UAV OS

Point Cloud Processing Using GRASS

UAV OS Lab Handout-03

This handout narrates how GRASS GIS can be used for generating DSM from 3D dense point cloud. GRASS GIS modules specially binning techniques is great for generating DSM straightaway from point cloud. Further, GRASS has rich set of tools for statistically analyze point cloud for its quality.

PS Singh
NESAC, 2019
UAV OS

What You Will be Learning

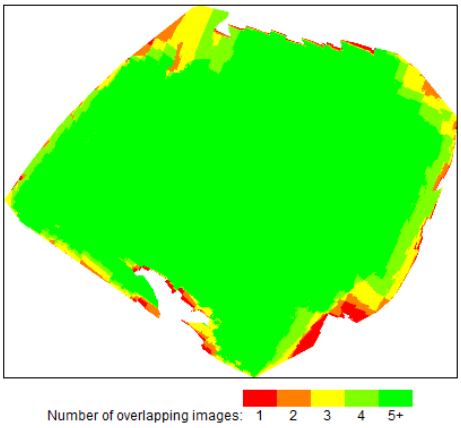
- Learn GRASS GIS Basic operations.
- Understand GRASS GIS analytical tools for Point Cloud Processing.
- Use GRASS GIS for statistically analyse the Point cloud for quality, spatial density, coverages etc
- Use GRASS r.in.lidar and other GRASS libraries for generation of 3D Terrain models from UAV-Based Photogrammetric Point Clouds.

Data To be Used

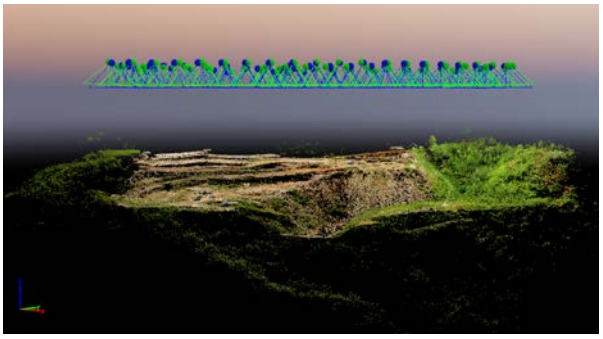
3 sets of Point Clouds in LAS/LAZ format

About Data1: Mixed scene (part of West Khasi Hills) covering a small area of about 9.6 ha. T600 DJI inspire series of UAV that has Zenmuse X3-FC350 camera with focal length of 4 mm and an effective resolution of 12.4 mega pixels giving a maximum image size of 4000x3000. This was flown at a height of 100m to collect about 42 images of the area.

Notice the overlappings between consecutive scenes

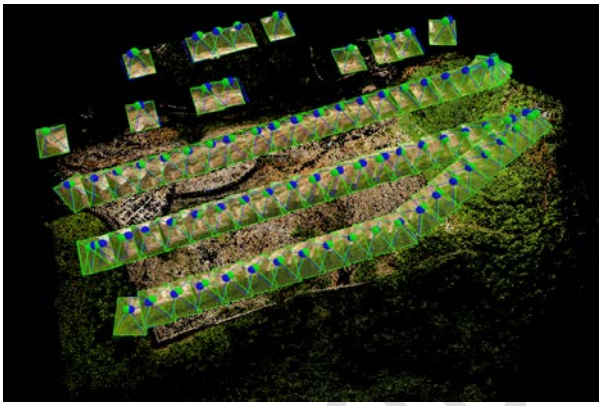


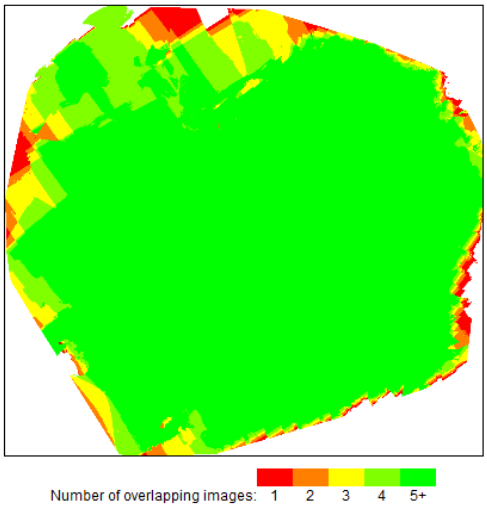
About Data2: Building structure surrounded by vegetation cover (part of East Khasi Hills) covering an area of about 14.98 ha. UAV with Zenmuse X3-FC350 camera with focal length of 4 mm was flown at 100m to collect about 81 images of the area.



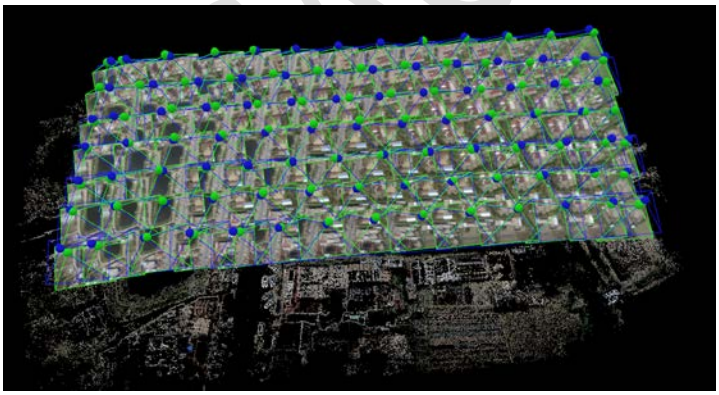
Top Image: Observe the constant flight height set via flight planner. But notice, the variation in terrain elevation. Final GSD/resolution of orthomosaic etc will be calculated based on average GSDs of individual images captures.

Right Images: Notice the flight planning done to carry out the aerial mapping of the area. Also notice, due to gaps or faulty image acquisition on upper left corner of the area, there are less images and overlapping due to which, you'll find gaps/coarse point clouds which may be in-accurate.

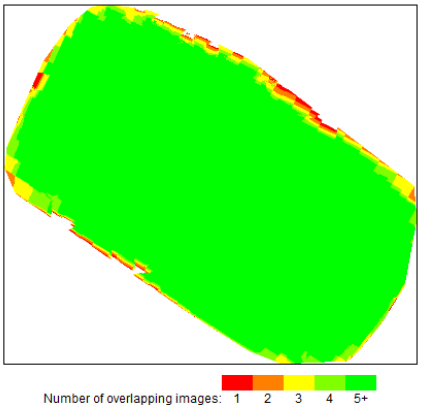




About Data3: Area with settlements and fewer vegetation



Notice the perfect sidelap and endlap for every adjacent scenes.



Note : Commands/module names etc have been highlighted in red colour for easy and quick reference

Learning Goal 1: Understand GRASS basic GUI interface. Defining GRASS GIS Spatial database for data importing and storing. Basic operations for data import and setting of computation region and understand some of its major modules.

About GRASS GIS

The point clouds are increasing in use for number of geospatial and engineering applications. There's challenges in point cloud data analysis due to its size or volume of these datasets. Because of these, the real and immediate practical use, its evaluation and interactive visualization are limited, difficult and/or time consuming. GRASS GIS has powerful modules and libraries to perform access and analysis on large point cloud datasets. We'll see how GRASS GIS can be use to statistically analyse Point cloud for quality, spatial density, coverages etc and most importantly how it can be use to generate useful data products.

Basic introduction to GRASS graphical user interface

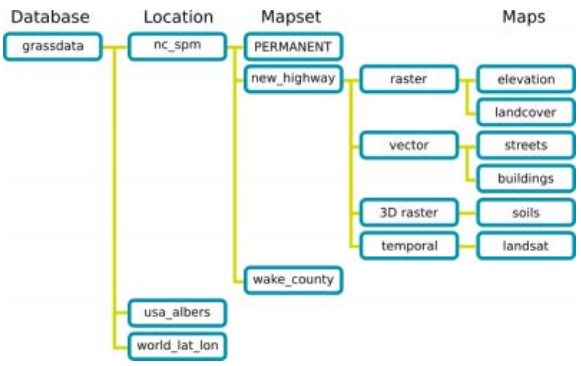
We give a brief overview of GRASS GIS. For our 3D point cloud processing exercise, we need not have full understanding of how to use GRASS GIS. But then, we need vary basic things of how to place our data in the correct GRASS GIS database directory, including some basic GRASS functionality.

GRASS GIS uses specific database structure. First we create a new ***location*** and import the required data into that location. Below briefly list step by step directions on how to download and place your data in the correct place.

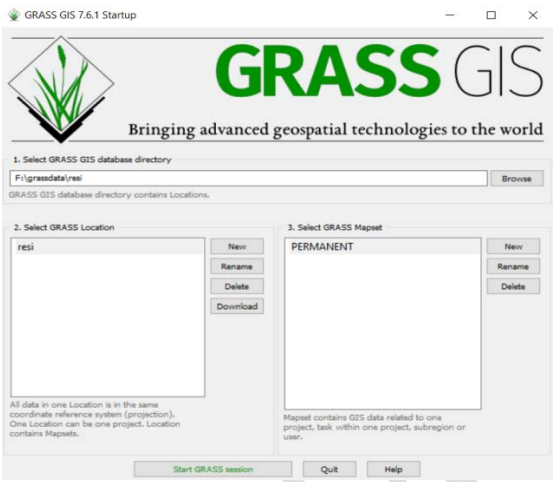
- A GRASS GIS Spatial Database (*GRASS database*) comprises of directory with specific Locations (projects) where data (data layers/maps) are stored.
- Location is a directory with data related to one geographic location or a project. All data within one Location has the same coordinate reference system.
- Mapset is a collection of maps within Location, containing data related to a specific task, user or a smaller project.

Start GRASS GIS, a start-up screen should appear. Unless you already have a directory called grassdata in your Documents directory (on MS Windows) or in your home directory (on Linux), create one. You can use the Browse button and the dialog in the GRASS GIS start up screen to do that.

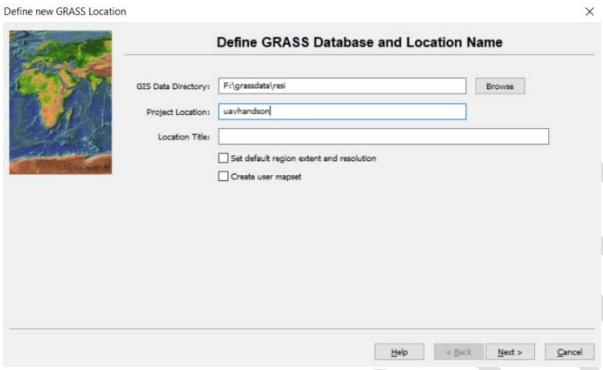
We will create a new *location* for our project with CRS (coordinate reference system) with EPSG code 32646 (for WGS 84, UTM Zone 46). Open Location Wizard with button *New* in the left part of the welcome screen. Select a name for the new location, select EPSG method and code 32646. When the wizard is finished, the new location will be listed on the start-up screen. Select the new location and mapset PERMANENT and press *Start GRASS session.*



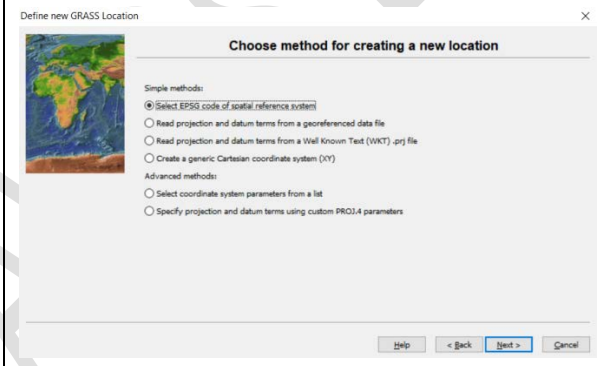
GRASS GIS Spatial Database structure



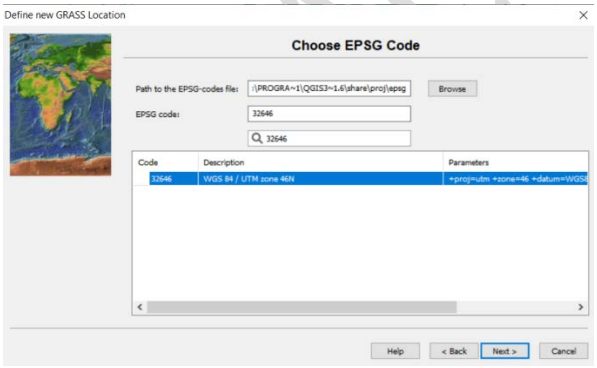
GRASS GIS 7.6.1 startup dialog



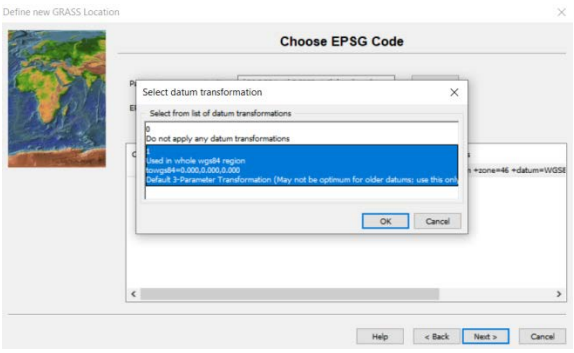
Start Location Wizard and type the new location's name



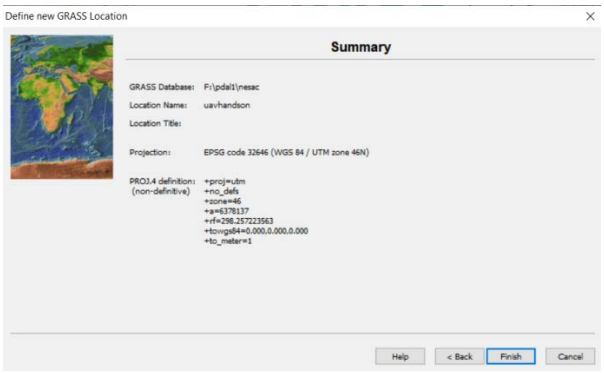
Select method for describing CRS



Find and select EPSG 32646



Confirm the use of the default datum transformation



Review summary page and confirm

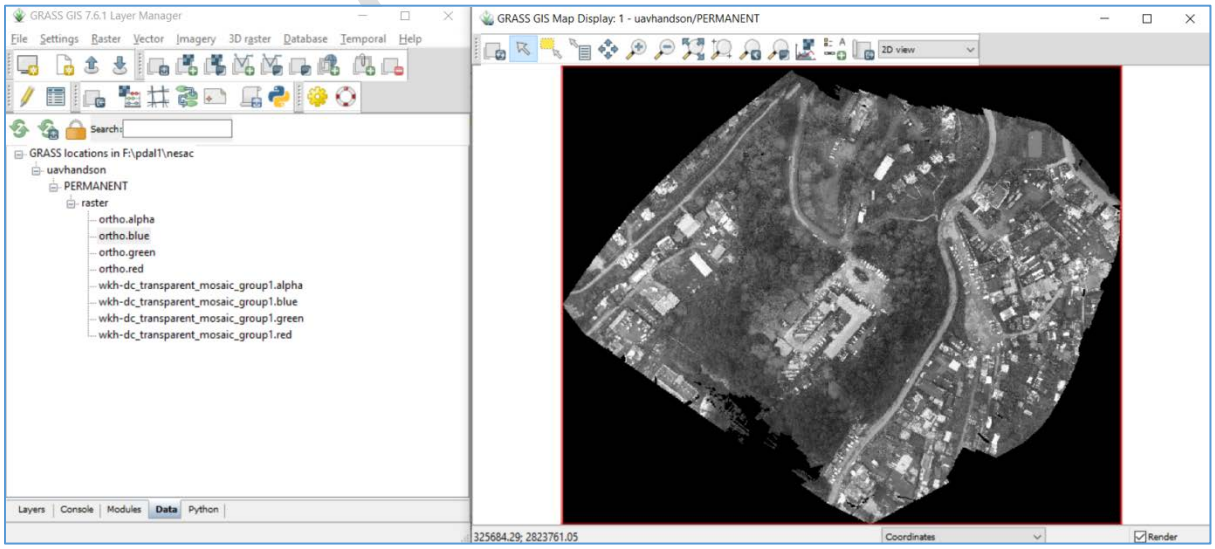
Note: Alternative and easy approach - If you already have a georefrence data (orthophoto etc). It can be used to read projection parameters from directly from this file without the need to specify the EPSG.

The current working directory can be changed from the GUI using *Settings → GRASS working environment → Change working directory* or from the Console using the cd command. This is advantageous when we are using command line and working with the same file again and again. This is often the case when working with lidar data. We can change the directory to the directory with our Point Cloud LAS file. In case we don't change the directory, we need to provide full path to the file.

Importing data

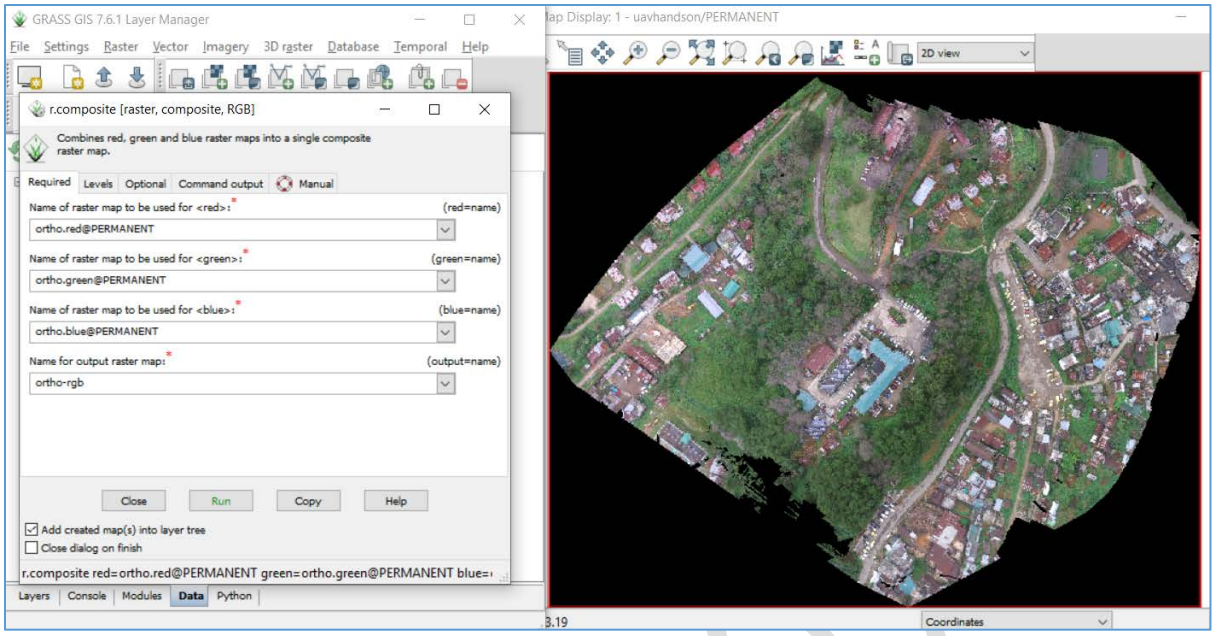
In this step we import the provided data into GRASS GIS. In menu *File - Import raster data* select *Common formats import* and in the dialog browse to find the orthophoto file, change the name to ortho, and click button *Import*. All the imported layers should be added to GUI automatically, if not, add them manually. Point clouds will be imported later in a different way as part of the analysis.

The equivalent command is: r.import input=ortho.tif output=ortho



The map layer with one channel loaded

Use RGB Compose Tool for composing the map for RGB channels and display actual ortho image.



Computational region

Before we use a module to compute a new raster map, we must (optionally) properly set the computational region. All raster computations will be performed in the specified extent and with the given resolution.

Computational region is an important raster concept in GRASS GIS. In GRASS a computational region can be set, subsetting larger extent data *for quicker testing of analysis or analysis of specific regions* based on administrative units. We provide a few points to keep in mind when using the computational region function:

- defined by region extent and raster resolution
- applies to all raster operations
- persists between GRASS sessions, can be different for different mapsets
- advantages: keeps your results consistent, avoid clipping, for computationally demanding tasks set region to smaller extent, check that your result is good and then set the computational region to the entire study area and rerun analysis

run `g.region -p` or in menu Settings - Region - Display region to see current region settings

Computational region can be set using a raster map:

g.region raster=ortho –p (assuming layer named **ortho** has been imported beforehand)

(Optional) Generate KMZ for overlaying on Google Earth

This method can be *used to carry out independent check for the positional accuracy of the products generated* (ortho, DTM etc). First export the raster with r.out.png. Then run gdal_translate to first convert the **png** to **tif** and finally to **kmz**.

Note. : r.out.png *may not* work in all versions of GRASS GIS. Tested on GRASS 6.4 sill

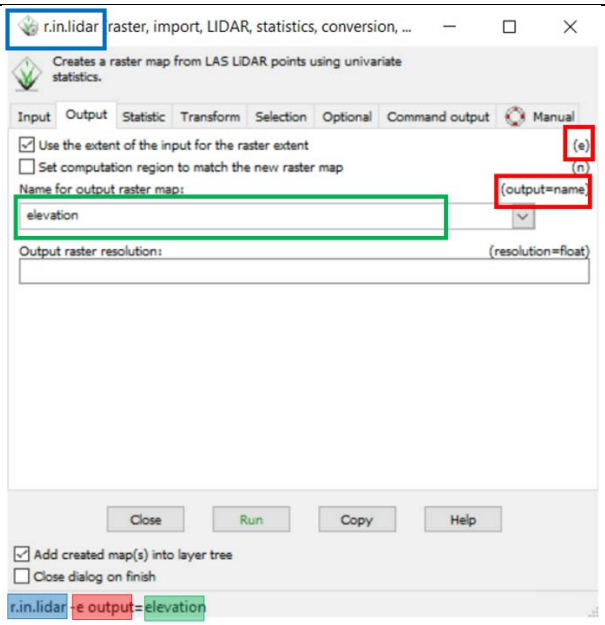
- r.out.png -t -w input=ortho@PERMANENT output=ortho_rgb.png (gives non-georeferenced Portable Network Graphics (PNG) image format)
- gdal_translate -a_srs EPSG:32646 ortho_rgb.png ortho.tif (transform and re-project)
- gdal_translate -of KMLSUPEROVERLAY -co FORMAT=PNG ortho.tif ortho.kmz (transform)

Modules

GRASS GIS functionality is available through modules (which are sometimes called tools, functions, or commands). Modules respect the following naming conventions:

Prefix	Function	Example
r.	raster processing	*r.mapcalc*: map algebra
v.	vector processing	*v.surf.rst*: surface interpolation
i.	imagery processing	*i.segment*: image segmentation
r3.	3D raster processing	*r3.stats*: 3D raster statistics
t.	temporal data processing	*t.rast.aggregate*: temporal aggregation
g.	general data management	*g.remove*: removes maps
d.	display	*d.rast*: display raster map

Above highlighted are the main groups of modules. There is few more for specific purposes. Note also that some modules have multiple dots in their names. This often suggests further grouping. For example, modules staring with ***v.net.*** deal with vector network analysis. The name of the module helps to understand its function, for example ***v.in.lidar*** starts with ***v*** so it deals with vector maps, the name follows with ***in*** which indicates that the module is for importing the data into GRASS GIS Spatial Database and finally ***lidar*** indicates that it deals with lidar point clouds. The same is applicable for UAV based photogrammetry point cloud saved in *.las/.laz* format.



r.in.lidar dialog with Output tab active and highlighted module name (blue), options and flags (red) and option values (green)

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

Example **r.in.lidar** command in Bash with highlighted module name (blue), options and flags (red) and option values (green)

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

Example **r.in.lidar** command in Python with highlighted module name (blue), options and flags (red) and option values (green) and import (grey)

One of the advantages of GRASS GIS is the diversity and number of modules that let you analyzes all manners of spatial and temporal data. GRASS GIS has over 500 different modules in the core distribution and over 300 addon modules that can be used to prepare and analyze data. The following table lists some of the main modules for point cloud analysis.

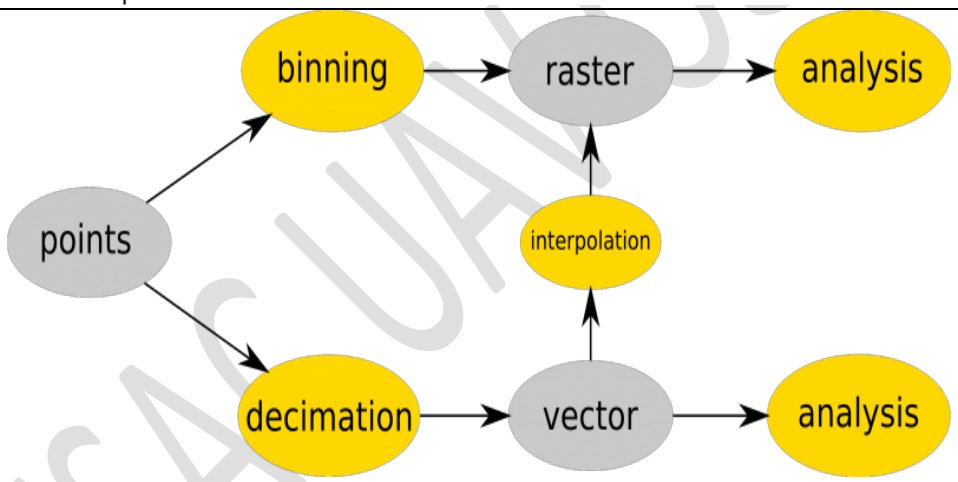
Module	Function	Alternatives
r.in.lidar	binning into 2D raster, statistics	r.in.xyz, v.vect.stats, r.vect.stats
v.in.lidar	import, decimation	v.in.ascii, v.in.ogr, v.import
r3.in.lidar	binning into 3D raster	r3.in.xyz, r.in.lidar
v.out.lidar	export of point cloud	v.out.ascii, r.out.xyz
v.surf.rst	interpolation surfaces from points	v.surf.bspline, v.surf.idw
v.lidar.edgedetection	ground and object (edge) detection	v.lidar.mcc, v.outlier
v.decimate	decimate (thin) a point cloud	v.in.lidar, r.in.lidar
r.slope.aspect	topographic parameters	v.surf.rst, r.param.scale
r.relief	shaded relief computation	r.skyview, r.local.relief
r.colors	raster color table management	r.cpt2grass, r.colors.matplotlib
g.region	resolution and extent management	r.in.lidar, GUI

Learning Goal 2: Understand the concept of *binning* for per-cell processing of point cloud, rasterize the point cloud and generate DSMs.

Binning is fast method for analyzing point clouds and generating DEMs using *Per-cell* processing:

- At least one point for each grid cell
- Analysis : number of points per cell, range, stddev
- Methods for DEM : mean, min, max, nearest
- Sufficient for many applications
- No need to import the points, on-fly raster generation
- May be noisy, include no-data spots

In fact, *the quickest way to analyze basic properties of a point cloud* is to use binning and create a raster map. We will now use r.in.lidar to create point count (point density) raster. At this point, we don't know the spatial extent of the point cloud, so we cannot set computation region ahead, but we can tell the r.in.lidar module to determine the extent first and use it for the output using the -e flag. We are using a coarse resolution to maximize the speed of the process. Additionally, the -r flag sets the computational region to match the output.

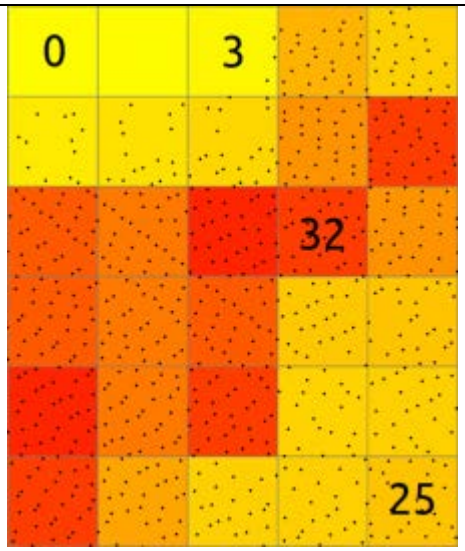


Option1: Point cloud is binned into a raster (e.g. r.in.lidar) and then analyzed as raster

Option2: Point Cloud is decimated (e.g. using v.in.lidar), converted to vector (still using v.in.lidar) and then interpolated (e.g. v.surf.rst) into raster and analyze

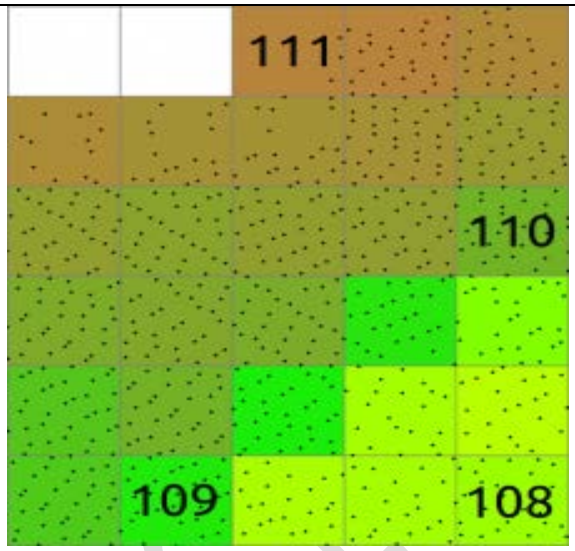
Option3: Point Cloud after decimation is analyzed as a vector (e.g. v.vect.stats).

Note: In the diagram above, Data are in gray, processes in yellow.



Binning for Point Statistics

Here, binning of points into a 2D raster consists of **counting the number of points falling into each cell**. The resulting cell value is then count of points in that cell. This is good way for initiate analysis of Point cloud for Point Cloud Spacings/Density etc

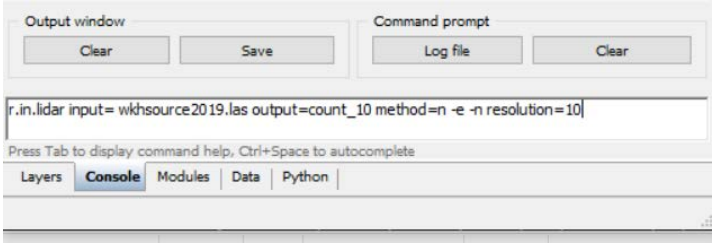


Binning for Raster DSM creation

Here, binning involves also **values associated with the points** and computes statistics on these values. Here the **mean/max/min etc of Z** coordinates of all the point in each cell is computed and stored in the raster. The cells without any points are NULL (NoData) shown in white here.

Lets first perform binning technique check point statistics by run r.in.lidar with method=n with resolution=10. Before running the command, ensure that you are in the directory where data is stored. The r.in.lidar commands can be run both from GRASS console window located within the GUI framework or directly from GRASS command shell.

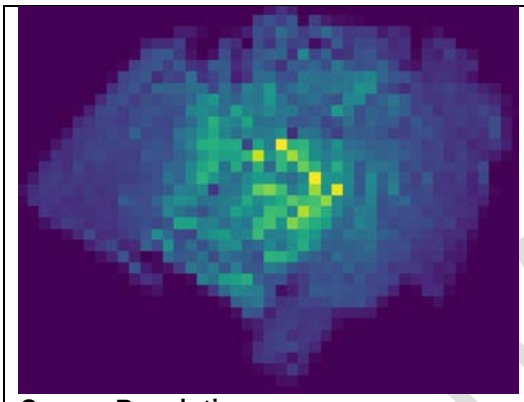
GRASS Command Shell



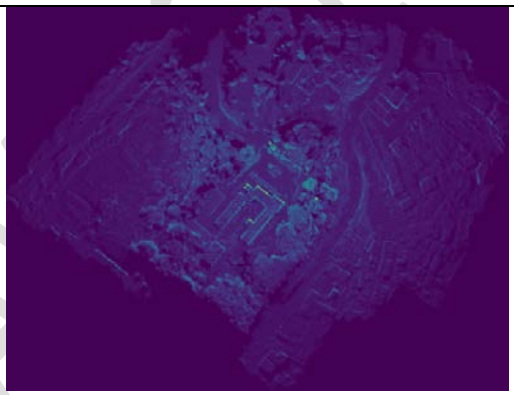
GRASS Console command window

Learning Goal 3: Analyze the output DSMs generated with various binning techniques based on the input point clouds presented.

Lets generate two simple DSMs with binning *based on the number of point clouds* – at different resolutions say 10 and 1. The command r.in.lidar input= input_file_name output=output_file_name method=n -e -n resolution=<> will does the job.

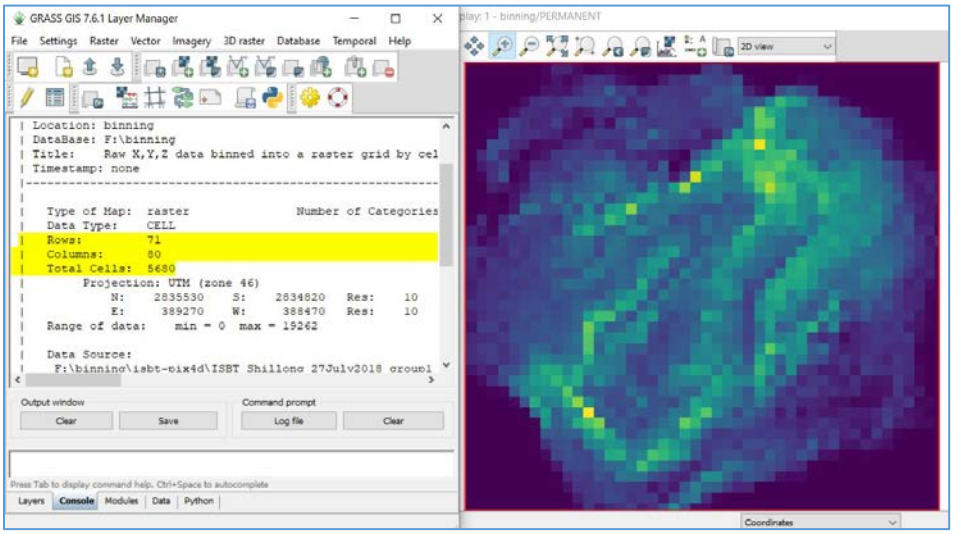


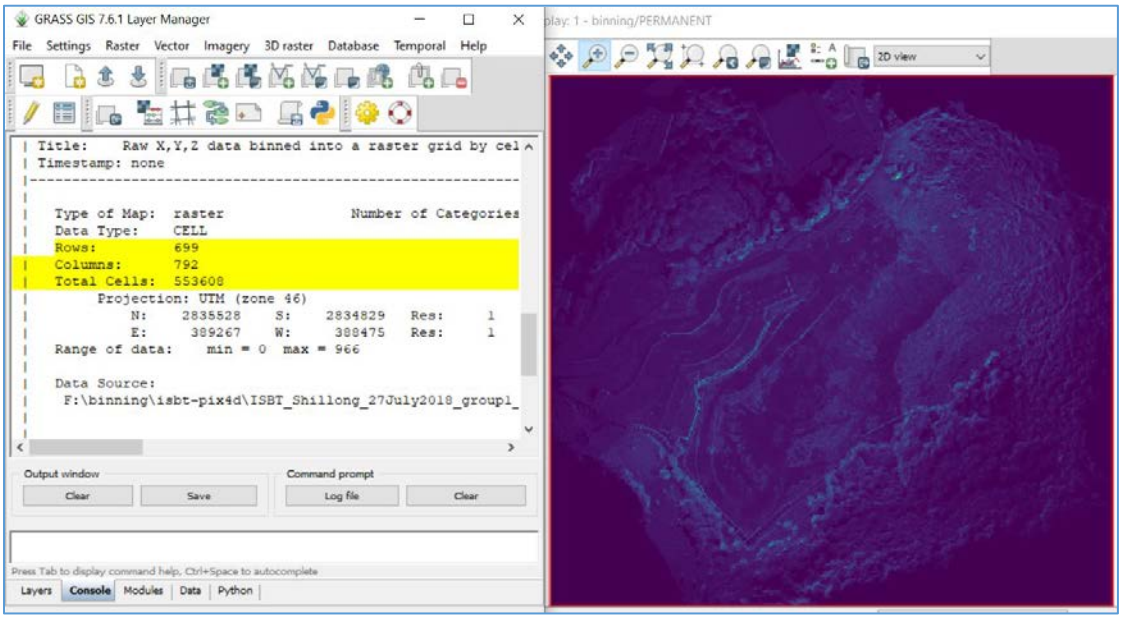
Coarse Resolution
r.in.lidar input= wkhsource2019.las output=count_10 method=n -e -n resolution=10



Finer Resolution
r.in.lidar input= wkhsource2019.las output=count_1 method=n -e -n resolution=1

Quick check on the statistics on both the outputs: By right clicking on the layer and select Metadata should display the metadata of the layer. Alternatively, r.info map=<layer_name> can also be used to see the statistics of the layer. Observe the total numbers of cells (or rows and columns) for each output.

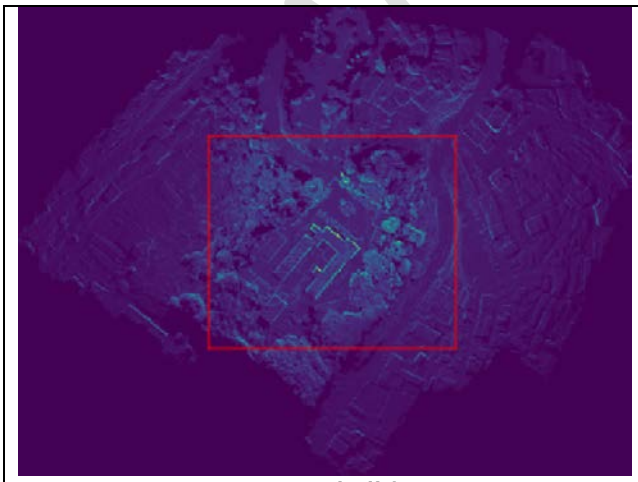




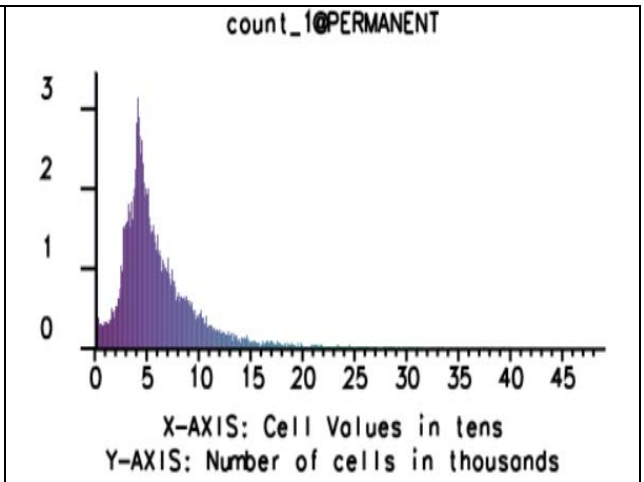
Now we can see the distribution pattern, but let's examine also the numbers (in GUI using right click on the layer in *Layer Manager* and then Metadata or using r.info directly):

r.info map=count_10

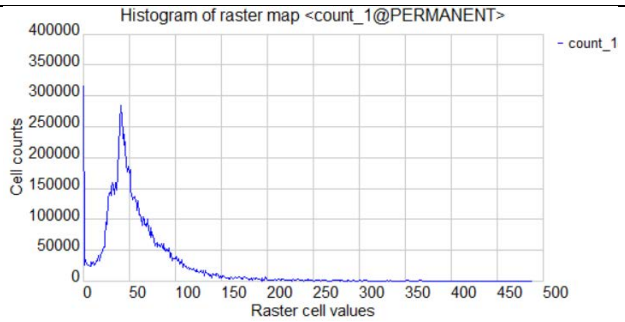
Look at the distribution of the values using a histogram. Histogram is accessible from the context menu of a layer in Layer Manager, from Map Display toolbar Analyze map button or using the d.histogrammodule (d.histogram map=count_1). Select a computational region interactively by drawing a rectangle for which the histogram to be computed.



Point count per cell using r.in.lidar with 1m cells (resolution=1) and -e flag

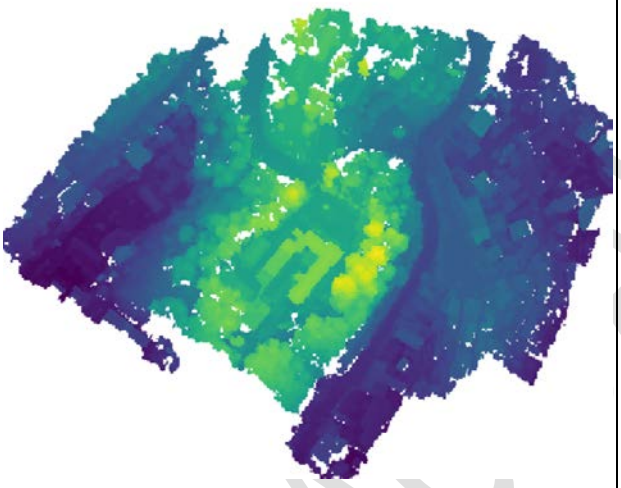


GRASS GIS Histogramming Tool (based on d.histogram)

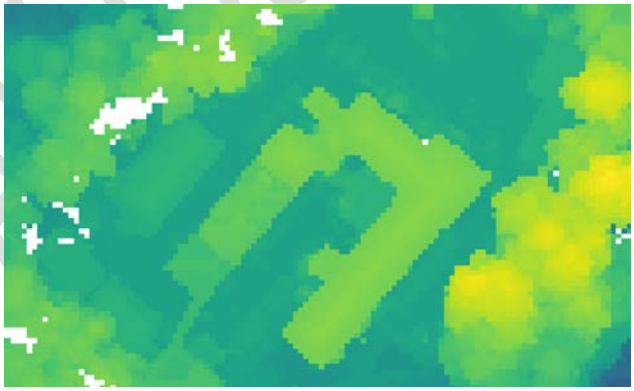


GRASS GIS Histogramming Tool (based on wxPython)

Now we have appropriate number of points in most of the cells and there are no density artifacts around the edges, so we can use this raster as the base for extent and resolution we will use from now on. Use binning to obtain the digital surface model (DSM) (resolution and extent are taken from the computational region):

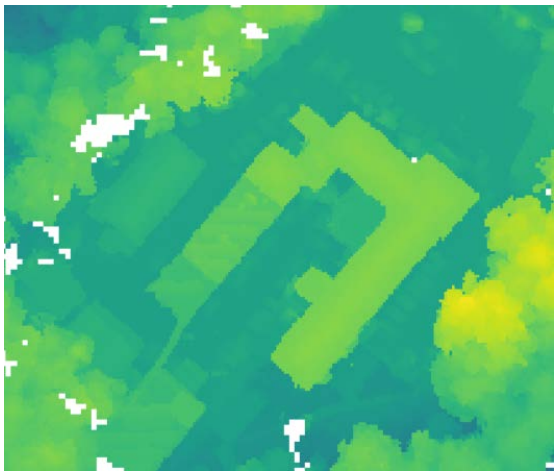


r.in.lidar input=wkhsource2019.las output=binned_dsm_1m method=max resolution=1



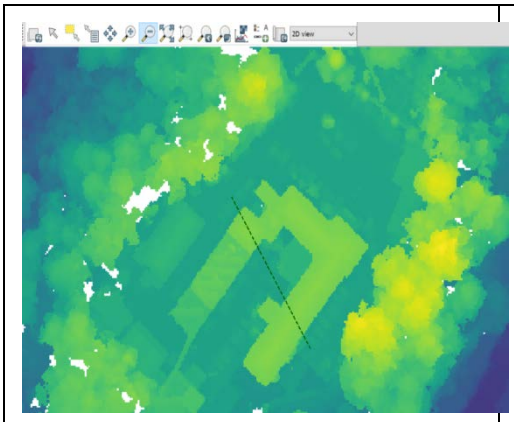
(Enlarged View)

r.in.lidar input=wkhsource2019.las output=binned_dsm_0_5m method=max resolution=0.5

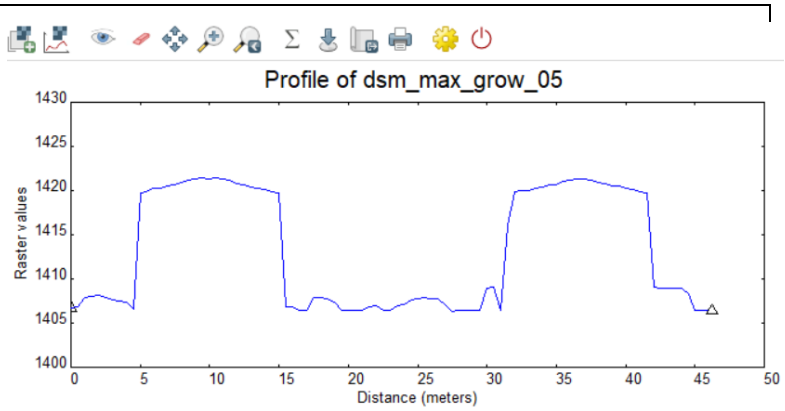


(Enlarged View)

▶ Further, you can run the Profile Analysis on the data output and make sure, its really a surface model. Profile analysis may also be used to analyze two different point clouds or difference in surface profile for two point clouds of the same area generated at different time periods etc. Below shows surface profile of a building structure. Notice the **height and curvy structure of the building** which can be clearly noticed from the plot.



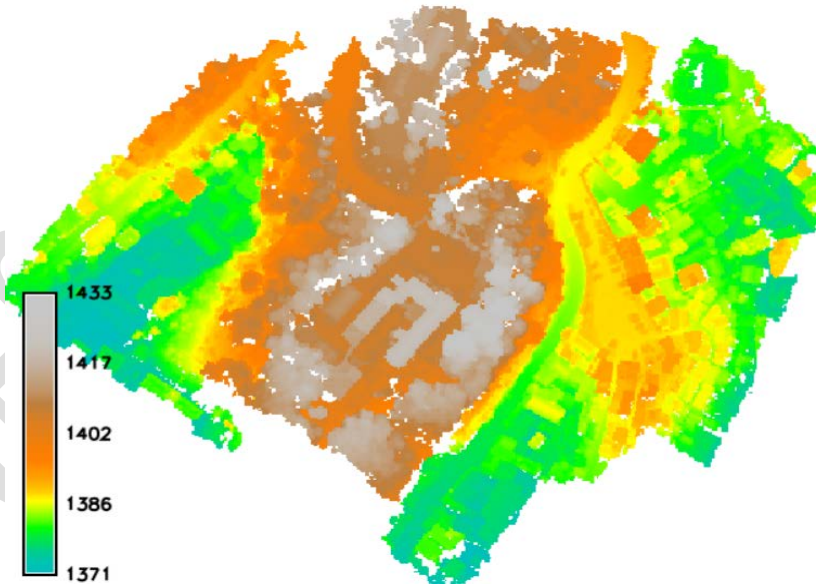
Surface Profile drawn across the building structure



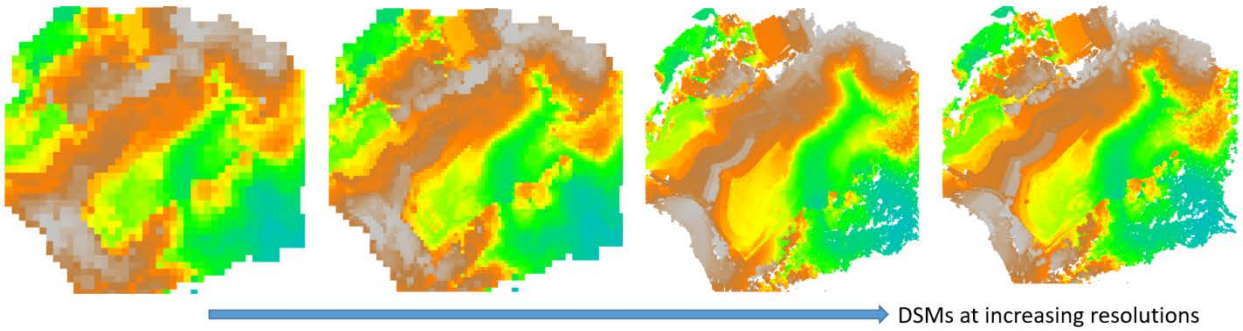
Surface Profile graph. Notice **equal height** and **curvy shape** of the building structure and length of the spreads in meters.

Optionally, Colour the DSM with elevation colour and add the legend.

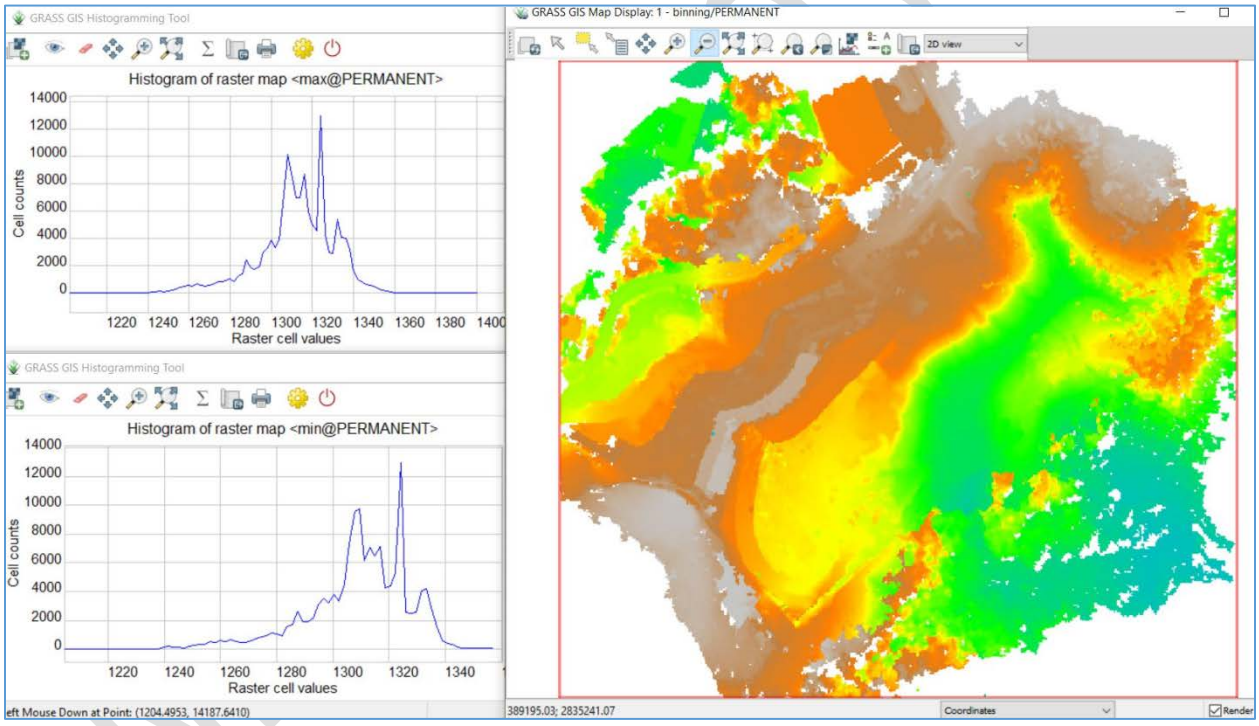
r.colors map=binned_dsm_1m color=elevation -e



The point cloud can be binned at various cell size based on the available density of 3D point cloud. The decision therefore to select optimum cell size is dependent on point cloud density. Below shows the DSMs created at different cell sizes. Notice that, increasing the resolution of cell size will create data gaps. We therefore, needs to set the appropriate cell size based on the density of the point cloud. The gap in the DSM however can be filled using various interpolation techniques such as IDW.



Check possible Outliers by plotting histogram for min & max methods: The histogram can be plotted to check and see the possible outliers in the point clouds. As observed below, the plot drawn both for min,max method shows similar trend signaling the absence of outliers or erroneous points within the point cloud.



Learning Goal 4: Comparisons of binning with *min, max* method performed on Datasets with varying terrain and mix scene (vegetation with building structure). Further, how you can also use other modules such as *r.grow* and *r.relief* to increase the area of DSMs created and generate shaded relief maps for easy comparisons of DSMs outputs.

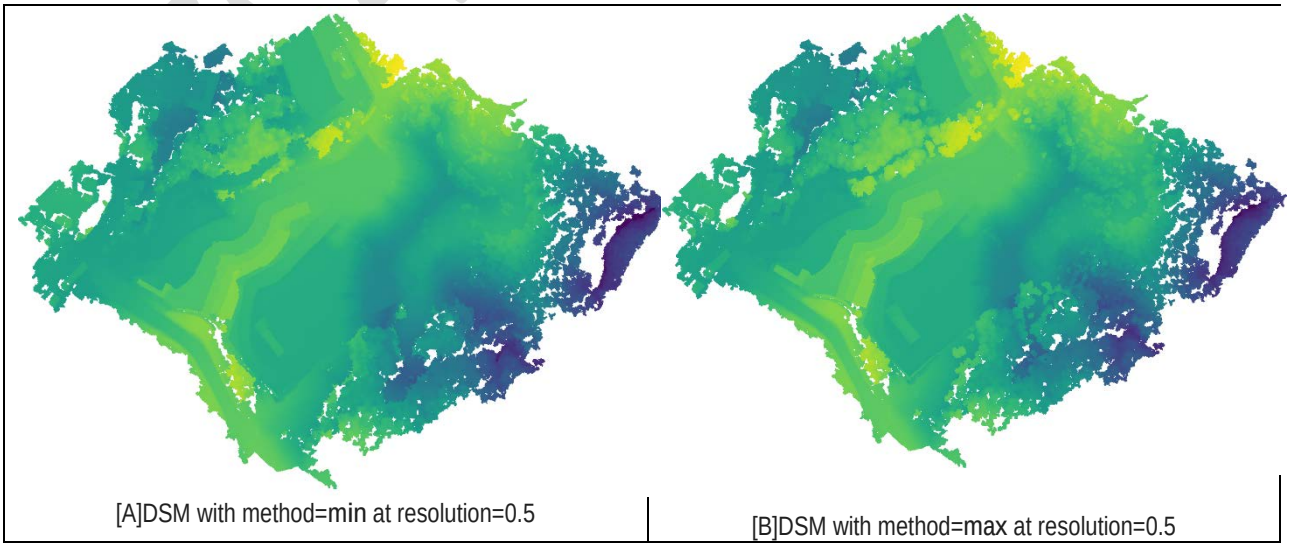
The method=min will generate more terrain/ground oriented DSM by considering the minimum value per cell. Its good when we want to exclude all tall rising points representing trees and other objects and tries to retain only the minimum cell value.

The method=max can be used when we want to create DSM and retain the vegetation covers like trees etc. Below, we'll use these 2 methods to create two separate DSM and examine the outputs.

- First, Compute the DSM with method=max and resolution=0.5m
  r.in.lidar -e -n -o --overwrite input=ISBT_Shillong_24July2019_group1_densified_point_cloud.las output=dsm_max_05 method=max resolution=0.5

```
C:\Users\Lenovo\Desktop\ISBT\ISBT_Shillong_24July2019>r.in.lidar -e -n -o --overwrite input=ISBT_Shillong_24July2019_gro
up1_densified_point_cloud.las output=dsm_max_05 method=max resolution=0.5
Reading data...
 100%
Writing output raster map...
 100%
14302348 points found in input file(s)
r.in.lidar complete. Raster map <dsm_max_05> created. 14302348 points found
in region.
```

- Fill the DSM with r.grow [Reason for Using r.grow: This is helpful for filling the data gaps]
  r.grow adds cells in the adjacent perimeters of DSM areas and creates a new raster map layer. The region can grow by one or more than one cell by changing the size of the radius parameter.
  r.grow --overwrite input=dsm_max_05 output=dsm_max_grow
- Finally create a shaded relief map with altitude=45 and azimuth=315
  r.relief --overwrite input=dsm_max_grow output=dsm_max_relief altitude=45 azimuth=315
- Repeat the above three commands for method=min and create a separate DSM

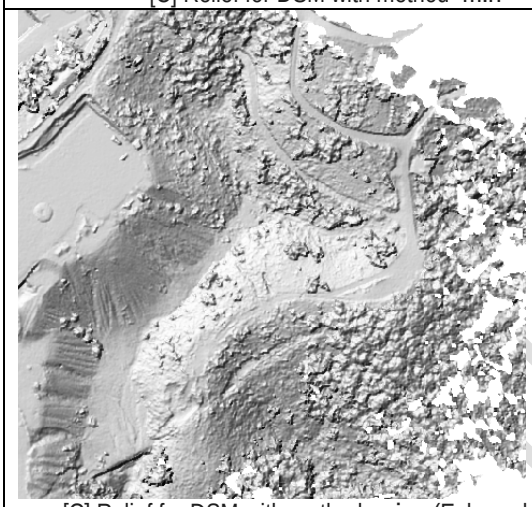


[A]DSM with method=min at resolution=0.5

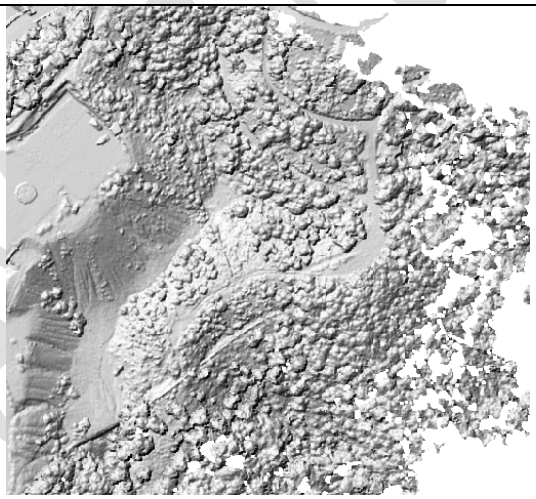
[B]DSM with method=max at resolution=0.5

[C] Relief for DSM with method=min

[D] Relief for DSM with method=max

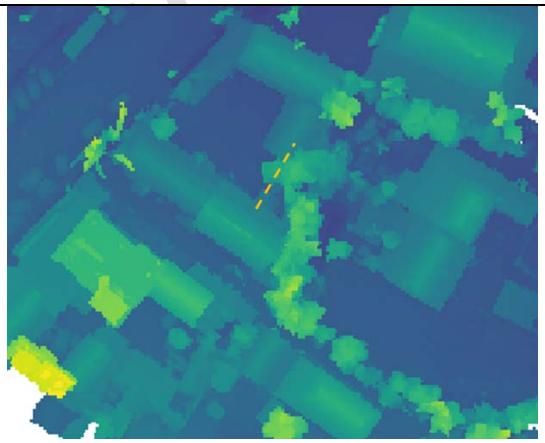


[C] Relief for DSM with method=min (Enlarged View)

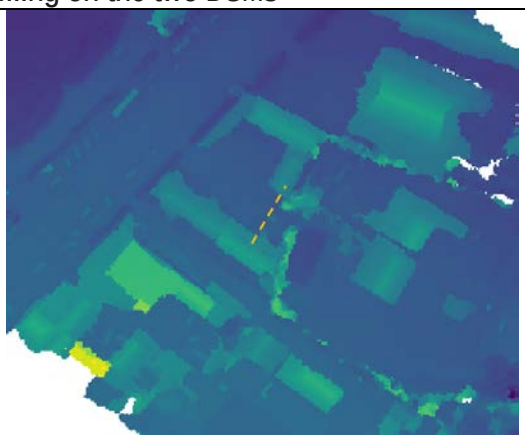


[D] Relief for DSM with method=max (Enlarged View)

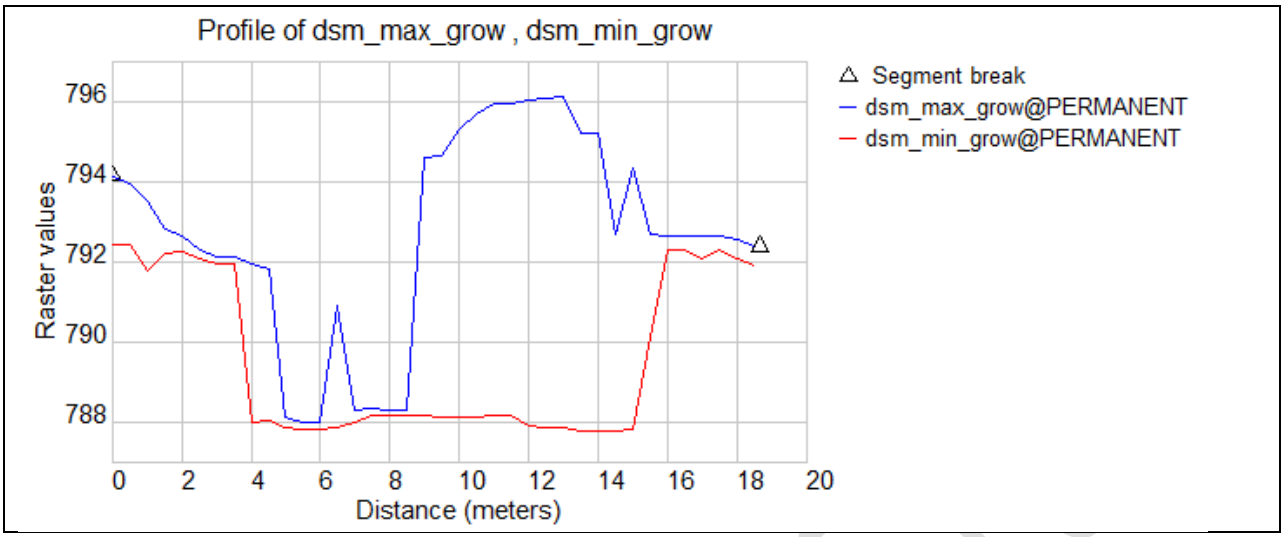
Difference on the Elevation Profiling on the two DSMs



dsm_max_grow (with max method)

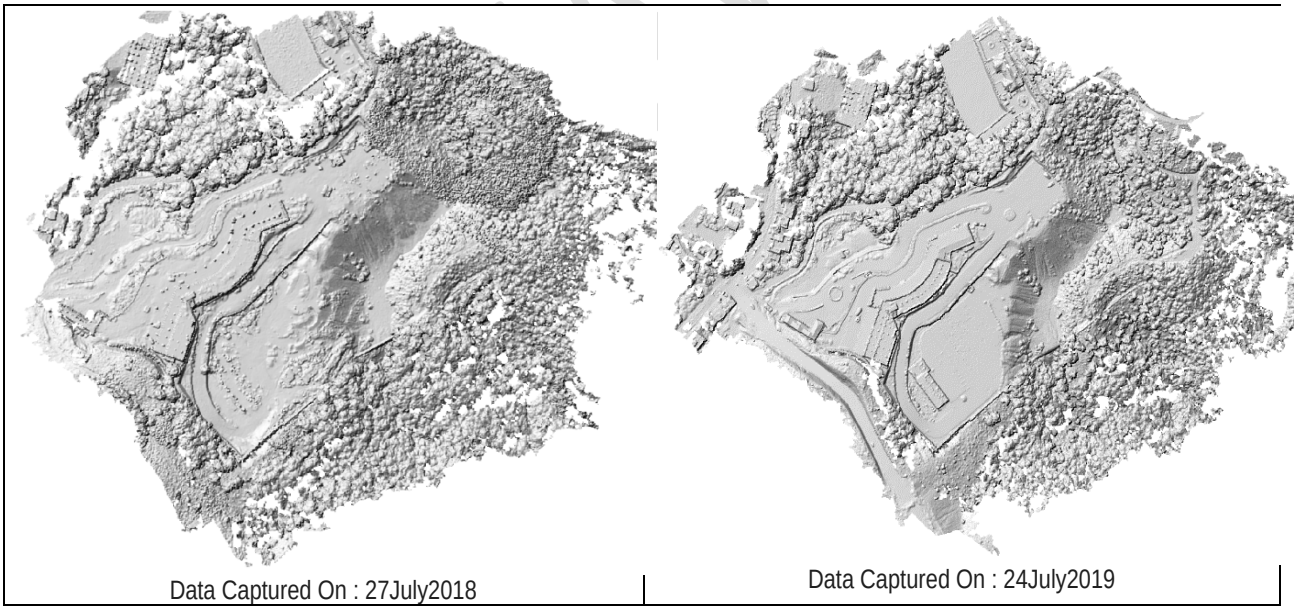


dsm_min_grow (with min method)



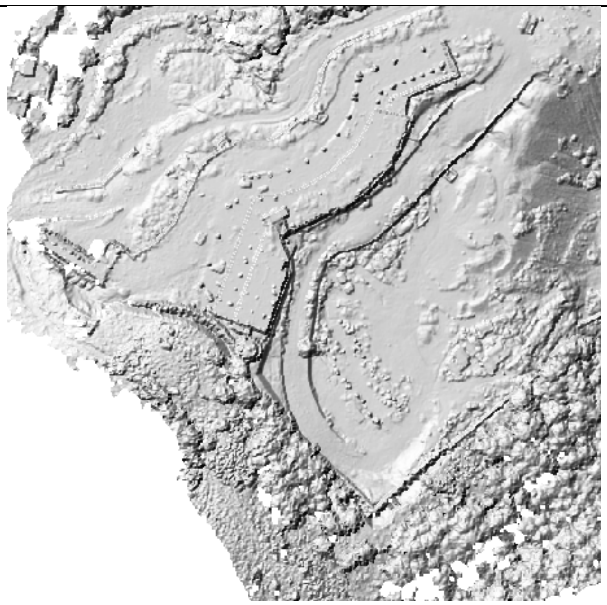
As observed from the above two methods, the method=max tries to generate DSM preserving the vegetation cover in the scene where method=min focuses only on the minimum cell value and creates terrain oriented DSM. Notice the clear difference in the elevation profile drawn on top of both the DSMs.

Additionally relief map generated from Point Clouds can also be used to see the changes in the 2 datasets captured at different time period as depicted below:



Data Captured On : 27July2018

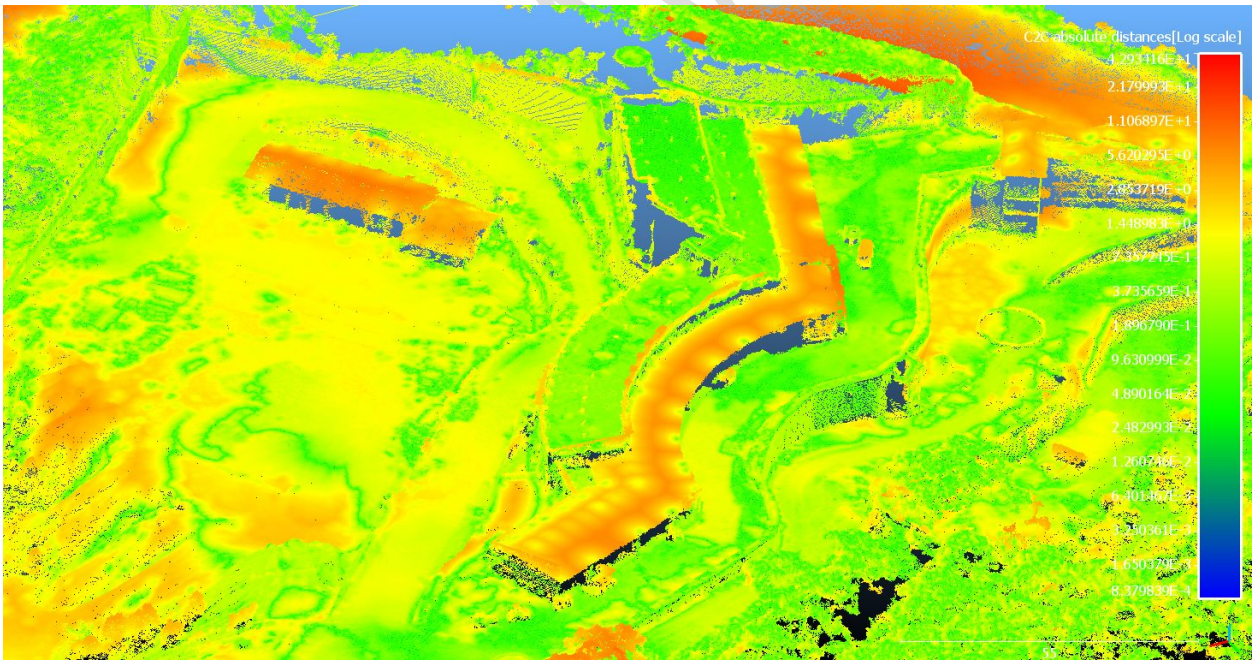
Data Captured On : 24July2019



(A part of the area Enlarged)

(A part of the area Enlarged)

Further, Cloudcompare can be used to find the differences in the point clouds and notices the changes in the areas.



The areas highlighted in yellow and red has significant changes compared to rest of the areas.

Learning Goal 5: Using RGB as Intensity (and Create RGB image from Intensities)

Unlike LiDAR generated point cloud that has Intensity value, the photogrammetry derived point does not have intensity value. PDAL Ferry command can be used to convert the R, G and B channels to different

Intensity values. Eg. The following pdal command converts the photogrammetry point cloud to different Intensity values taking all colour channel:

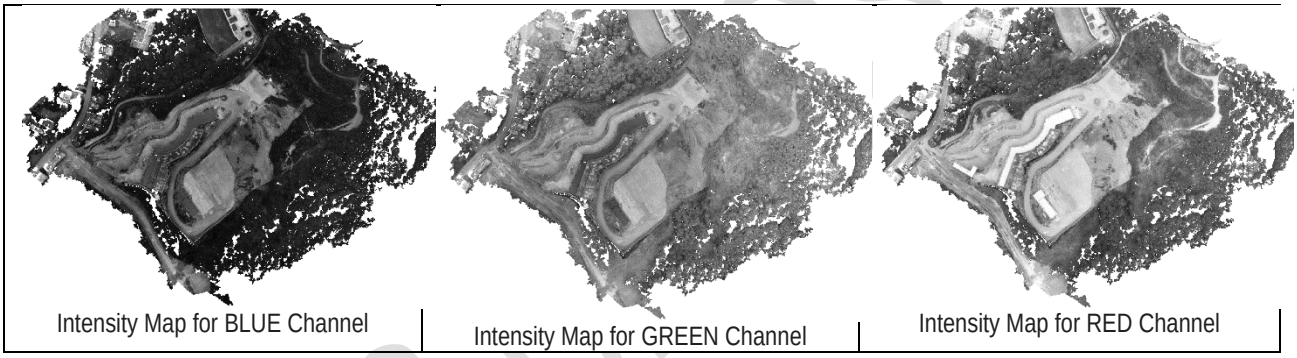
```
pdal translate -i ISBT_Shillong.las ISBT_Shillong_b2.las -f ferry --filters.ferry.dimensions="Blue = Intensity"
pdal translate -i ISBT_Shillong.las ISBT_Shillong_b4.las -f ferry --filters.ferry.dimensions="Red = Intensity"
pdal translate -i ISBT_Shillong.las ISBT_Shillong_b1.las -f ferry --filters.ferry.dimensions="Green = Intensity"
```

We can use the above datasets and generate raster maps at defined resolution, say at 0.5 as follows:

```
r.in.lidar -e -o -i -n --overwrite input=ISBT_Shillong_b4.las output=ISBT_Shillong_04 resolution=0.5
r.in.lidar -e -o -i -n --overwrite input=ISBT_Shillong_b1.las output=ISBT_Shillong_01 resolution=0.5
r.in.lidar -e -o -i -n --overwrite input=ISBT_Shillong_b2.las output=ISBT_Shillong_02 resolution=0.5
```

Finally, we convert all the intensity outputs to grey colours:

```
r.colors map=ISBT_Shillong_04,ISBT_Shillong_01,ISBT_Shillong_02 color=grey
```



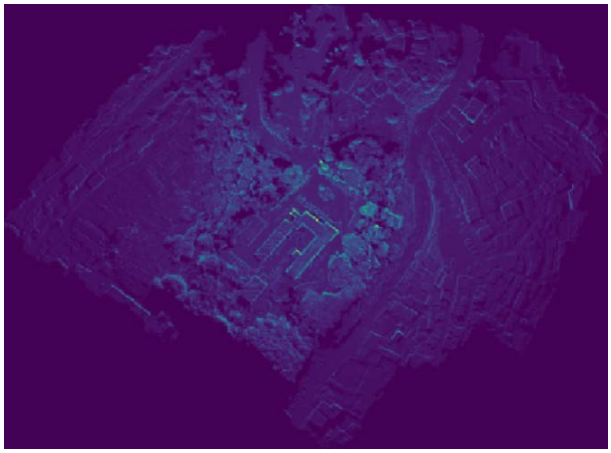
Intensity Map for BLUE Channel | Intensity Map for GREEN Channel | Intensity Map for RED Channel

Combining these also gives an RGB image of the area (using r.composite):

An RGB ortho image directly from the Point cloud!

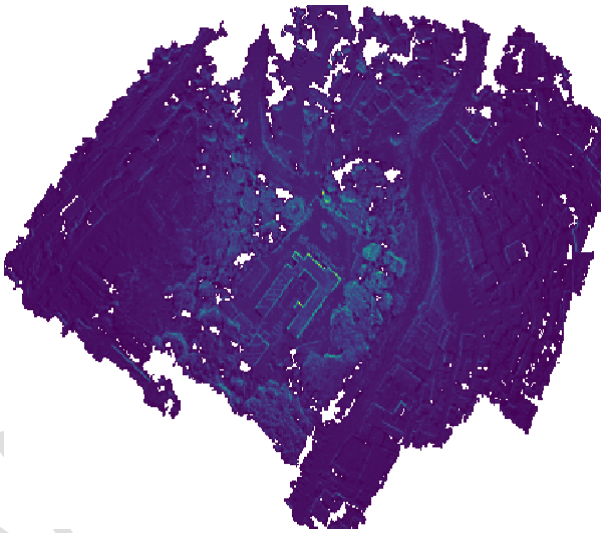
Learning Goal 6: Checking for Spatial Distribution and Regularity of Point Clouds generated (Assessing the holes in point clouds)

- How Many Cells with zero points? Cells with many zero points shows sparseness or gaps in points in the area. These needs to be looked upon. If there are too many cells with empty points, it may not be perfect for generation of good surface model.



Output with cell with zero point/cell

r.in.lidar input= wkhsource2019.las output=count_1 method=n -e -n resolution=1



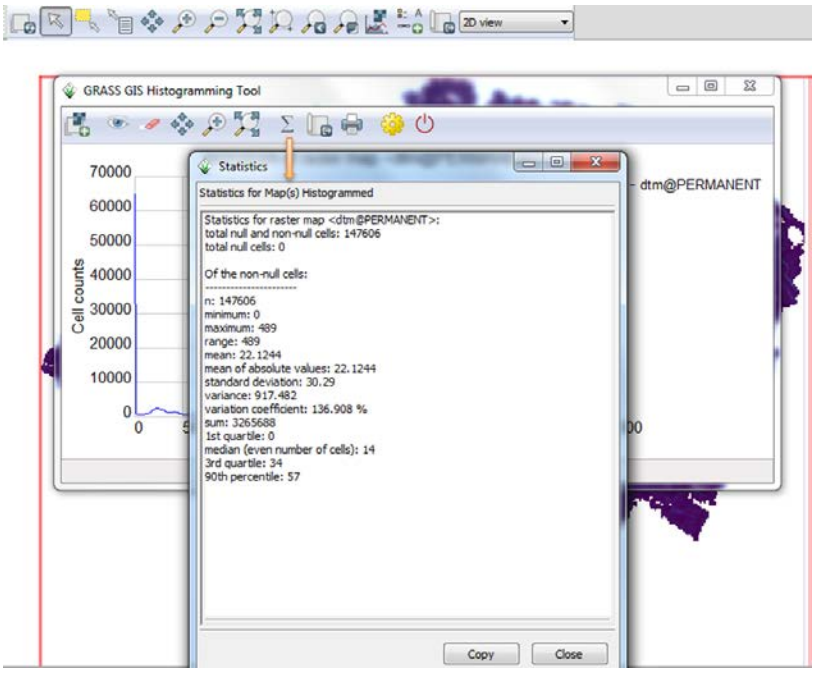
Clean Output with null cell removed

r.mapcalc "count_1m_null = if(count_1m==0,null(),count_1m)" --overwrite

r.univar map= count_1m (The Univar gives statistics on points)

```
r.univar map=dtm@PERMANENT
total null and non-null cells: 147606
total null cells: 0
Of the non-null cells:
----------------------
n: 147606
minimum: 0
maximum: 489
range: 489
mean: 22.1244
mean of absolute values: 22.1244
standard deviation: 30.29
variance: 917.482
variation coefficient: 136.908 %
sum: 3265688
(Wed Oct 09 14:30:39 2019) Command finished (0 sec)
```

The above dsm has an average or mean density = 22.1244 points/meter and number of points (sum) as 3265688. You can also check the statistics from the Map Display – by first plotting histogram and clicking on Statistic icon:



- Coverage Assessment: Where are the 0 count (null/nodata) cells?

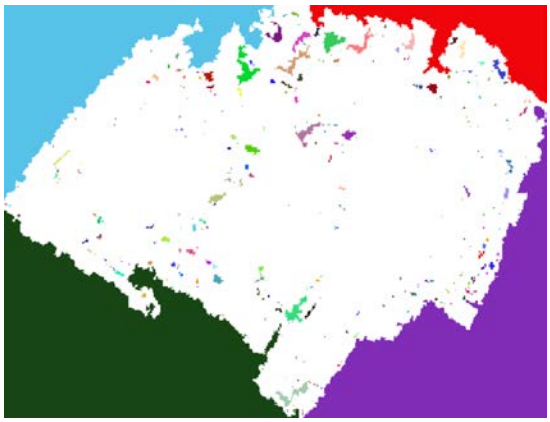
  First Verify the region: g.region –pa

- Create Inverse Raster Mask

  r.mapcalc "count_1m_null_mask = if(isnull(count_1m_null),1,null())" –overwrite

- Clump/Aggregate the 0 cell data

  r.clump –d --overwrite input=count_1m_null_mask output=count_1m_null_clump



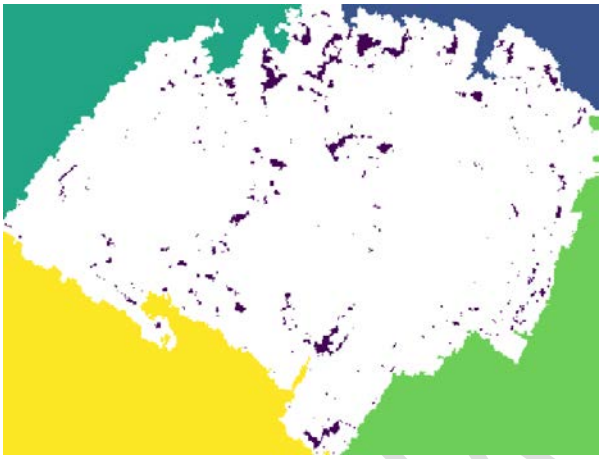
Clumps or groups cells that form physically discrete areas into unique categories.

```
F:\pdal1\nesac>r.clump -d --overwrite input=count_1m_null_mask output=count_1m_null_clump
Pass 1 of 2...
 100%
Generating renumbering scheme...
 100%
Pass 2 of 2...
 100%
r.clump complete. 307 clumps.
```

We got 307 clumps. We'll now select those clumps with at-least 3 cells.

- Filter the Mask>3 cell (Find areas for areas with more than 3 cells)

  r.area --overwrite input=count_1m_null_clump output=count_1m_null_3cells lesser=3



- Vectorize the Mask and
  r.to.vect --overwrite input=count_1m_null_3cells output=count_1m_null_mask type=area
- Add Area Column – Calculate Size

  v.db.addcolumn map=count_1m_null_mask columns="acres double precision"
  v.to.db map=count_1m_null_mask option=area columns=acres units=acres

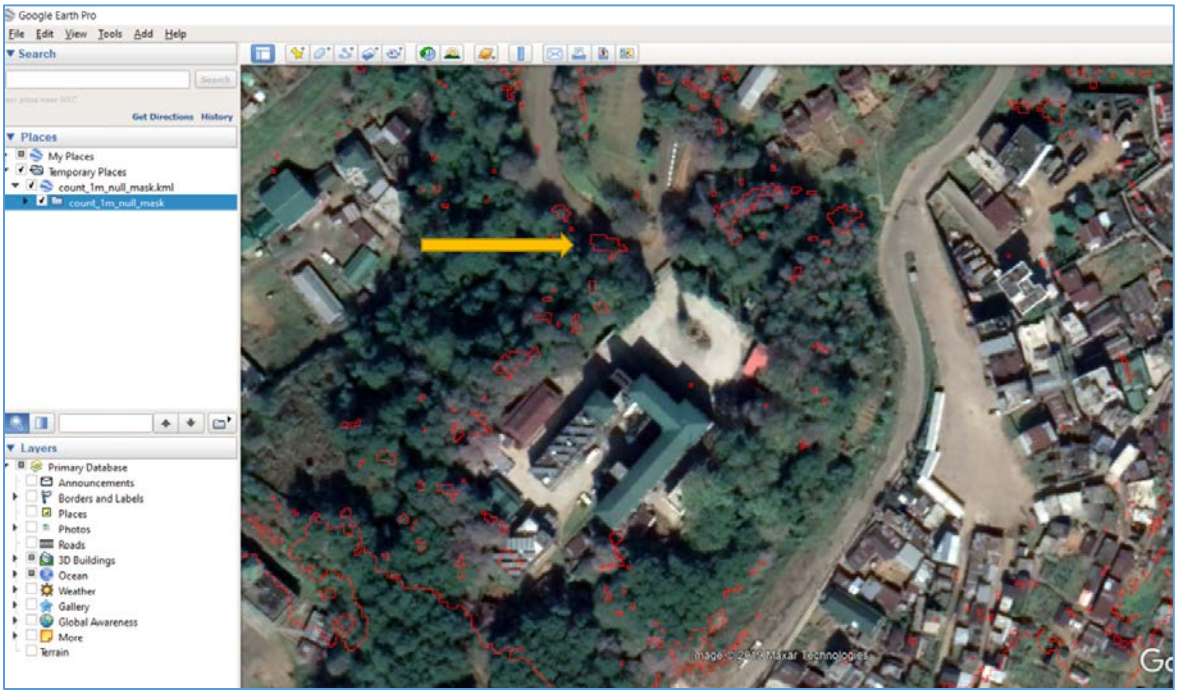
```
F:\pdal1\nesac>v.db.addcolumn map=count_1m_null_mask columns="acres double precision"

F:\pdal1\nesac>v.to.db map=count_1m_null_mask option=area columns=acres units=acres
WARNING: Values in column <acres> will be overwritten
Reading areas...
 100%
Updating database...
 100%
398 categories read from vector map (layer 1)
398 records selected from table (layer 1)
398 categories read from vector map exist in selection from table
398 records updated/inserted (layer 1)
```

- Output to KML or other OGR format

  v.out.ogr --overwrite input=count_1m_null_mask output=F:\pdal1\nesac\count_1m_null_mask.kml format=KML

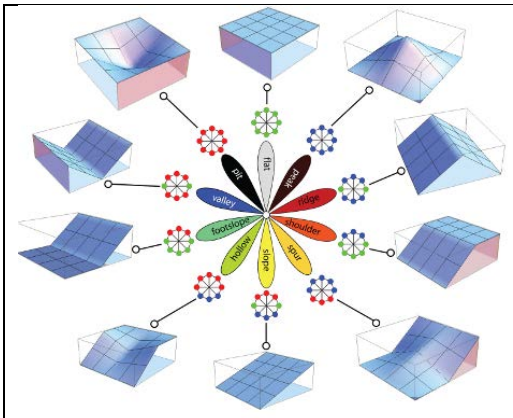
```
F:\pdal1\nesac>v.out.ogr --overwrite input=count_1m_null_mask output=F:\pdal1\nesac\count_1m_null_mask.kml format=KML
Exporting 402 areas (may take some time)...
 100%
WARNING: 4 features without category were skipped. Features without
         category are written only when -c flag is given.
v.out.ogr complete. 398 features (Polygon type) written to
<count_1m_null_mask> (KML format).
```



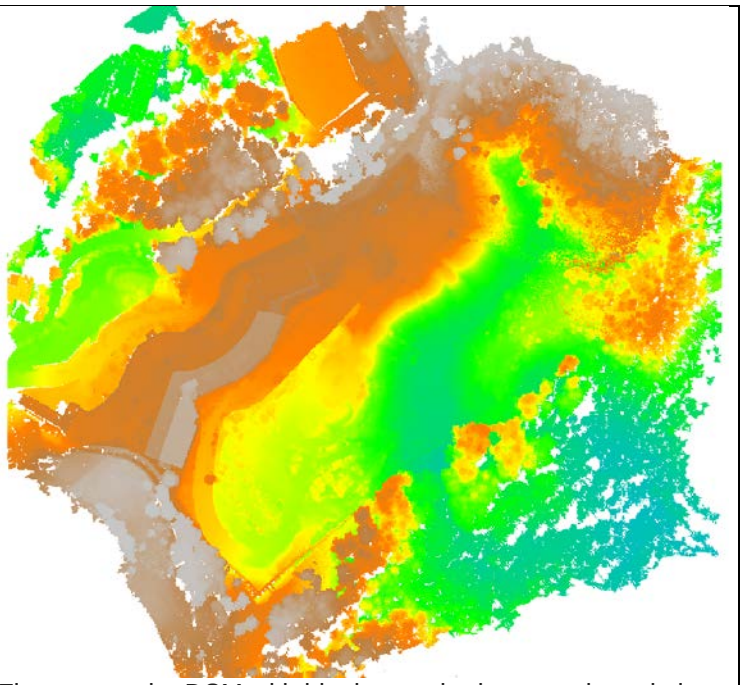
Notice the area where point could not be collected (encircled with red polygons). This way, we can find out important areas in the scene where points are missed and may therefore need to regenerate it with more images over the area.

Bonus Application: Vegetation Analysis Using *r.geomorphon* module

r.geomorphon - Calculates terrain forms and associated geometry using machine vision approach. It returns geomorphic map with the following 10 most popular terrain forms. In the legend, the shoulder would represent the shape of tall trees. We'll use r.geomorphon to identify and locate the trees from the scene.

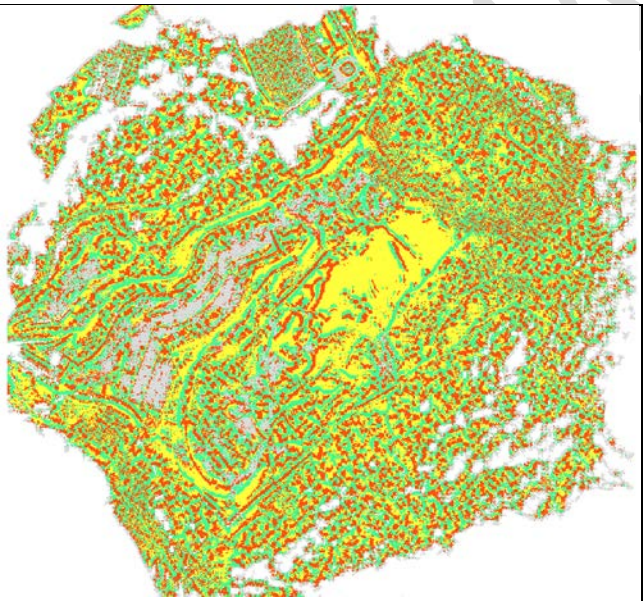


10 terrain forms r.geomorphon can represent



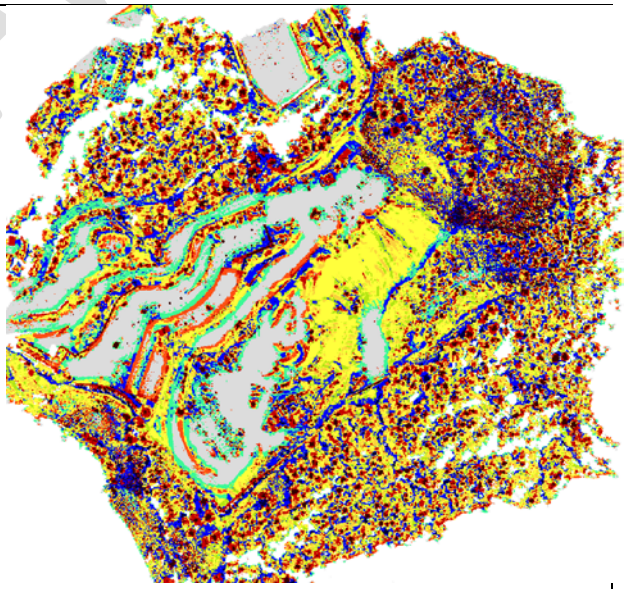
First we get the DSM with binning method max and resolution at 0.5

r.in.lidar –e –n input=data.las output=dsm method=max resolution=0.5

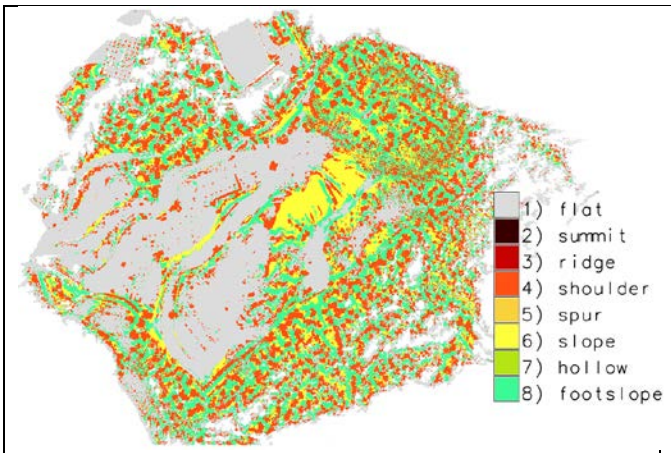


Then, we find the forms keeping search window at 7 and with skip value to 4 to ignore objects with small extends

r.geomorphon elevation=dsm forms=dsm_forms search=7 skip=4

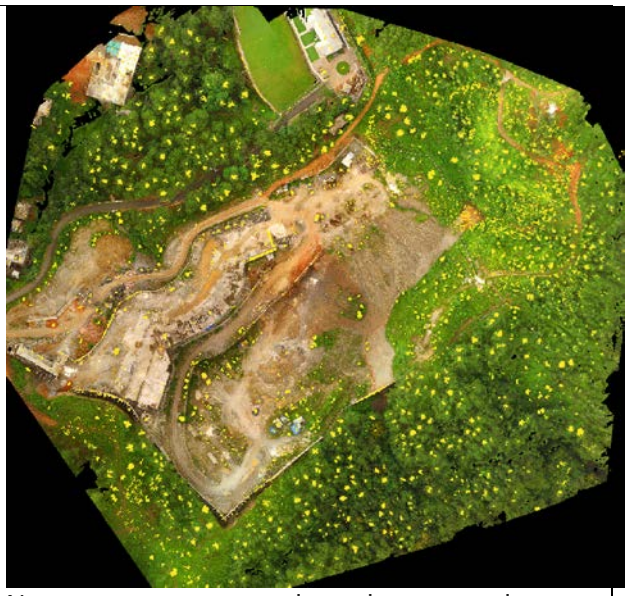


r.geomorphon elevation=dsm_isbt_max_05 forms=dsm_forms search=12 skip=8 flat=10 --o



Lowering the skip parameter decreases generalization and brings in summits which often represents tree tops. Note : Red area shows where trees are grown

r.geomorphon elevation=dsm_isbt_max_05 forms=dsm_forms search=12 skip=2 flat=10 --o



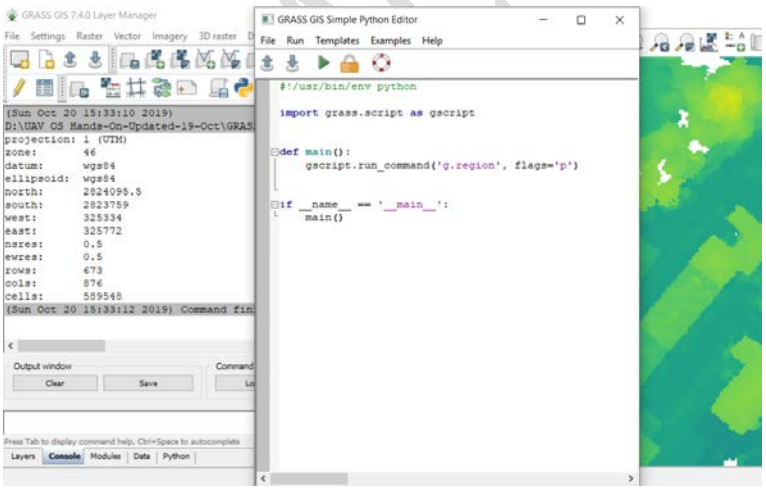
Now we extract summits using r.mapcalc raster algebra expression

r.mapcalc trees = if(dsm_forms==2, 1, null())

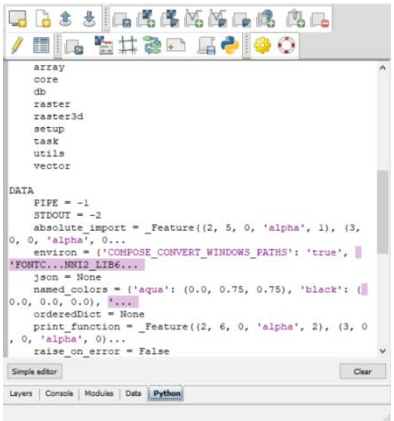
Note : Yellow spots indicates the presence of tall tress

(*optional* Reading) Basic introduction to Python interface

The simplest way to execute the Python code which uses GRASS GIS packages is to use *Simple Python Editor* integrated in GRASS GIS (accessible from the toolbar or the Python tab in the *Layer Manager*). Another option is to use your favorite text editor and then run the script in GRASS GIS using the main menu ***File*** → ***Launch*** *script*.



Simple Python Editor integrated in GRASS GIS



Python tab with an interactive Python shell

We will use the Simple Python Editor to run the commands. You can open it from the Python tab. When you open Simple Python Editor, you find a short code snippet. It starts with importing GRASS GIS Python Scripting Library:

```
import grass.script as gscript
```

In the main function we call g.region to see the current computational region settings:

```
gscript.run_command('g.region', flags='p')
```

Note that the syntax is similar to command line syntax (g.region -p), only the flag is specified in a parameter. Now we can run the script by pressing the Run button in the toolbar. In Layer Manager we get the output of g.region.

In this example, we set the computational extent and resolution to the raster layer ortho which would be done using g.region raster=ortho in the command line. To use the run_command to set the computational region, replace the previous g.region command with the following line:

```
gscript.run_command('g.region', raster='ortho')
```

The GRASS GIS Python Scripting Library provides functions to call GRASS modules within Python scripts as subprocesses. All functions are in a package called grass and the most common functions are in grass.script package which is often imported import grass.script as gscript. The most often used functions include:

- script.core.run_command(): used with modules which output raster or vector data and when text output is not expected
- script.core.read_command(): used when we are interested in text output which is returned as Python string
- script.core.parse_command(): used with modules producing text output as key=value pair which is automatically parsed into a Python dictionary
- script.core.write_command(): for modules expecting text input from either standard input or file

More on GRASS Python https://grasswiki.osgeo.org/wiki/GRASS_Python_Scripting_Library

For More info on various other useful functionalities on r.in.lidar (GRASS), go to:
https://grass.osgeo.org/grass76/manuals/r.in.lidar.html