UAV-OS

Hands-On Guide

OpenDroneMap : Understanding Complete UAV Data Processing Pipeline Using Open Source

A Practical Approach

This detail hands on will guide you for Drone data processing Using OpenDroneMap. The entire process pipelines has been tested on real drone datasets captured by NESAC. You'll learn necessary concepts from ODM installation to how you can master the advance settings and tweak so that desired UAV data products can be generated.

PS Singh

2/27/2024

What You Will be Learning

- Self-install the ODM by yourself and make it Ready to Use!
- Understand and operate the powerful ODM Process Pipeline for Drone Data Processing
- Use ODM to generate useful Dense 3D Point Clouds, 3D Mesh Model and immediately usable geo-rectified (assuming images has geo-locations) Orthomosaics, DSM and DTMs etc
- Experiment the powerful Task Options and tweak to generate different data products as per the requirement with relevant screenshots tested on NESAC UAV images.
- Process huge volume of data by splitting datasets.
- Assess the accuracy of data products generated.

Learning Goal 01: Why You Must Use OpenDroneMap (ODM in short)

- OpenDroneMap is built based on open standards and has many features for effective generation of UAV data products. The users has been given a **freedom** and the flexibility to play around with different settings and configurations for generating the product of their choice *as oppose to the proprietary based software where you are given a black-box solution with no option but to trust that the outputs are correct!*
- The ODM therefore helps in understanding the insights of the UAV derived data captured by today's consumer grade inexpensive drones.
- Free as in Freedom: ODM gives the users the freedom to use, modify, examine, distribute the software! Further, To make ODM more accessible and powerful, ODM can be installed with no cost on various flavours of Operating System and make it more powerful by scaling it to hundreds or thousands of clusters with no licensing cost.
- Best Software: ODM has best features among all existing Open Source Drone Processing Software.
- Highly Configurable : A highly reconfigurable UAV processing pipelines for immediate generation of the desired data product.
- Use of ODM in wide variety of applications - Although OpenDroneMap has been built for processing of aerial imagery usually captured from UAVs, the ODM can also be used in other allied fields such as architectural and archaeological heritage mapping, building indoor 3D mapping and modeling for which photos were captured with consumer grade handheld camera or phones.
    - In case of UAV derived images, ODM can be used to generate immediate ready to use and useful product such as sparse/dense geo-referenced point clouds, orthomosaics, digital terrain/surface models and other useful maps such as textured maps and 3D models etc – with our without GCPs/GPSs information.
    - ODM has provision for adding GCPs for enhance accurate data products.
    - Some of the immediate applications where ODM can be used are : Crops growth monitoring, Mapping of land use, monitoring of infrastructure construction and its progresses, Classification tasks such as tree and their counting trees, Analysis of stockpile volumes, Inspections of roofs and mobile towers, verification of work completions and even stitching of historical aerial images and create 3D models!

Learning Goal 2: Installing and getting ready for the Software

- **Components available under OpenDroneMap** are : [1]**ODM** is the core processing engine and to be used from the command line approach. It does take 2D images as its input and generates variety of outputs such as 3D sparse/dense point clouds, 3D models representing surface and terrain and orthomosaics. [2]**NodeODM** is an API built on the platform of ODM. With this, It will allow various ODM users for accessing the functionalities of ODM over local/remote network. [3]**WebODM** is web based interface and has 2D/3D map viewer and exposes complete required functions for taking 3D images and generate outputs. Further, it has configurable task functions which can be tweak and changed based on the product we want to generate. [4]**CloudODM** is a client for communicating with ODM through NodeODM API. [5]**PyODM** is Python based SDK that allows to create tasks via the NodeODM API. [6]**ClusterODM** act as a load balance between multiple NodeODM instance.

- **ODM on Docker**: Usually OpenDroneMap runs via a program called as docker. Docker is a platform for running multiple *containers* that packages necessary dependent tools and libraries required for the specific software. These containers than are run within a virtual environment. On Ubuntu Linux 16.04 and above, it's more comfortable to run ODM components natively. For Windows, it has to be run on a Virtual Machines (VM) (VirtualBox) with little overhead. Recommended system requirements can be Latest generation CPU with atleast 100 GB of disk space of free space and 16 GB RAM.

  Note: System CPUs with more cores and RAM size allows for faster processing. Presently graphics card (GPU) has no visible impact on processing performance.

Installing OpenDroneMap on Windows

Step 1. Enable Virtualization Support in the 64 bit Windows OS.
Step 2. Install **Git** and **Python** (the latest 3.x version)
Step 3. Install Docker Toolbox/ Docker Desktop
Install one of them but do NOT install both of it!
Step 4. Allocate 60-70% of available memory and 50% of CPU processor (via VM)
Step 5. Download/Clone WebODM via Git
Step 6. Launch WebODM

From **Git Bash** type the following to start installing WebODM:

```
$ ./webodm.sh start
```

This will download and install OpenDroneMap components such as WebODM, NodeODM and ODM.
When done successfully, start the WebODM from a web browser using the following URLs:
http://localhost:8000 (**Docker Desktop**). In case of **Docker Toolbox,** first find the docker-machine IP by typing `$ docker-machine ip` and use the output URL for opening the WebODM

Installing OpenDroneMap on Linux based OS

For installing ODM on Ubuntu, the programs to be installed on prior are Docker, Git, Python, Pip
The following are commands for installing on Ubuntu:

Step 1. Install the initial 4 programs

```
$ sudo apt update
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sh get-docker.sh
$ sudo apt install -y git python python-pip
```

Step 2. Install it **Docker-compose** by using **pip:**

```
$ sudo pip install docker-compose
```

Step 3. Download and Launch WebODM

```
$ git clone https://github.com/OpenDroneMap/WebODM
$ cd WebODM
$ ./webodm.sh start
```

The above command will start WebODM and also sets up a processing node by default (node-odm-1). For stopping WebODM, press **CTRL+C** or alternatively use the following command: $ ./webodm.sh stop

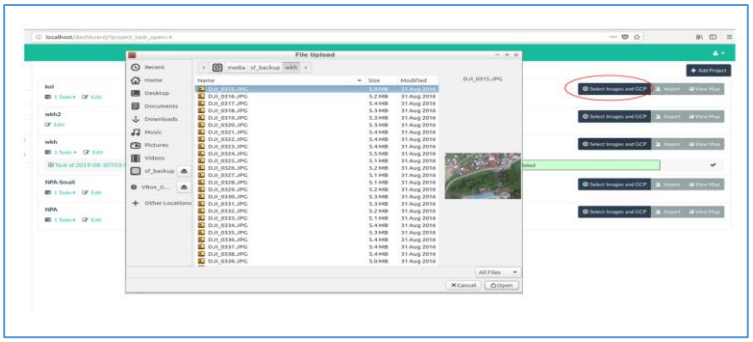
Type the following command, if you wish to run WebODM on other available ports instead of default 8000
$ ./webodm.sh restart --port <port number>

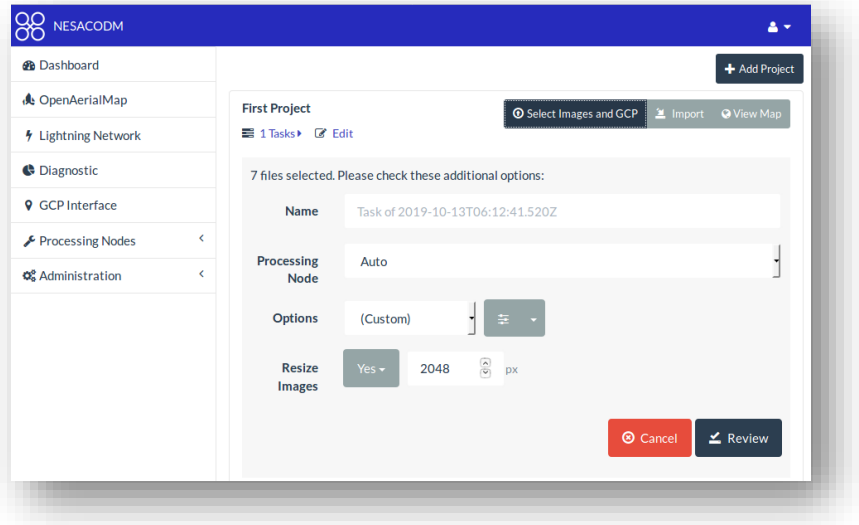
Once **./webodm.sh start** and opens the WebODM in the browser, Note that under the **Processing Nodes** menu there's already *node-odm-1* configured for immediate use. The WebODM now is ready for processing the UAV data. Start with few images to test the entire working pipelines and understand the system. Start with 100 or less images.

Learning Goal 3: Understanding the ODM GUI, explore its functionalities, processing of Drone Image and visualize its output.

Presently, the OpenDroneMap supports only JPEG files. But the beauty is that the ODM **can process for images that are captured by different cameras with varying sensors** and also taken from different viewpoints and angles. In case of images captured by drone/phones, the geo-locations of the images are generally stored in Exchangeable Image File Format (EXIF) tags. These information were then used by the software to create geo-referenced orthomosaic and 3D elevation models. **But the software can also accepts with no geo-location information to generate dense 3D point clouds or 3D models** which does not require geo-referencing but cannot be used to create orthomosaic. In such cases, Ground Control Point (GCP) points must be added to the images to create more accurate and geo-referenced data products. Software such as GIMp may be used to check view Metadata of the images. Further, it may be used to add/modify existing GPS information.

To start processing UAV images, click (or drag/drop) on **Select Images and GCP** button.



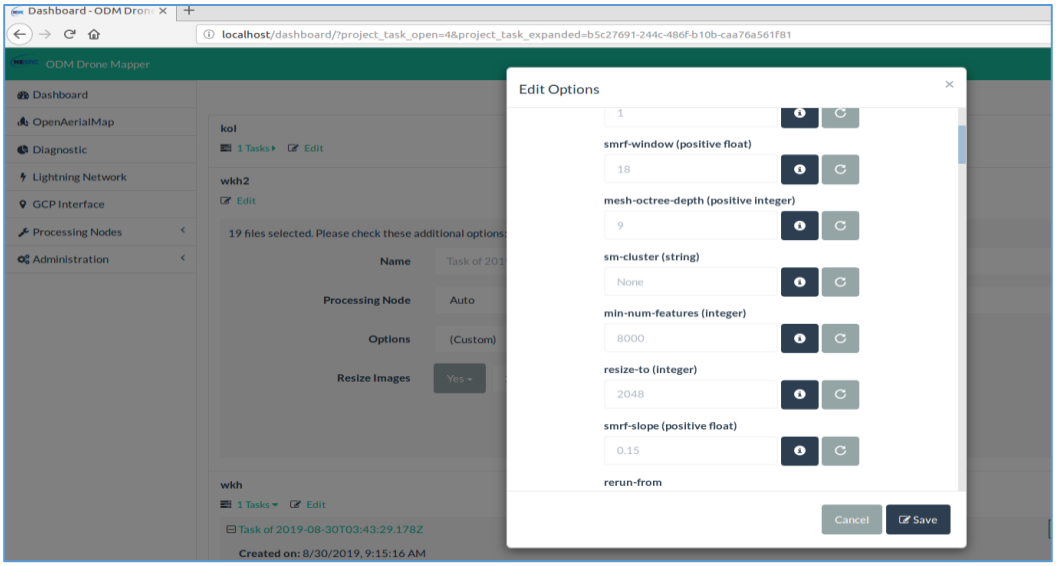


There are a few settings you can choose:

• **Name:** Give a name for the job.

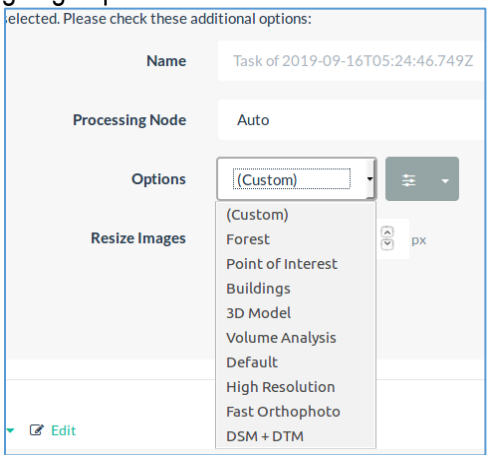
• **Processing Node: Auto** will simply pick a processing node for you or manually select a processing node in case there are multiple processing nodes.

• **Options:** Choose one from the predefined list of templates with default presets. We can also create other default settings based on our requirements. For now simply choose the **Default** preset.

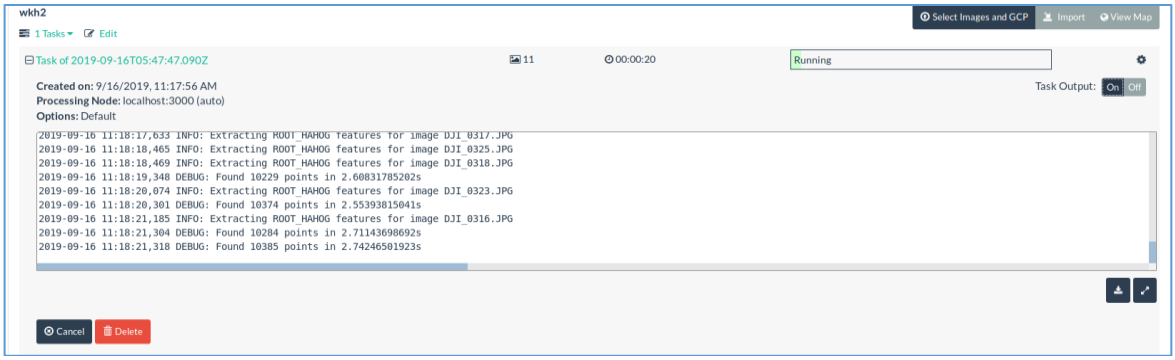


There are several components involved in the data processing pipeline. **Each component has several adjustable settings that influence the output**. The ODM exposes a subset of these available knobs through various options. When creating a task, a user can choose to tweak one or more options to change 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! Pl note that, the 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!.

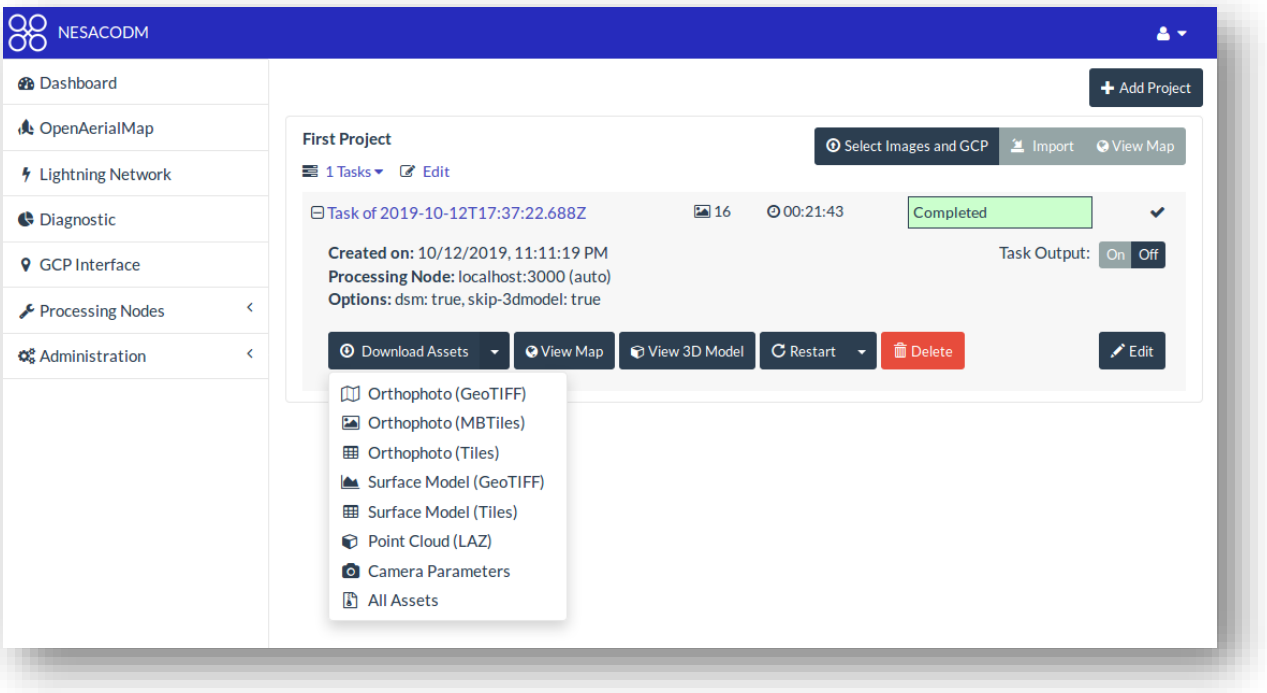
To start with, you can also choose the **predefined option templates** based on the type of scene you are going to process.



• **Resize Images:** Select this for less memory usage and to lessen overall processing speed. **Review** and then 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. A task now will be started and necessary information such as time elapsed, console output are refreshed and displayed for every 3-4 seconds. After the task is completed the output results are transferred from the processing node to WebODM.



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



Output results assets can be downloaded or viewed using 2D/3D interfaces.

View Map: Now, let's start checking our data products. Clicking on 'View Map' will opens a 2D viewer windows where you can **interactively visualize** 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.

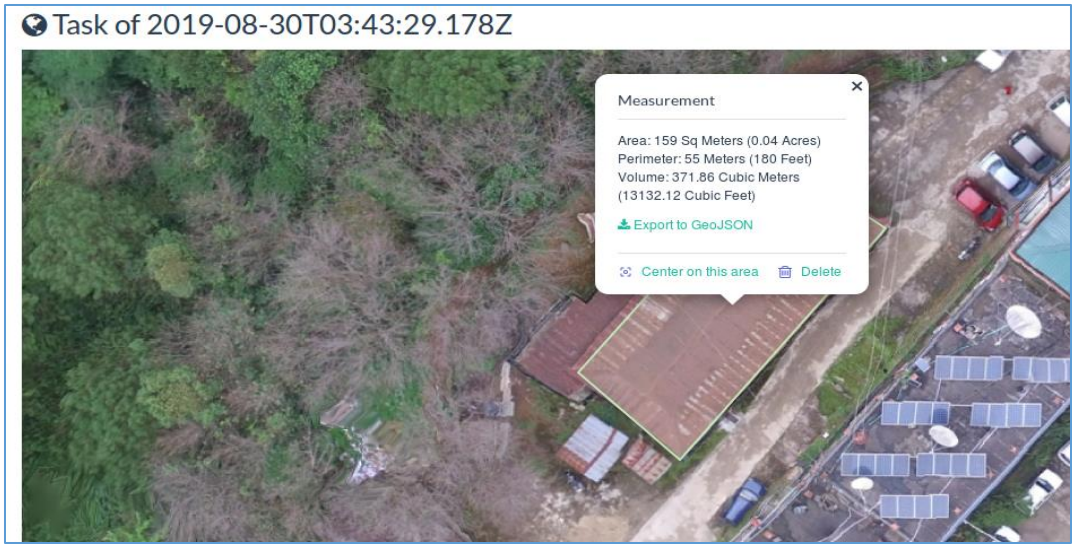


Figure: Measurement of Area, Perimeter and Volume on an object

Figure: Google Map View of the Area under Study

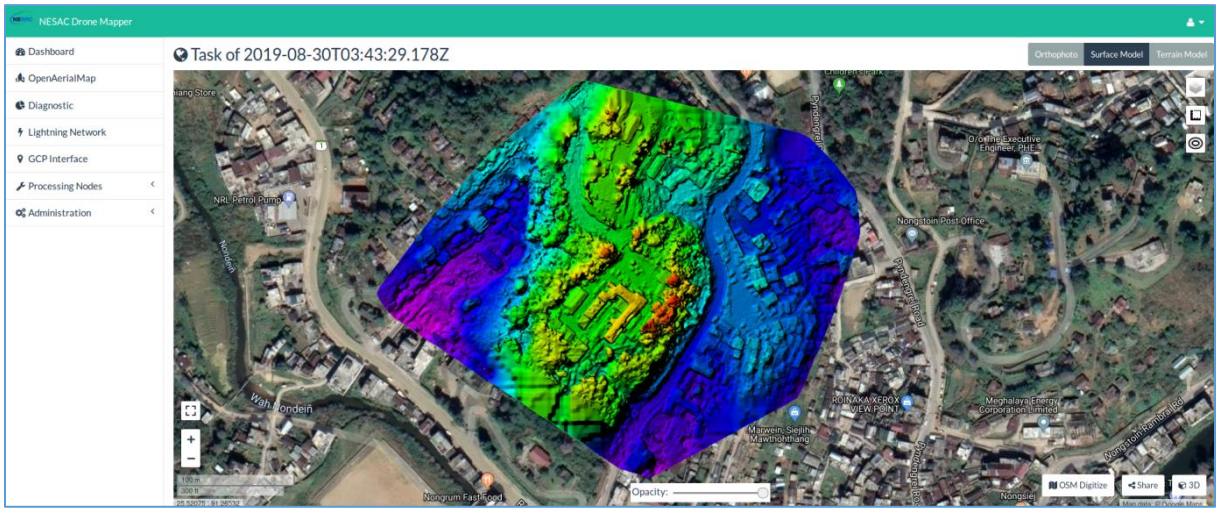


Figure: The Digital Surface Model

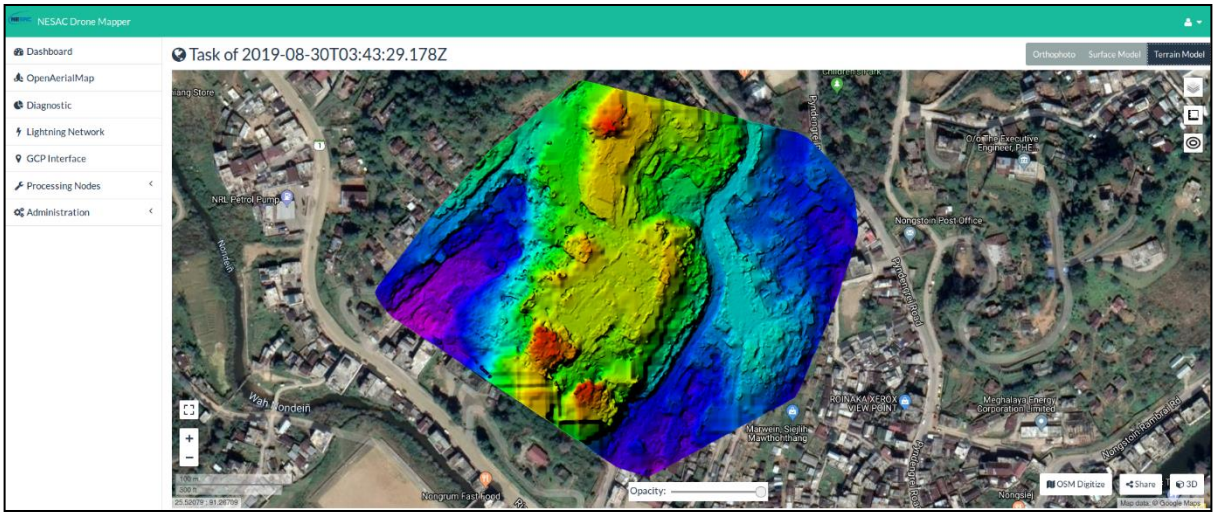


Figure: The Digital Terrain Model

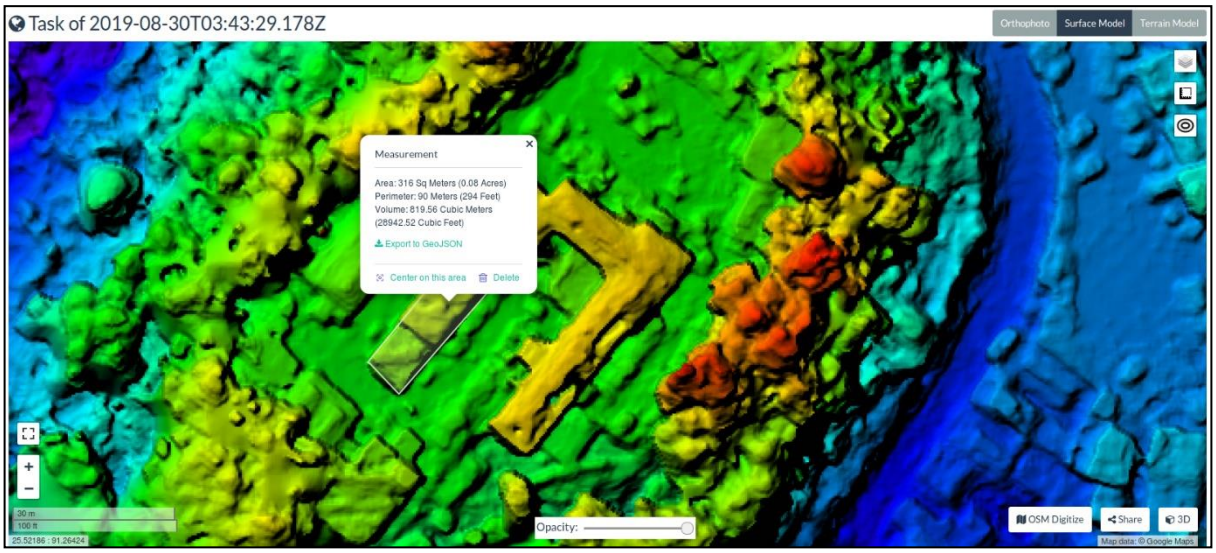


Figure: Measurement Using The Digital Surface Model

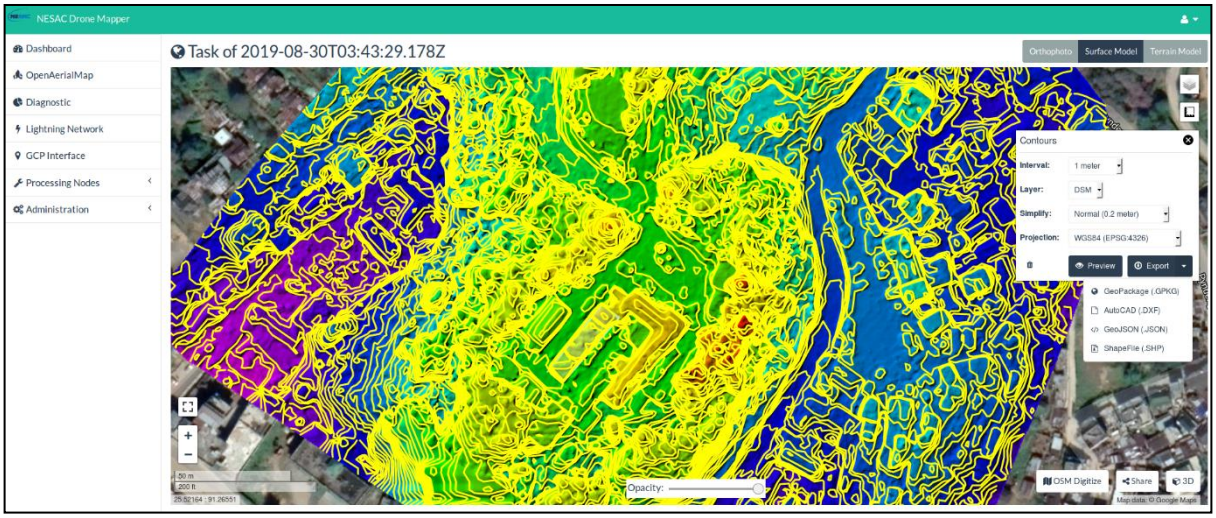


Figure: Generation of Contour Using The Digital Surface Model

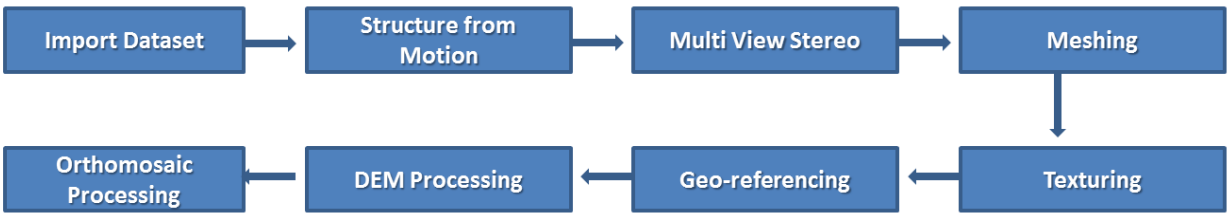
View 3D Model **:** : The 3D Map viewer allows you to **visualize the dense 3D point Cloud / 3D Texture Model**. You can manage the visualization by changing its settings in appearance. Now, you can also **perform analysis of the point cloud with regard to measurement of distance, area, height, volume,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

Download Assets — Clicking on **All Assets** button, a zip file will be downloaded containing following directories:

• **dsm_tiles** should contains the tiles for the color shaded digital surface model. Tiles are useful for it can be used to display the product on the web viewer such as Cesium or Leaflet.
• **dtm_tiles** tiles for the digital terrain model.
• **orthomosaic_tiles** tiles for the orthomosaic.
• **odm_dem** for the digital elevation model files.
• **odm_georeferencing** has the dense point clouds.
• **odm_texturing** basically stores the textured 3D model (geo-referenced/geo-referenced version).
• **entwine_pointcloud** the point cloud that can be streamed over the web.

Learning Goal 4: ODM Process pipeline components and what each components does/contributes for processing the images.

The process for generation of images to 3D models and orthomosaic consists of **incremental** steps/blocks of process modules where each **successive module depends on the output of the previous module** as depicts in the diagram below. Each module has been explained below.



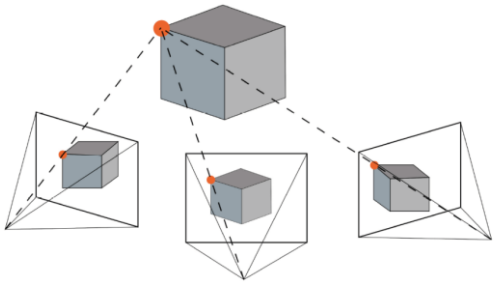
Step 1 : Import Dataset
Input**:** images + GCP (optional) Output**:** image database
There may be instances where Input Images can be corrupted where few images may contain full or partial GPS coordinates (embedded within the EXIF tags). This module thus **counts the images, extracts dimensions and parses GPS information from all available images** and exclude the corrupted images.

Step 2 : Structure From Motion
Input: images + GCP (optional) Output: camera poses + 3D sparse point cloud + transform
Structure From Motion (SfM) is a photogrammetry based computer vision technique which is primarily meant for **estimating 3D scene structures given multiple overlapping image sequences** (from the UAV motion taking pictures as it fly along). SfM **solves for camera pose and estimation simultaneously** along with the scene reconstruction given overlapped images taken from multiple view-points. SfM does well in accurately calculating the pose and orientation of the cameras. Further, **it doesn't require any GPS information**. SfM performs the following steps sequentially:

[1] First it **extracts the camera information** from the images' EXIF tags (if its available). The information from EXIF tags is used to initially estimate of the camera parameters (focal length, sensor size and others which is then refined in subsequent steps later on.

[2] Every images is run for distinguishable **feature detection or keypoint or interest point detection**. The features of interest may be the edges, the corners, the blobs or other unique features present on the image. This is very important step as robust and well detection of interest points will decide in accurately deriving of the UAV data products.

[3] For every image pairs, the **keypoints are compared or tracked based on its feature descriptors** defined for every keypoints detected and matched. Then optimizations are run to discard likely impossible matches or images which are far from each other using GPS information and or by thresholding. Initially, the process starts with a good image pair and add more image. This way, the SfM recovers both the external orientation of the camera and the 3D scene structure resulted in creating the 3D sparse point cloud.



SfM for 3D model reconstruction

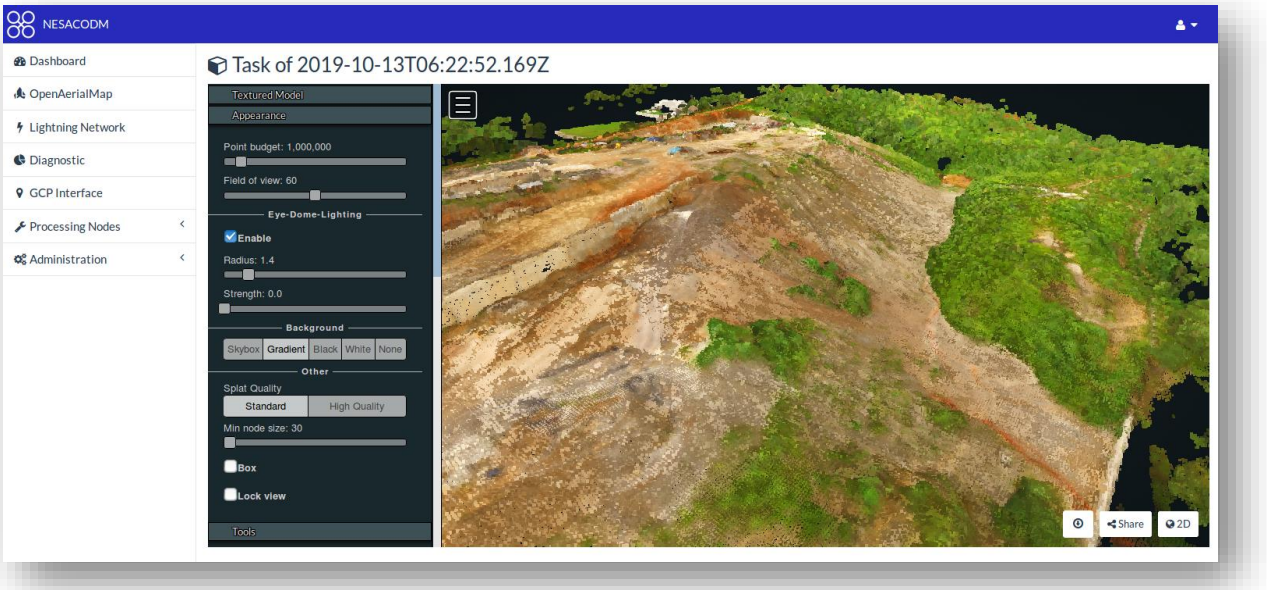
ODM is dependent on OpenSfM(Open Structure From Motion) for efficiently solving the SfM problem and **generating sparse 3D point cloud**.

Step 3 : Multi View Stereo

Input: images + camera poses + (sometimes) sparse 3D point cloud Output: dense 3D point cloud

SfM gives the sparse point cloud, For 3D scene reconstruction, we need to **densify the sparse point cloud**.

Multi-View Stereo (MVS) uses the principle of Multiple View Geometry to start looking for possible keypoint matches where searching of the corresponding keypoint on the other image is carried out (in 1D search) on its epipolar line. This way, it can find many more keypoints previously missed by the SfM and starts building more 3D keypoints from these new correspondences via triangulations followed by bundle adjustments that minimizes or removes the re-projection errors. ODM presently offers 2 options for carrying out MVS: Multi-View Environment (MVE) and OpenSfM.

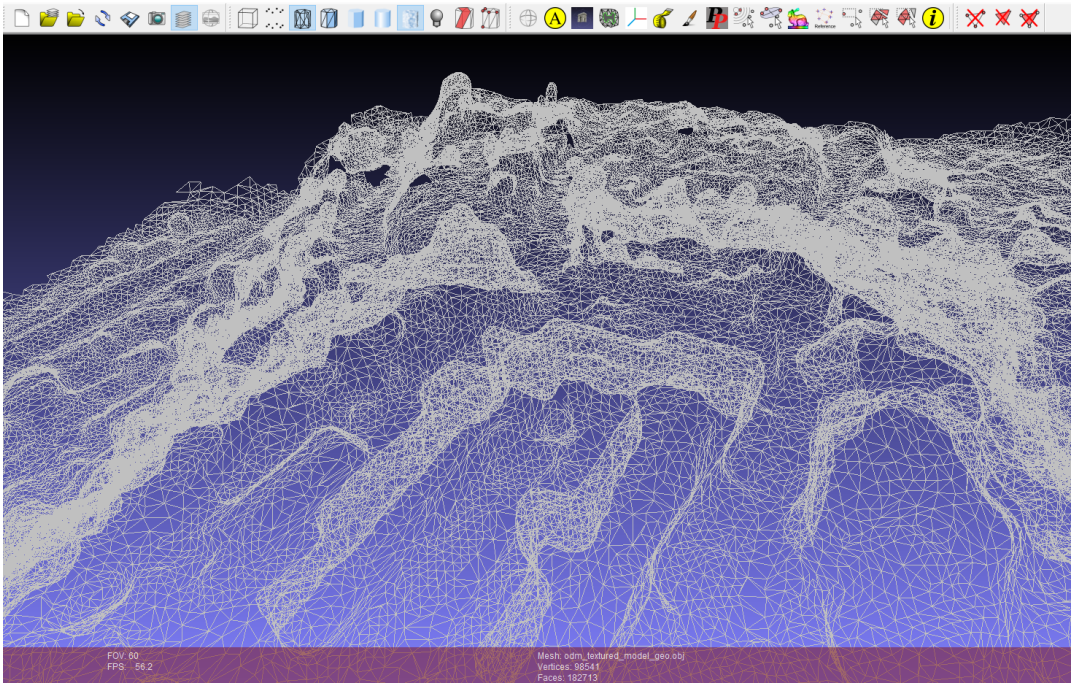


Lots of 3D points make a dense point cloud

Step 4 : Meshing

Input: Dense/Sparse 3D point cloud Output: 3D and 2.5D meshes

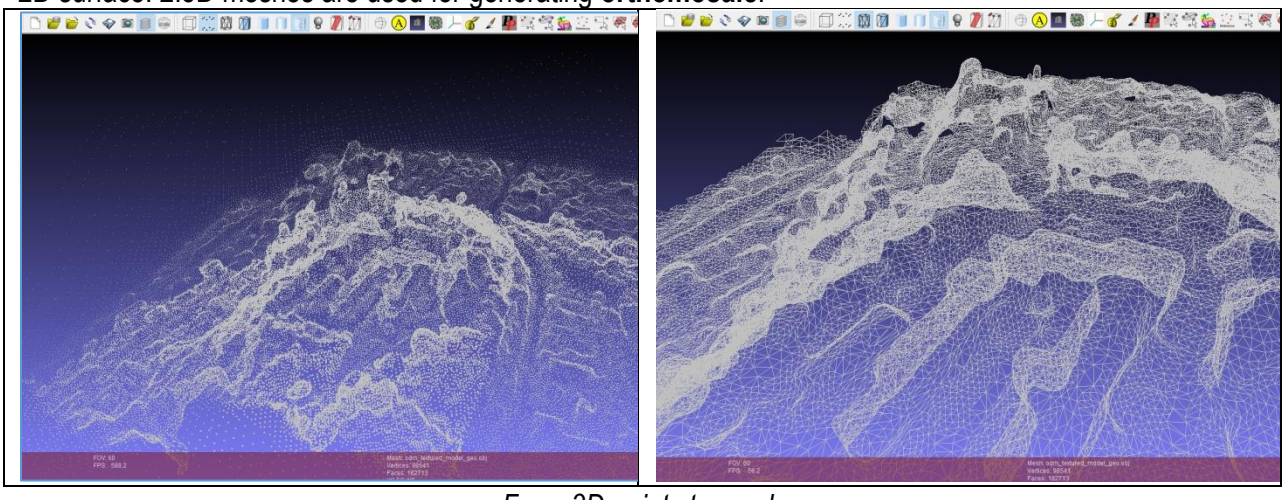
The 3D models are more appropriately called *polygonal meshes* or *meshes* for short.



3D mesh

For generating the polygonal meshes from the 3D dense/sparse points, the following steps are performed:
1. “Join the dots” forming multiple triangles and obtain a mesh. Here, the points may be moved or deleted to create a better and refined looking mesh.
2. Add colour (from the original 2D images) to the mesh created. This process is referred as *texturing*.

Meshing is the process of “joining the dots”. ODM currently supports 2 different algorithms for polygonal meshing and uses one of the two based on the images and the user defined settings/configurations:
1. **Screened Poisson Surface Reconstruction** for highly robust and memory algorithm for creating 3D surfaces and used for generating full 3D models with a high accuracy.
2. **dem2mesh** is a program for generating **2.5D** meshes (meshes that look 3D with simple *extrusions* of a 2D surface. 2.5D meshes are used for generating **orthomosaic**.

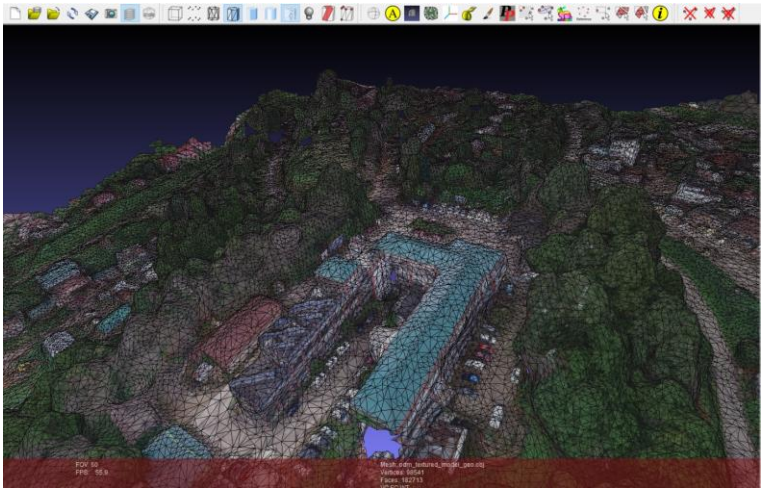


From 3D points to mesh

Step 5 : Texturing

Input: images + camera poses + meshes Output: textured meshes

Initially, the meshes does not have any colours information associated with it. Texturing thus is the process of giving the colours to meshes. This is done via specially computed images called texture images and by then each polygon is assigned a part of the texture images. This process is done with a tool called *Mvs-Texturing.*



Textured mesh (A place in Meghalaya)

Note : There being a randomness in the Mvs-texturing algorithm, the results may differ slightly even though its run from the same mesh.

Step 6 : Georeferencing

Input: transform + point cloud + textured meshes Output: geo-referenced point cloud + textured meshes + crop Boundaries

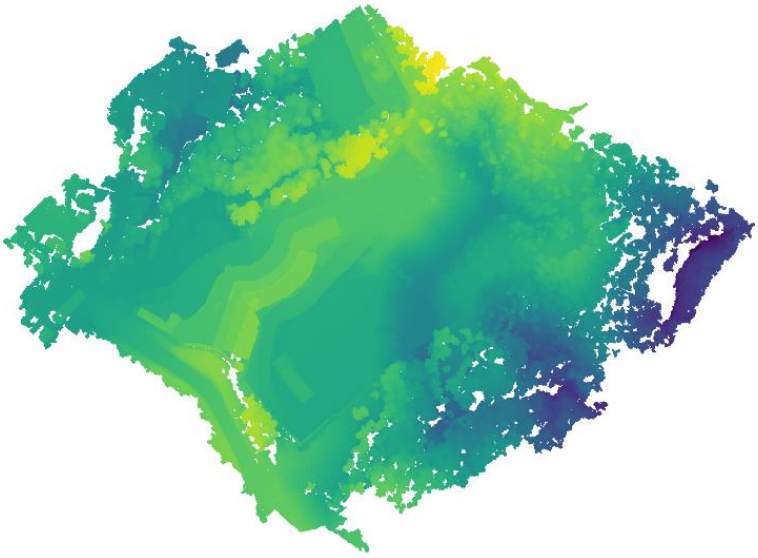
The 3D models and meshes formed on the previous module will be represented on a *local* coordinate system (a made up coordinate system) which therefore has no correlation to real world positions. Georeferencing process will do the task of **converting/*transforming* this local coordinate system into a world coordinate system**. But, ODM can do this only if the GPS coordinates available with the input images or a GCP file are provided before processing the images. The GPS coordinates if available, are used during the SfM step for aligning the 3D scene reconstruction. On the other hand, if GCPs are provided, the GPS information will be ignored and instead the GCPs will be used for the alignment to enhance the accuracy. One of the outputs of the SFM module is a *transform* file. This will convert 3D point clouds/models from local coordinate to world coordinates. After georeferencing, this point cloud is used for generating an estimate of the geographical bounds of the dataset provided which will be subsequently used for cropping orthomosaic and digital elevation models.

Step 7 : Digital Elevation Model Processing

Input: geo-referenced point cloud + crop boundaries Output: digital surface models + digital terrain models + classified geo-referenced point cloud

In this stage, the ODM takes the geo-referenced point cloud and make surface model. The holes in the model created due to missing relevant point in the area are then filled using Inverse Distance Weighing interpolation technique. The model is then smoothened using median filter by removing noise or bad

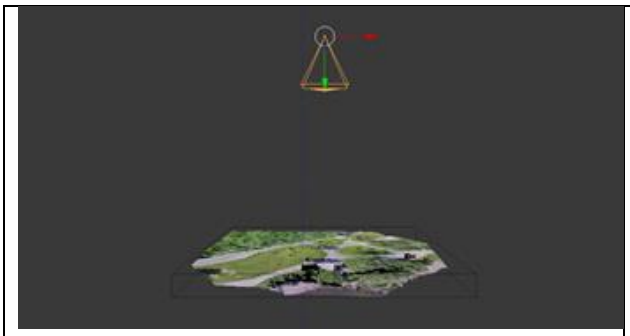
values. With additional other settings, ODM can also can classify the point clouds as ground vs. non-ground points which can help generate a Digital Terrain Model by first removing all non-ground points, filling the gaps and smoothing etc. Finally, the product is cropped based on the geographical bounds of the given datasets.



Digital surface model

Step 8 : Orthomosaic Processing
Input: textured mesh + crop boundaries Output: orthomosaic

The orthomosaic is generated by capturing a picture of the full textured 3D mesh model from the top. A dedicated application in the ODM will load the 3D textured mesh model into an orthographic scene and then the result is saved to an image using the appropriate resolution set. The final image thus obtained is then geo-referenced and converted to a GeoTIFF using the information computed in the geo-referencing stage above. Finally, the result is cropped based on the dataset extent.



Orthographic camera taking a picture of the 3D model from the top

	Orthomosaic

Learning Goal 4: Detail Understanding the various Task Options in ODM. These can change the behavior of your ouput. Understanding these settings is critical for generating good data products.

In the ODM, the data processing pipeline can be tweak by playing around each component with their adjustable settings. This can greatly influence the product that we are going to generate. **You can Set Enable/Disable for activating each Option. Assign Integer/Float/String value to each Option.**

pc-classify	smrf-scalar	opensfm-depthmap-min-patch-sd	smrf-window	mesh-octree-depth	sm-cluster	min-num-features	dem-euclidean-map
resize-to	smrf-slope	rerun-from	use-3dmesh	orthophoto-compression	mve-confidence	texturing-skip-hole-filling	skip-3dmodel
texturing-skip-global-seam-leveling	texturing-nadir-weight	texturing-outlier-removal-type	orthophoto-resolution	dtm	orthophoto-no-tiled	dem-resolution	matcher-neighbors
mesh-size	ignore-gsd	build-overviews	use-opensfm-dense	opensfm-depthmap-min-consistent-views	texturing-skip-local-seam-leveling	texturing-keep-unseen-faces	pc-csv
debug	use-exif	mesh-samples	split	matcher-distance	split-overlap	orthophoto-bigtiff	end-with
dem-decimation	orthophoto-cutline	pc-filter	fast-orthophoto	pc-ept	crop	pc-las	depthmap-resolution
merge	dsm	dem-gapfill-steps	mesh-point-weight	max-concurrency	texturing-tone-mapping	cameras	texturing-data-term
texturing-skip-visibility-test	opensfm-depthmap-method	use-fixed-camera-params	smrf-threshold	verbose	use-hybrid-bundle-adjustment		

Different Options available with WebODM when creating a task

Please note that, task options available on WebODM are just the subset of the total available options with ODM. For more task settings, native OpenDroneMap installed at Ubuntu may be looked at to explore other available options. But then WebODM has already exposes important and majority of the options than be used for generating a much reliable UAV data products and thus can be used to generate much complex workflows based on our needs. There are no clear guidelines for choosing of options for generating optimal products, as certain options that work well for a particular datasets might not possibly work for another datasets for different UAV scene. Last but not the least, these task options will keep on changing based on the ODM version changes. I will highlight on the details of the task options currently available with recent version of WebODM 1.1.0

build-overviews
When enabled, overviews are added to the orthomosaic. This does not affect other 2D outputs such as DEMs. Overviews are an optimization available for GeoTIFFs that **reduces the time it takes to open them**, for the tradeoff of a larger filesize and some computational time. Think of overviews as downsized copies of the orthomosaic, stored inside the orthomosaic file itself. Overviews are **useful when opening the *large* orthomosaic in GIS programs that support them, such as QGIS**. When overviews are available, the viewer program can load the overview most appropriate for the current zoom level instead of loading the entire file. If it takes forever to display a 400MB orthomosaic in a GIS program, it's probably because overviews weren't built! ODM creates overviews at 1/2, 1/4, 1/8 and 1/16 of the original resolution using an average interpolation. If an orthomosaic is 1000x1000 pixels, **build-overviews** will store copies of the same orthomosaic at 500, 250, 125 and 62 pixels resolution for faster visualization.

cameras
During the SfM process, **camera parameters are estimated from the input images**. Instead, with this option, it's possible to **choose a pre-computed set of camera parameters** *from previous known operations that delivers good product*, either as a path to a **cameras.json** file or by providing the contents of a **cameras.json** file. A **cameras.json** file is always computed after processing a dataset, so you can use the camera parameters computed from one dataset to process another. The **cameras.json** file can also be generated using a special calibration target and a calibration software. *Specifying a pre-computed set of camera parameters can be useful to increase the accuracy of certain reconstructions*, especially those that exhibit *doming* effects (point clouds that look concave or convex when they should be straight).

dem-decimation.
All DEMs are computed from the 3D point cloud output. The larger the point cloud, the longer it takes to compute a DEM. To speed things up or to quickly generate DEM for a quick review, at the trade-off of possible accuracy loss, this option reduces or decimates the number of points used to compute DEMs. The value specifies how many points to "skip" during the decimation. Let's look at an example by setting this option to 3. By taking all the points in the point cloud and placing them in a straight line, the program will reduce the number of points by keeping one every three.

Only black points are included. Gray points are skipped

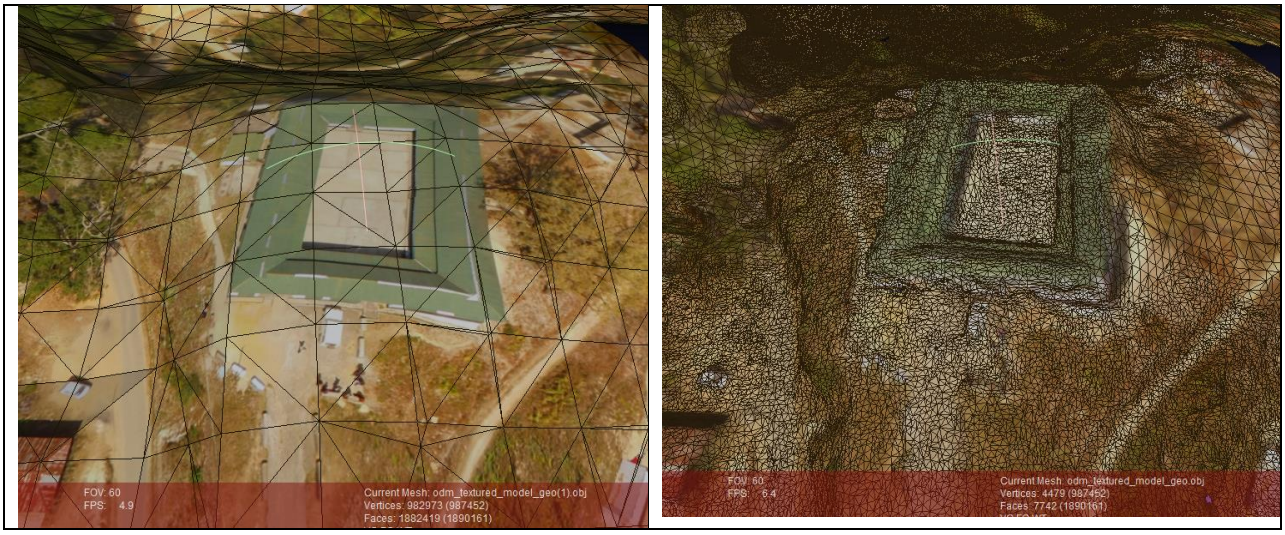
If you use a value of 1 (the default), then all points are included. You can compute this percentage by doing: (1 / decimation) * 100. It should be noted that points are skipped sequentially and do not take into account spatial information, so this option should be used with care. Decimation also does not affect the original point cloud. The decimation is only used for the purpose of computing DEMs.

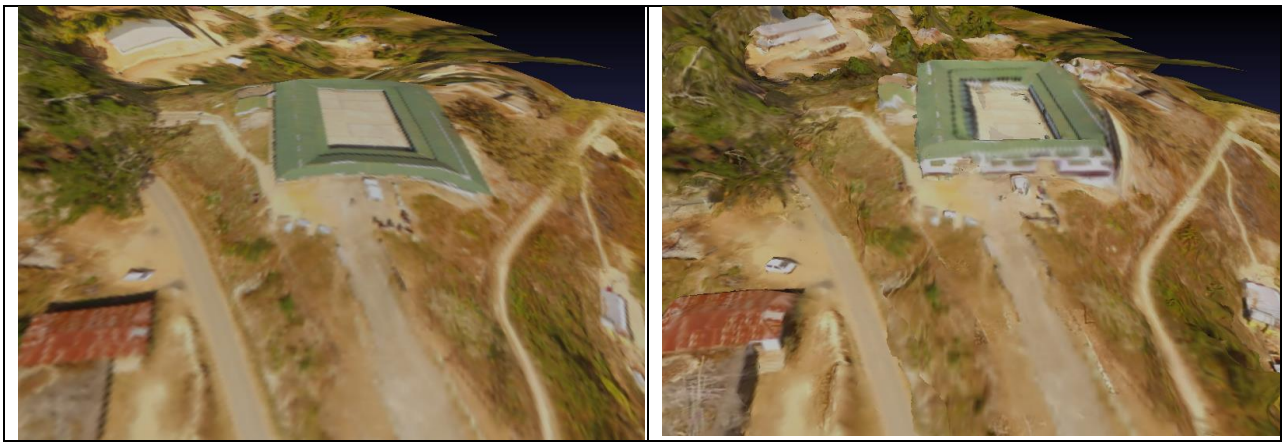
mesh-octree-depth

When it comes to generating 3D models, this is probably the most important option. It specifies a key variable for the Screened Poisson Reconstruction algorithm, which is responsible for generating a mesh from the point cloud. To understand how this option affects the output, it helps to visually understand the concept of an octree. First, octree means *eight-tree* (okta is *eight* in Greek). Why eight? Because at each level (or *depth*) of the tree, each box (or *node or branch*) of the tree is divided in eight parts. At the first level there's only one branch. At the second level there's 8. At the third there's 64 and so forth.

An octree with depth 1, 2 and 3

Lower depths in an octree allow finer details to be captured. The practical aspect of this option is that the higher the value, the finer the resulting mesh will be. The trade-off is exponentially longer run-time and memory usage. The default value of 9 works well for a lot of different cases. Flat areas can benefit from lower values (6-8) and urban areas can improve by setting this value higher (10-12). When increasing this option, **mesh-size** should also be increased as finer meshes require more triangles.





mesh-octree-depth 6 and mesh-size 10000 (left) vs. mesh-octree-depth 11 and mesh-size 1000000 (right)

Notice the meshes generated with different settings and their resulted textured model. More octree-depth and large mesh size does gives better 3D textured model. Tweak these settings, if you have better computing resources and wish to generate fine grained 3D textured model.

mesh-point-weight

Similarly to **mesh-octree-depth**, this option specifies a key variable for the Screened Poisson Reconstruction algorithm, which is responsible for generating a mesh from the point cloud. It affects the mesh by giving more importance to the location of the points. In practical terms, higher values can help create higher fidelity models, but can also lead to the generation of artifacts (undesired alterations). In general the default value works fairly well.

mesh-point-weight set to 0 (left) and 20 (right).

Notice the lack of edges in the top image and the excessive bumps in the bottom image

mesh-samples

Similarly to **mesh-octree-depth**, this option specifies a key variable for the Screened Poisson Reconstruction algorithm, which is responsible for generating a mesh from the point cloud. It specifies how many points should fall within a node of the octree during its construction. In practical terms, this value

should be tweaked between 1 and 20 to improve the smoothness of a model. If there's noise in the point cloud, this value should be increased. If there's little or no noise in the point cloud, this value should be set to 1 (the default).

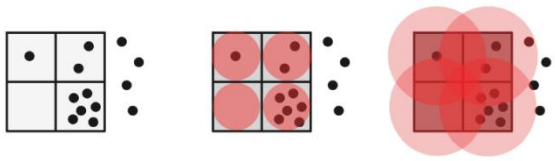
mesh-samples set to 1 (left) 20 (right). Ideal values for this option are between 1 and 20

mesh-size

To keep memory usage and run-time under control, after a mesh is generated the program simplifies it by setting an upper limit on the number of triangles the mesh can contain. The more triangles a mesh contains, the longer it takes to process it in subsequent steps of the pipeline. A low triangle count can sometimes degrade the quality of the mesh. If details seem to be missing from the 3D model or sharp triangles are present, increasing this option could improve results. **This is especially beneficial in urban areas where buildings require fine details for more appealing results.**

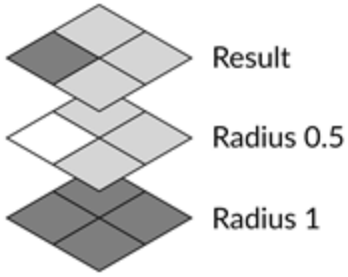
dem-gapfill-steps

The process of going from point cloud to DEMs is not as straightforward as it may seem. Since DEMs are *rasters* (images), they have *cells* (pixels). Each cell, should have a value. Depending on the resolution of the raster, certain cells may have zero, one or more points that fall within it. All cells need a value, even if no points fall directly into it, otherwise there will be empty areas (gaps) in the DEM! One way to overcome this is to use a radius around each cell. Every point that falls within the radius is considered part of the cell.



Pixels and points (left), radius of 0.5 (middle) and radius of 1 (right)

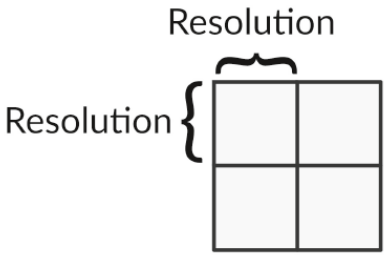
But how big should the radius be? If too small, as in the 0.5 radius example above, some cells might remain empty. If too big, there will be too much smoothing and accuracy will suffer. Since different point clouds have varying degrees of density, one solution is to compute multiple DEMs with different radiuses and stack them.



Gap fill interpolation with 2 DEM layers

Results with smaller radiuses (more accuracy, more gaps) are placed at the top, while results with bigger radiuses (less accuracy, less gaps) are placed at the bottom. The initial value for the radius for the first layer is set to half of the raster resolution. Subsequent layers have twice the radius of their predecessors.

dem-resolution
This option specifies the output resolution of DEMs in cm / pixel.



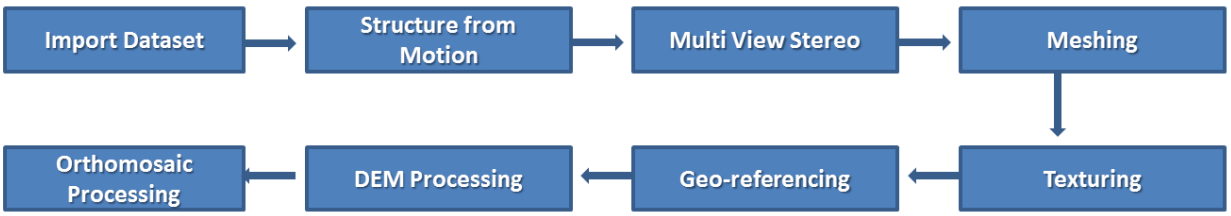
Each square represents a pixel in a raster DEM

As an example, if the area covered by the point cloud is 100x50 meters and dem-resolution is set to 10 cm / pixel, the final image size of the DEM in pixels can be calculated by:

(100 meters * 100 cm/meter) / 10 cm/pixel = 1000 pixels
(50 meters * 100 cm/meter) / 10 cm/pixel = 500 pixels

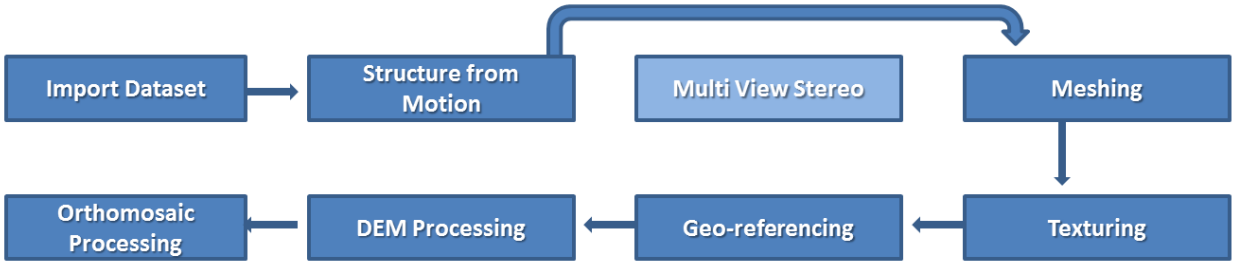
So the output DEM will be an image of 1000x500 pixels. The multiplication by 100 in the parenthesis is necessary because there are 100 centimeters in a meter.

fast-orthomosaic
Sometimes all that a user wants is an orthomosaic, generated as fast as possible, using the least amount of resources. If we revisit the overview of the processing pipeline, this is how it's generally executed:



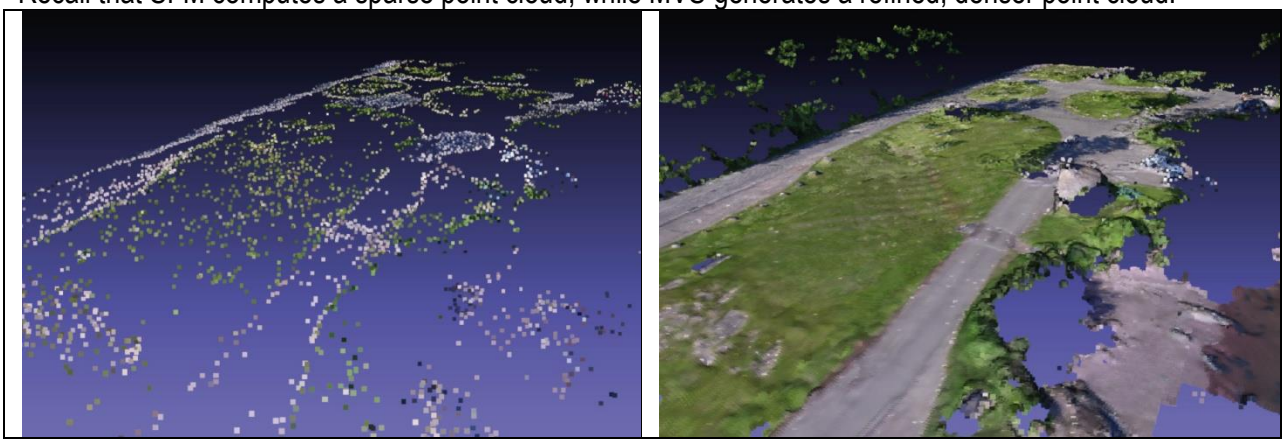
Processing Pipeline (normal)

When **fast-orthomosaic** is used, the program is instructed to skip the expensive MVS step.



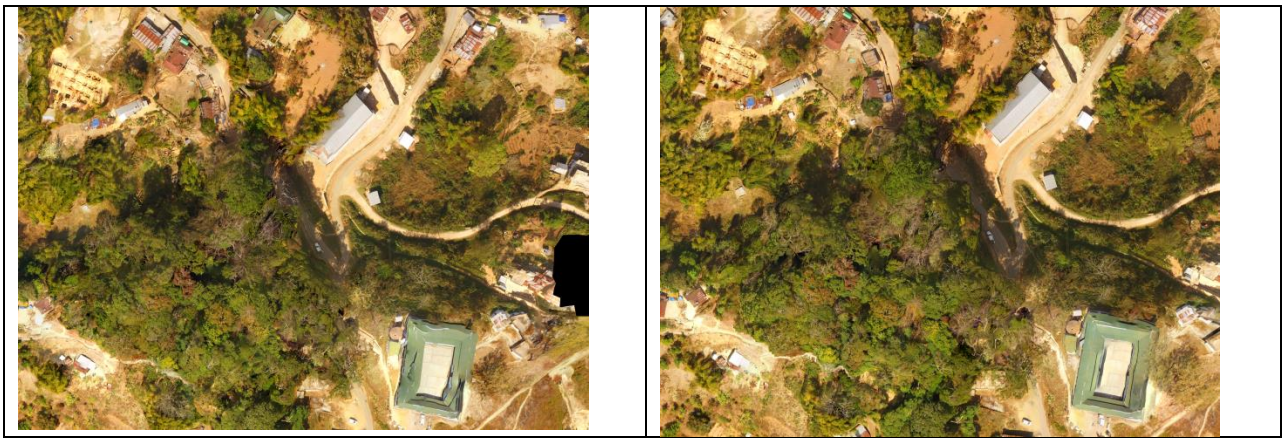
Processing Pipeline (fast-orthomosaic)

Recall that SFM computes a sparse point cloud, while MVS generates a refined, denser point cloud.



Sparse (left) vs. dense (right) 3D point cloud outputs

Both point clouds can be used to generate a mesh. However, it's better to have more points, as meshes can be created with more details. In the dense point cloud screenshot above, the building in the middle of the scene is well defined, but it's almost missing in the sparse point cloud. Buildings are especially difficult to model without a dense point cloud, so this option tends to yield poor results in urban areas. For flat areas such as farmlands, however, the results are quite good, since there are fewer buildings or structures to model. This option also tends to work well with images captured from a really high altitude or with images that have less than 50% overlap (such as many historical aerial images).



fast-orthomosaic (left) Vs. Normal (right) - A part of Meghalaya

Comparing the two images above (mix scene for an area in Meghalaya, India), the building on the right is much more defined than the left, but the terrain between the two is almost identical. This option can make a big difference in run-time, especially for very large datasets. If your datasets lots of terrain and almost no building structure you can use fast-orthomosaic option to quickly get the output without losing much quality.

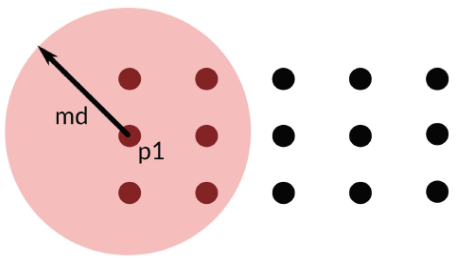
dsm
This option generates a digital surface model (DSM). DSMs are generated by **taking the maximum elevation values in a point cloud**, including both terrain and other structures such as buildings or trees. If two points fall on top of each other, only the tallest point is used. Gaps in the point cloud are filled using the **dem-gapfill-steps** option. The result is stored in *odm_dem/dsm.tif*.

dtm
This option generates a digital terrain model (DTM). DTMs are generated by classifying the point cloud using a simple morphological filter (SMRF). Setting this option implicitly turns on the **pc-classify** option, which classifies point either as ground or non-ground. Non-ground points are discarded before computing the DTM. Gaps in the point cloud are filled using **dem-gapfill-steps** option. The result is stored in *odm_dem/dtm.tif*.

DSM (left) vs. DTM (right)

matcher-distance
The SfM process involves matching image pairs, that is, finding images that share features between them (images showing the same objects). The naive, brute force approach is to compare each image against each other image. This results in an exhaustive, but slow search. Finding all images pairs in a 100 images dataset requires a lot of comparisons. To be exact, it requires: 100 * (100 - 1) = 9900 comparisons.

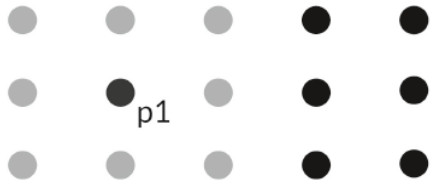
This number increases rapidly with larger datasets. To speed things up, the program uses an optimization. The core idea is that, when processing datasets collected in a uniform pattern, most images will be paired with images that are within a short distance from each other. Since each image often contains GPS location information, the program can set a distance threshold for which image pairs should not be considered. This is called preemptive matching.



Dots represent approximate image locations, extracted from EXIF tags. The md value represents the maximum distance that the image p1 will search for other images. Images outside of the circle will not be considered for matching

The distance is expressed in meters. It can be set to zero to disable it. If no location information is embedded in the EXIF tags of the images, this option is disabled. This option works in conjunction with the **matcher-neighbors** option.

matcher-neighbors
Similarly to **matcher-distance**, this option performs preemptive matching by considering only the nearest neighbors of each image. The illustration below shows the result of setting this option to 8:

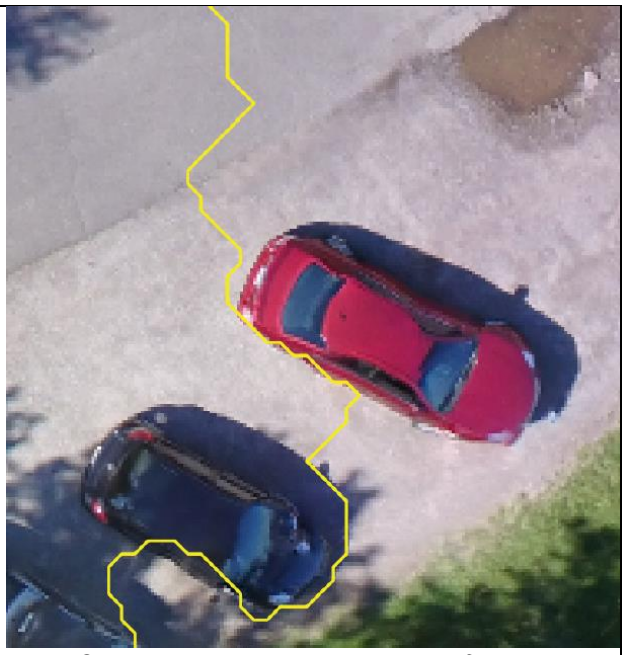


Dots represent approximate image locations, extracted from EXIF tags. When the matcher-neighbors is set to 8, only the 8 nearest neighbors (highlighted in gray) are considered for matching with image p1

For datasets with lots of overlap, it can be beneficial to increase this value since it's likely that valid matches will not be taken into consideration and decrease the accuracy of the reconstruction. It can be set to zero to disable it. If no location information is embedded in the EXIF tags of the images, this option is disabled. This option works in conjunction with the **matcher-distance** option.

orthomosaic-cutline
By turning on this option the program will generate a cutline. A cutline is a polygon within the orthomosaic's crop area that attempts to follow the edges of features.

Cutline

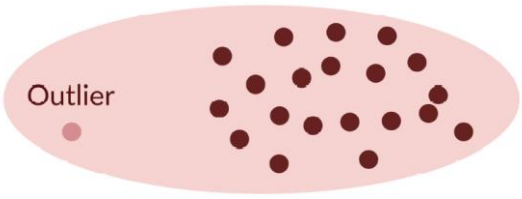


Cutline going around the edges of a car

A cutline can be **used to merge overlapping orthomosaics** by minimizing the color differences between seams. It's used to merge orthomosaics when processing large datasets using the split-merge pipeline. The cutline is saved in *odm_orthomosaic/cutline.gpkg*.

pc-filter

Noise from the point cloud can be partially removed using a statistical filter. This option sets the standard deviation threshold value for the filter. Standard deviation is a measure of how spread out points are relative to their neighbors. The filter looks at the closest 16 neighbors for each point and computes their standard deviations (how far each point deviates from the average distance to each other point). If a point is found to be too far away relative to its neighbors, thus having a standard deviation higher than the threshold, the point is labeled as an outlier.

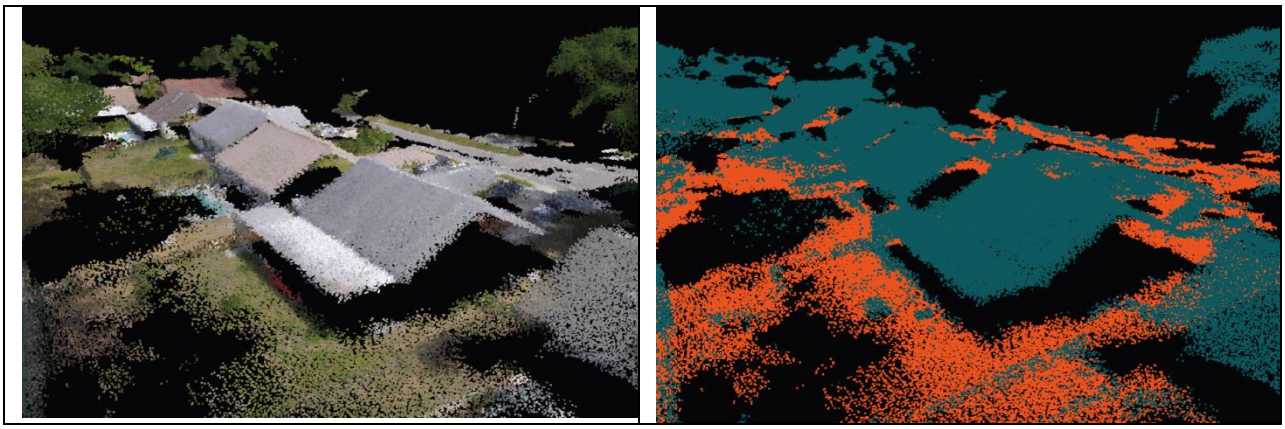


The gray point has a high standard deviation, so it's labeled as an Outlier

Setting this value too high will keep some noisy points, while setting this value too low will possibly remove valid points. Filtering can be disabled by setting this option to zero.

pc-classify

Points in a point cloud can be assigned one of several classification values to indicate whether a point is part of the terrain (ground), of a building, of a tree (vegetation) and several other possible classifications. By default every point is simply labeled as *unclassified* and the software does not attempt to label what each point represents. By turning on this option, a Simple Morphological Filter (SMRF) is used to attempt to find the points that are part of the terrain (ground) and assign to them a ground classification value. The end result is a point cloud that is divided between ground and non-ground points. The ground point cloud can then be used for the purpose of computing a DTM.



Point cloud (left) and classification results with SMRF (right)

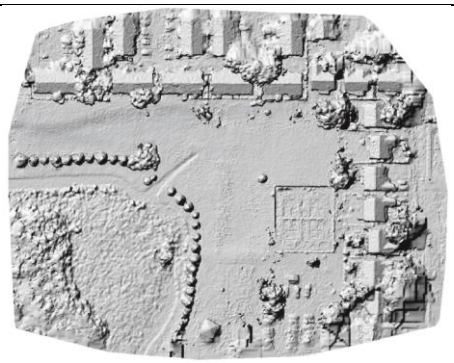
The SMRF algorithm can be controlled via four options. It's usually recommended to try the default values, examine results and then make tweaks as needed.

• **smrf-scalar:** is used to make the threshold parameter dependent on the slope. To improve results, this value can be decreased slightly if the **smrf-threshold** value is increased and vice-versa.

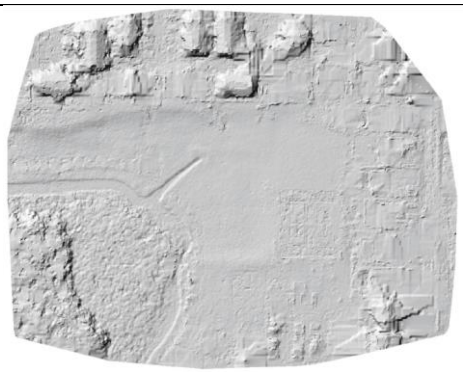
• **smrf-slope:** should be set to the *largest common terrain slope*, expressed as a ratio between change in elevation and change in horizontal distance (if elevation changes by 1.5 meters over a 10 meter distance, that's 1.5 / 10 = 0.15). It should be increased for terrains with large slope variation (hills, mountains) and decreased for flat areas. For best results it should be higher than 0.1, but not higher than 1.2.

• **smrf-threshold:** specifies the minimum height (in meters) of non-ground objects. For example, setting a value of 5 will likely be sufficient to identify buildings, but will not be sufficient to identify cars. To identify cars the value should be lowered to 2 or even 1.5 (the average car height). This parameter alone has the biggest impact on results.

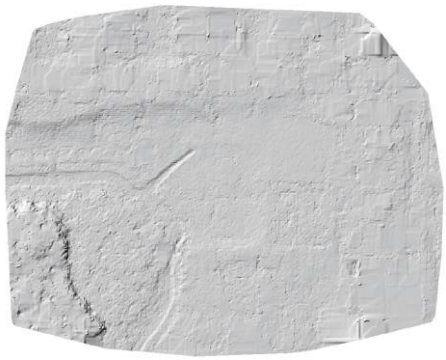
• **smrf-window:** should be set to the size of the largest nonground feature (in meters). For example, if a scene is full of small objects (trees), this value can be decreased. If the scene contains large objects (buildings), this value can be increased. It's recommended to keep this value above 10. SMRF has limitations and it's important to understand them. In particular, the filter will sometimes mistakenly classify points that belong to buildings or trees as ground points (type II errors).



Input surface model



Terrain model obtained with default SMRF options. Note some houses were mistakenly included and artifacts are lingering around the edges of removed objects

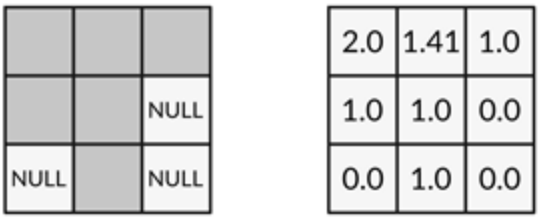


A much improved terrain model obtained by setting smrf-threshold 0.3 (decreased), smrf-scalar 1.3 (increased), smrf-slope 0.05 (decreased) and smrf-window 24 (increased)

Automated methods for reliable classification of point clouds are an active area of research. SMRF performs remarkably well, but often requires some tweaking. Users wishing to generate high quality DTMs should always double check the results and adjust the SMRF options as needed. It's also sometimes worth comparing the classification results with those obtained from supervised or trained classification methods. CloudCompare is a free and open source software that implements such methods.

dem-euclidean-map

An euclidean map is a geo-referenced image derived from DEMs (before any holes are filled) where each pixel represents the geometric distance of each pixel to the nearest *void*, *null* or *NODATA* pixel. It's an indicator (map) of how far a value in the DEM is to an area where there are no values. This can be useful in cases when a person wishes to know which areas of a DEM were derived from actual point cloud values and which ones were filled with interpolation. Looking at the euclidean map, every pixel that has a value of zero indicates that the corresponding location on the DEM was filled with interpolation (because the distance of a *NODATA* pixel to itself is zero).



DEM before hole filling (left) and corresponding euclidean map(right)

Euclidean map results are stored in the *odm_dem* directory.

depthmap-resolution

A depthmap is an image containing information relative to the distance of objects in a scene. Depthmap images are computed during the photogrammetry process to compute the dense point cloud. The higher the resolution of depthmaps, the longer the run-time to triangulate points. Higher resolution depthmaps increase the number of points, but also increase the amount of noise. Increasing the depthmap resolution will increase run-time quadratically (twice the resolution will take 4x the time to compute). **If you need to change the density of your point clouds, this is the option to tweak**. Point clouds are the basis for 3D models, which in turn are used to generate orthomosaics. When mapping urban areas, if the buildings come out with holes, look malformed or "wavy", a higher density point cloud could help obtain better looking buildings. Increasing this value too much can increase noise, so there's a subtle balance between point density and quality of results.

end-with
The processing pipeline is composed of several steps and processing is executed sequentially. Sometimes the results don't turn out as expected or a user might wish to compare the results of using different options. Since changing an option sometimes affects only a certain stage of the pipeline, there's no need to execute every single step all the way to the end. By using this option, the program will stop the execution at the chosen step. Possible values (in order of execution) are:
• **dataset**
• **split**
• **merge**
• **openSfM** (Structure From Motion)
• **mve** (Multi-View Stereo)
• **odm_filterpoints**
• **odm_meshing**
• **odm_25dmeshing**
• **mvs_texturing**
• **odm_georeferencing**
• **odm_dem**
• **odm_orthomosaic**

This option is often used in conjunction with **rerun-from**.

gcp
By default the program looks for a file named *gcp_list.txt* in the project directory. If it exists, it's used as a ground control point file to increase the geo-referencing accuracy of the results. With this option users can specify an alternate path for the GCP file. This option is not shown in WebODM and is automatically set if a GCP file is uploaded with a dataset.

help
Shows all possible options and exits.

ignore-gsd
To achieve good processing speed, the program relies on optimizations. One of these optimizations uses the average Ground Sampling Distance (GSD) value from all images (plus some buffer) to achieve two goals:
1) Put a limitation on the resolution of orthomosaics and DEMs.
2) Compute a good target size for the images when texturing 3D models.
The reason for placing a limit on resolutions is simple. While the program allows output resolutions to be set via **orthomosaicresolution** and **dem-resolution**, it can be tedious to estimate what the maximum resolution can be. For example, if photos are captured at 400ft, it makes no sense to set the resolution to 0.1 cm / pixel. The photos don't contain enough details to achieve that target resolution. So the optimization automatically lowers the resolution to a more reasonable value. Similarly, when the program knows that the orthomosaic is going to have a target resolution of 5 cm / pixel, it's wasteful to use full resolution images to create the textures for the 3D models. The orthomosaic will be downsized anyway, so the program resizes the images prior to texturing, speeding things up and lowering memory usage.

There are a few caveats with this approach:

1) The GSD value is computed by averaging the GSD value of all images in the scene. Furthermore, the flight altitude necessary for the computation is estimated from an average plane height computed from the sparse point cloud.

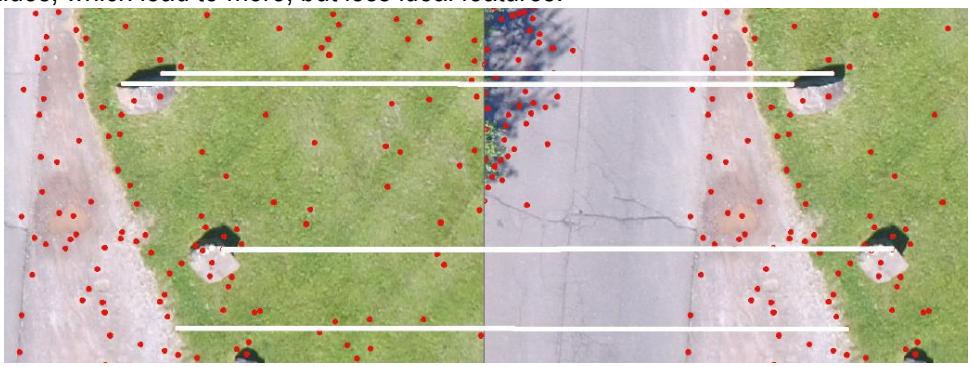
Average plane height (dotted gray line) and terrain (black line)

This means areas that have large changes in elevation (hills, mountains) might turn into an inaccurate estimate for the GSD value. In such scenario, the resolution could be possibly capped at a value lower than ideal.
2) To colour a mesh, the texturing program has to choose the best section of a photo, from a set of photos. To decide which photo and section is best, the algorithm uses several factors When **texturing-data-term** is set to gmi (the default), one of these factors is an indicator of *sharpness*. Resizing images prior to texturing has the side effect of reducing the *sharpness* of all images, thus decreasing the importance of this factor relative to others. This can result in more blurred areas. When noticing excessive blur in an orthomosaic, or when the resolution of the outputs is lower than expected, turning on **ignore-gsd** can improve results. The trade-off is longer runtime and higher memory usage.

max-concurrency
By default the program will attempt to use all available CPU resources. There are scenarios where this might not be desirable, for example on shared servers or when wanting to use the computer for other tasks while processing. This option attempts (but does not guarantee) to limit the maximum number of CPU cores that will be used at the same time.

merge
This option controls what assets should be merged during the merge step of the split-merge pipeline. By default all available assets are merged, but users can choose to merge only specific ones.

min-num-features
During the SFM process, images have to be matched into pairs. The way the pairing happens is by means of finding matches between features in the images. An example of a feature is the corner of a building or the edge of a car. If the same feature is found to be present between two images, it's used as evidence of a possible match. But features can be ambiguous. An identical looking corner of a building between two images could be from two corners of two different building that just happen to share the same architecture style. So the program finds lots and lots of features in each image and uses all of them to find possible pairs. This option controls the minimum number of features the program tries to finds in each image, thus increasing the likelihood of finding matches between images. It does so by progressively lowering certain threshold values, which lead to more, but less ideal features.



min-num-features controls the target number of red points in each image

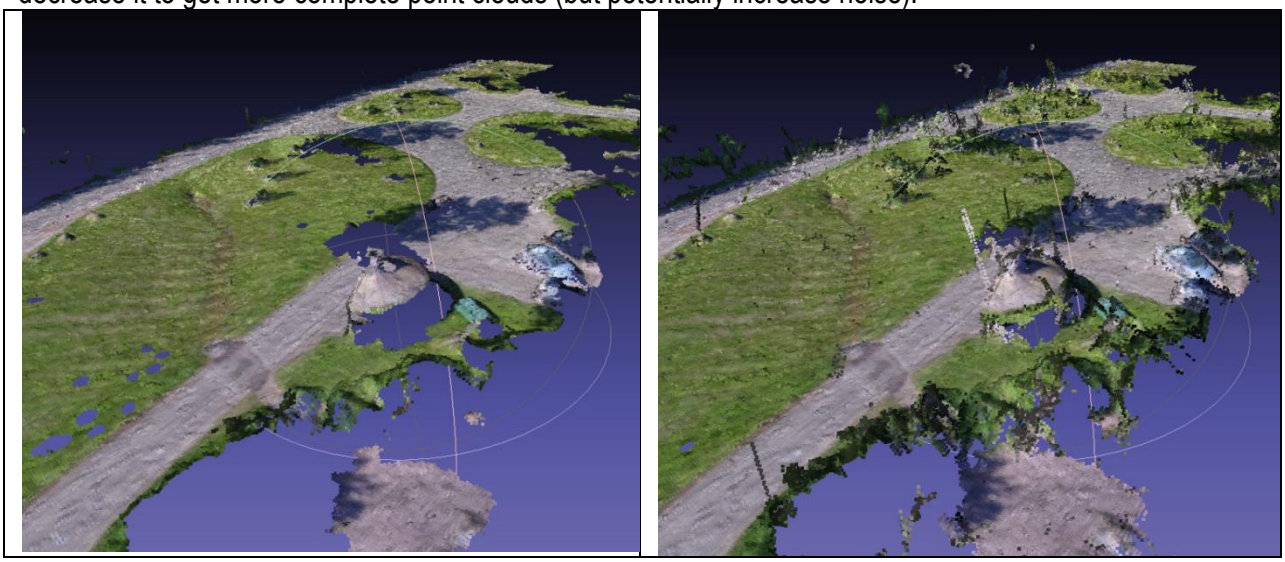
This option should be increased when trying to map areas that have few distinguishable features, such as forest areas:

In the above scene, it's difficult to find good distinct features / reference points in the image above. increasing the number of features can increase the chances of a match between such images

Increasing this option results in longer run-time, but increases the chances that a reconstruction will be properly generated. In certain instances, the program might also be able to generate only a partial reconstruction. In such cases users will notice that some areas for which images exists do not appear in the final results (a chunk of the orthomosaic will appear to be missing). Increasing this option could help find matches that generate a more complete reconstruction. An interesting side effect is that increasing this option also increases the number of points in the sparse point cloud, so it can be used in conjunction with **fast-orthomosaic** to produce slightly better results.

mve-confidence

When the dense point cloud is computed using MVE (the default), each point is assigned a confidence value between zero and one. A value of zero indicates that the point is most likely noise, while a value of one indicates that the point is most likely a good point. Points below a certain confidence threshold are discarded. Users can increase this option to decrease noise (but potentially eliminate valid points) or decrease it to get more complete point clouds (but potentially increase noise).



Confidence set to 0.6 (left) and 0 (right). Notices some areas are missing in the left image, but much more noise is present in the right image

openSfM-depthmap-method
When **use-openSfM-dense** is set, this option affects the point cloud by specifying the method used to compute depthmaps
The three possible options are:
• **PATCH_MATCH** (default)
• **PATCH_MATCH_SAMPLE**
• **BRUTE_FORCE**

PATCH_MATCH and **PATCH_MATCH_SAMPLE** are faster, but sometimes miss some valid points, which can result in a point cloud with some empty areas. **BRUTE_FORCE** is slower, but does a more exhaustive job and can produces more complete results. The default value tends to work well and users should switch to **BRUTE_FORCE** only if the point cloud is missing significant chunks. **PATCH_MATCH** is slightly slower than **PATCH_MATCH_SAMPLE** but tends to create slightly denser point clouds.

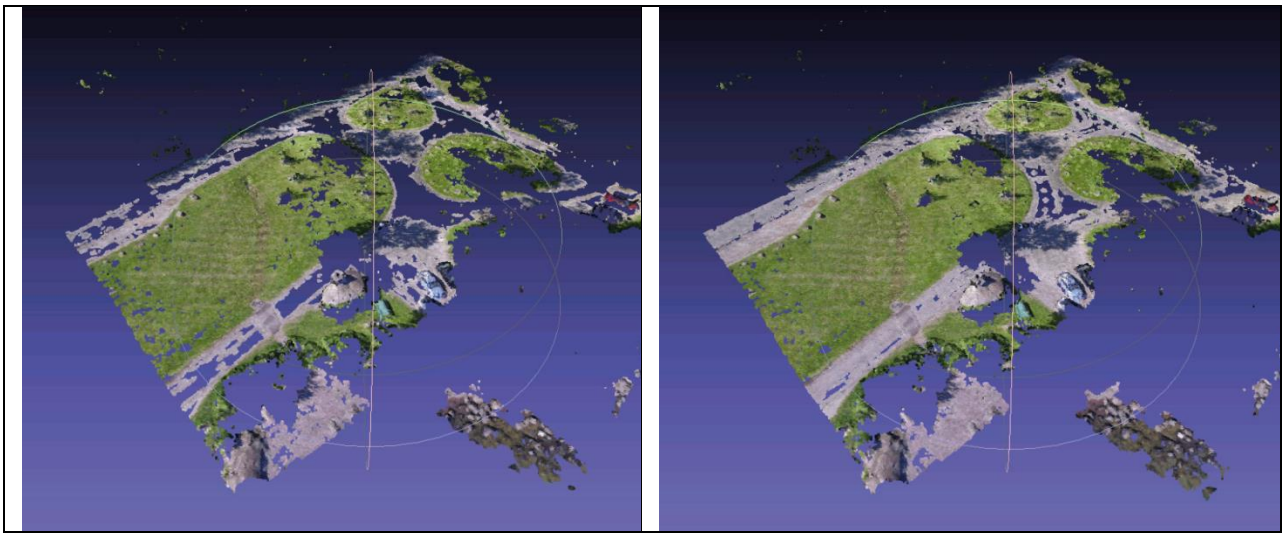
openSfM-depthmap-min-patch-sd
When **use-openSfM-dense** is turned on and **openSfMdepthmap-method** is set to **PATCH_MATCH** or **PATCH_MATCH_SAMPLE** this option controls a key variable that helps define areas in the depthmaps that should be skipped for improving run-time performance. Patches are simply small sections in an image:

Patch in an image

During computation, input images are split into patches. Each patch is assigned a variance value computed from its pixels. The key idea is that areas in the image that are uniform (contain similar values), such as the sky, contain little to no information useful for triangulating points.

Patch with high variance (left) vs. patch with low variance (right)

Variance is simply standard deviation multiplied by itself. So this option defines the minimum standard deviation that an image patch must have during the OpenSfM depthmap calculation process for it to be considered in the computation. Setting this value too low will result in longer run-time, while setting this value too high could potentially make areas with uniform values to be ignored. The screenshots below illustrates this behavior:



openSfM-depthmap-min-patch-sd set to 5 (left) and 2.5 (right).

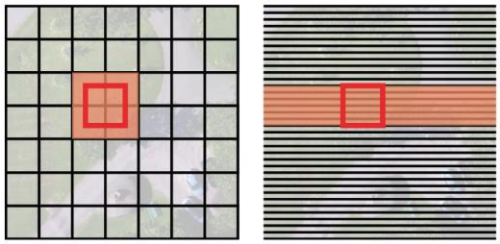
Notice the lack of roads in the image on the left. Roads have uniform colors (low variance). Setting a high value caused the roads to disappear!

orthomosaic-bigtiff
BigTIFF is an extension of the TIFF format to support files larger than 4GB. The possible values for this option are:
• **YES**: force the output orthomosaic to use BigTIFF
• **NO**: force the output orthomosaic to use classic TIFF
• **IF_NEEDED**: will use BigTIFF if it is needed (image larger than 4GB and not using compression, see **orthomosaiccompression**)
• **IF_SAFER** (default): will use BigTIFF if the resulting file might exceed 4GB. This is a heuristics that might not always work depending on compression. The BigTIFF format is not backward compatible with classic TIFF (programs compatible with TIFF do not necessarily support BigTIFF). A viewer needs to have explicit support for BigTIFF. Luckily most GIS programs support BigTIFF. Some legacy applications however might not. If you use one of these legacy applications, set this option to **NO**. If receiving a *TIFFAppendToStrip:Maximum TIFF file size exceeded* error, the heuristic used for **IF_SAFER** failed to guess the final size of the image. In this case changing this option to **YES** can fix the error.

orthomosaic-compression
Compression is a method to save space in exchange for slightly longer run-time. The possible values for this option are:
• **JPEG:** Uses JPEG compression with quality value of 75. JPEG is a lossy compression method, meaning some image quality is lost during compression.
• **LZW:** Uses Lempel–Ziv–Welch compression. This is a lossless compression method, meaning image quality is not lost during compression.
• **PACKBITS:** This compression method, like LZW, is lossless. It's arguably more supported than LZW, but achieves less compression than LZW.
• **DEFLATE** (default): also referred as ZIP compression, deflate is a lossless compression method. It tends to yield slightly smaller file sizes when compared to LZW.
• **LZMA:** another lossless compression method.

• **NONE:** Skips compression. Speeds up the generation of the orthomosaic, but creates larger files.

orthomosaic-no-tiled
By default the program will generate tiled TIFFs, with a tile size of 256x256 pixels. Tiling in this context is not the same as generating tiles for web viewers. Instead its related to the arrangement of data within the file. Data can be arranged either in tiles or stripes. When reading and displaying an entire file, tiles and stripes are equivalent in performance, since all data must be read regardless. When accessing a subsection of the image however, for example when zooming into an area, using tiles often results in needing to read less data.



Tiles (left) vs. stripes (right). The red rectangle is the area being accessed. The highlighted area shows the amount of data that needs to be read from the file. The smaller the highlighted area, the quicker it is to access the file

Tiles are the default. Tiled orthomosaics take slightly longer to create compared to striped. To use stripes, users can turn on this option.

orthomosaic-resolution
Same as **dem-resolution**, but applied to orthomosaics instead of DEMs.

pc-csv
By default the output point cloud is exported in a compressed LAZ format. The LAZ format is not human readable and cannot be opened with a simple text editor. Users can export a copy of the point cloud to CSV (Comma Separated Value) format by turning on this option. The CSV file format is not ideal for point cloud data, but can sometimes be useful for debugging the values of the point cloud or for import in programs that do not understand LAS/LAZ. The resulting point cloud is stored in *odm_georereferencing/odm_geo-referenced_model.csv*.

pc-ept
By setting this option users can export the point cloud in Entwine Point Tile (EPT) format, which can be used to efficiently stream point clouds across networks. EPT datasets can also be efficiently analyzed, queried and transformed using tools such as PDAL. The resulting EPT dataset is stored in the *entwine_pointcloud* directory.

pc-las

By default the output point cloud is exported in a compressed LAZ format. Since not all programs support LAZ, users can export a copy of the point cloud in uncompressed LAS format by turning on this option. The resulting point cloud is stored in *odm_georereferencing/odm_geo-referenced_model.las*.

rerun
Shorthand for **rerun-from <step> end-with <step>**.

rerun-all
Shorthand for **rerun-from dataset**, while also removing all output folders before starting the process.

rerun-from
Same as **end-with**, except that it instructs the program to resume execution from a specific point in the pipeline, skipping previous steps. When using WebODM the rerun-from option is automatically set when using the **Restart** button dropdown.
It's not always possible to restart a task from a certain step in WebODM. The processing node has to support task restarts and the task intermediate results need to have been kept on disk (by default they are kept only for 2 days).

resize-to
During the SfM process the program needs to extract features from each image. To speed things up, the program resizes all images prior to performing feature extraction. It's important to note that the input images are not affected by this option and neither are other stages of the pipeline. Changing this option will not degrade the quality of resulting orthomosaics or 3D models. This option can be lowered with datasets that have lots of recognizable features (cars, buildings, etc.) and should be increased with datasets that lack them (forest areas, deserts, etc.).

skip-3dmodel
Sometimes all that a user wants is an orthomosaic. In that case, it's not necessary to generate a full 3D model. This option saves some time by skipping the commands that produce a 3D model. A 2.5D model (a 3D model where elevation is simply *extruded* from the ground plane) is still generated. 2.5D models are not *true* 3D models as they cannot represent the true shape of objects such as overhangs, but work well for the purpose of rendering orthomosaics.

3D model (top) vs. 2.5D model (bottom). Note the absence of overhangs on the bottom

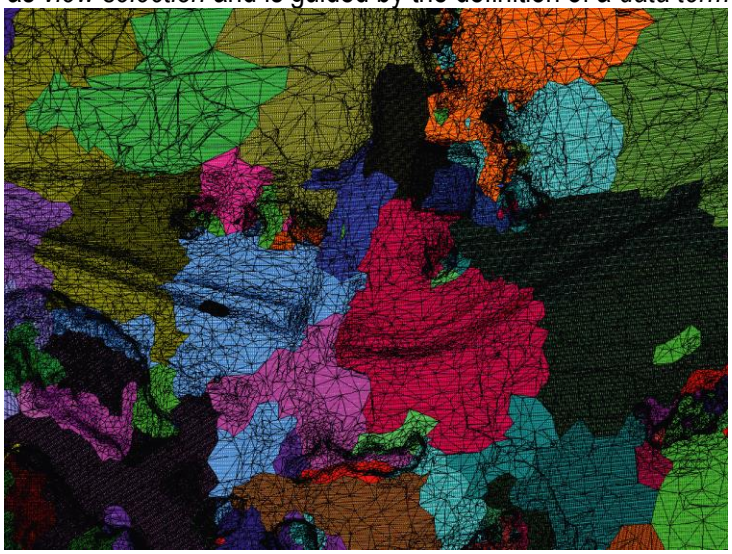
sm-cluster

Specifies a URL to a ClusterODM instance. When combined with the **split** option, it enables the distributed split-merge pipeline for processing large datasets in parallel using multiple processing nodes.

split
When set to a number lower than the number of input images, enables the split-merge pipeline.

split-overlap
Specifies the amount of overlap (in meters) that sub-models should have during the split-merge pipeline.

texturing-data-term
A difficult part of texturing a mesh is answering the question of how to choose the best image for each part of the mesh, since due to overlap each part of the mesh has likely been photographed by multiple images. This process is known as *view selection* and is guided by the definition of a *data term* or a *cost function*.



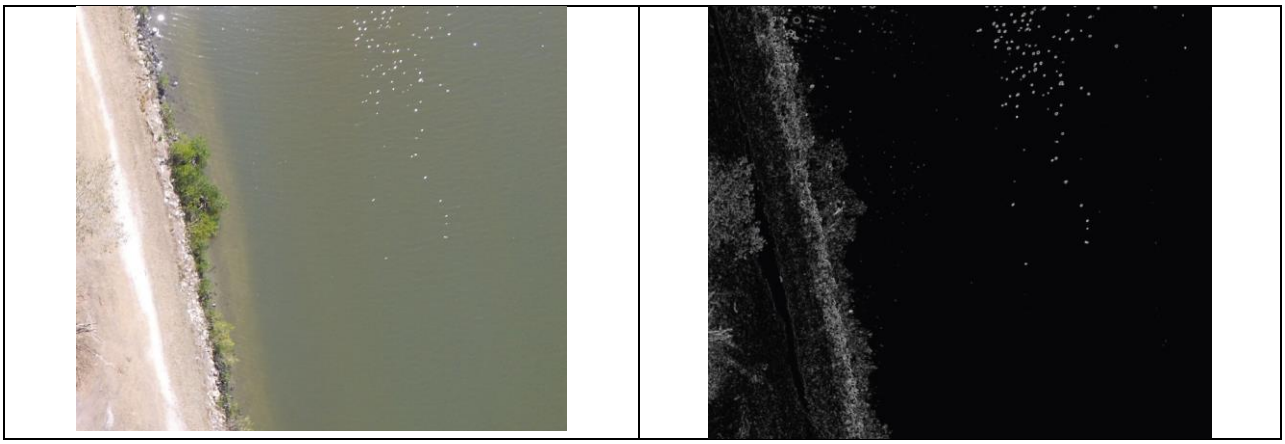
The view selection process assigns different areas of a mesh with an image. In the screenshot above each color represents a different image assigned to an area

Two data terms are available:
1. gmi (default)
2. area

GMI stands for *Gradient Magnitude Image*. A gradient is a change in values and is often represented graphically for ease of understanding. Magnitude is a measure of how much the gradient *changes*. If a gradient changes gradually it will have lower magnitude than if the gradient changes abruptly.

A low magnitude gradient (black to white)

The purpose of the gradient is to prioritize areas in an image that are in focus (they will exhibit a higher gradient magnitude). The gradient image is computed by running a *Sobel* edge detector and computing a gradient over the result.



An image (left) vs. Sobel edge detector mapped to a black-white gradient (right). The water shows small changes in gradient, whereas the shoreline has a higher gradient magnitude

The *area* data term works differently. It simply prioritizes images that provide the largest area coverage for a particular section of the mesh.

A triangle in the textured mesh

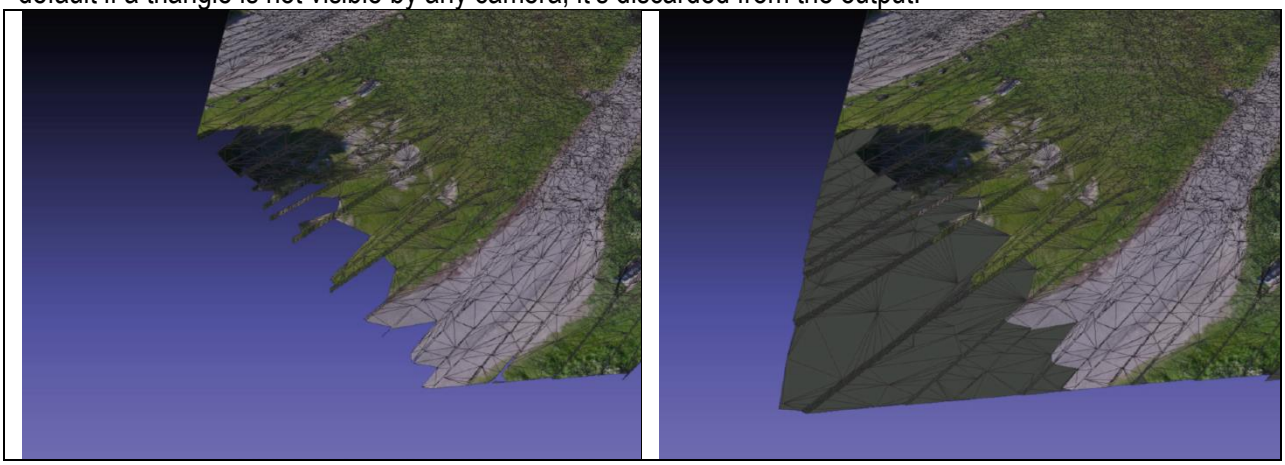
The same triangle projected on two images. Because the triangle's area in the left image is bigger, the left image is chosen for coloring the triangle

Data term gmi (left) vs. area (right). On the left, notice the sharpness of the ditch and the blue car, but the presence of a tree on the road

The tree was wrongly placed on the top image in this case. Trees have high gradient magnitudes, whereas roads do not.

texturing-keep-unseen-faces

The input to the texturing part of the pipeline consists of a mesh, cameras and images. The input mesh is composed of triangles. The program at some point checks which triangles are visible by the cameras. By default if a triangle is not visible by any camera, it's discarded from the output.



Unseen faces are removed from the textured mesh (top) vs. faces are kept with no color (bottom)

This option instructs the program to keep all triangles, regardless of whether they are seen by a camera or not.

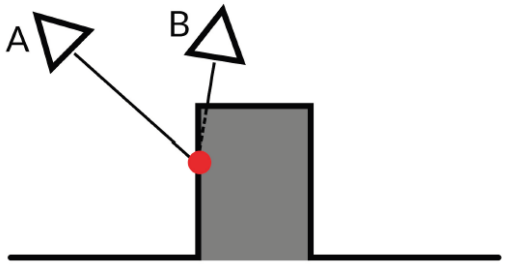
texturing-nadir-weight

During texturing the program needs to choose the best image for each section of the mesh. In addition to using the data term (see **texturing-data-term**), if a 2.5D mesh is used for generating the orthomosaic, the program enables a special mode called nadir. While in nadir mode, the program behaves differently with parts of the mesh that are vertical or almost vertical (think walls of buildings).

1. The visibility test for such vertical parts is disabled (see **texturing-skip-visibility-test** for a description of the visibility test).

2. The quality score obtained either via the *gmi* or *area* data term is replaced with a value proportional to the *nadir-ness* of the images (how straight down is an image?), multiplied by a weight. Images that are captured straight down are given more priority than images that are captured at an angle.

This option controls the weight of the nadir factor. The higher this option, the more nadir images are favored for texturing vertical areas of the mesh and vice versa.



Camera A has a better view of the side of the building and Camera B is occluded. But because visibility testing is skipped and Camera B is more nadir, in nadir mode camera B is selected.

Nadir mode can substantially improve the quality of orthomosaics, especially around the corners of buildings.

texturing-nadir-weight set to 0 (left) and 32 (right). Note the quality improvements near the edges of the building

This option affects only the 2.5D textured mesh (not the 3D textured mesh). The default of 16 works well for most datasets. A higher value can increase the quality of buildings, but can lose details in other areas of the orthomosaic.

texturing-outlier-removal-type

Aerial imagery is not always static. Sometimes moving objects (cars, bicycles, cats, etc.) are captured between multiple photos. Because the objects are moving however, they could end up appearing in multiple parts of the textured mesh during the texturing process.

Orthophoto (**Pix4D**)

Orthophoto (**ODM**)

Moving object, captured between two pictures

To prevent these artifacts, the program checks for consistency between two or more images. If inconsistencies are found, they are labeled as outliers and removed. There are two methods to do that:

1. gauss_clamping (default)
2. gauss_damping

Both are based on evaluating each photo in which an outlier could be visible using a statistical method. Gauss Clamping is more aggressive. Upon finding a high likelihood of an outlier, it will reject the photo containing the outlier from use in the area being textured. Gauss Damping on the other hand will progressively lower the picking priority of the offending photo, so it's a less aggressive approach and can lead to smoother results.

Gauss Damping (left) vs. Gauss Clamping (right). Note a part of the cart was missed in the left image

texturing-skip-global-seam-leveling

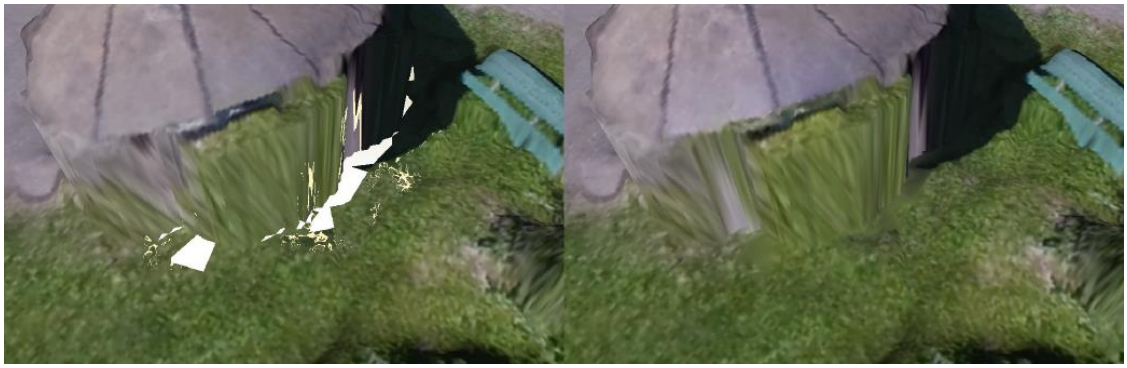
During texturing the program needs to merge together images that have different characteristics, for example different light intensities.

Seams in the textured model due to different light intensities (left) and global seam leveling applied (right)

This kind of seam blending is referred to as *global* because it's evaluated on all texture patches. The goal is to minimize the difference between the patches as to achieve consistent luminosity throughout.

texturing-skip-hole-filling

During the texturing process, some parts of the mesh cannot be assigned a texture. This could happen because not enough information is available to assign a particular face of the mesh to one of the input images. To mitigate this, small holes in the mesh are filled by interpolating the textures of nearby faces.



Small holes (left) are filled via interpolation (right). Interpolation comes out as a smooth blur, which is not very noticeable for small areas This option disables the hole filling feature (not recommended).

texturing-skip-local-seam-leveling

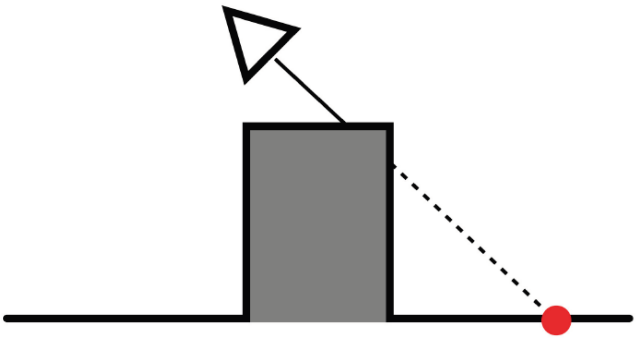
When texturing, the program needs to merge together images that have different characteristics, for example different light intensities. Applying global seam leveling (discussed in **texturing-skip-global-seam-leveling**) is a necessary but often insufficient step to remove all visible seams from the texture patches. To overcome this problem, the program applies localized Poisson editing (a way to blend two images) on all texture patches. The method is *local* because it affects only a local buffer around the boundary of the texture patches to keep runtime under control.

Result with global seam leveling, but not local seam leveling (left) and with local seam leveling (right)

This option disables local seam leveling (not recommended).

texturing-skip-visibility-test

During the texturing process, mesh faces are checked for visibility. If an obstacle exists between a face and a camera (such as a building), that face will ignore image information coming from that particular camera. This is a good way to assure consistency.



A camera ray projected onto a face (red dot). A building is in the way, so the camera is ignored. Without visibility testing, images from building rooftops would show up on the ground!

texturing-tone-mapping
This option can sometimes help enhance the quality of textured meshes and thus the quality of the orthomosaics. When setting this option to **gamma**, the program applies gamma correction to the texture maps, prior to applying local and global seam leveling. Gamma correction is a method for mapping luminance values in an image (for which an in-depth discussion is available at https://www.cambridgeincolour.com/tutorials/ gamma-correction.htm). The use of gamma correction in the context of texturing can generate more vivid results.

Raw input image (left), gamma corrected image (right)

time
Generates a *benchmark.txt* file stored in the project directory showing the time it took to process each step of the pipeline. Useful for measuring performance. By default no benchmark file is generated.

use-3dmesh
By default a 2.5D textured mesh is used to render the orthomosaic. 2.5D meshes tend to work well for most aerial datasets, but can sometimes lead to sub-par results, especially if there are no nadir images in the datasets (images with the camera pointed straight or almost straight at the ground). The reason for it is that the texturing step is performed differently between 3D and 2.5D meshes. 2.5D meshes give priority to nadir images, and if these are missing, the texturing might be of lesser quality compared to a 3D mesh. For points of interests, such as one obtained by orbiting a single building at close range with oblique images, a 2.5D mesh will also perform poorly. This option instructs the program to use the full 3D model for generating an orthomosaic and to skip the generation of the 2.5D model. See also **skip-3dmodel**.

use-exif
A Ground Control Point file will always be used if either a *gcp_list.txt* file exists in the project directory (ODM) or if a GCP file has been uploaded with a dataset (WebODM). By turning on this option, the program ignores the GCP file and relies on the location information from the image's EXIF tags instead.

use-fixed-camera-params
During the SfM process, the camera's internal parameters need to be estimated and refined to achieve a good solution. Sometimes, due to poor image collection practices or excessive lens distortion, the camera parameters are wrongly estimated and can result in sceme reconstructions that exhibit a *doming* effect. By turning on this option it's possible to instruct the software not to optimize camera parameters at all (keeping the parameters fixed). This can sometimes improve results if there's little to no geometric distortion in the images and the focal length value embedded in the images' EXIF tags is accurate.

use-hybrid-bundle-adjustment
Bundle adjustment (BA) is a refinement step during the SFM process that improves the location of cameras, 3D points and camera parameters. This process needs to be done at regular intervals during the reconstruction to avoid the accumulation of errors, but is also computationally expensive. It comes in two varieties, local and global. Local BA only refines a subset of the reconstruction, global BA refines the entire reconstruction. Performing global bundle adjustment requires re-evaluating the entire scene, which is slow. By default, the program will perform global BA only after the number of new triangulated points in the scene has increased by 20%. It will also perform local BA every time a new camera is added to the scene by comparing the cameras that are within 3 levels of connectivity (capped to 30 cameras).

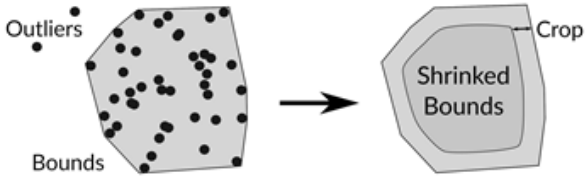
By turning on this option, the program changes the logic that triggers the local and global bundle adjustment. First, a global bundle adjustment is forced every time 100 new cameras are added to the scene, regardless of the number of new points added to the reconstruction. Second, when performing local bundle adjustment, the program only looks at the the cameras that are within 1 level of connectivity instead of 3. In short, this option increases the number of times that global BA is performed, but reduces the number of local BA operations.

For small datasets (< 1000 images) there's not much difference. As the number of images increases, the cost of running more local BA operations starts to outweigh the cost of running more global BA. For very large datasets, turning on this option can reduce the total run-time. It can also increase the accuracy of the reconstruction since a global bundle adjustment is performed more frequently.

use-openSfM-dense
The program has two options for generating the dense point cloud:
1. MVE (default)
2. OpenSfM
By default MVE (Multi-View Environment) is used. By turning on this option OpenSfM's depthmap reconstruction algorithm is used instead.

verbose
This option enables verbose messages. When this option is enabled, the processing output will contain more messages which can be used to diagnose possible issues.
crop

In its raw form, the orthomosaic contains irregular, jagged edges. These are the result of the texturing program attempting to fill areas that have little or no information. Cropping attempts to remove those edges to give us a nice, smooth looking orthomosaic. First, the boundaries of the orthomosaic are estimated by looking at the point cloud. The point cloud is first thinned and filtered to remove outliers. Then a polygon encompassing the result is saved in *odm_georeferencing/odm_geo-referenced_ model.bounds.gpkg*. This polygon is further smoothed and shrunk by the value specified in the **crop** option (as a value in meters). The final polygon is then used to crop the orthomosaic.



Point cloud (left) and shrinked bounds that define the crop area (right)

Estimating where the bounds are is not a perfect science. Sometimes the area of an orthomosaic that looks perfectly good will be removed in the cropping process. This option can be set to zero to skip cropping. For very large datasets, skipping crop can lower the run-time, since no computations have to be performed to estimate boundaries.

debug
Enable additional debug messages in the console output. Mostly useful for development purposes.

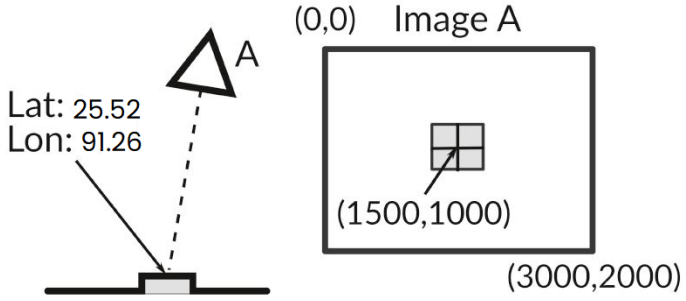
version
This option causes the program to print the ODM version number and exit.

Learning Goal 05: Using of ODM GCP interface for adding Ground Control Points

A ground control point (GCP) is a position measurement made on the ground, typically using a high precision GPS. Measurements are made near identifiable structures such as street corners or by placing visible markers on the terrain.

A ground control point marker.

Images that contain the visible markers can then be *tagged* by creating a correspondence between the image location of the markers and their real world positions.



A GCP marker is photographed by camera A to produce Image A. In a second step, the pixel location of the marker (1500,1000) from Image A can be manually tagged with its real world coordinates (latitude 25.52, longitude 91.26).

Using ground control points can increase the geo-referencing accuracy of a reconstruction, since measurements of static (nonmoving) objects using a high precision GPS are often better than those obtained from the GPS of moving UAVs. **The ideal number of ground control points ranges between 5 to 8**, placed evenly across the area to be flown. **Adding more than 8 ground control points does not necessarily result in increased accuracy**. If the same marker is visible from multiple images, it should be tagged multiple times for each image. **Ideally each marker should be tagged at least 3 times**. Another way to think of it is to capture each marker on at least 3 images. **This is so that the marker's location can be triangulated during computation**. Ground control points can be used by providing an additional text file along with the input images. The file follows a simple format:

<spatial reference system>
<geo_x> <geo_y> <geo_z> <im_x> <im_y> <image_name> [<extras>]
<geo_x> <geo_y> <geo_z> <im_x> <im_y> <image_name> [<extras>]

...

The GCP for the above image, for example, would be represented as:

```
EPSG:4326
91.26 25.52 0 1500 1000 ImageA.jpg gcp1
```

The **geo_z** field in this case may be set to zero if it doesn't have an altitude value.

Creating a GCP file using POSM GCPi

The task of manually finding and measuring pixel coordinates from all images and tying marker locations to world coordinates can be tedious and error prone at best. The Portable OpenStreetMap Ground Control Point Interface (POSM GCPi) provides a way to make this process a bit easier. The application is already loaded as a default plugin in WebODM and can be accessed via the **GCP Interface** panel.

Step 1. Create a GCP file stub
After measuring the location of your markers it's likely that you'll end up having a list of labeled point coordinates. Exporting them to CSV format and opening them in a spreadsheet application such as LibreOffice Calc should yield something similar to:

```
X,Y,Z,Label
91.263877, 25.522005, 1346, gcp1
91.262870, 25.521703, 1332, gcp2
91.262929, 25.522513, 1340, gcp3
91.264279, 25.522433,1350, gcp4
91.263534, 25.522086,1346, gcp5
```

From here, let's add two new columns for the *px*, *py* fields (initialized to zero) and shift the *Label* column to the right, so that our file now looks like:

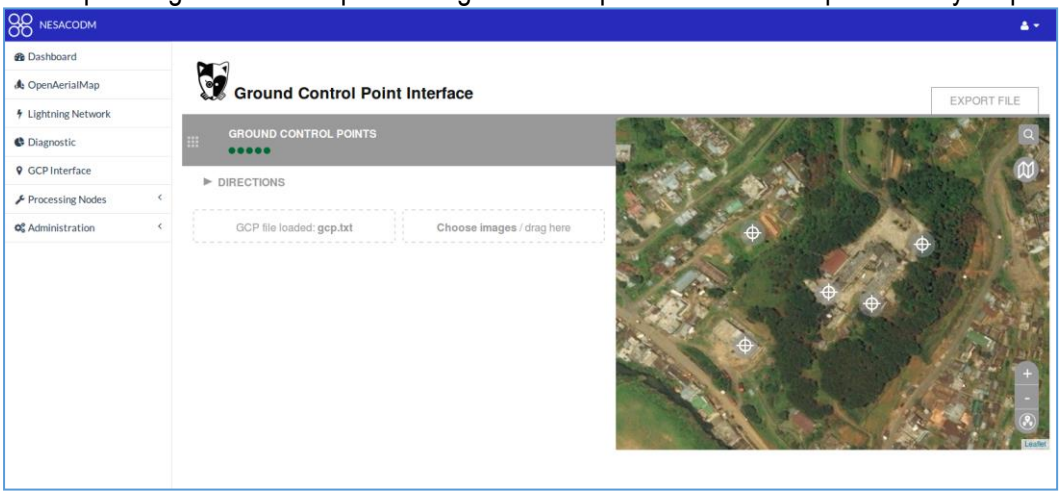
```
X,Y,Z,px,py,Label
91.263877, 25.522005,1346, 0,0,gcp1
91.262870,25.521703, 1332, 0,0,gcp2
91.262929, 25.522513,1340, 0,0,gcp3
91.264279, 25.522433,1350, 0,0,gcp4
91.263534, 25.522086,1346, 0,0,gcp5
```

Finally, we remove the first line, replacing it with a SRS definition and save the CSV file making sure to use *tabs* or *spaces* instead of *commas* as a separator character. In LibreOffice Calc you can achieve this by clicking **File - Save As...** and from the bottom left of the save window check **Edit Filter Settings**. The final result opened in a text editor should look like:

```
EPSG:4326
91.263877 25.522005 1346 0 0 gcp1
91.262870 25.521703 1332 0 0 gcp2
91.262929 25.522513 1340 0 0 gcp3
91.264279 25.522433 1350 0 0 gcp4
91.263534 25.522086 1346 0 0 gcp5
```

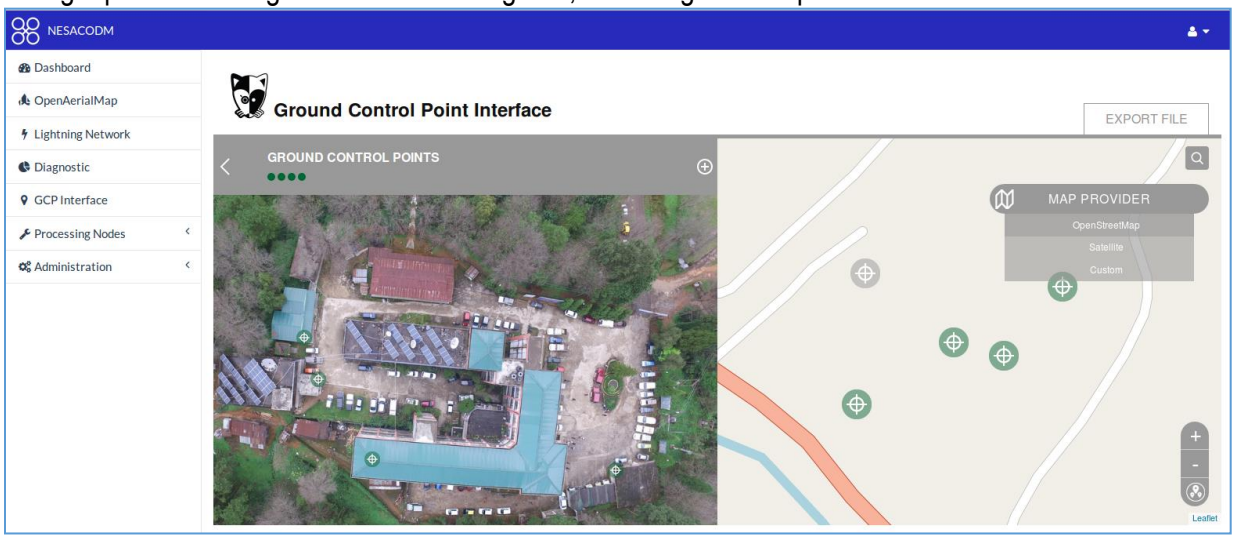
Step 2: Import the GCP file stub

From POSM GCPi, press the **Load existing Control Point File** button and select the file you just created in step 1. After pressing **Load** the map on the right should update to reflect the position of your points.



Loading a GCP stub into POSM GCPi

Step 3. Import the images and start tagging
Now import the images by pressing **Choose images**. Clicking an image will expand the panel on the left. By default for the sake of user friendliness, the interface will add a new ground control point, which we need to remove. You can click the point on the right panel and delete it. Now from the left panel you can pan and zoom around the image to move the target icon at the location of your marker. Once it's in the proper position, simply click the target icon once from the left panel and click the corresponding target icon from the right panel. The target icon should turn green, indicating a correspondence has been set.



Tagging images with points using POSM GCPi

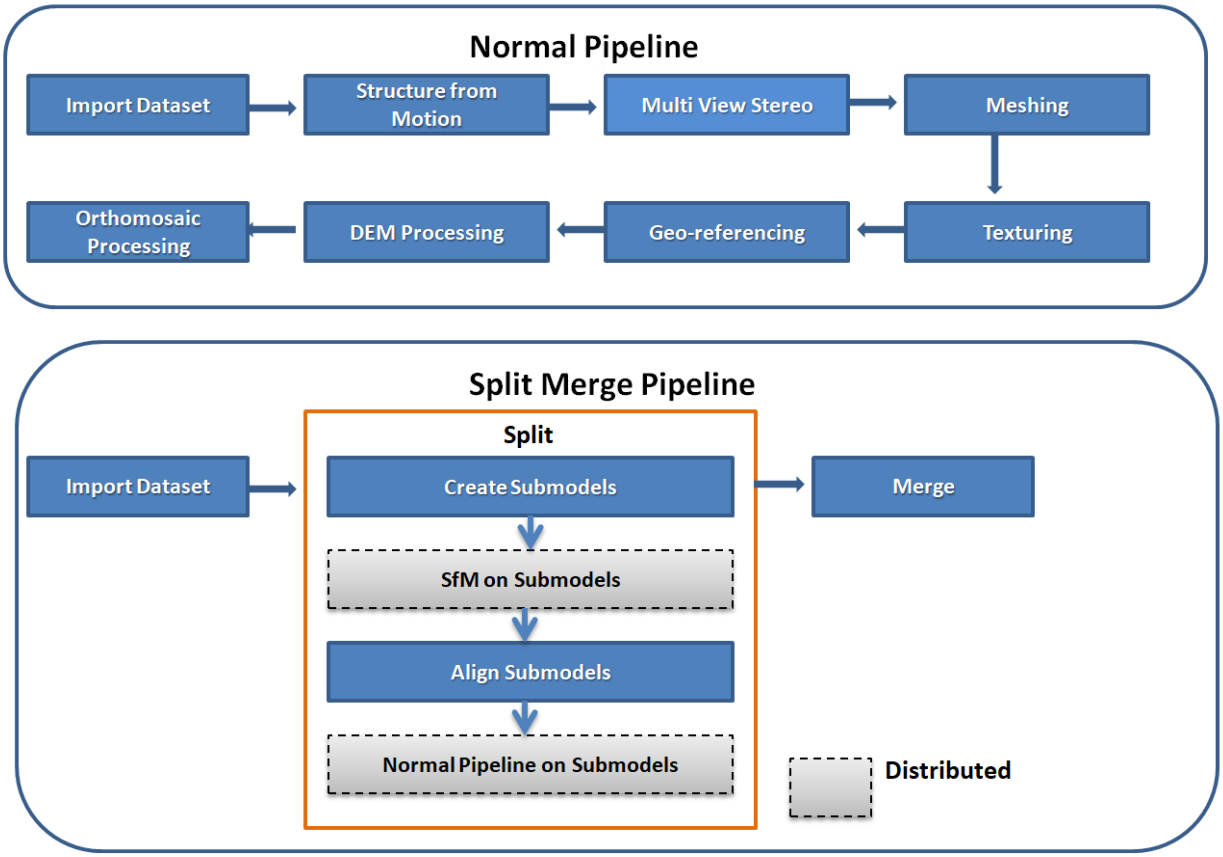
Now you can repeat the process for every marker and every image. If you get tired or want to save your work, simply press **Export File** and save the result in a location of your choice. You can resume your work by reloading the images and the exported file. Notice the green dot on the top left corner. The dot shows the number of points that have been connected to at least one image (in this case, four). At the end of the process you'll want to have at least 5 or 8 green dots.

Step 4. Export the GCP file
Once you are done, you can press the **Export File** button to export your finished GCP file. This is the file that you will be using for the next steps.

Using GCP files
With WebODM and NodeODM using a GCP file is as simple as including it along with the images during file upload. With ODM, the GCP file should be placed in your project folder and be named **gcp_list.txt**. Alternatively, you can use the **gcp** option to specify a path to a GCP file. For example, if images are stored in **D:\odm\project\images** and your GCP file is stored in **D:\odm\project\gcp.txt**, you can simply invoke from the command line:
$ docker run -ti --rm -v //d/odm/project:/datasets/code opendronemap/odm --project-path /datasets --gcp/datasets/code/gcp.txt

Learning Goal 06: Processing Large Datasets by splitting and merging

ODM/WebODM can needs lots of memory while processing. The memory requirements should increase almost proportionally with the number of UAV images and their resolution provided. But what if you have 1000s of high resolutions UAV images to process. The WebODM/ODM's **split-merge** will help splits a large dataset into smaller, manageable parts called sub-models **with some overlap between them**. Each sub-model is processed independently or in parallel on multiple machines base on the settings. After all the sub-models are processed, the results are then combined/merged together to form a single large consistent model. This way, users can process much larger datasets with less powerful computers with lesser RAMs!.



Split-merge pipeline. Step 2 and 4 of the split section can be performed in parallel on separate machines when using distributed split-merge

Split-Merge Options

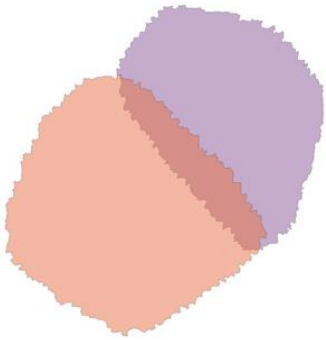
Split-merge can be used either from the command line (ODM) **or from WebODM** and is turned on/off by setting the **split**
option. Split-merge will be turned on anytime the following condition is true:

Split Option < Number of Images>

This option specifies the average number of images that should be included in each sub-model. Note that this does not guarantee that your sub-models will have this exact amount of images. In fact, some sub-models might end up having twice as many images as others. It just means that the following expression holds true:

split-overlap

In order to align and merge results, each sub-model must be reconstructed with a certain amount of overlap and redundancy with other sub-models.



Overlap area between two sub-models

The amount of overlap in meters is specified with this option. Datasets captured at higher altitudes should use a larger value, while those captured at lower altitudes can use lower values. More overlap will significantly slow down computation because of the redundancy of processed data, but can help achieve better model alignment during the merge step. If the resulting DEMs and point clouds from different sub-models show big gaps in elevation, try increasing the overlap.

sm-cluster
This option enables distributed split-merge by specifying a URL to a ClusterODM instance.

merge
After splitting and computing each individual sub-model, results need to be merged back together. By default all outputs (point clouds, DEMs and orthomosaics) are merged. A user can use this option to specify that only a particular type of output should be merged. Valid options are:
• all
• pointcloud
• orthomosaic
• dem

ClusterODM
ClusterODM is a program to connect together NodeODM API compatible nodes. It allows for tasks to be distributed across
multiple nodes while taking into consideration factors such as maximum number of images, queue size and slot availability
A ClusterODM looks like a normal NodeODM node from the outside and can operate with any tool that also works with
NodeODM. To start ClusterODM, simply type:

```
$ git clone https://github.com/OpenDroneMap/ClusterODM
$ cd ClusterODM
$ docker-compose up
```

Quick Comparison of ODM Vs Pix4D(Proprietary) Products

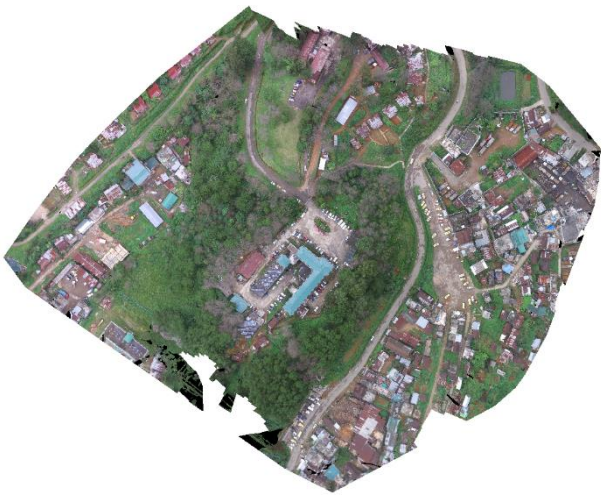
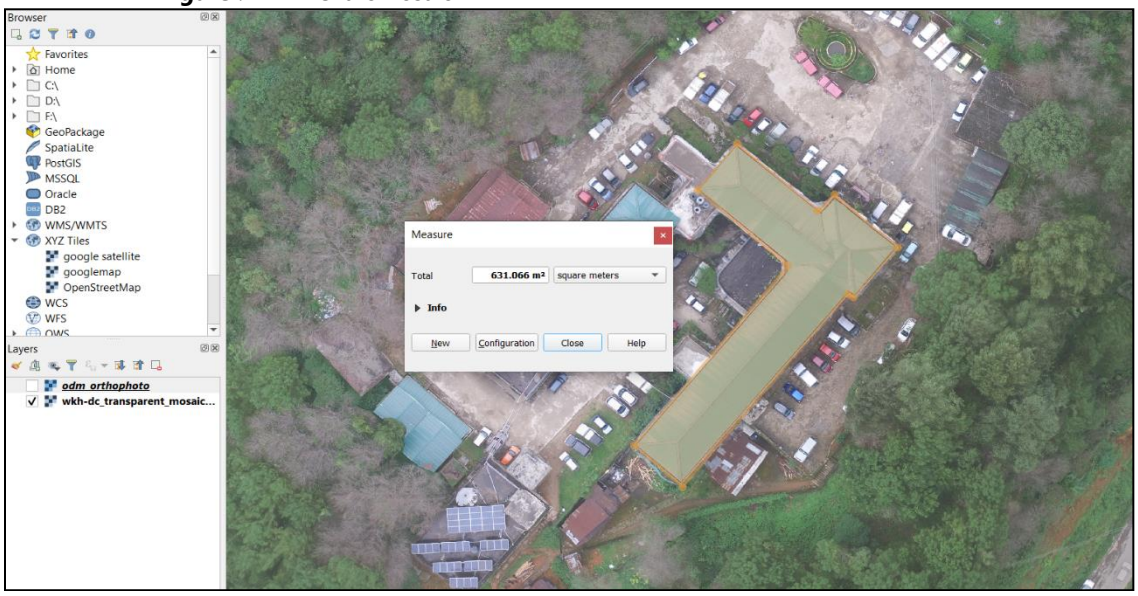


Figure : Pix4D Orthomosaic ODM Orthomosaic

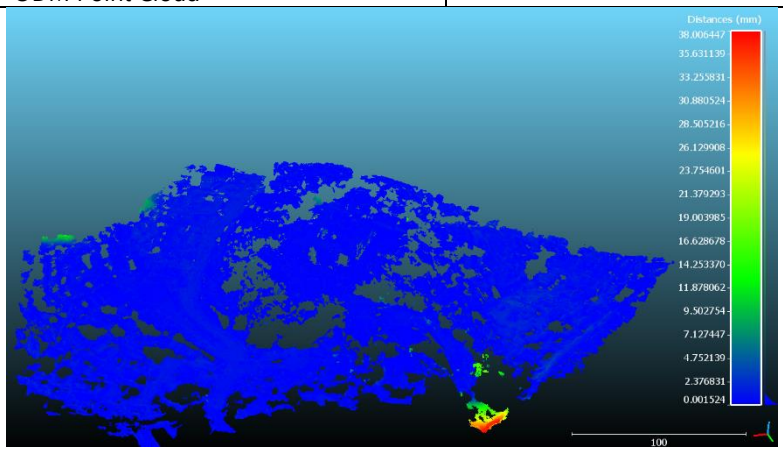


Measurement of Area on Pix4D orthophoto [631.066m^2]

Measurement of Area on ODM orthophoto [630.839m^2]

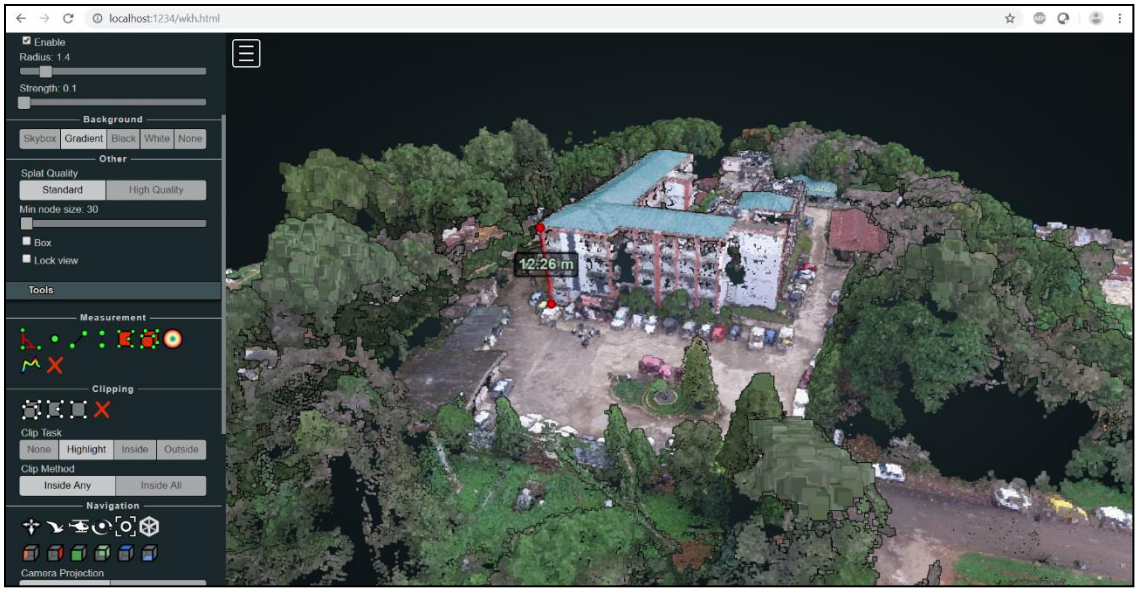
ODM Point Cloud

Pix4D Point Cloud

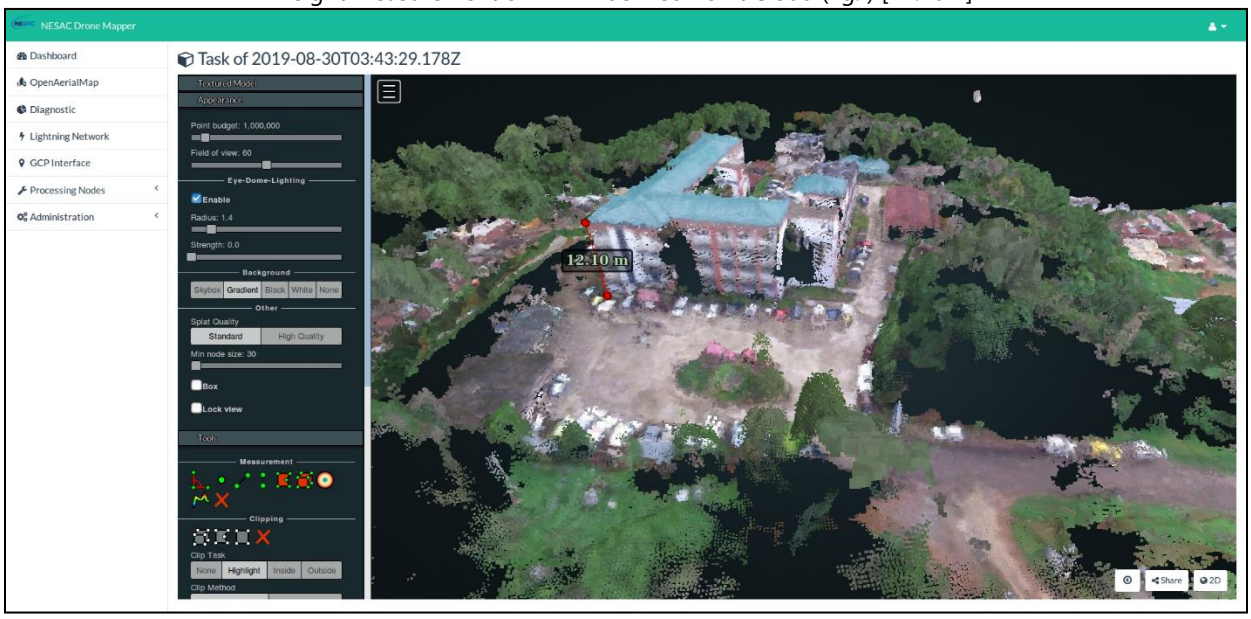


Point Cloud Comparison

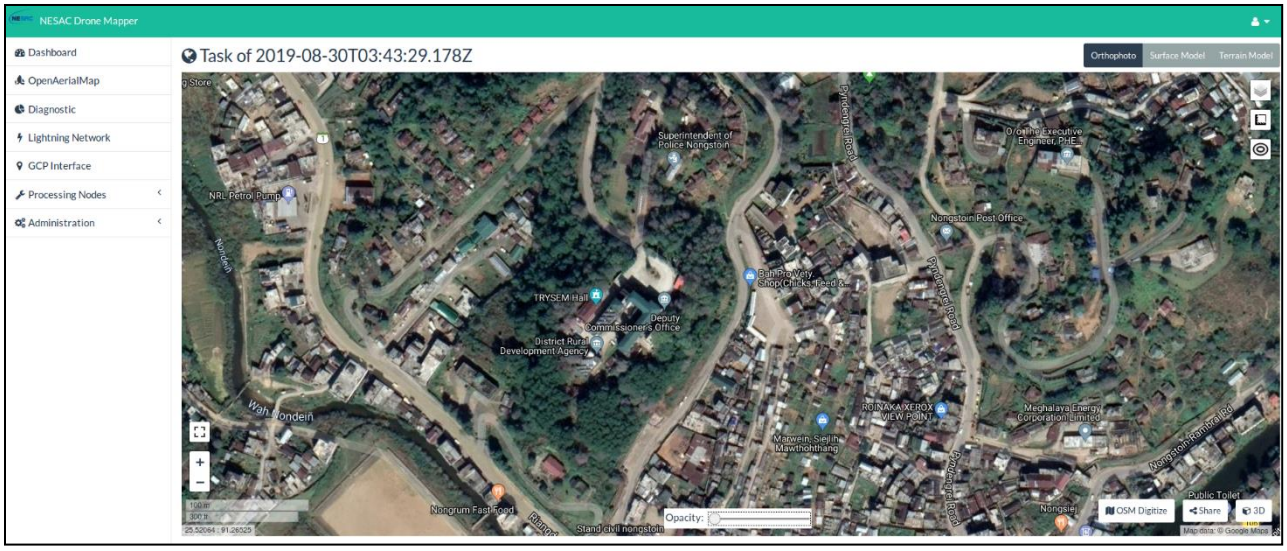
The Pix4D derived Point cloud has much denser point cloud than ODM. But, ODM still gives 3D point cloud representing accurate 3D scene structures. A comparison has been made (taking ODM Point cloud as a base) after proper registration and computing Point-to-Point distance. We observed almost zero distances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



Height Measurement on Pix4D derived Point Cloud (Eg.) [12.26m]



Height Measurement in ODM derived Point Cloud [12.10m]



Google Map View of the Area under Study

Orthomosaic overlaid on Google Map

Image as seen in Google Map

UAV Orthophoto

Observe the fine details in UAV Orthophoto

Glossary

2.5D Model: A model where elevation is simply *extruded* from the ground plane and thus is not a true 3D model.
Artifacts: undesired alterations generated as the result of a digital process.
API: Application Programming Interface. A set of functions allowing the creation of applications that access the features or
data of another application.
AWS: Amazon Web Services is a cloud service provider.
Bundle Adjustment: a refinement step during the Structure From Motion process that improves the location of cameras,
the 3D points of the scene and the camera parameters.
CloudODM: A command line tool to process aerial imagery in the cloud.
ClusterODM: A NodeODM API compatible autoscalable load balancer and task tracker for connecting multiple NodeODM
nodes under a single network address.
CRS: Coordinate Reference System. A CRS is a coordinate based system used to locate geographical entities.
CSV: Comma Separated Value is a textual file format where fields are typically separated by commas or some other character such as a space or a tab.
cURL: a software providing a library and command-line tool for transferring data using many protocols.
DEM: Digital Elevation Model (either a DSM or a DTM).
Depthmap: An image containing distance information for objects in a scene relative to the camera plane.
Docker: a tool used to launch *containers*, lightweight standalone packages of software. OpenDroneMap uses docker
to run many of its software. Docker is also the name of the company that develops the tool.
DSM: Digital Surface Model. A 2D representation of elevation that includes terrain, buildings, trees and other structures.
DTM: Digital Terrain Model. A 2D representation of elevation that includes terrain only.
EXIF: Exchangeable file format for images and their auxiliary tags. EXIF tags are pieces of information embedded within
an image. They can include information such as camera model, GPS location of where the image was shot, focal length
information and many more.
GCP: Ground Control Point. A GCP is a position measurement made on the ground, often taken at the location of a clearly
identifiable marker, to increase the georeferencing accuracy of a reconstruction.
GSD: Ground Sampling Distance. In an aerial photo, it's the distance between pixels measured on the ground.
Mesh: A collection of vertices, edges and faces that define the shape of a 3D model. A mesh does not include color
information such as textures.
MVE: Multi-View Environment is a suite of software packages developed to ease the work with multi-view datasets and to
support the development of algorithms based on multiple views. It features Structure from Motion, Multi-View Stereo and

Surface Reconstruction algorithms.
MVS: Multi-View Stereo is a branch of study in computer vision that focuses on the reconstruction of 3D models from multiple overlapping image pairs. MVS programs expect that information about cameras has already been computed.
NodeODM: A lightweight REST API to access aerial image processing engines such as ODM or MicMac.
Noise: An unwanted interference. When applied to point clouds, it indicates points that should not be present or that
were missed during outlier filtering.
Nadir: The direction pointing directly below a particular location.
ODM: A command line toolkit to generate maps, point clouds, 3D models and DEMs from drone, balloon or kite images.
OpenSfM: Open source structure from Motion library written in Python. The library serves as a processing pipeline for
reconstructing camera poses and 3D scenes from multiple images.
Orthomosaic: An image that has been *orthorectified*, warped in such a way that distances and scales are uniform.
Photogrammetry: the process of obtaining reliable information about physical objects and the environment through the
process of using photographic images.
Preemptive Matching: During the Structure From Motion process, the act of reducing the possible number of pair
candidates by using the location information stored in the EXIF tags of the images.
PyODM: A Python SDK for adding aerial image processing capabilities to applications.
RTK: Real Time Kinematics. A satellite navigation technique used to enhance the precision of position data derived from
satellite-based positioning systems.
SDK: Software Development Kit. A set of libraries, tools and examples that help software developers to build software with a particular technology.
SFM: Structure From Motion is a photogrammetry technique for estimating 3D objects (structures) from overlapping image
sequences (from motion).
Texturing: The act of creating 2D images suitable for use on 3D models, also known as *texture maps* or *texture skins*. Textures give 3D models a realistic appearance.
UAV: Unmanned Aerial Vehicle. An aircraft piloted by remote control or on-board computers.
UI: User Interface.
UTM: Universal Transverse Mercator is a coordinate reference system. It ignores altitude and treats the earth as a perfect
ellipsoid.
Virtualization: the act of creating a virtual version of computer hardware platforms, storage devices, and computer
network resources.
VM: Virtual Machine. A software program or operating system that works like a separate computer.
WebODM: User-friendly, extendable application and API for processing aerial imagery.
WSL: Windows Subsystem for Linux. A feature of Windows 10 that allows Linux programs to run natively on Windows