

# Efficient processing of dense UAV point clouds

Class project presentation

Vaclav Petras (Vashek)

North Carolina State University, Center for Geospatial Analytics

**NC STATE UNIVERSITY**

December 1, 2015

GIS595/MEA792: UAV/lidar data analytics course



## Questions

- ▶ How many points are really necessary to create a detailed DEM?
- ▶ Which method of point decimation preserve more information?

## Implementation

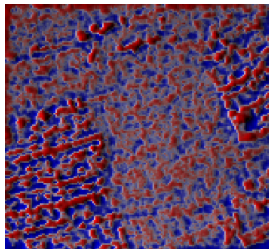
- ▶ Open source implementation for further review and improvement.
- ▶ Methods implemented in GRASS GIS so that they can be used by a broad audience.

# Count-based decimation influence on interpolated elevation



```
g.region nsres=0.3 ewres=0.3 rows=149 cols=161 (cells=23989)  
v.surf.rst ... npmin=120 tension=20 smooth=2 segmax=40
```

# Count-based decimation influence on local relief model



all  
0 %



skip=5  
20 %



preserve=20  
90 %



preserve=100  
99 %

```
r.local.relief input=... output=... shaded_output=... neighborhood=11
```

# Progressiveness of count-based decimation



# Influence of grid-based decimation resolution



resolution=0.1

0 %

resolution=0.3

81 %

resolution=0.9

98 %

resolution=1.5

99 %

# Resolution of grid-based decimation



# 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

# Crop the point cloud by polygon

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



areas



`v.in.lidar mask=`

# 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=`

# 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=`



`v.patch -nz`

# Count-based decimation

*v.in.lidar* – count-based decimation during import



full



preserve/skip



offset



limit

*v.decimate* – point cloud decimation of vector maps (also supports grid-based decimation with preserving point properties)

# Store return and class information as category

*v.in.lidar* can store return or class information as category  
using layers and categories for something else than ID and class



Also: read coordinates only – speed improvement (-c flag)

# Binning of points from multiple LAS files

*r.in.lidar* – read multiple LAS files in one run

The original workflow

```
r.in.lidar input=tile_01.las output=tile_01
r.in.lidar input=tile_02.las output=tile_02
...
r.patch input=tile_01,tile_02,... output=elevation
```

is replaced by

```
r.in.lidar file=tile_list.txt output=elevation
```

where `tile_list.txt` is

```
tile_01.las
tile_02.las
...
```

# Compute height above a given raster during binning

*r.in.lidar* – derive height above ground of features



The resolutions of binning and ground raster can differ, so different statistics can be computed during binning.



# Export vector points from GRASS GIS as LAS

*v.out.lidar* – export 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

## Summary

- ▶ count-based and grid-based decimation perform the same on a *given* point cloud
- ▶ analysis needed for every dataset → need for tool to create a report
- ▶ improvements needed for the project integrated into GRASS GIS

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

