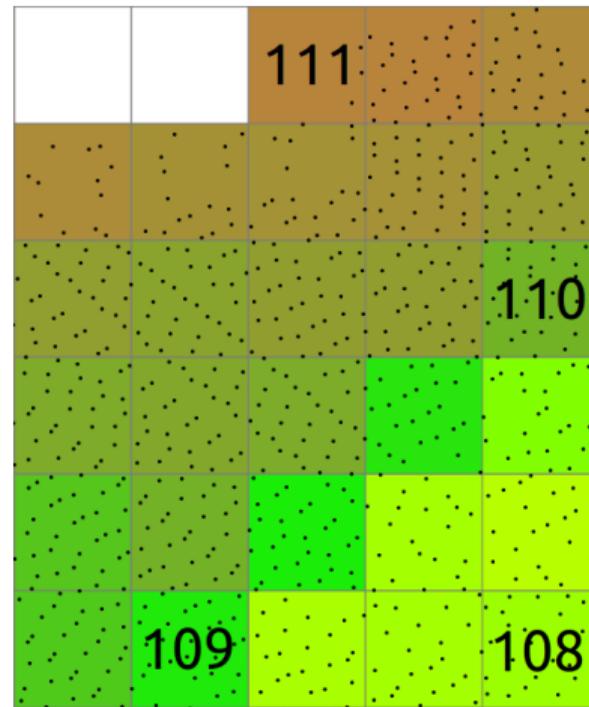
# Binning as 2D histogram

- ▶ counts number of points in cell



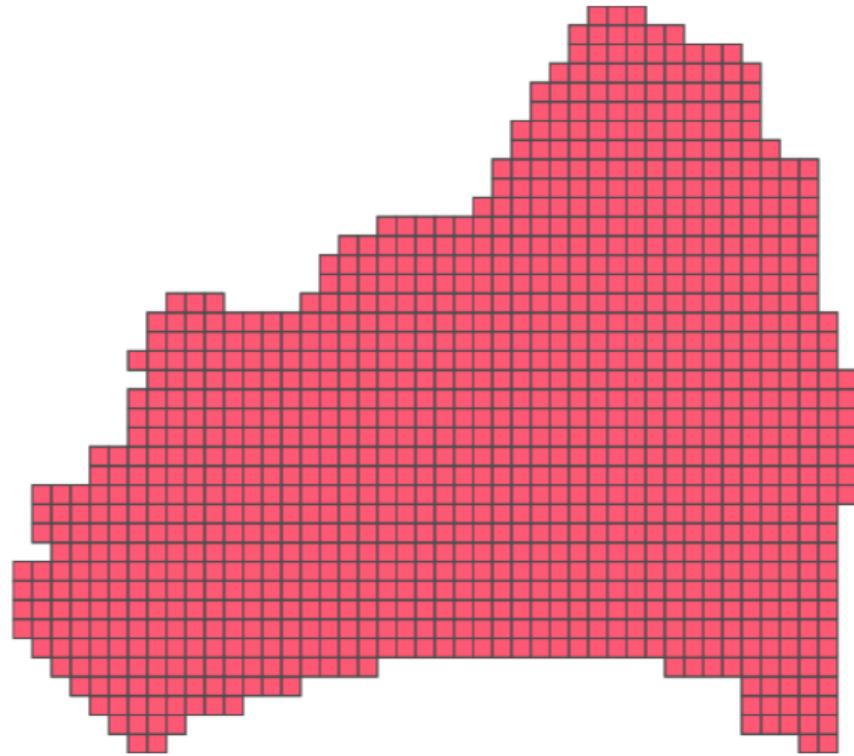
# Binning points to raster

- ▶ *r.in.lidar* (import and analysis)
- ▶ statistics of point counts, height and intensity
  - ▶ n, min, max, sum
  - ▶ mean, range, skewness, ...



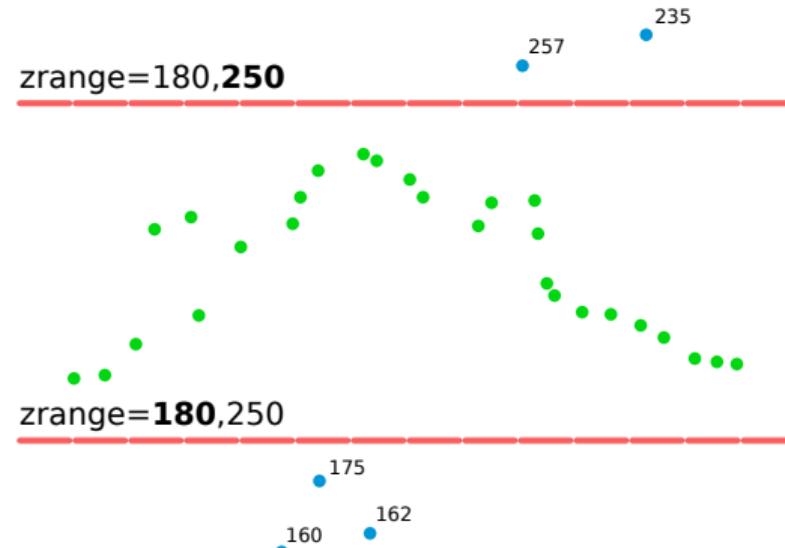
# Practical functions

- ▶ analytical and practical functions in *r.in.lidar*
- ▶ read multiple tiles as one
  - ▶ no merging
  - ▶ 0.5 billion points in 90 files in minutes



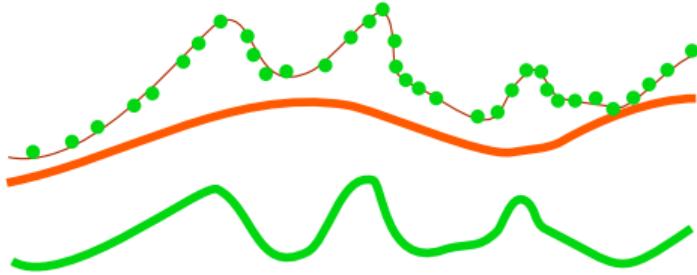
# Filtering points

- ▶ filter points by
  - ▶ range of Z
  - ▶ return
  - ▶ class
  - ▶ ...
- ▶ at the time of binning with *r.in.lidar*
  - ▶ minimal additional cost

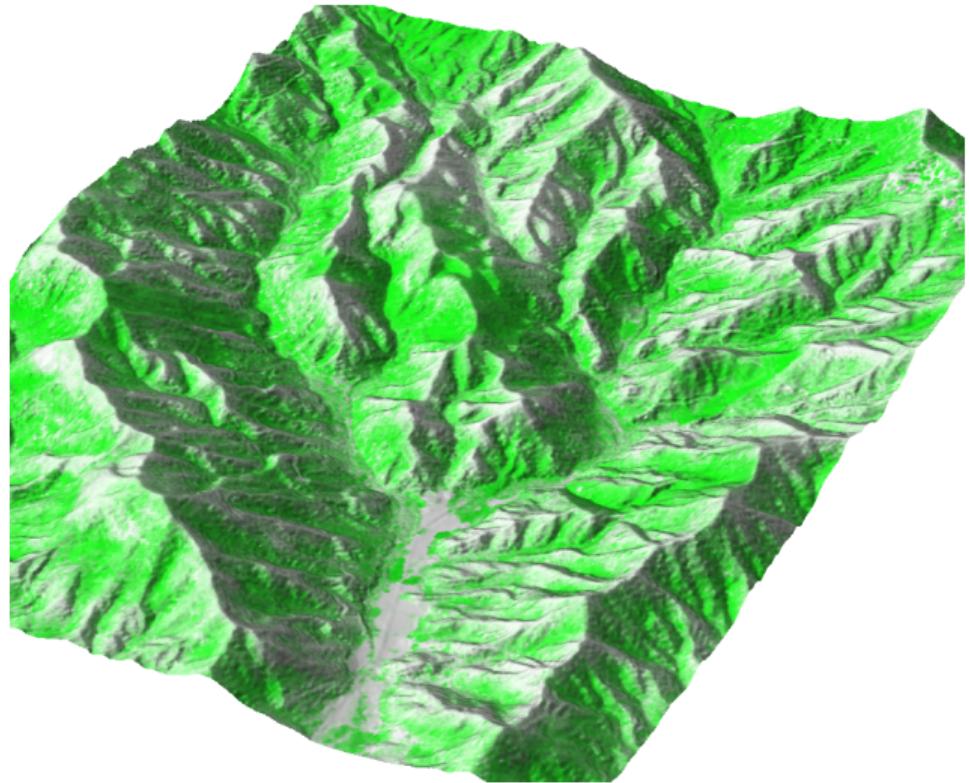


# Height above a surface

- ▶ new base raster feature in  
*r.in.lidar*
- ▶ given surface + points cloud  
→ height of features

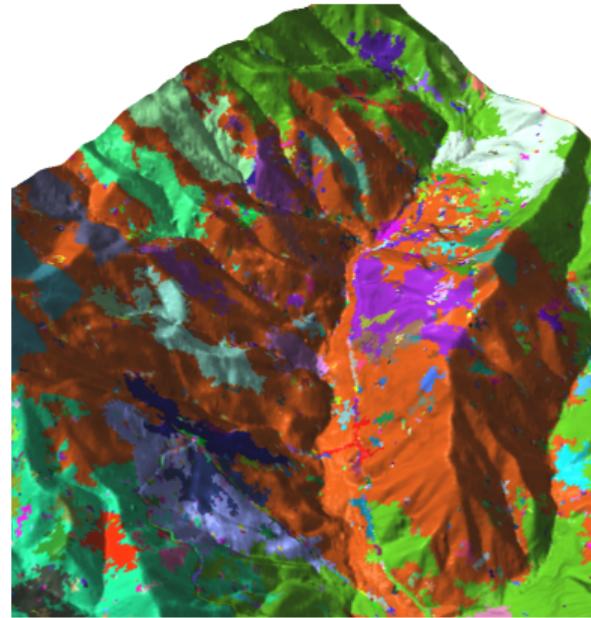


- ▶ not limited by memory



# Rasterize early

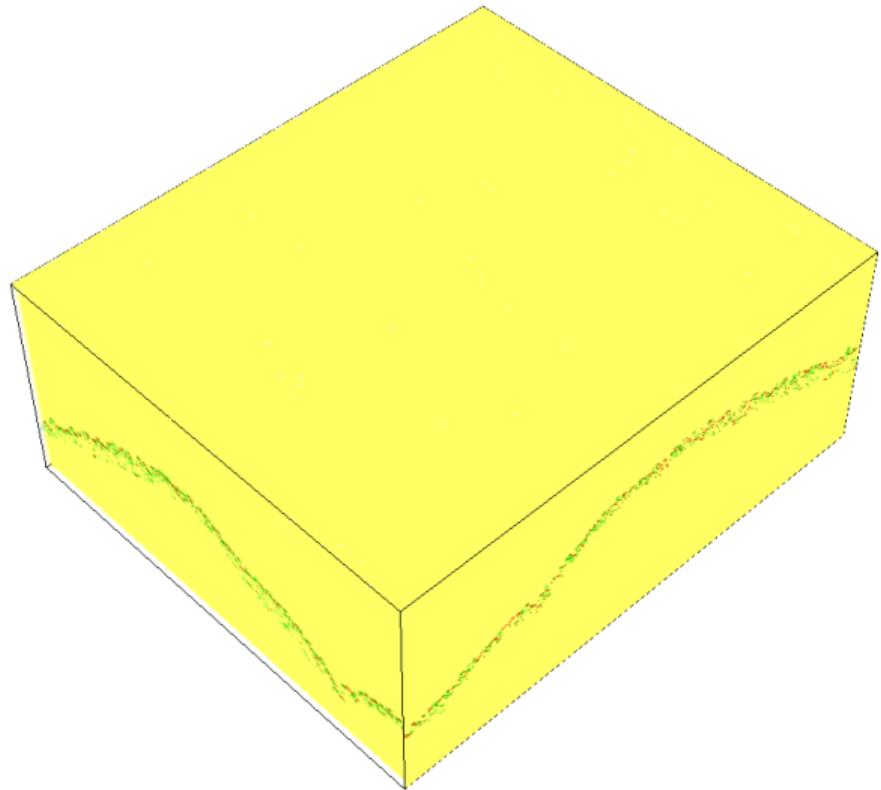
- ▶ less cells than points
  - ▶ 578 mil points (ground 30 mil)
  - ▶ 15 mil cells in 8km × 7km at resolution 2m
    - ▶ faster to loop through
    - ▶ less disk space
- ▶ raster
  - ▶ natural spatial index
  - ▶ that's what the algorithms use



*i.segment on different point counts*

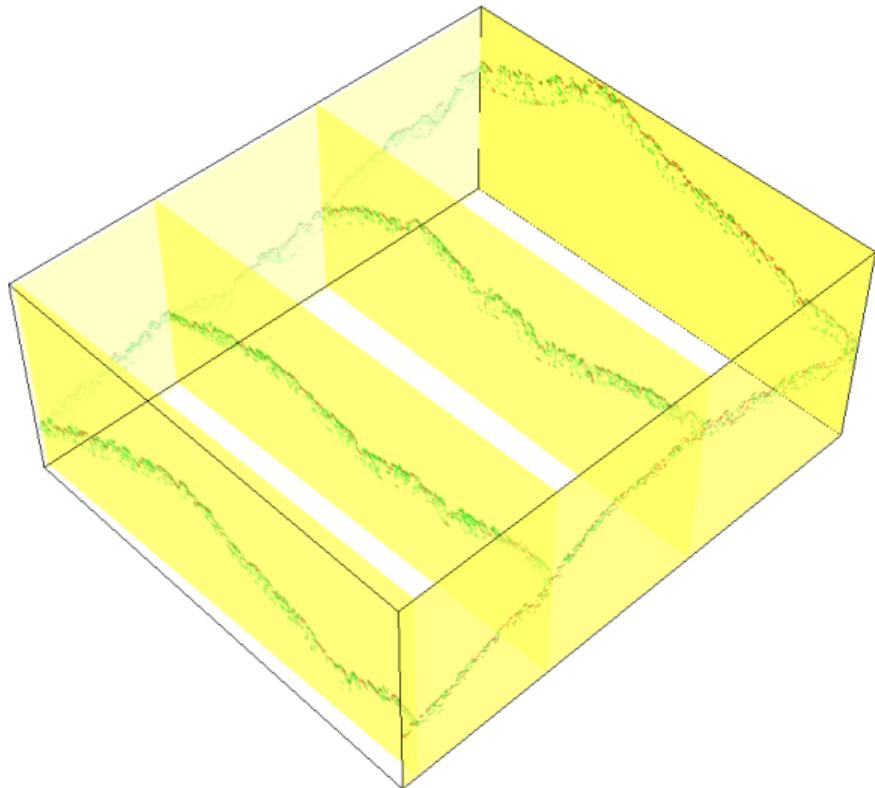
## 3D raster

- ▶ stacked 2D rasters
- ▶ challenging to visualize
- ▶ same principles as in 2D
  - ▶ e.g. 3D raster map algebra



# 3D raster

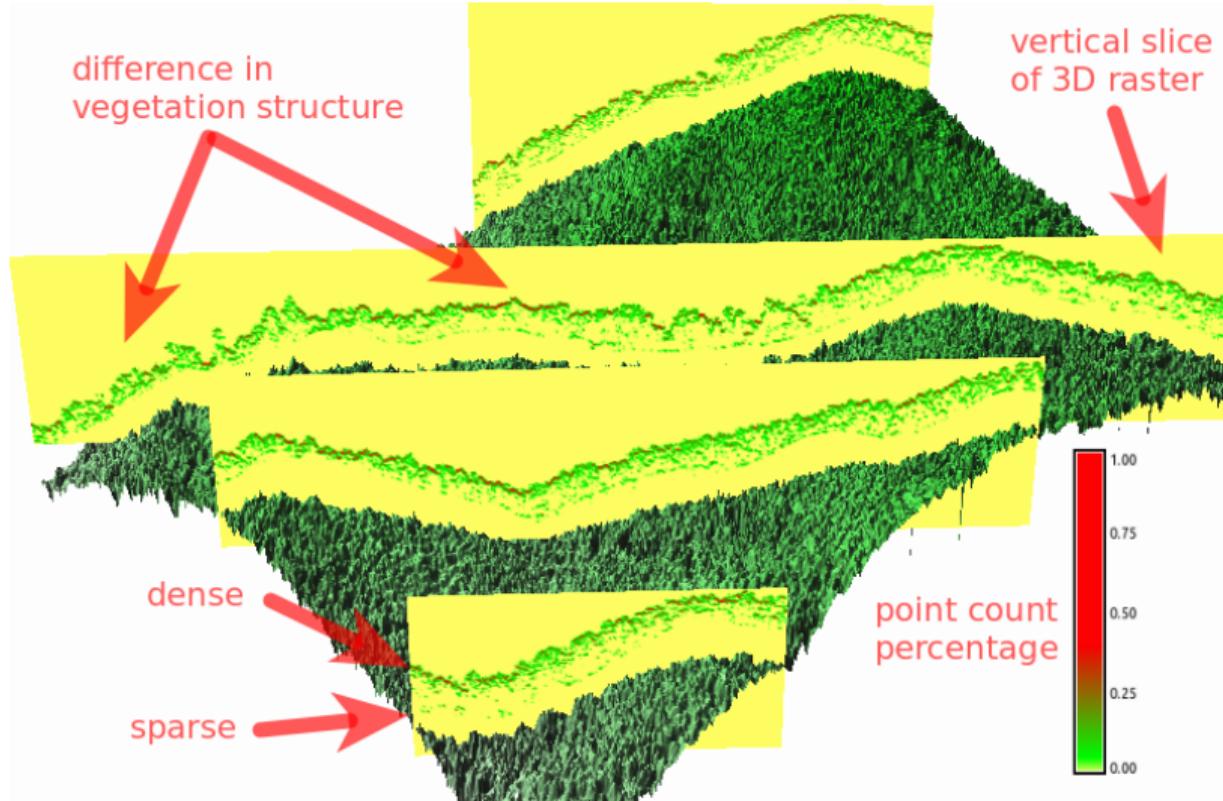
- ▶ stacked 2D rasters
- ▶ challenging to visualize
- ▶ same principles as in 2D
  - ▶ e.g. 3D raster map algebra



# Binning points to 3D raster

- ▶ *r3.in.lidar*
- ▶ proportional count
  - ▶ count per 3D cell relative to the count per vertical column
- ▶ intensity can be used instead of count

height reduction by base raster under development (analysis and space efficient)



# Large point clouds

## Rasters (binning of points)

- ▶ trade-off: memory (RAM) or slow
- ▶ 64bit version
  - ▶ your operating system may limit max memory

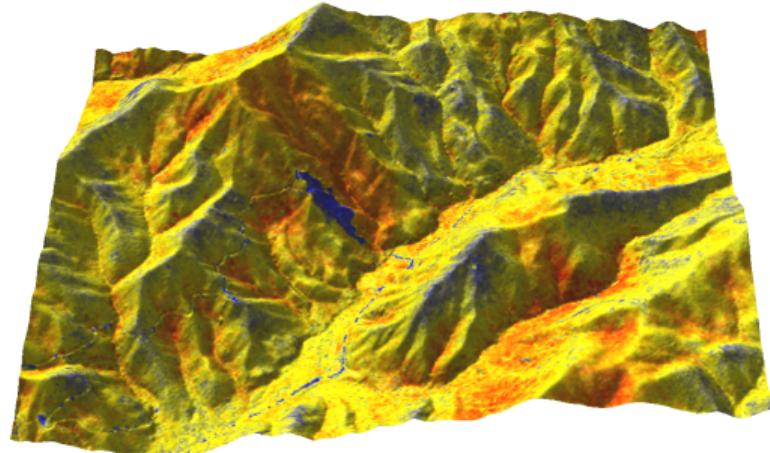
Brunswick county: binning,  $\approx 1050$  files,  
 $> 9$  billion points

Hyde county: binning,  $\approx 950$  files,  $> 4$  billion  
points, base elevation 5ft raster, 60ft height  
raster

$\approx 0.5\text{-}3$  hours, 1-13GB of memory (in-memory mode)

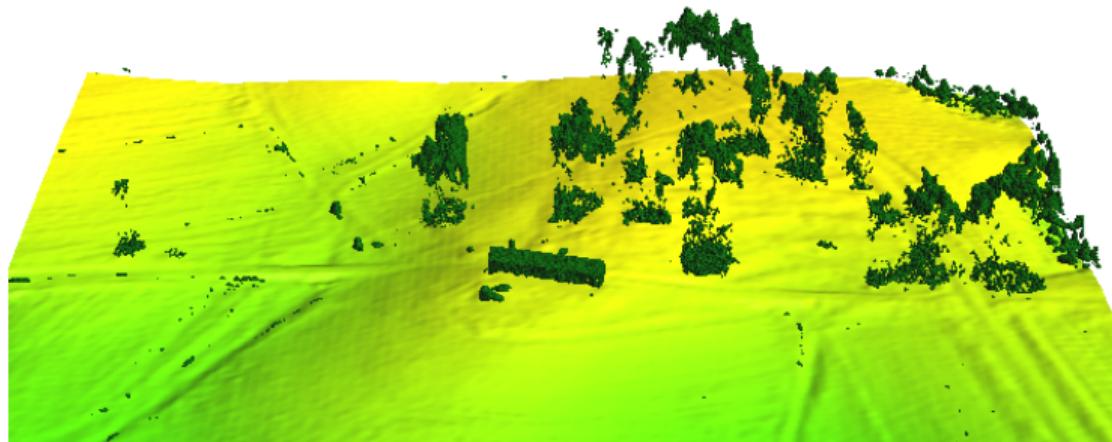
## Vectors (points as points)

- ▶ point cloud specific optimizations
  - ▶ no IDs stored
  - ▶ no attribute table
  - ▶ no topology created



# Ground detection

- ▶ *v.lidar.edgedetection*,  
*v.lidar.growing*,  
*v.lidar.correction*
  - ▶ uses returns
- ▶ *v.lidar.mcc*
  - ▶ multiscale curvature  
based classification  
algorithm<sup>1</sup>



---

<sup>1</sup> Evans, J. S. & Hudak, A. T. 2007: A Multiscale Curvature Algorithm for Classifying Discrete Return LiDAR in Forested Environments.

# Sky-view factor

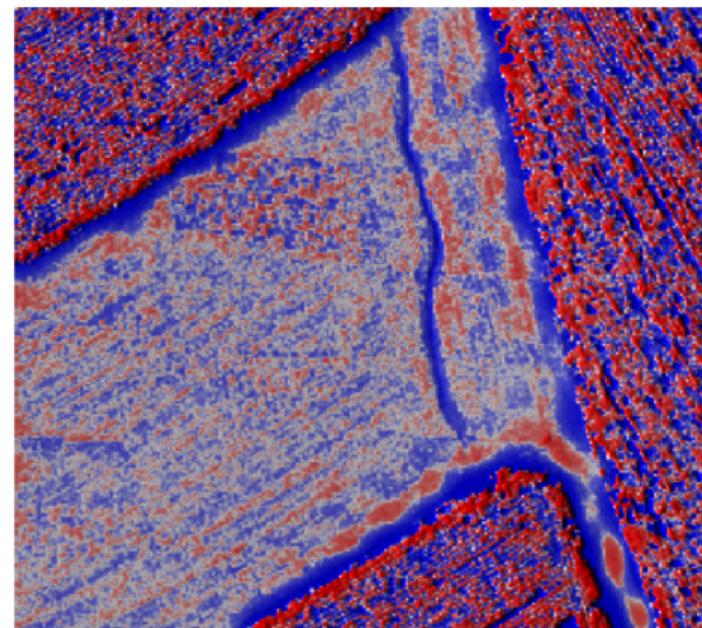
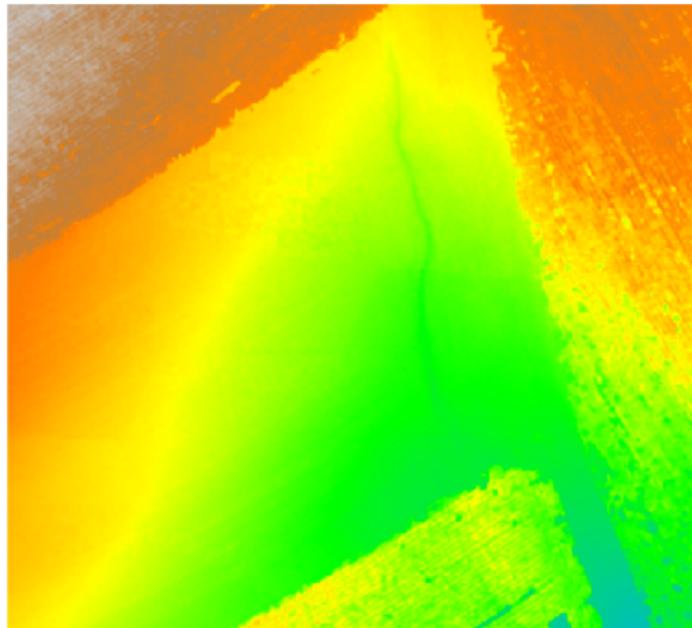
- ▶ *r.skyview* (percentage of visible sky)



comparison of shaded relief and sky-view factor

# Local relief model (LRM)

- ▶ `r.local.relief` (micro-topography, features other than trend)

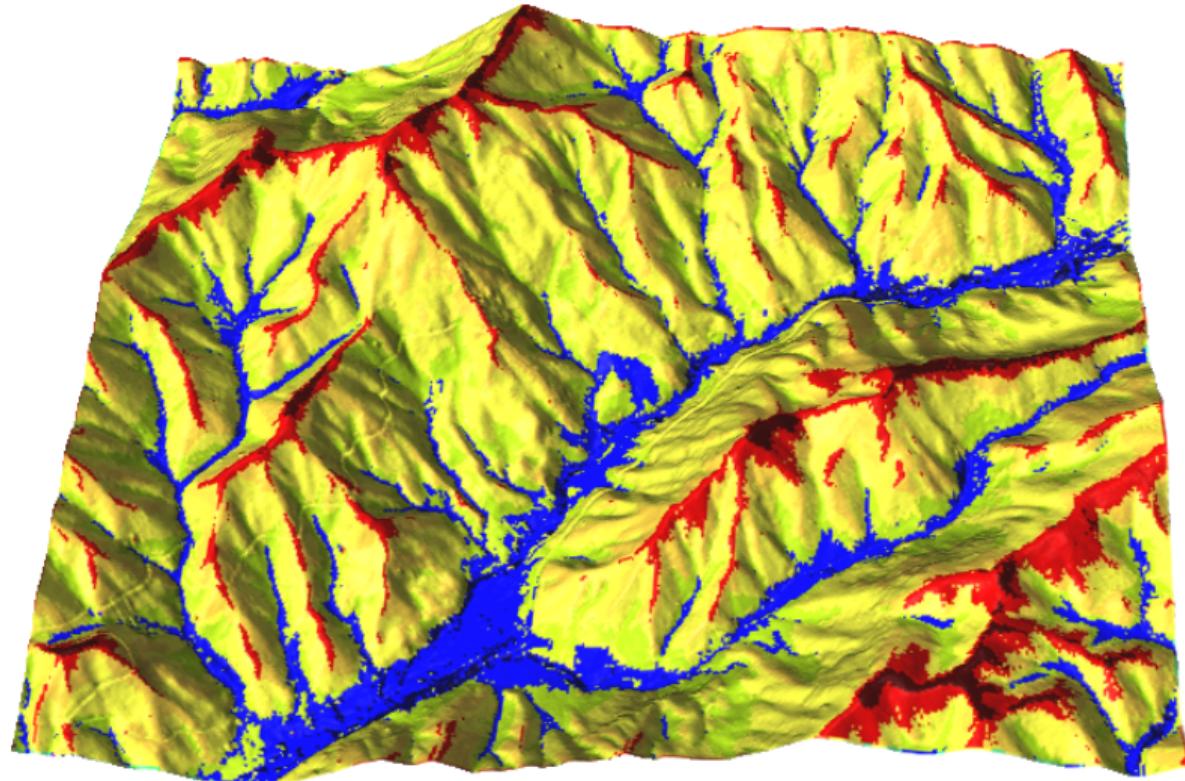


30-60cm wide, 30cm deep, 60m long gully (resolution 30cm)

# Landforms

- ▶ *r.geomorphon*
- ▶ geomorphons - a new approach to classification of landform<sup>1</sup>

<sup>1</sup> Jasiewicz, J., Stepinski, T., 2013, Geomorphons - a pattern recognition approach to classification and mapping of landforms, *Geomorphology*



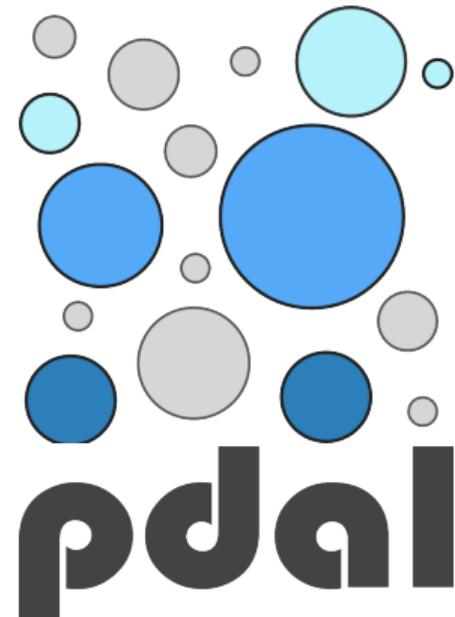
# Integration with PDAL

## PDAL

- ▶ Point Data Abstraction Library
- ▶ formats besides LAS/LAZ
- ▶ algorithms, filters, decimations

## Experimental integration

- ▶ *v.in.pdal*
- ▶ reprojection during import
- ▶ ground filter
- ▶ compute height as a difference from ground

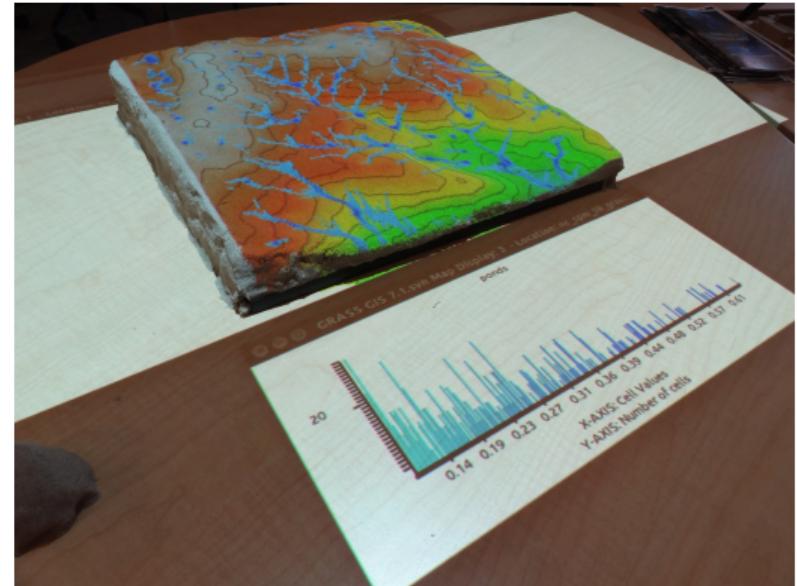


# libfreenect2 + PCL + GRASS GIS = *r.in.kinect*

## *r.in.kinect*

- ▶ scans using Kinect
- ▶ Point Cloud Library (PCL)
- ▶ OpenKinect libfreenect2
- ▶ GRASS GIS
  - ▶ C API
  - ▶ raster processing
  - ▶ regularized spline with tension interpolation

used in Tangible Landscape



## Summary

- ▶ rasterize early
- ▶ make use of existing methods for raster and vector processing
- ▶ 3D rasters, PDAL integration



Get GRASS GIS 7.1 development version at  
[grass.osgeo.org/download](http://grass.osgeo.org/download)

Slides and paper available at  
[wenzeslaus.github.io/grass-lidar-talks](http://wenzeslaus.github.io/grass-lidar-talks)

GRASS user mailing list  
[lists.osgeo.org/listinfo/grass-user](http://lists.osgeo.org/listinfo/grass-user)



# Acknowledgements

## Software

Presented functionality is work done by Vaclav Petras, Markus Metz, and the GRASS development team.

Thanks to users for feedback and testing, especially to Douglas Newcomb, Helena Mitasova, Markus Neteler, Laura Belica, and William Hargrove.



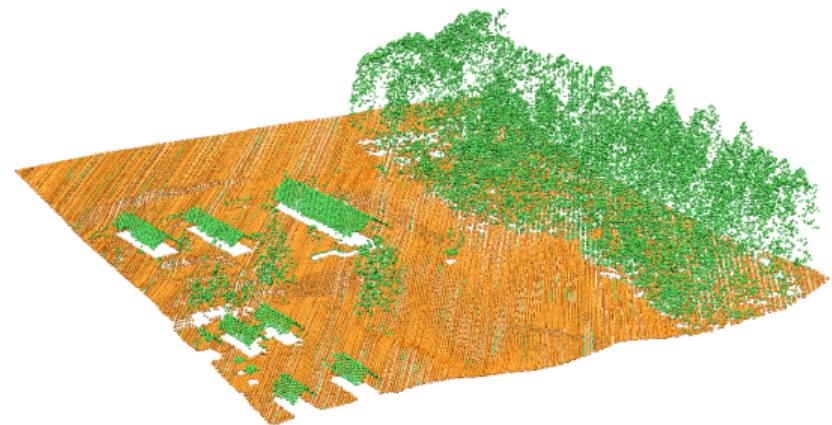
**GRASS** GIS

# Acknowledgements

## Datasets

Lidar and UAV Structure from Motion (SfM) data for  
GIS595/MEA792: UAV/lidar Data Analytics course

Nantahala NF, NC: Forest Leaf Structure, Terrain and  
Hydrophysiology. Obtained from OpenTopography.  
<http://dx.doi.org/10.5069/G9HT2M76>



# Acknowledgements

## Presentation software

Slides were created in L<sup>A</sup>T<sub>E</sub>X using  
the BEAMER *class*.