9/24/2019

Point Cloud Processing Using ODM, Meshlab and GRASS

UAV OS Lab Handout-01

PS Singh
NESAC, 2019

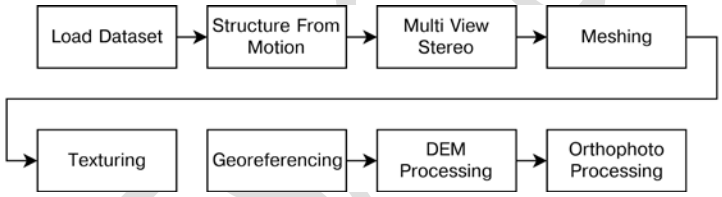
In this lab exercise, we'll learn to use some of the important Open Source Tool for processing drone imagery. OpenDroneMap is great tool for processing drone derived imagery and contains entire UAV SfM processing pipeline and delivers important ready to use geo-referenced data products such as Orthomosaics, DTM, DSM etc. in addition to the dense 3D Point Clouds and Textured Models. Further, It has inbuilt potree based 3D point viewer and analysis platform for perform important measurement analysis such as Distance, Area, Volume, Height, Angles, Surface Profiles etc. It gives quality products almost as good as the products generated with other commercial software such as Pix4D etc. The OpenDroneMap is evolving and much improvement are being done by its various contributors in the Open Source Community. Recent version of ODM allows for addition of GCP to enhance the accuracy of its various products.

Meshlab is wonderful tool for editing, cleaning and meshing of Point Clouds. It has tons of algorithms for Surface Reconstructions. Further, the platform is great for understanding the quality of 3D point cloud generated.

GRASS is an Open Source tool for GIS data analysis. We'll explore some of its module for processing UAV point clouds to generate DEMs.

1. Generation of Drone Data Products Using WebODM

Going from images to 3D models and orthophotos is a process best visualized as a series of incremental steps. Each step relies on the work of previous steps.



[A] Installation of WebODM on Windows

Multiple ways to install it. Command Line based option or choose WebODM Installer which one can buy with nominal payment. (https://www.opendronemap.org/webodm/download/). We'll use manual command line option to install it. (https://github.com/OpenDroneMap/WebODM/)

1. Install Docker Toolbox (https://docs.docker.com/toolbox/toolbox_install_windows/)
2. Install Git (https://git-scm.com/downloads)
3. Install Python (https://www.python.org/downloads/)
4. Docker for Windows users should set up their Docker environment before launching WebODM
5. Create Virtual Machine with more space

a. Under C:\Program Files\Docker Toolbox
b. Modify start.sh with the following: Modify create VM parameters

```
create -d virtualbox >> create -d virtualbox --virtualbox-disk-size "100000"
```

6. From the Docker Quickstart Terminal, Issue the following commands -

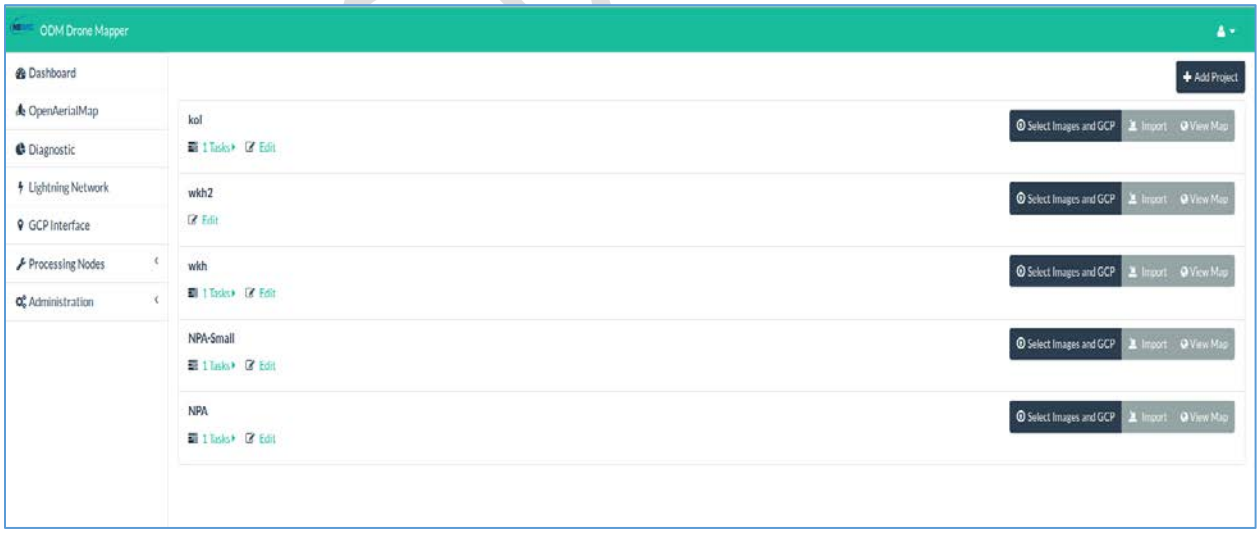
```
git clone https://github.com/OpenDroneMap/WebODM --config core.autocrlf=input
--depth 1
cd WebODM
./webodm.sh start
```

7. Issue > docker-machine ip. If the output is 192.168.1.100
8. Open a Web Browser to http://192.168.1.100:8000.
9. To stop WebODM press CTRL+C or run: ./webodm.sh stop
10. To update WebODM to the latest version use: ./webodm.sh update

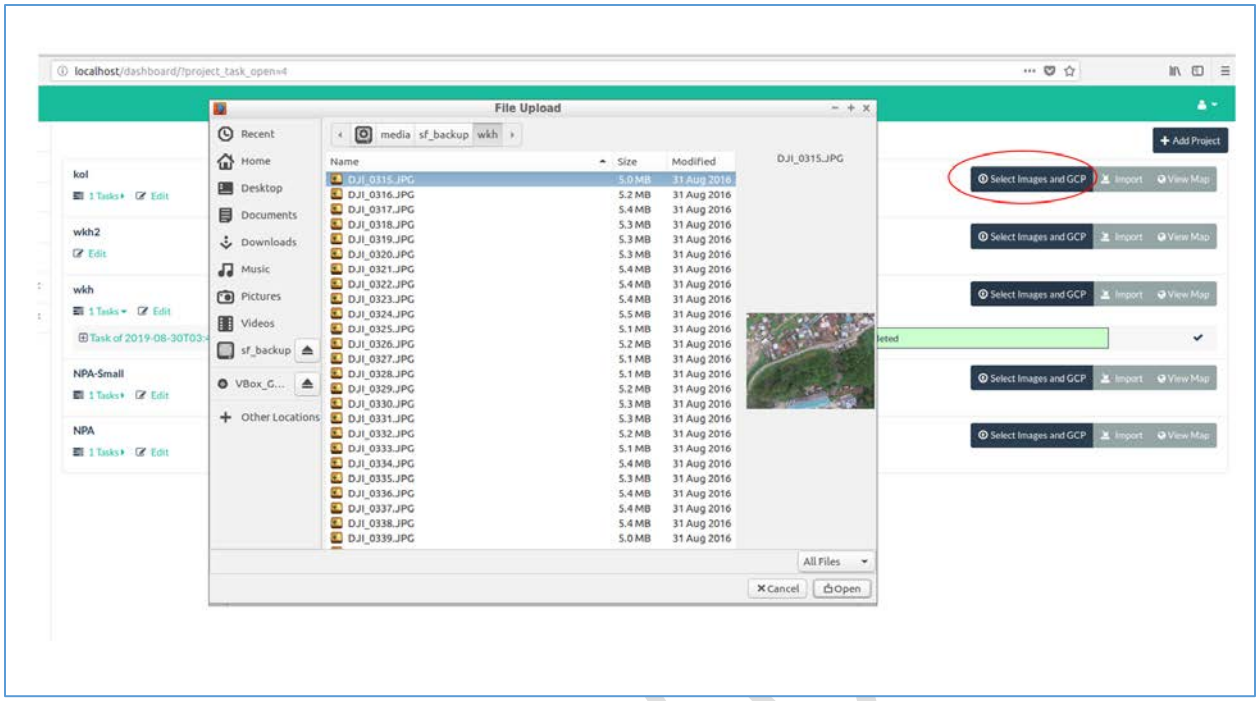
[B] Drone Data Processing Using WebODM

The WebODM has simple and intuitive web interface with necessary tools for processing the drone imagery. Follow the steps as given below to start processing your drone imagery.

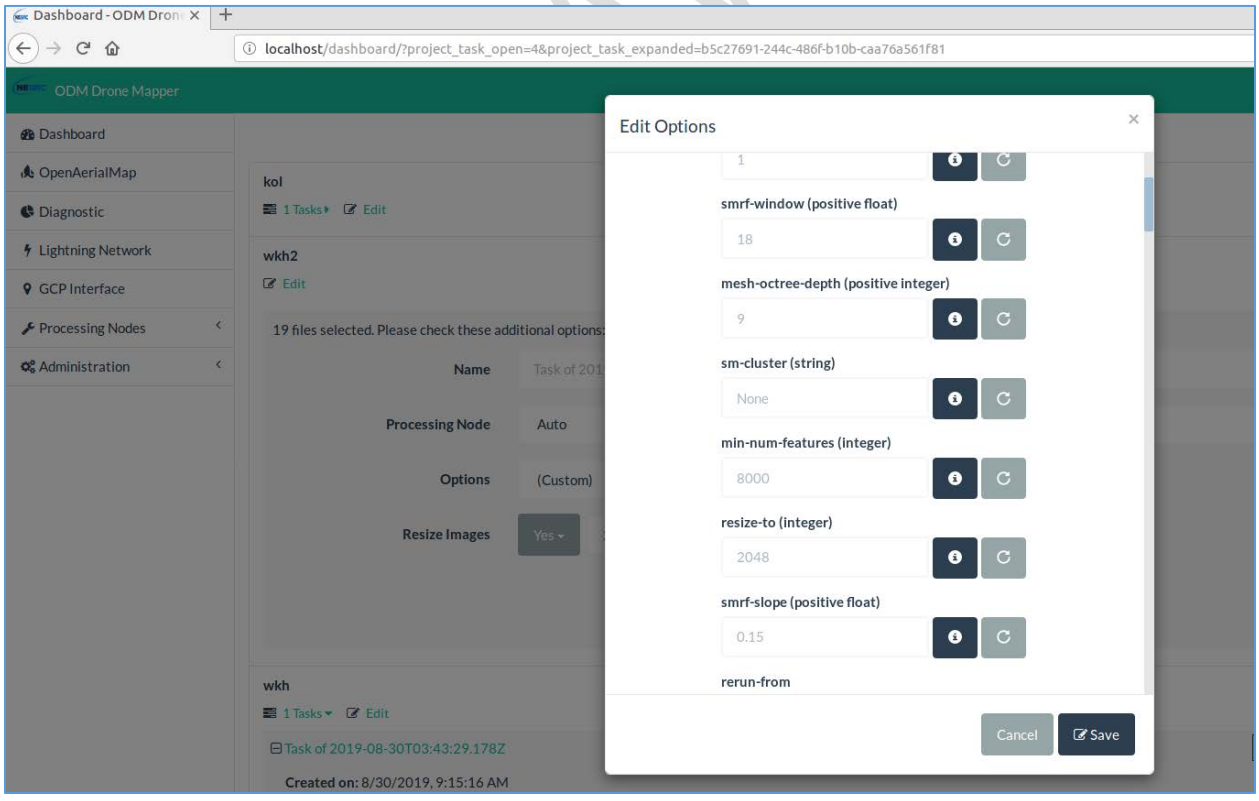
1. Once, the WebODM interface opens up, click on the **Dashboard** and start configuring the drone imagery. Click on **Add Project** to name your project.



2. Click on **Select Images and GCP** to start adding your Drone imagery

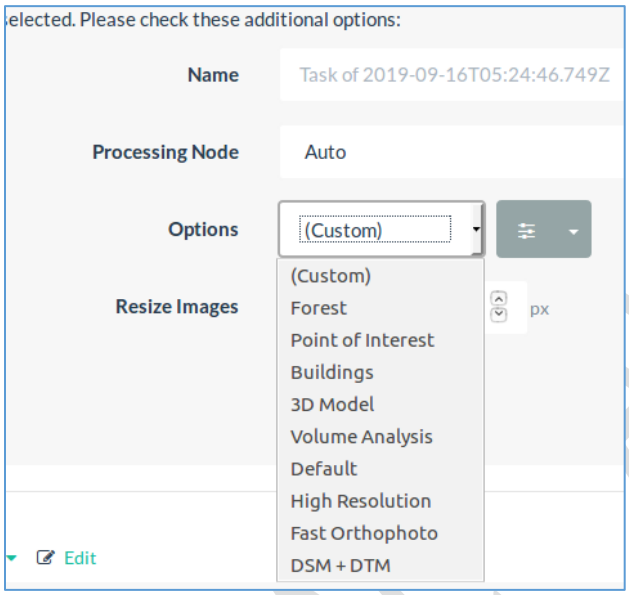


3. Edit processing options based on your product output requirement

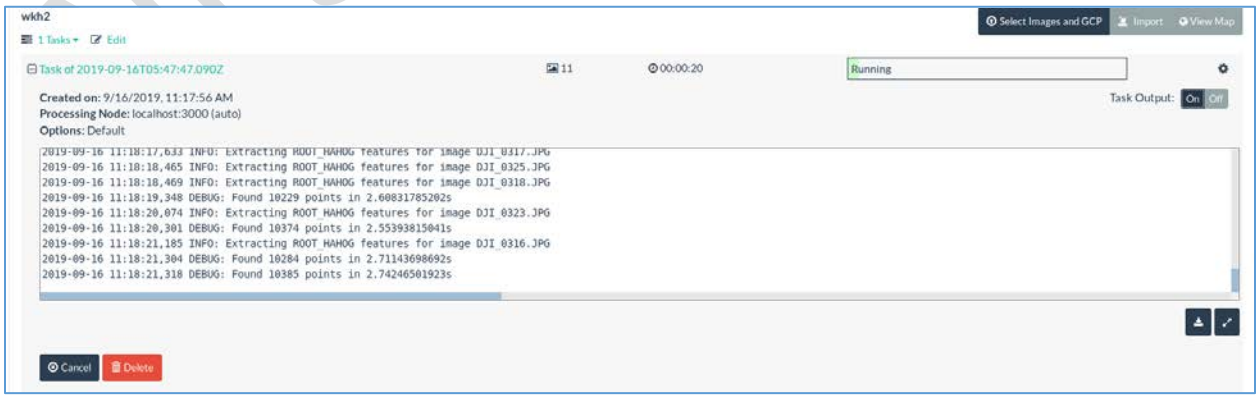


There are several components involved in the data processing pipeline. Each component has several adjustable settings that influence the output. The software exposes a subset of these available knobs through various options. When creating a task, a user can choose to tweak one or more options to change the behavior of the pipeline. Pl note that, the WebODM exposes just a subset of options (but useful options) which can influence the output product. For much of the other options, you can use the parent ODM! Tuning options is more art than the science. There are no clear guidelines on how to tune options to achieve optimal results. That's mostly because the best options for a certain dataset do not necessary works for other datasets!.

But then, there are predefine option templates based on the type of scene you are going to process.



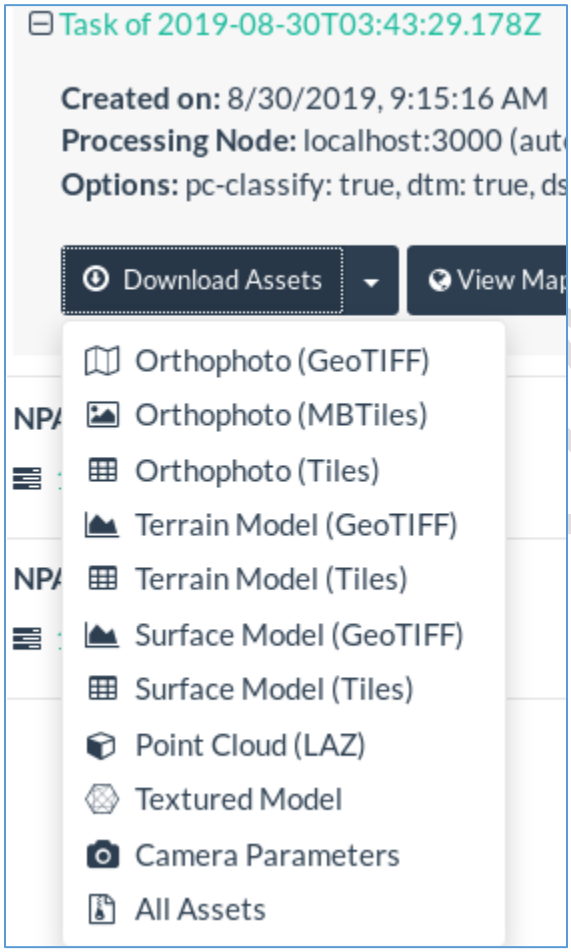
4. Click on **Start Processing** to start processing. Optionally, you can set the **Task Output** knob to **On** to see the background processes running behind. Depending upon your system configurations and number of images used, the overall processing time will vary.



Note : You can also add multiple projects and run in parallel. But, ensure that you have good system with good system configurations.

5. Once, its completed, you can now download the data products based on your configurations made.

6. You can download all the 3D models including the Orthomosaic and 3D Point Clouds. Click to All Assets and download all the files for our use later on.



7. Now, let's start checking our data products. Clicking on '**View Map**' will opens a 2D viewer windows where you can see the processed outputs (Orthomosaic, DSM, DTM etc.) overlayed on top of google map (assuming that, you've internet connection with the system). Further, you can also perform some kind of analysis such as Volume, area and length based on your selection on the study area. The interface also allows you to create

Contour map in chosen interval based on DSM/DTM and export it different formats including the popular shapefile format for immediate use in GIS analysis.

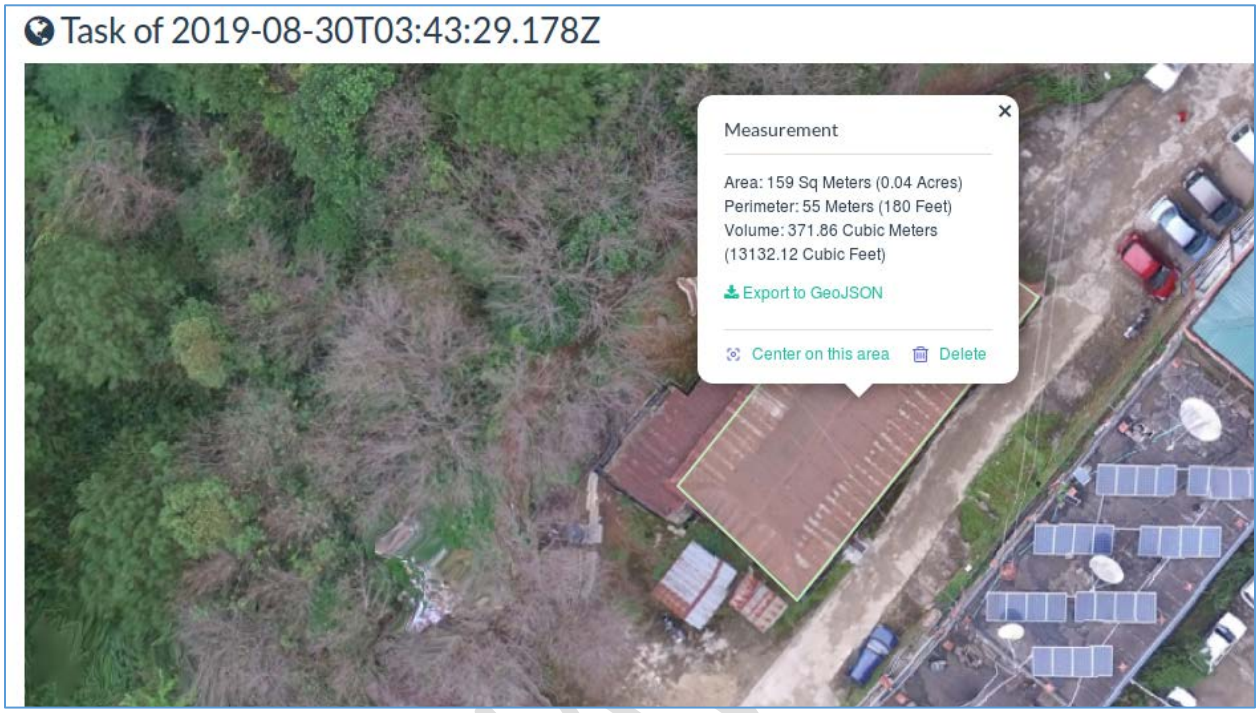


Figure: Measurement of Area, Perimeter and Volume on an object

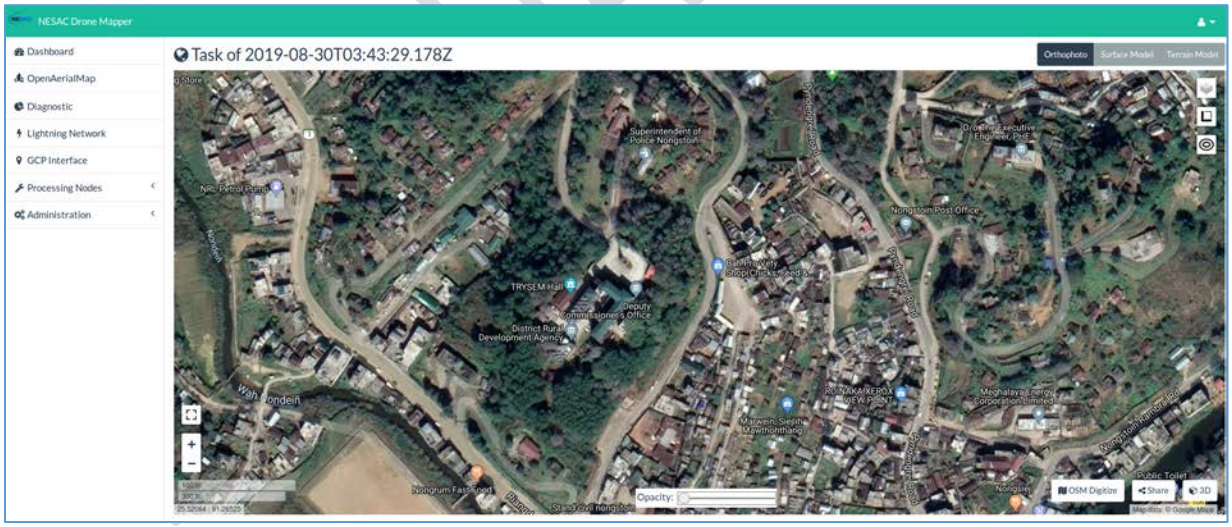


Figure: Google Map View of the Area under Study

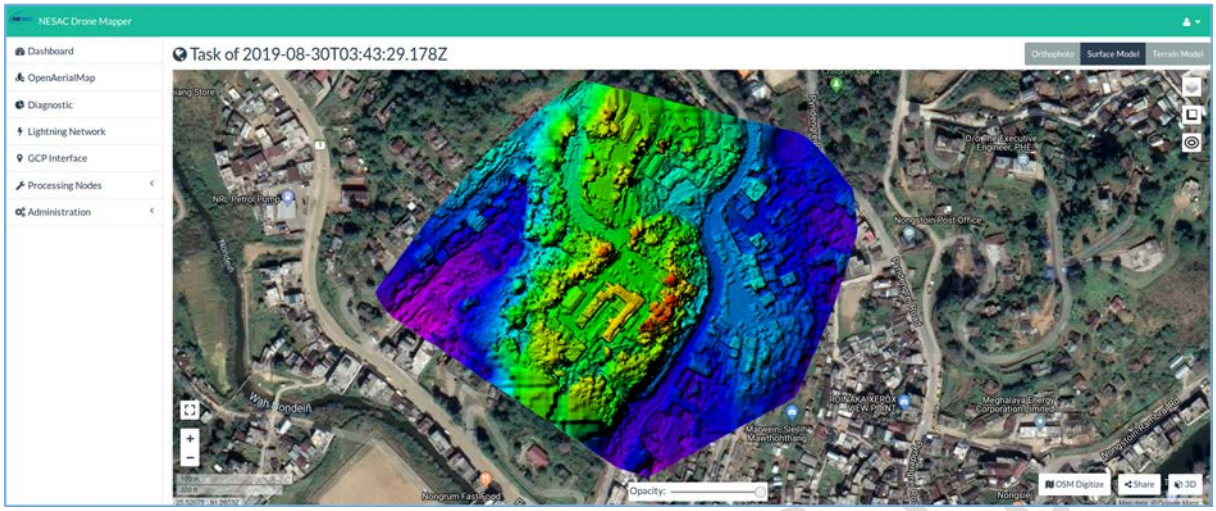


Figure: The Digital Surface Model

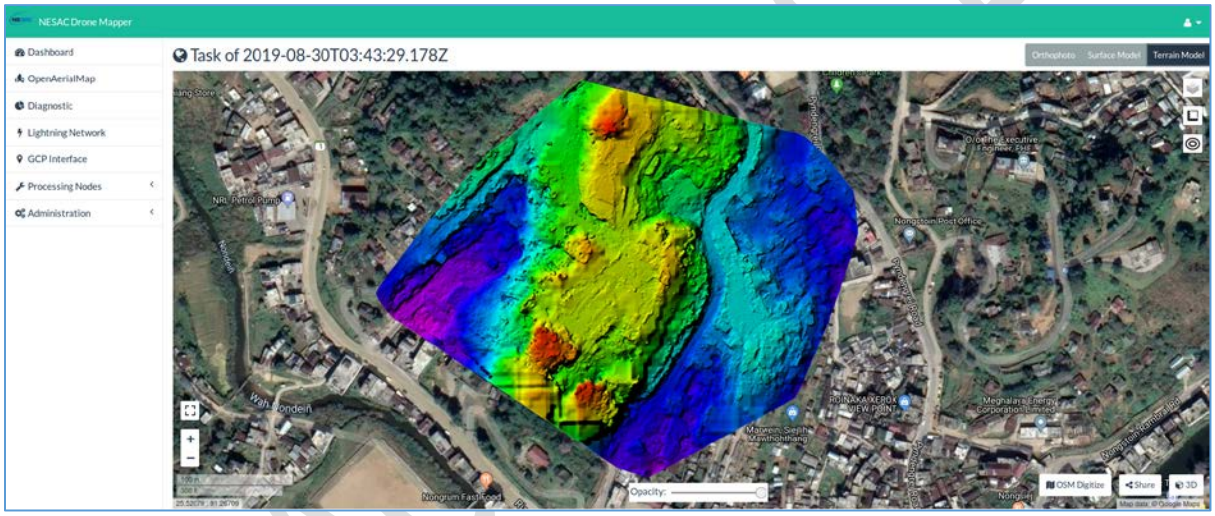


Figure: The Digital Terrain Model

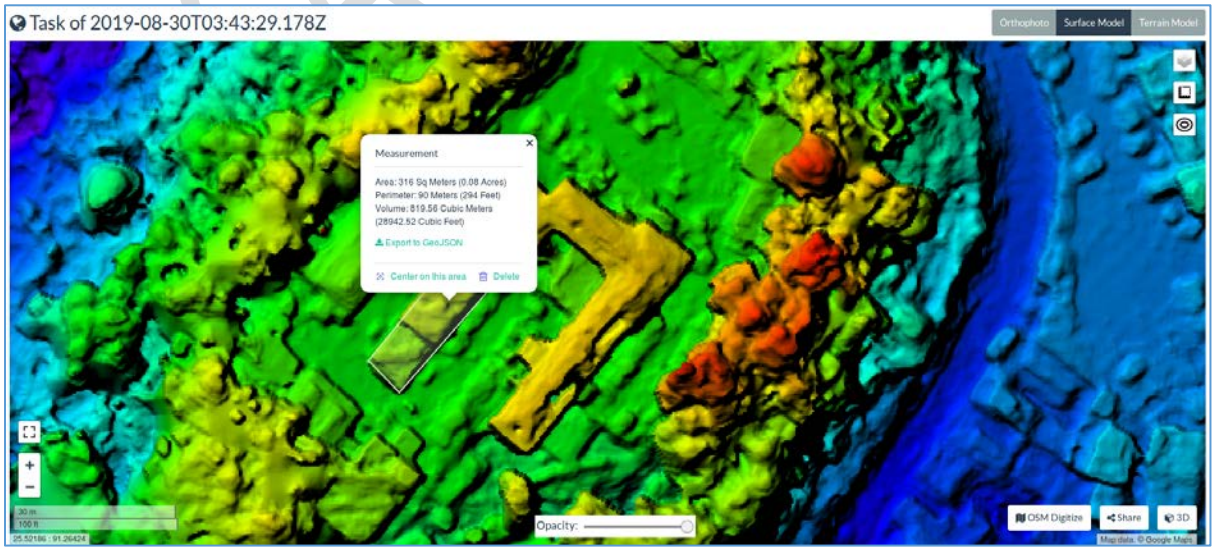


Figure: Measurement Using The Digital Surface Model

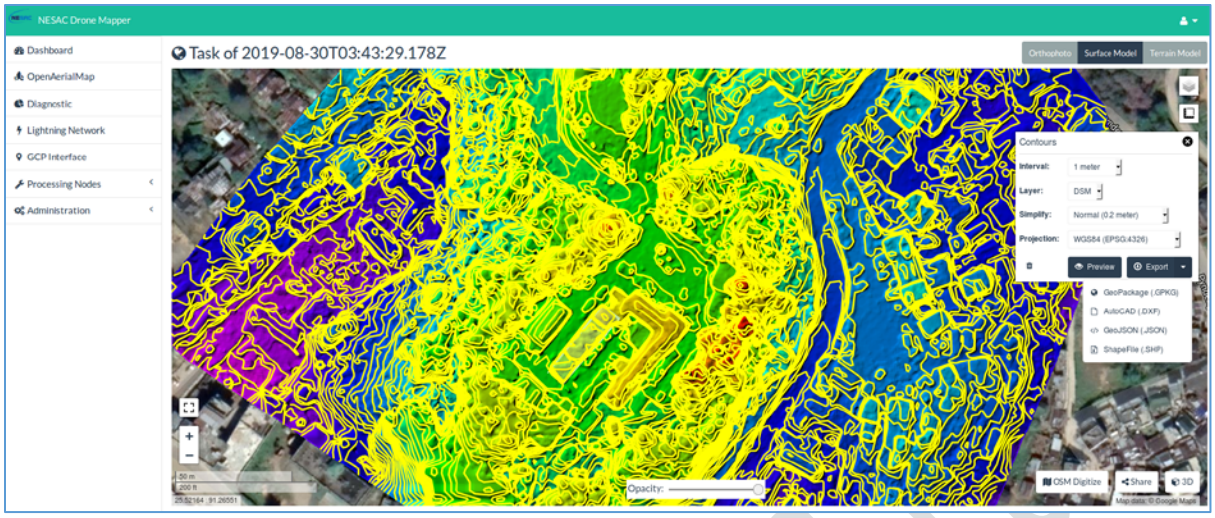


Figure: Generation of Contour Using The Digital Surface Model

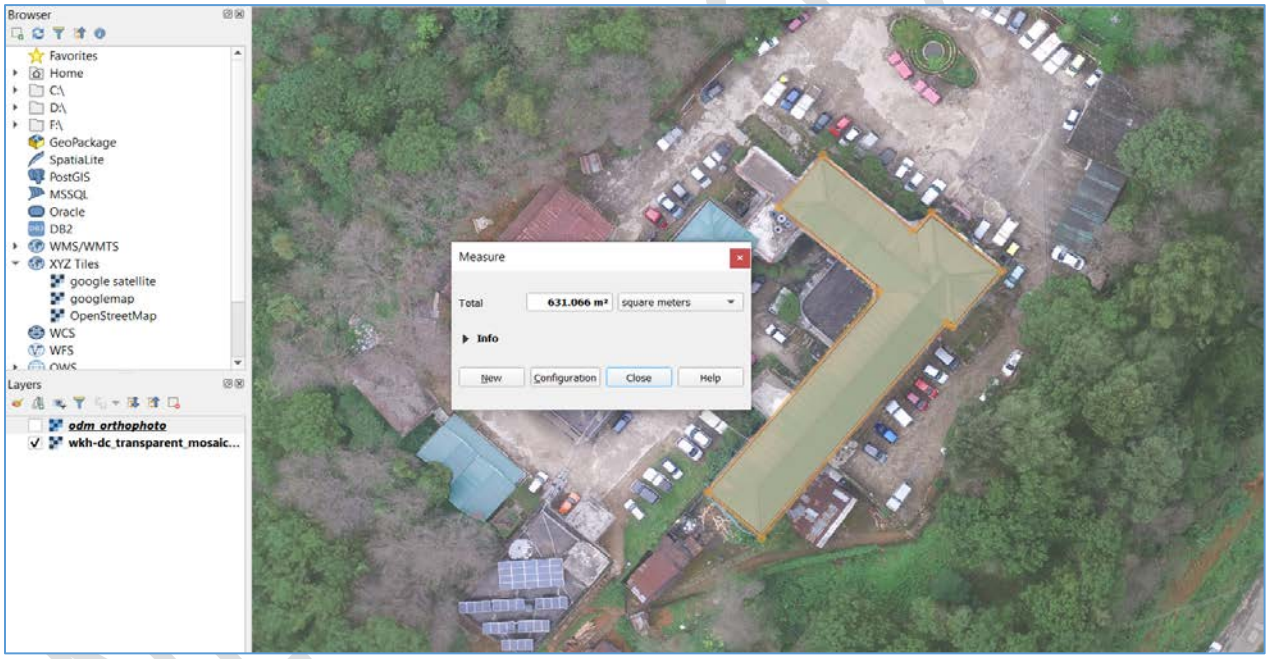
8. The 3D Point Cloud Analysis in WebODM : The 3D Map viewer allows you to visualize the dense 3D point Cloud / 3D Texture Model. You can manage the visualization by changing its settings in appearance. Now, you can also perform analysis of the point cloud with regard to measurement of distance, area, height, volume, surface profile etc.

Figure: 3D Point Cloud Visualization and Analysis on ODM

(Optional) : You can start using the geo-referenced orthomosaic for digitization and other GIS analysis on QGIS.

9. ODM Product Accuracies (in Comparison to other products)

Figure : Pix4D Orthomosaic ODM Orthomosaic

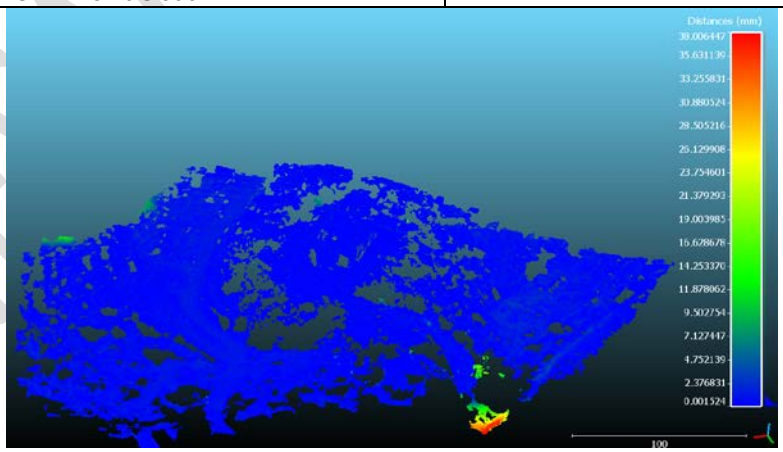


Measurement of Area on Pix4D orthophoto

Measurement of Area on ODM orthophoto

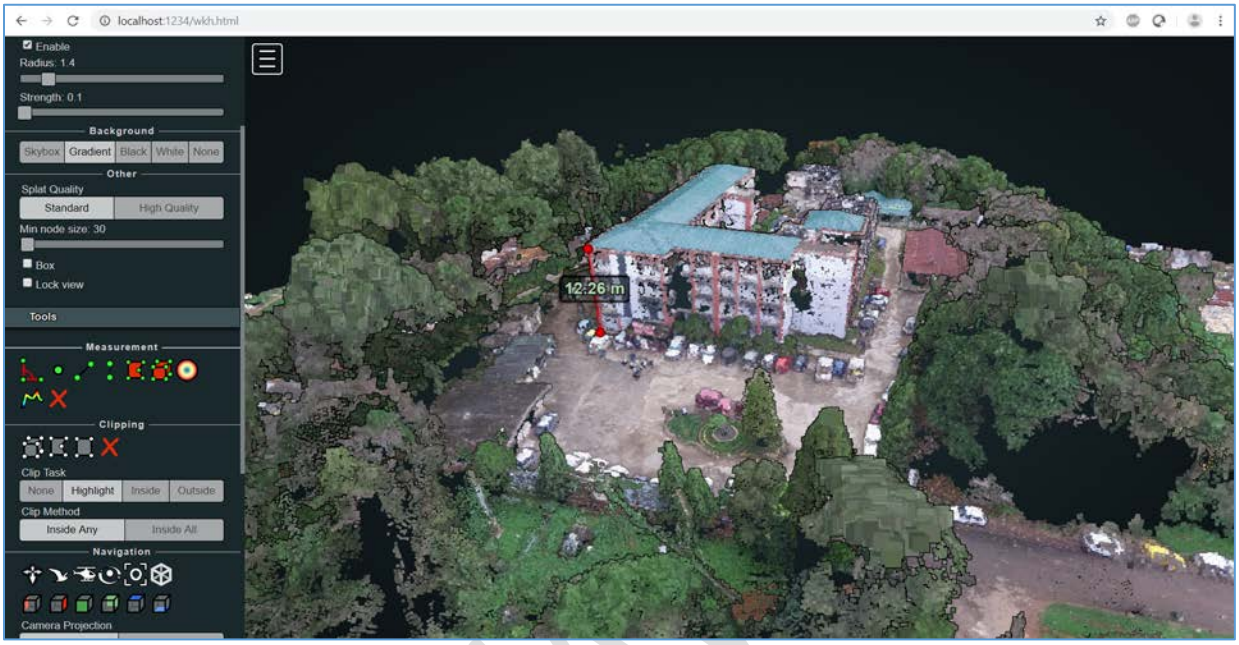
ODM Point Cloud

Pix4D Point Cloud

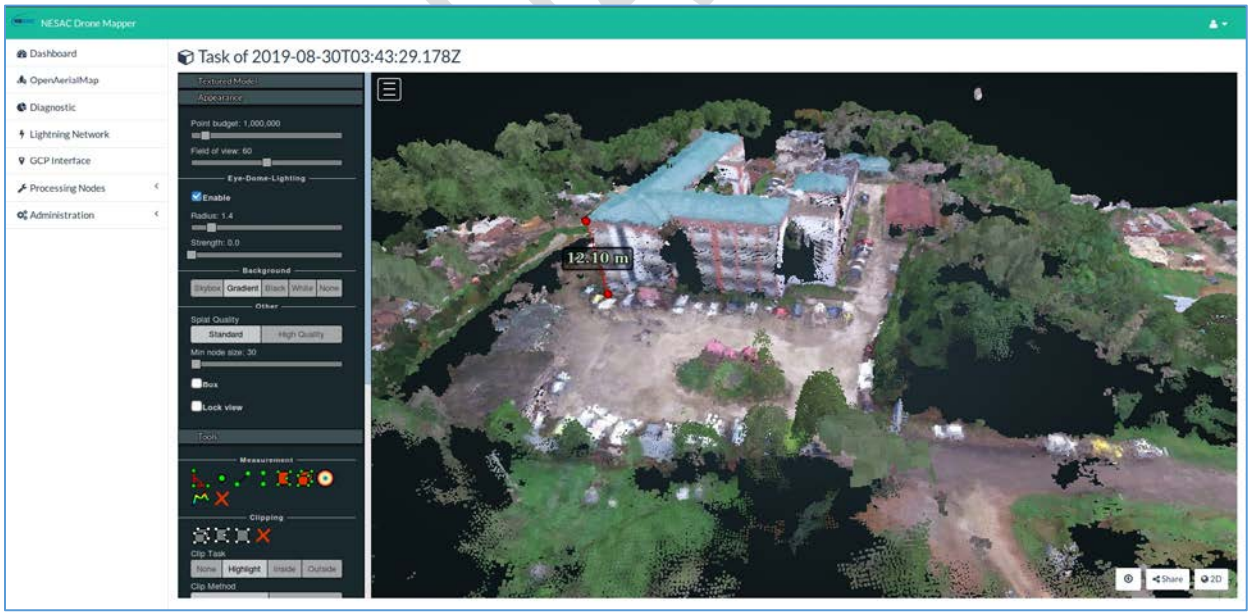


Point Cloud Comparison

The Pix4D derived Point cloud has much denser point cloud than ODM. But, ODM still gives 3D point cloud representing accurate 3D scene structures. A comparison has been made (taking ODM Point cloud as a base) after proper registration and computing Point-to-Point distance. We observed almost zero distances in almost all the point pairs except few.



Height Measurement on Pix4D derived Point Cloud (Eg.)



Height Measurement in ODM derived Point Cloud

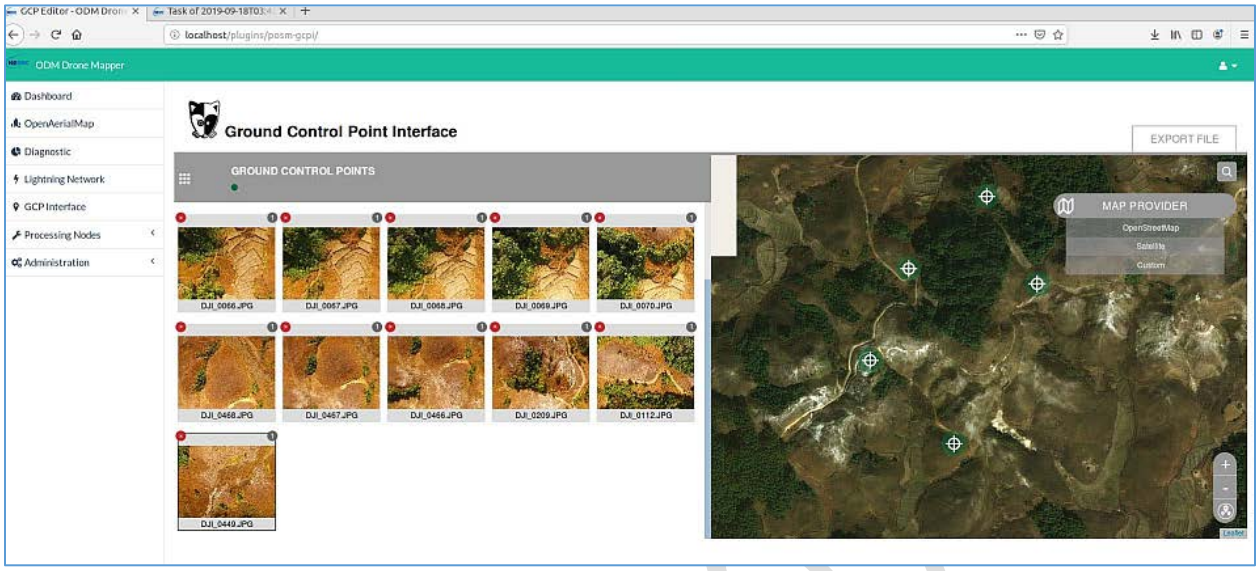
Google Map View of the Area under Study

OrthoPhoto overlaid on Google Map

Image as seen in Google Map UAV Orthophoto

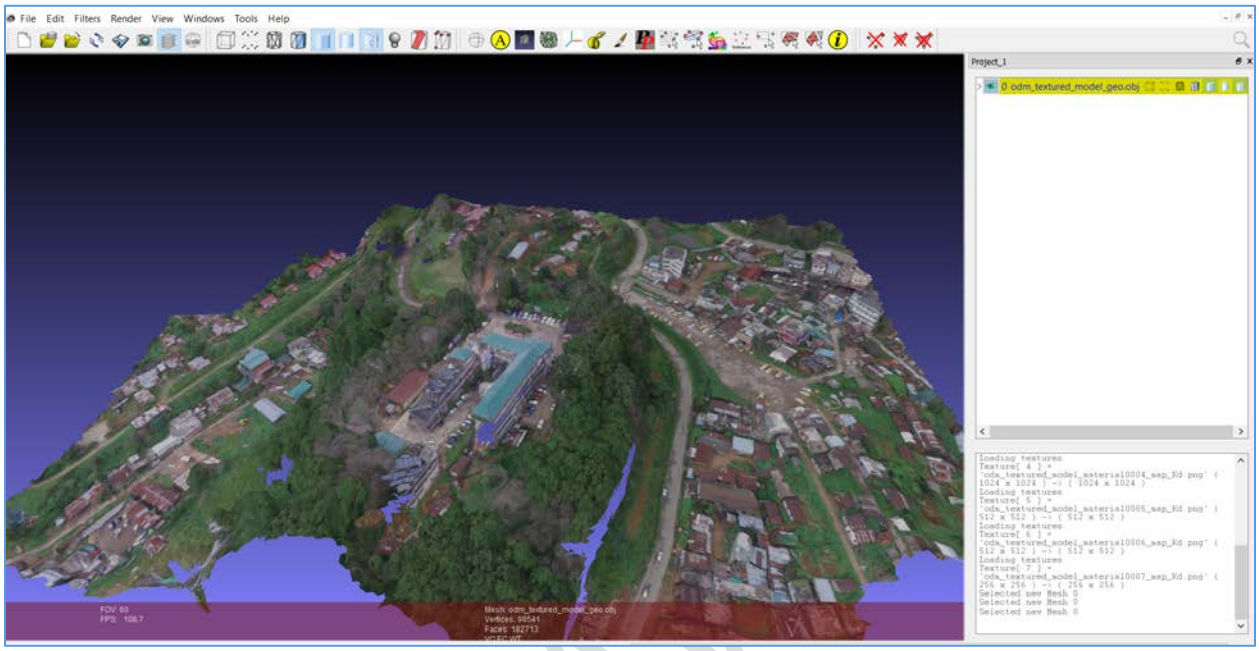
Observe the fine details in UAV Orthophoto

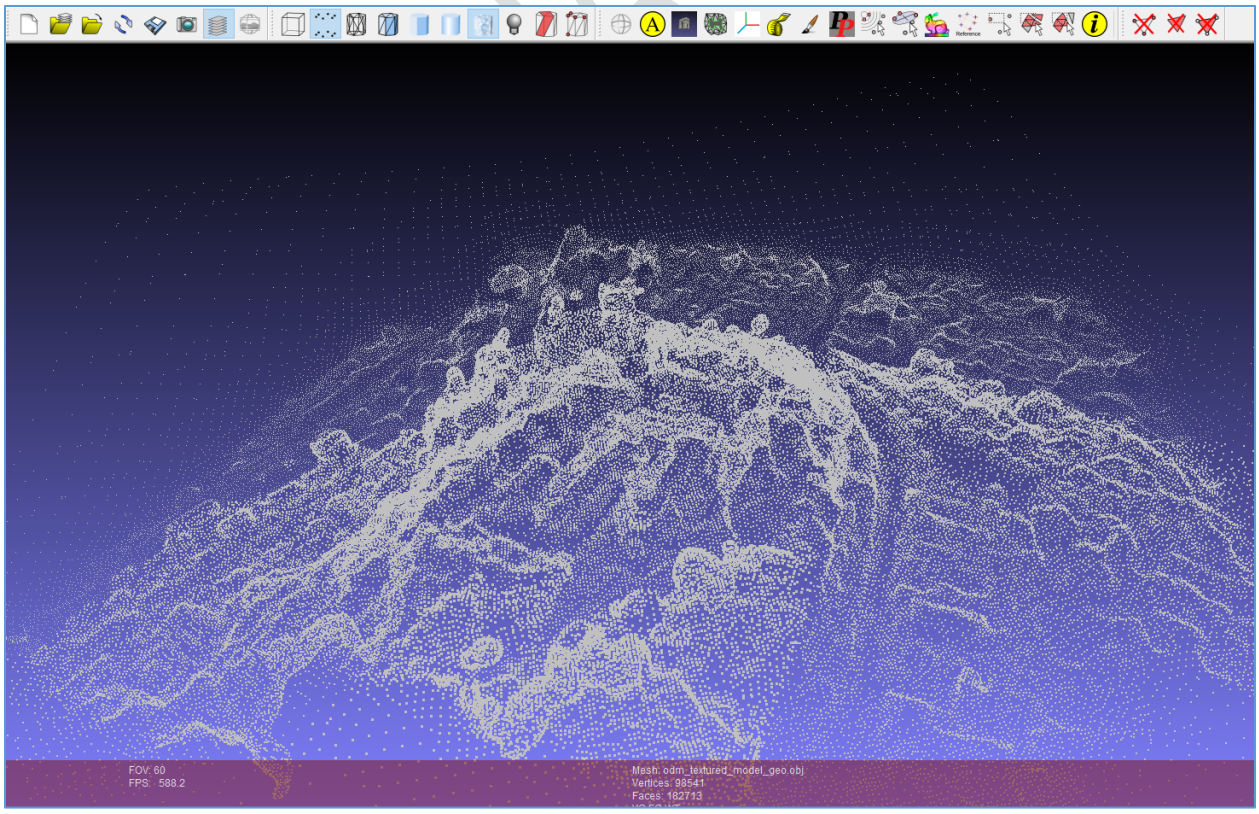
(Optionally) Use GCP for more accurate generation of Products.



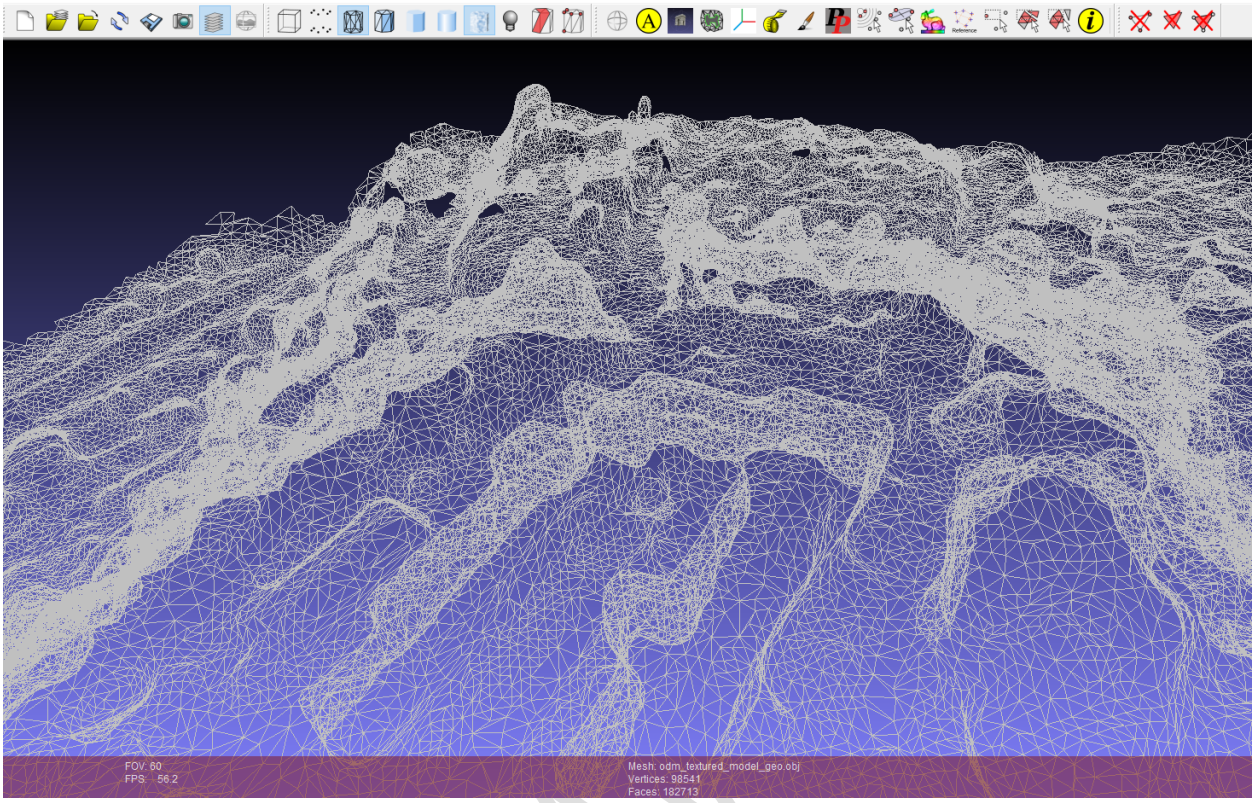
2. Meshlab : Point Cloud Editing, Meshing and Visualization

1. Load the .obj file (one of the output from WebODM) into Meshlab.

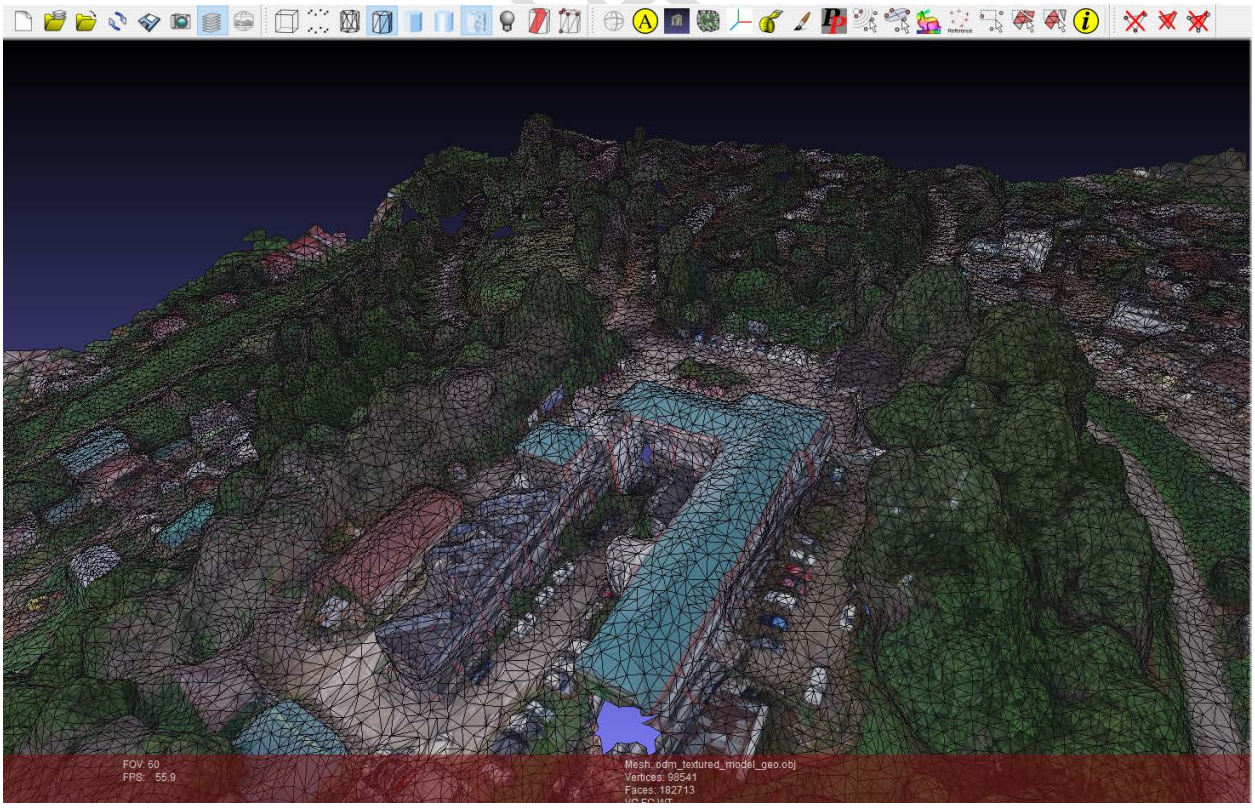




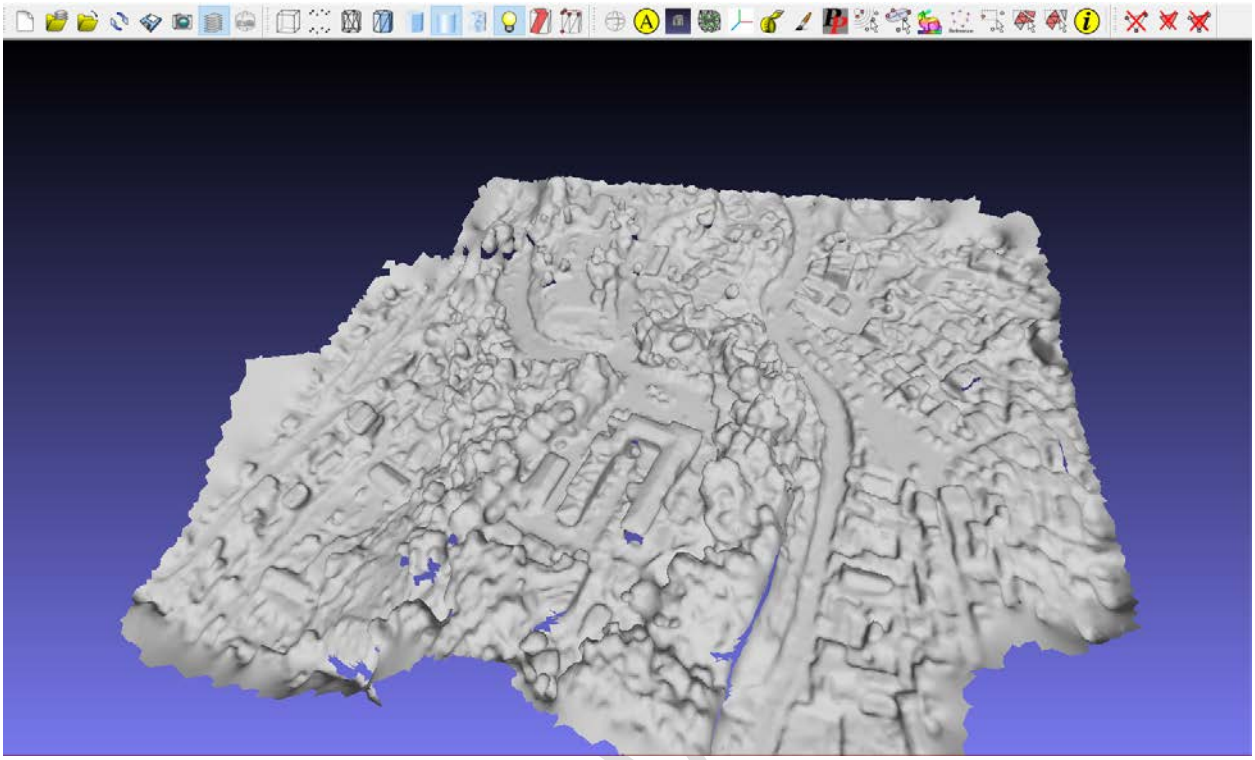
Observer the 3D Points used to create the 3D model



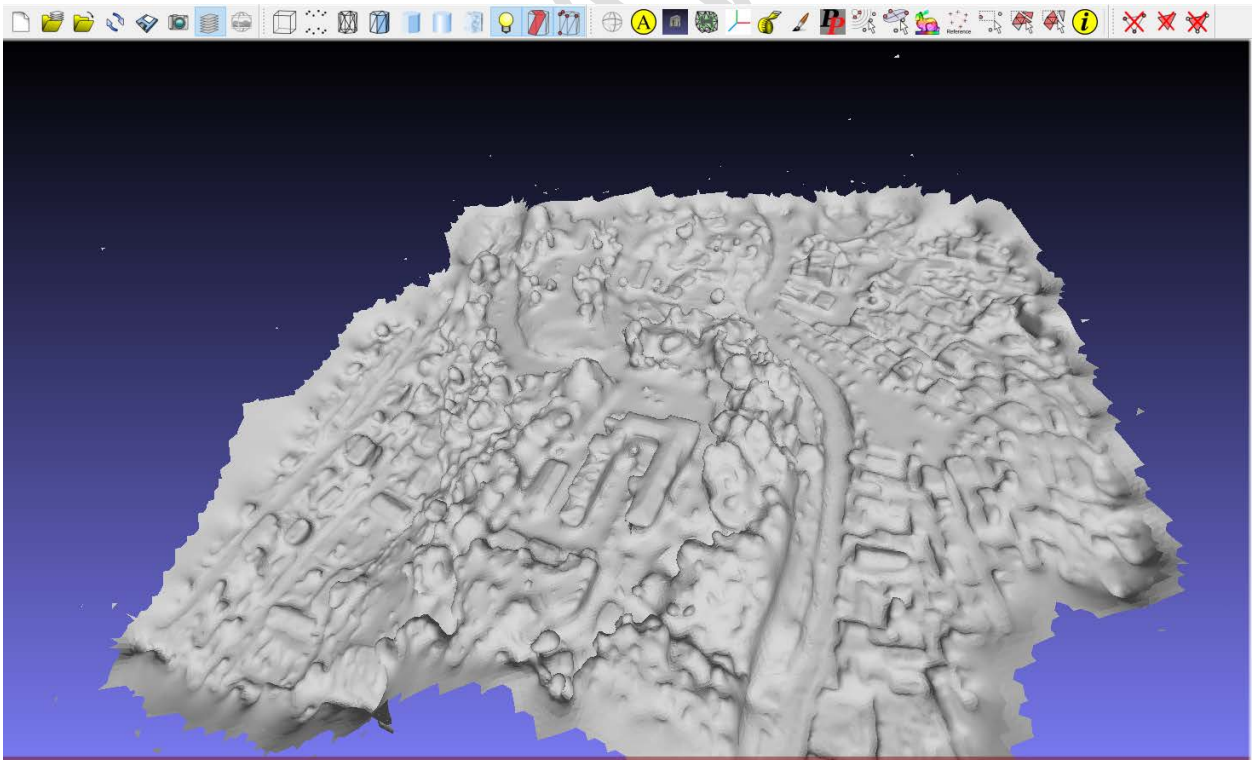
Observer the triangular facets formed by joining the 3D points to create 3D surface



Observe the Texturing of the Triangular facets to form 3D Textured Model



Shaded Model without Texturing



Filling of Holes from the Shaded Model with Surface Reconstruction Algorithms (Filters>Poisson Surface Reconstruction). **For More Details on Meshlab, Refer to next Lab Handout-2**

3. Processing UAV point clouds in GRASS GIS

Basic introduction to graphical user interface

GRASS GIS Spatial Database

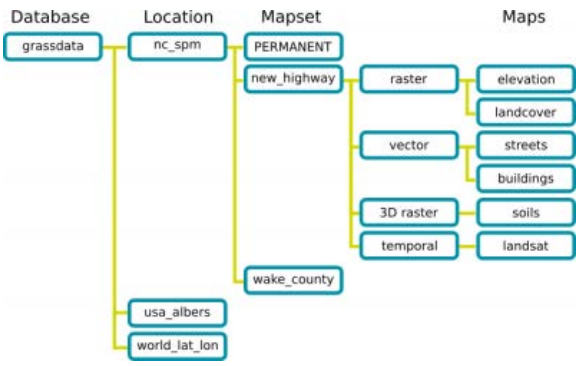
Here we provide an overview of GRASS GIS. For this exercise it's not necessary to have a full understanding of how to use GRASS GIS. However, you will need to know how to place your data in the correct GRASS GIS database directory, as well as some basic GRASS functionality.

GRASS uses specific database terminology and structure (GRASS GIS Spatial Database) that are important to understand for working in GRASS GIS efficiently. You will create a new *location* and import the required data into that location. In the following we review important terminology and give step by step directions on how to download and place your data in the correct place.

- A **GRASS GIS Spatial Database** (*GRASS database*) consists 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.

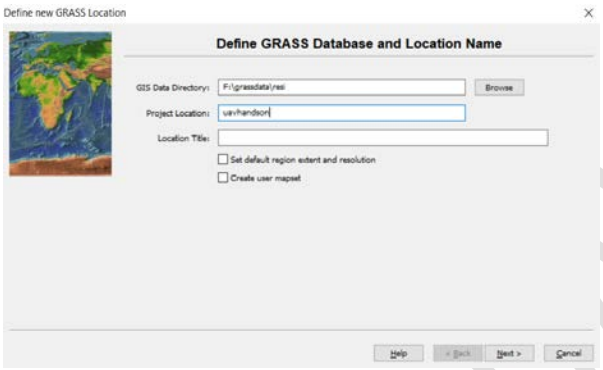
Start GRASS GIS, a start-up screen should appear. Unless you already have a directory called grassdata in your Documents directory (on MS Windows) or in your home directory (on Linux), create one. You can use the Browse button and the dialog in the GRASS GIS start up screen to do that.

We will create a new *location* for our project with CRS (coordinate reference system) with EPSG code 32646. 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*.

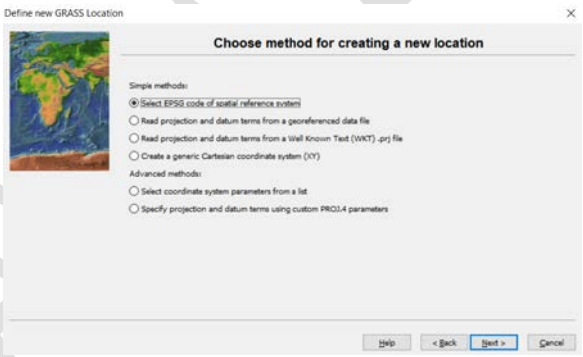


GRASS GIS Spatial Database structure

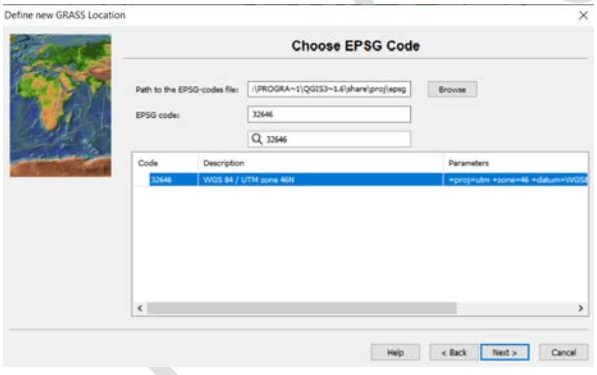
GRASS GIS 7.6.1 startup dialog



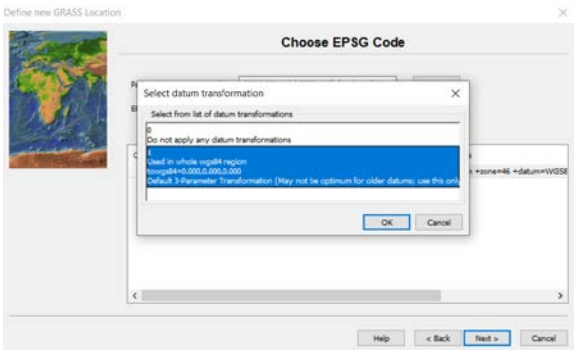
Start Location Wizard and type the new location's name



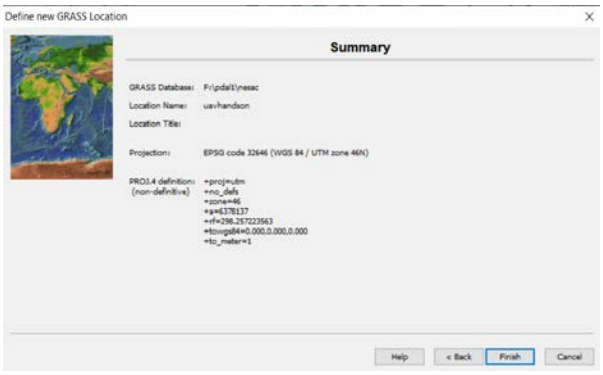
Select method for describing CRS



Find and select EPSG 32646



Confirm the use of the default datum transformation



Review summary page and confirm

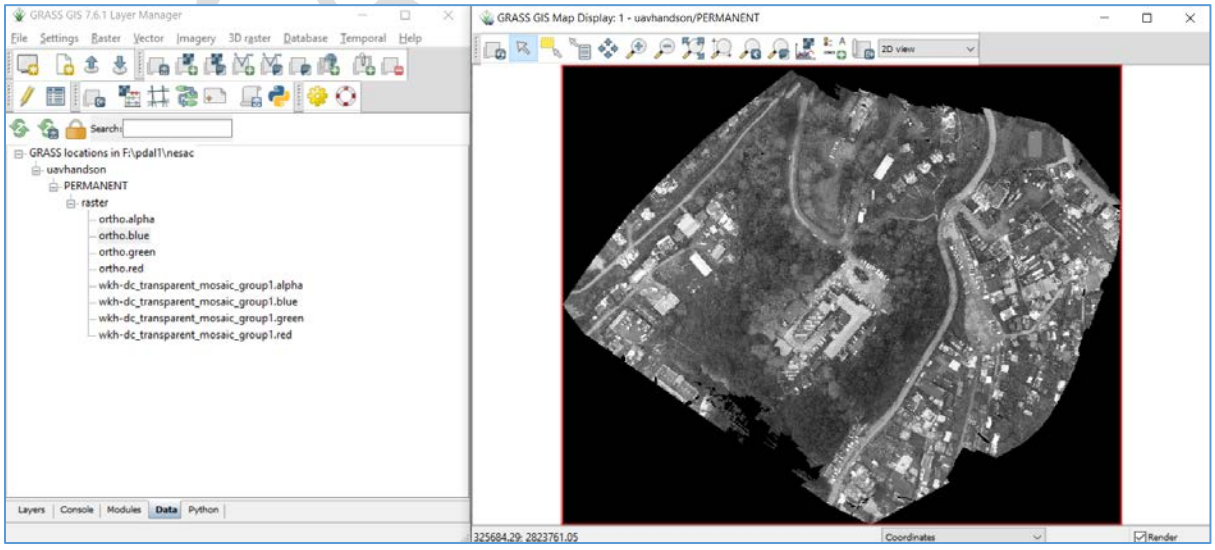
Note: Alternative and easy approach - If you already have a georefrence data (orthophoto etc). It can be used to read projection parameters from directly from this file without the need to specify the EPSG.

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 the downloaded LAS file. IN case we don't change the directory, we need to provide full path to the file.

Importing data

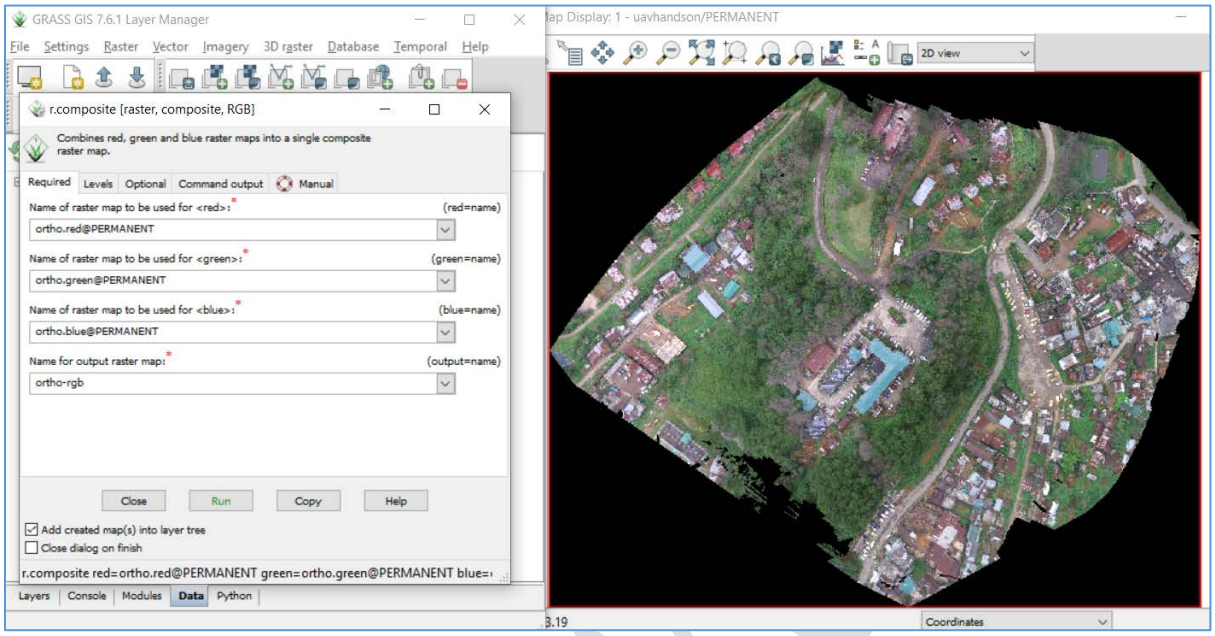
In this step we import the provided data into GRASS GIS. In menu ***File - Import raster data*** select ***Common formats import*** and in the dialog browse to find the orthophoto file, change the name to **ortho**, and click button ***Import***. All the imported layers should be added to GUI automatically, 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

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 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:

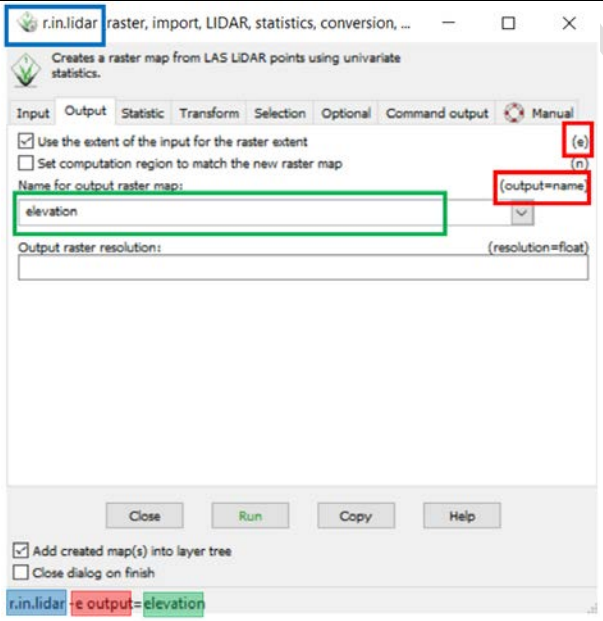
```
g.region raster=ortho -p
```

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

These 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.



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 analyze 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

Basic introduction to Python interface (*optional* Reading)

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

Here we use parse_command to obtain the statistics as a Python dictionary

```
region = gscript.parse_command('g.region', flags='g')
print region['ewres'], region['nsres']
```

The results printed are the raster resolutions in E-W and N-S directions. In the above examples, we were calling the g.region module. Typically, the scripts (and GRASS GIS modules) don't change the computational region and often they don't need to even read it. The computational region can be defined before running the script so that the script can be used with different computational region settings.

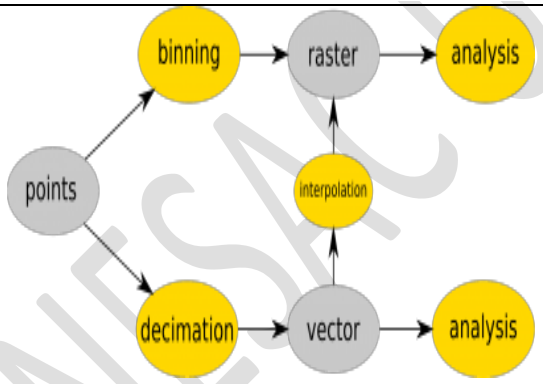
The library also provides several convenient wrapper functions for often called modules, for example script.raster.raster_info() (wrapper for r.info), script.core.list_grouped() (one of the wrappers for g.list), and script.core.region() (wrapper for g.region).

When we want to run the script again, we need to either remove the created data beforehand using g.remove or we need to tell GRASS GIS to overwrite the existing data. This can be done adding overwrite=True as an additional argument to the function call for each module or we can do it globally using os.environ['GRASS_OVERWRITE'] = '1' (requires import os).

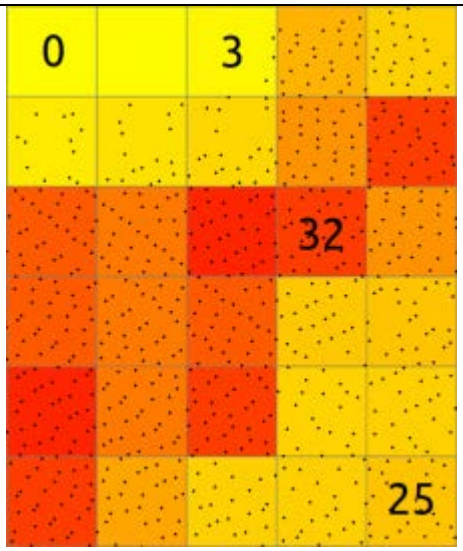
Finally, you may have noticed the first line of the script which says #!/usr/bin/env python. This is what Linux, Mac OS, and similar systems use to determine which interpreter to use. If you get something line [Errno 8] Exec format error, this line is probably incorrect or missing.

Binning of the point cloud

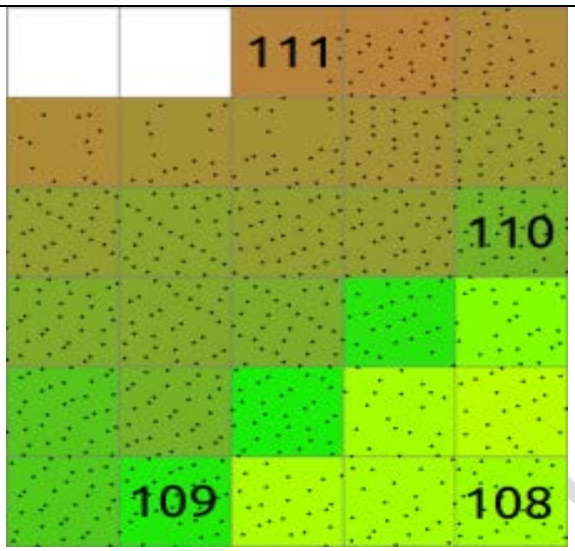
Fastest way to analyze basic properties of a point cloud is to use binning and create a raster map. We will now use **r.in.lidar** to create point count (point density) raster. At this point, we don't know the spatial extent of the point cloud, so we cannot set computation region ahead, but we can tell 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.



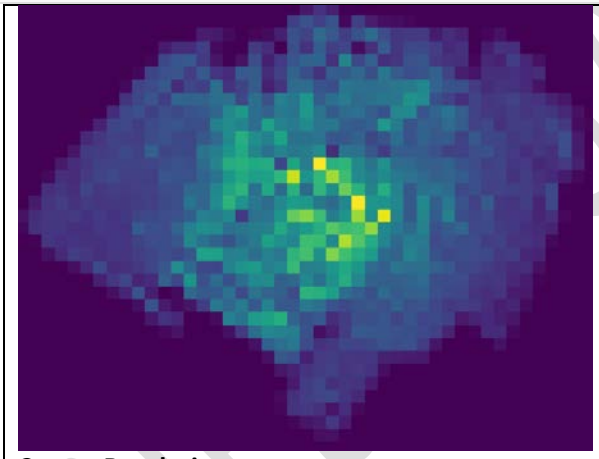
Point cloud is either binned into a raster (e.g. r.in.lidar) and then analyzed as raster or optionally decimated (e.g. using v.in.lidar), converted to vector (still using v.in.lidar) and then interpolated (e.g. v.surf.rst) into raster or analyzed as a vector (e.g. v.vect.stats). Data are in gray, processes in yellow.



In basic case, 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.

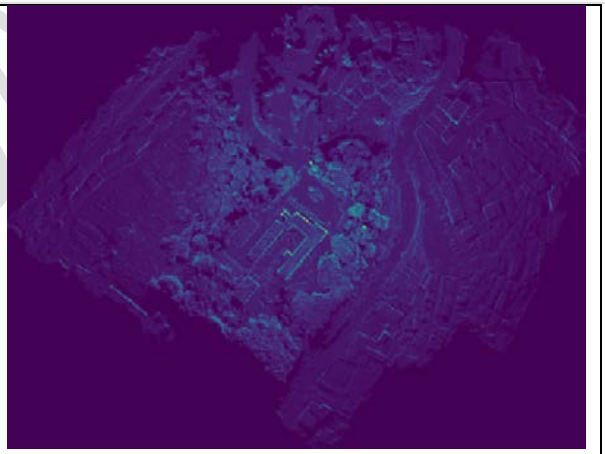


In general binning involves also values associated with the points and computes statistics on these values. Here the mean of Z coordinates of all the point in each cell is computed and stored in the raster. The cells without any points are NULL (NoData) shown in white here.



Coarse Resolution

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



Finer Resolution

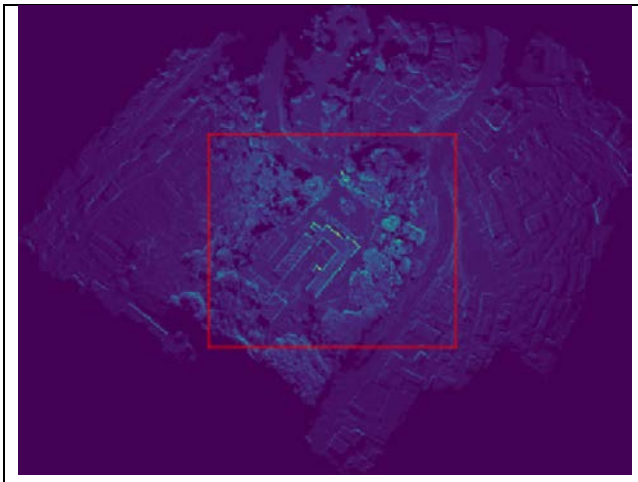
```
r.in.lidar input= wkhsource2019.las
output=count_1 method=n -e -n
resolution=1
```

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):

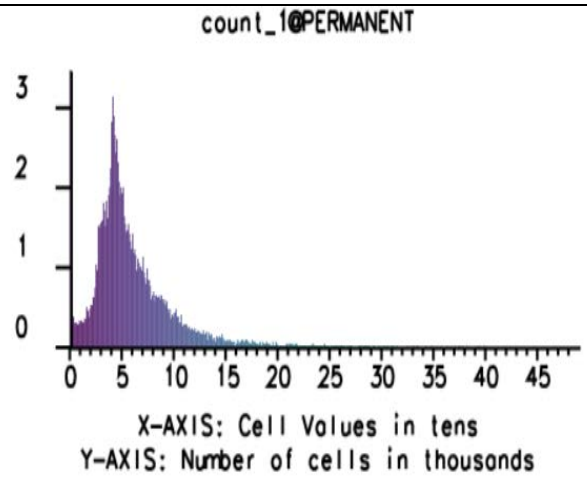
```
r.info map=count_10
```

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

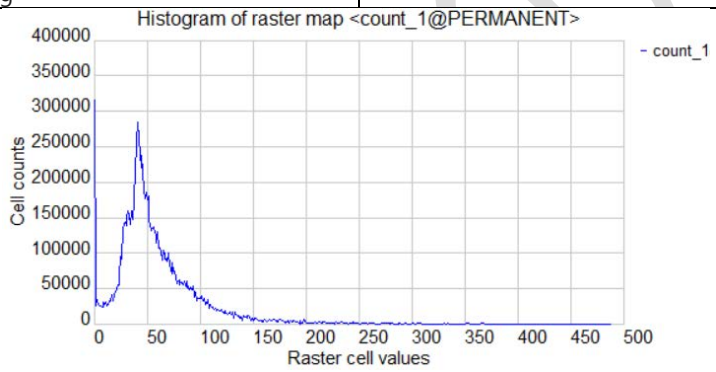
using the **d.histogrammodule** (**d.histogram map=count_1**). Select a computational region interactively by drawing a rectangle for which the histogram to be computed.



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

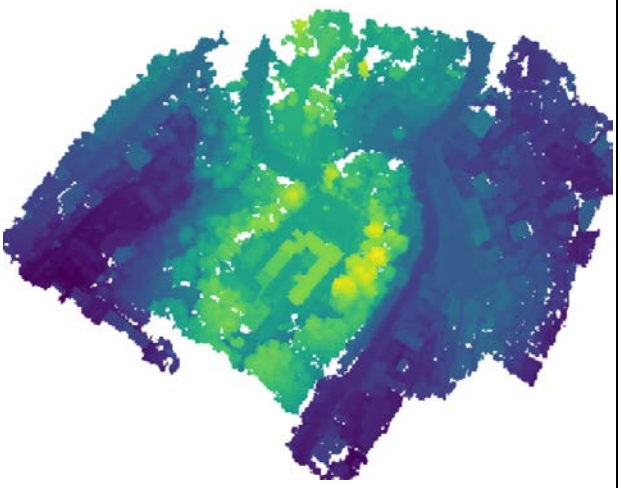


GRASS GIS Histogramming Tool (based on d.histogram)

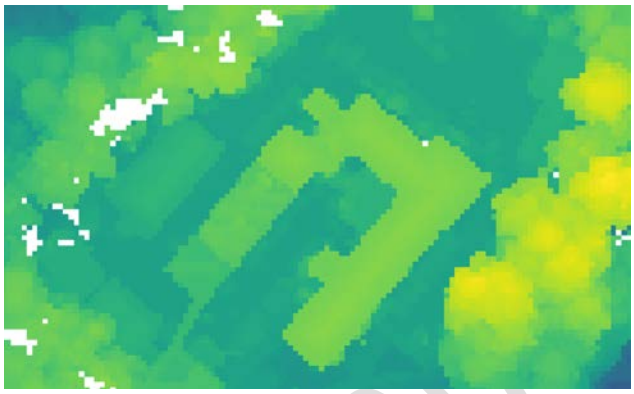


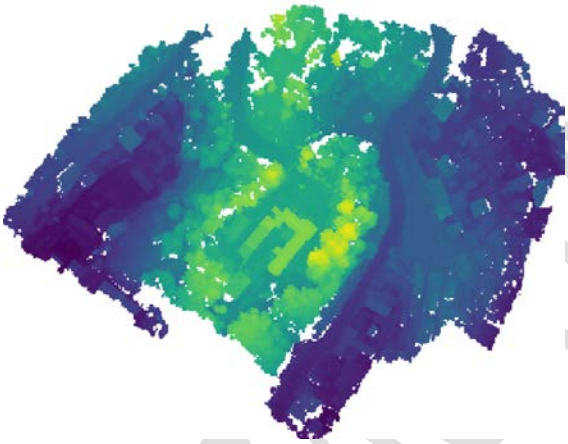
GRASS GIS Histogramming Tool (based on wxPython)

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):

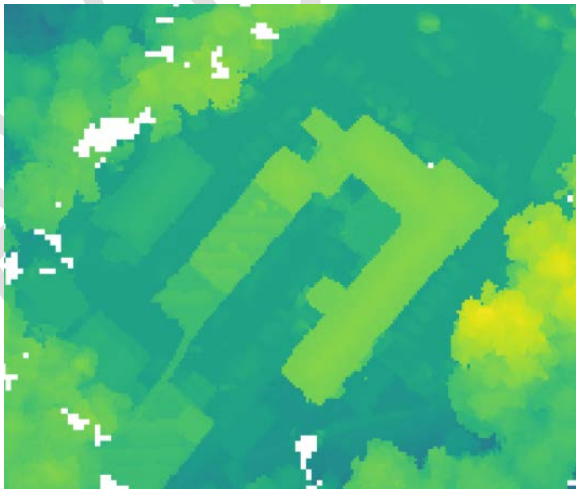


```
r.in.lidar input=wkhsource2019.las
output=binned_dsm_1m method=max
resolution=1
```



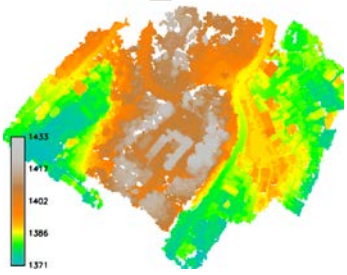


```
r.in.lidar input=wkhsource2019.las
output=binned_dsm_0_5m method=max
resolution=0.5
```

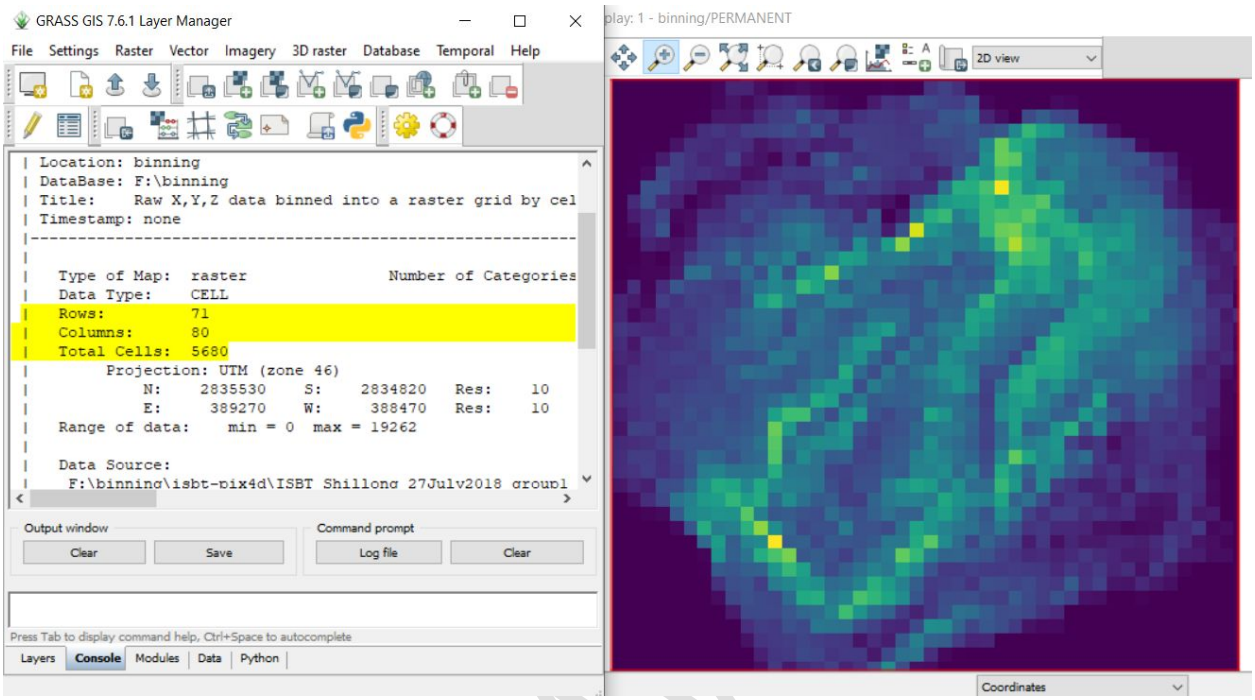


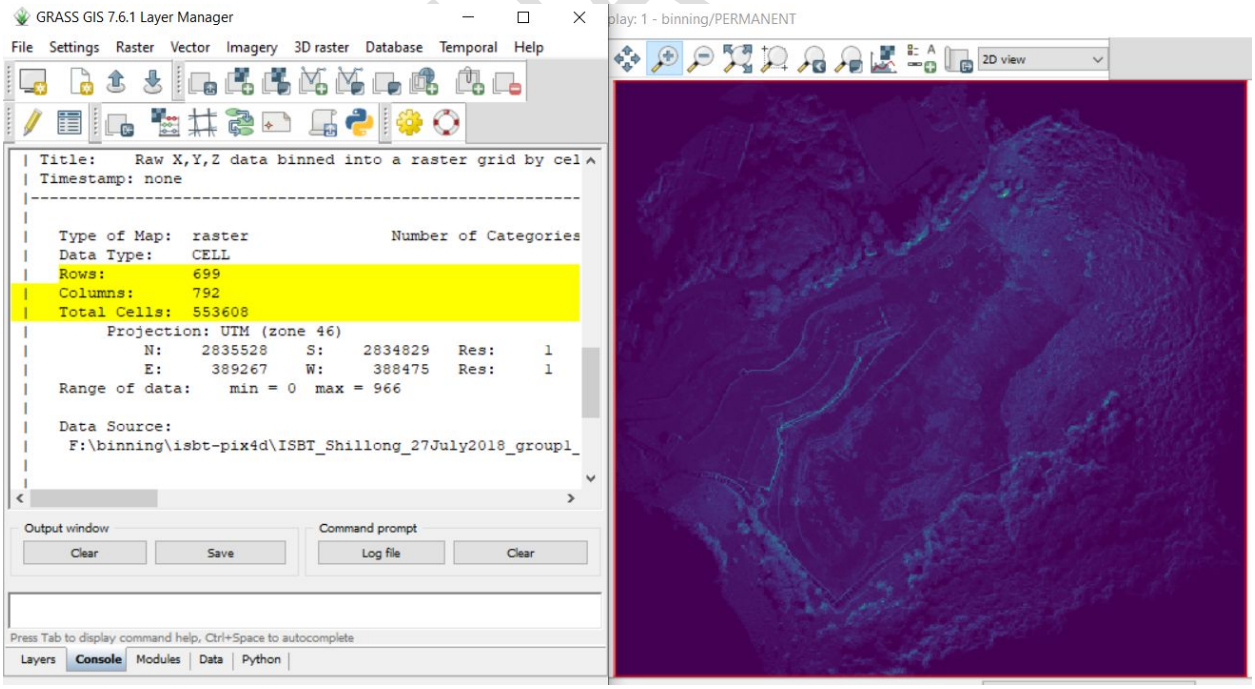
Colour the DSM with elevation colour.

```
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.





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