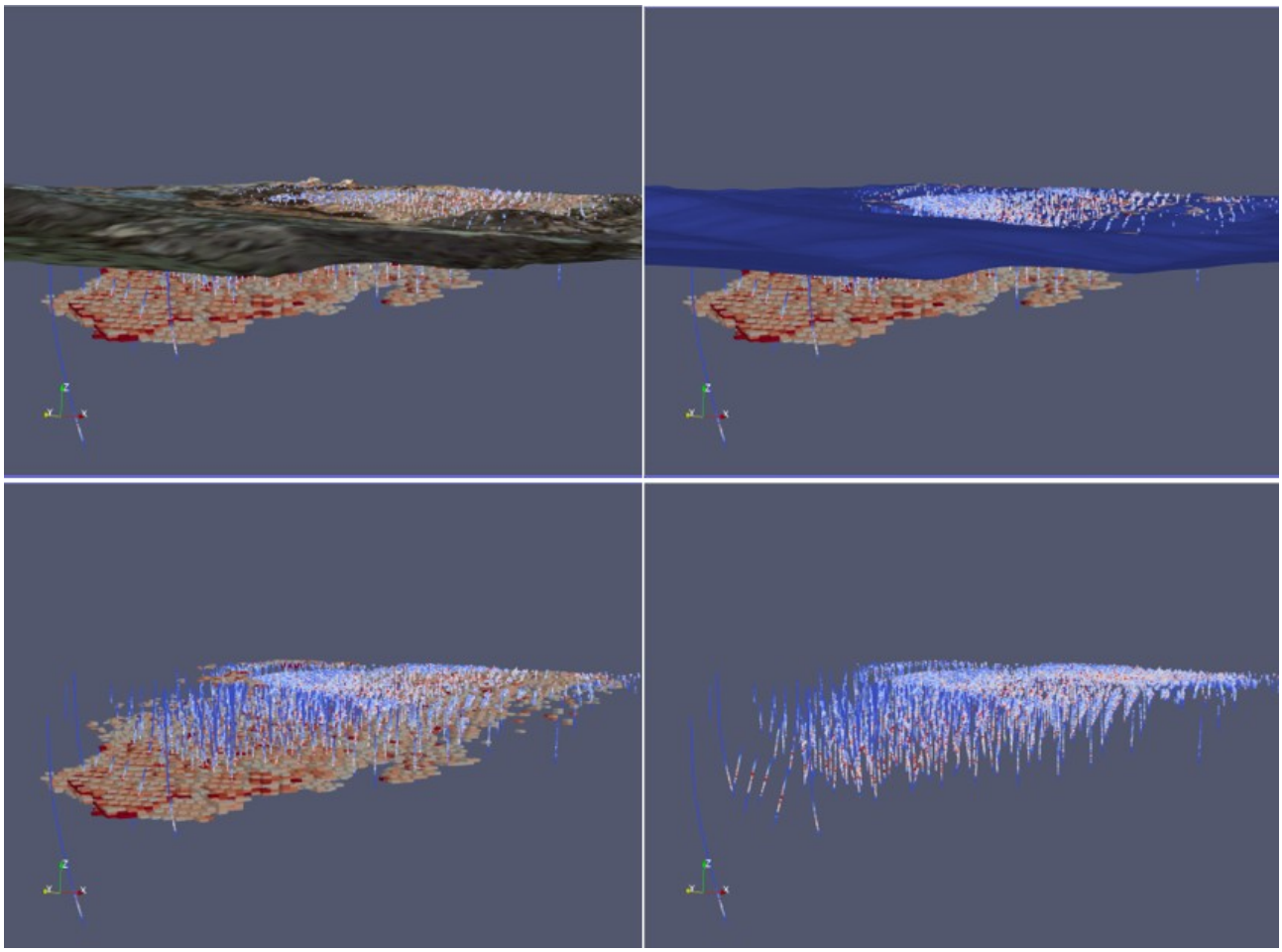
Using Free Open source software for preliminary geological resource modeling on an open pit mine. Parts 1 and 2



André Luiz Lima Costa

What it is needed to use this tutorial

- A computer that can run the software listed below;
- Any DJI drone with GPS (mini, mavic air 2, mavic 2 pro, mavic 2 zoom, phantom4, etc)
- RTK or PPK GPS system to survey ground control points; and
- Drillhole data (Collar,Survey,Assay).

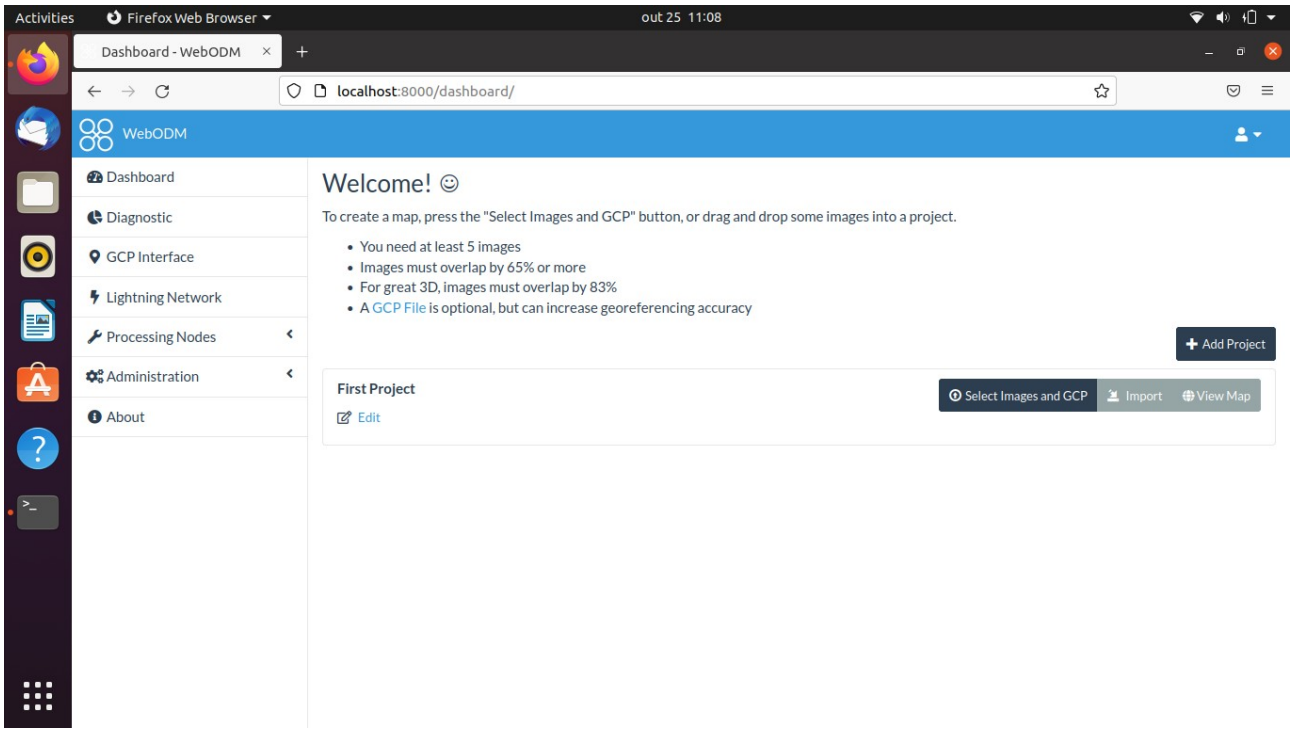
FOSS software used

- **WebODM 1.9 https://docs.opendronemap.org/installation/#installation**

Follow the instruction above to install and run WebODM on your computer:
You will need to install the following applications (if they are not installed already):

git *docker* *docker-compose* *python3* *pip3*

Open a Web Browser to **http://localhost:8000** and you should get the user login page and this page next.



To stop WebODM press CTRL+C or run:

```
./webodm.sh stop
```

Install the following programs too:

- **CloudCompare v2.11or higher** https:/www.danielgm.net/cc/
- **QGIS 3.10 or higher https://qgis.org/en/site/forusers/download.html**
- **Paraview 5.8.0 or higher https://www.paraview.org/download/**
- **Python 3.6 https://docs.conda.io/en/latest/miniconda.html#linux-installers**
- Install this version environment within Anaconda or Miniconda (conda) distributions because pygslib is not supported by higher versions yet.

Create a conda environment with python 3.6
$ conda create -n env36 python=3.6 anaconda
Activate the environment using:
$ conda activate env36

Under this environment use pip to install:
numpy,
pandas,
matplotlib and
psycopg2

Installing *pygslib* → The easiest way to install and work with PyGSLIB is using Anaconda or Miniconda (conda) distributions. To install PyGSLIB in the root environment (3.6) of your anaconda distribution simply type in a terminal:

```
conda install pygslib -c opengeostat -c conda-forge
```

- **R 3.6 or higher https://cloud.r-project.org/**
- **Julia https://julialang.org/downloads/**
- **PostgreSQL/Postgis https://www.postgresql.org/download/**

With these tools up and running it will be covered here all the steps to achieve block model and its data visualization. It is not the objective of this tutorial to cover in-depth aspects of geostatistical theory but some explanation of what it is being covered will be done.

It is assumed that you have enough knowledge to install the software above on your system of choice and have a basic knowledge using command line (cmd, monitor, etc) and creating text files with a text editor such as notepad, nano, vim, etc. Some aspect or capabilities of QGIS, Paraview or CloudCompare will not be covered but it is assumed that you can perform basic tasks on these applications such as locate a menu item that will be needed (open, save as, use a tool and change parameters).

It is also assumed that you can create basic scripts using R, python and Julia.

This is an guide/tutorial for those interested in new ways to do resource modeling without the need to spend a lot of money in proprietary software but it is not intended to replace them (yet). It is more a conceptual start that may lead to a complete open source methodology in the future.

PART 1
Preparing the Ground

Drone DTM and Orthophoto creation process

Drone topography survey it is an important tool to quickly evaluate a daily, weekly, bi-weekly or monthly topography and volume changes on an open-pit operation. The interval difference in volume can be used effectively for reconciliation. Below are the steps to transform drone orthophotos into a digital terrain model (DTM) using WebODM. If you already have TIF files of DTM and orthophoto image of your area jump to the next section (**Creating the stl and vtk files**).

Any modern drone from DJI (Phantom 4, mavic 2 pro, mavic 2 air and even mavic mini) can be used to take orthophotos using applications such as DroneLink or Maps Made Easy. Sorry, but no free tools yet to create drone missions.

After taking the photos from your area of interest, with at least 65% image overlap, load the photos from the drone memory and the ground control points (surveyed with the RTK or other) file into a folder on your computer.

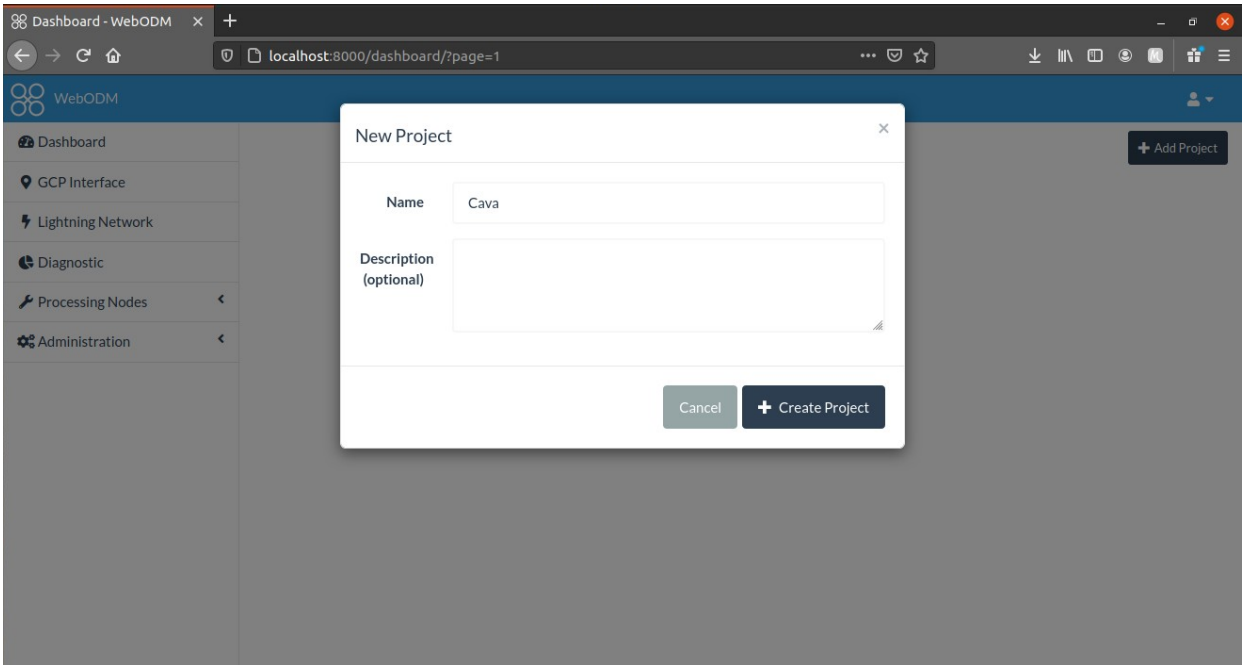
The ground control points file (named **gcp_list.txt**) must have the following format:

- The first line should contain the name of the projection used for the geo coordinates. This can be specified either as a PROJ string (e.g. +proj=utm +zone=10 +ellps=WGS84 +datum=WGS84 +units=m +no_defs), EPSG code (e.g. EPSG:4326) or as a WGS84 UTM <zone>[N|S] value (eg. WGS84 UTM 16N)
- Subsequent lines are the X, Y & Z coordinates, Their associated pixel coordinates, the image filename and optional extra fields, separated by tabs or spaces:
- Elevation values can be set to “NaN” to indicate “no value”
- The 7th column (optional) typically contains the label of the GCP.

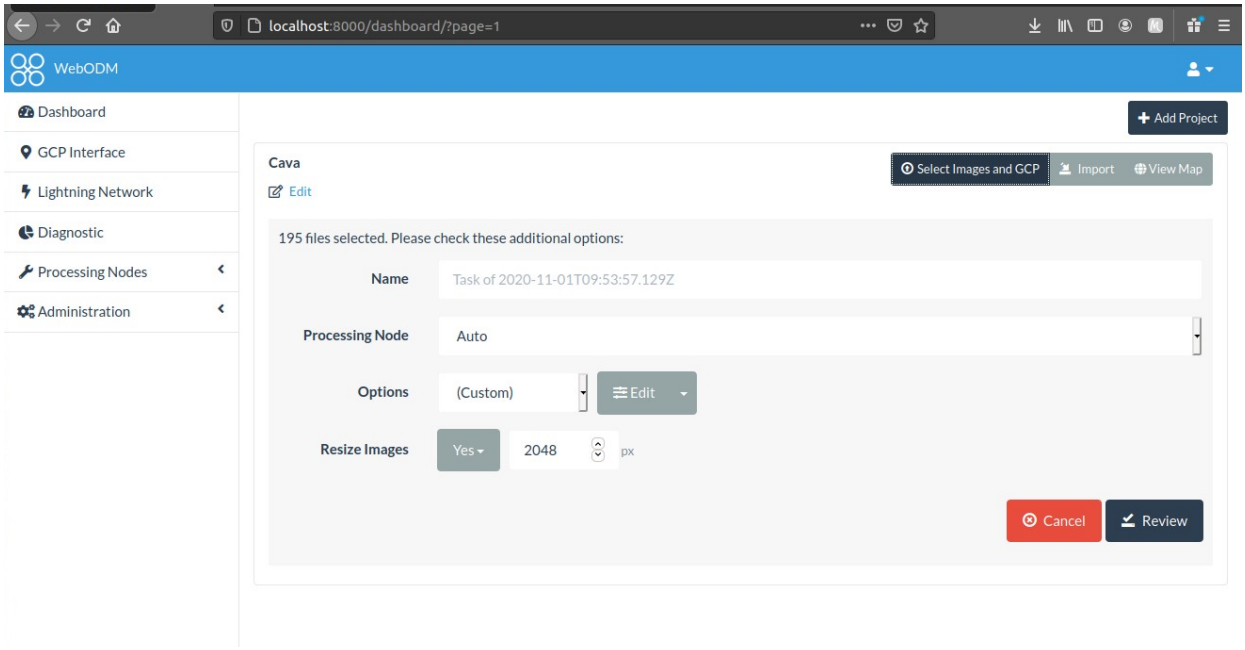
Example:

```
WGS84 UTM 24S
552913.5924 9501921.1768 102.0962 5130 592 DJI_0163.JPG
. . .
. . .
553040.3082 9501999.3036 81.6663 2392 2821 DJI_0247.JPG
```

Start WebODM up to process the drone photos and create a new project

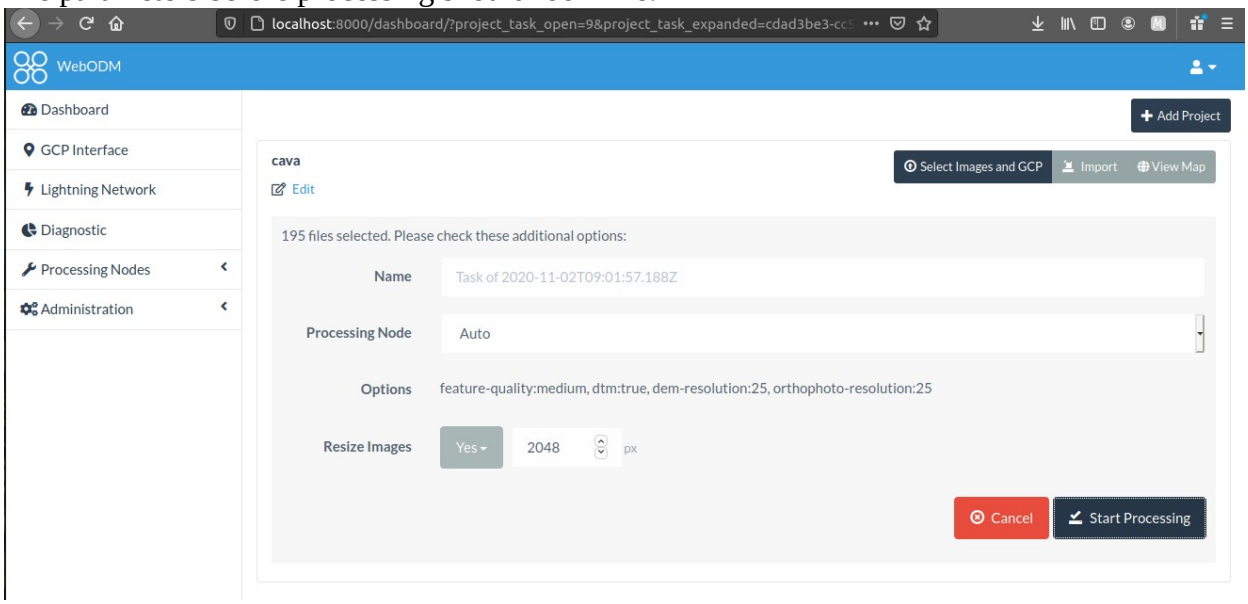


Select the drone photo images and the GCP file



Now click the ‘Edit’ button and fine tune the following processing parameters.
The ones important for now are:
feature-quality: medium
dtm: enable
Enter the dem-resolution and orthophoto-resolution in centimeters

The parameters before processing should look like:



Click the ‘Start Processing’ button. It may take a while depending on the number of photos, resolution used and computer processing speed and memory. It takes 2 hours to process 200 photo images using medium quality at 25 cm resolution on a Notebook with a core i5 processor, 4GB RAM (running Ubuntu). Consider a modern multi core processor with 32GB RAM to process higher resolution or several images.

A full list of the WebODM parameters / arguments and what they are can be found using this link https://docs.opendronemap.org/

After processing, save all the created files using the button at the bottom-left of the screen.
Unzip the file and extract these two files:
/odm_dem/dtm.tif
/odm_orthophoto/odm_orthophoto.tif

Now we can create point cloud with R to generate a stl file using CloudCompare. This stl will be used to clip the modeled blocks in Part 3. The DTM, integrated with the orthophoto mosaic, will create a VTK in QGIS file that will be used to illustrate the surface.

Creating the stl and vtk files

The stl file

Use the **dtm.tif** file, the R script below and CloudCompare to create the clipping surface stl file (**clipsurf.stl**)

First, from the folder where the **dtm.tif** is located, run the R script below to create a point cloud:
You will need to install the library ‘ raster’ but, from command line install gdal-dev first with (Ubuntu):

```
sudo apt-get install libgdal-dev
```

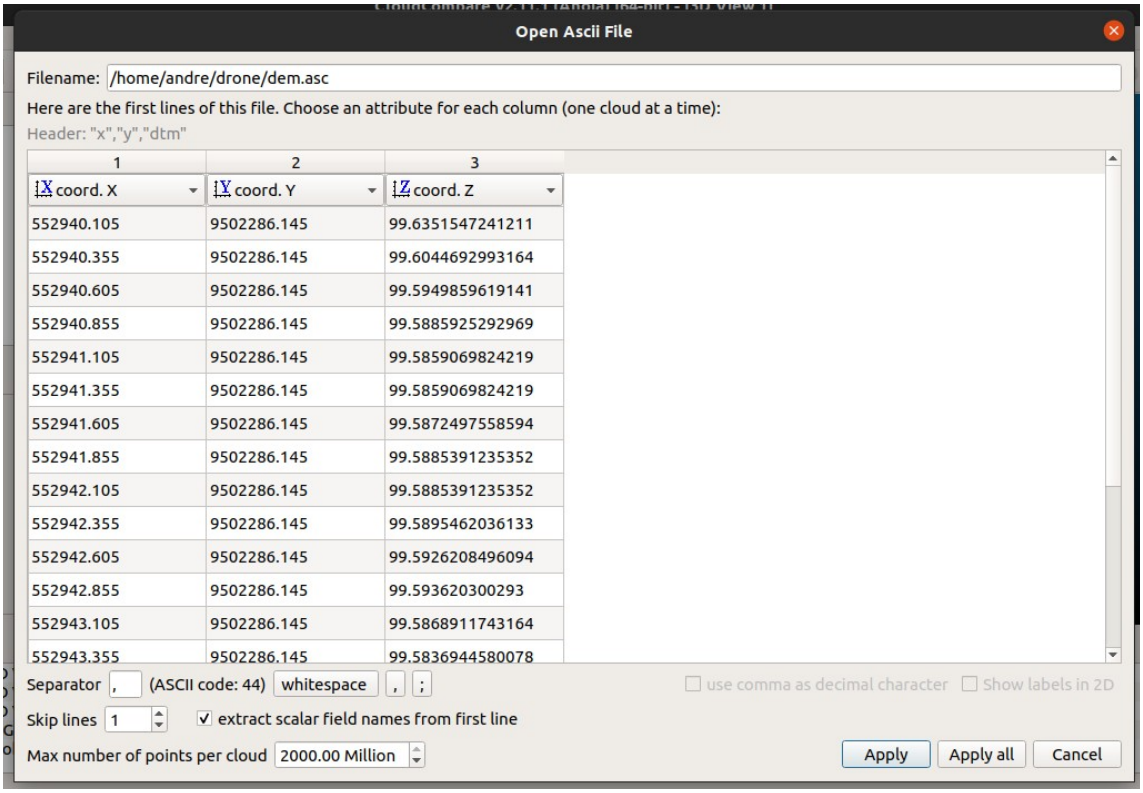
In R install the packages

```
install.packages('raster')
install.packages('rgdal')
```

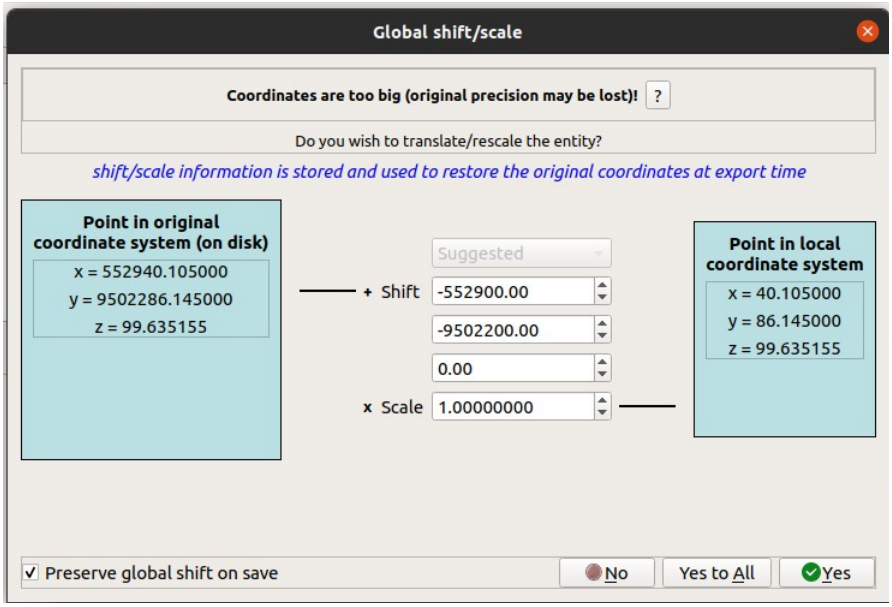
Now in R use

```
library(raster)
dem<-raster('dtm.tif')
xyz <- rasterToPoints(dem)
write.csv(xyz,"dem.asc", row.names = FALSE)
```

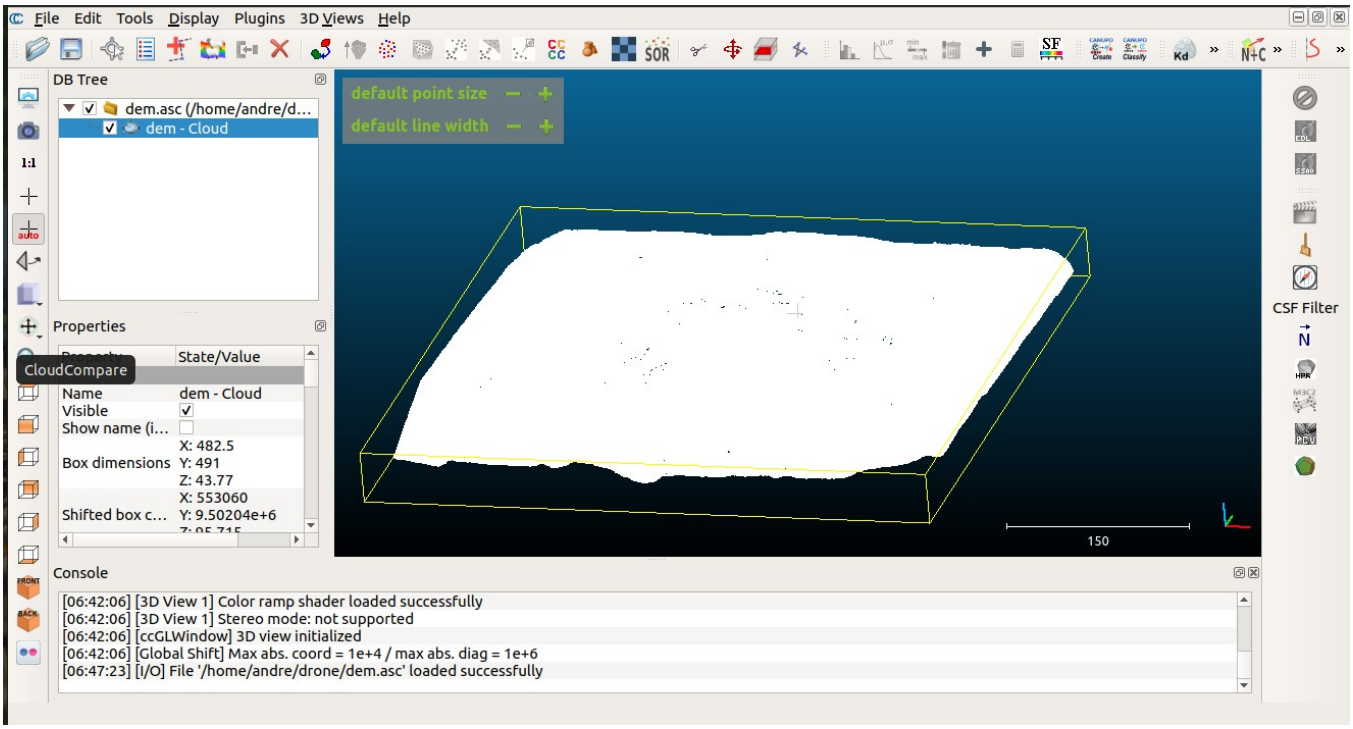
This will create a **dem.asc** point cloud file that we will now open using CloudCompare. The screen below should appear when the file is opened, select skip one line on skip lines box and click Apply:



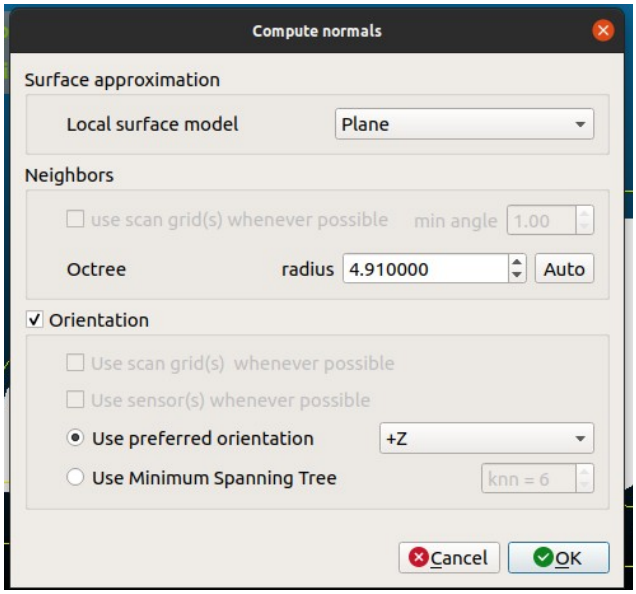
Click ‘No’ on this next dialog box



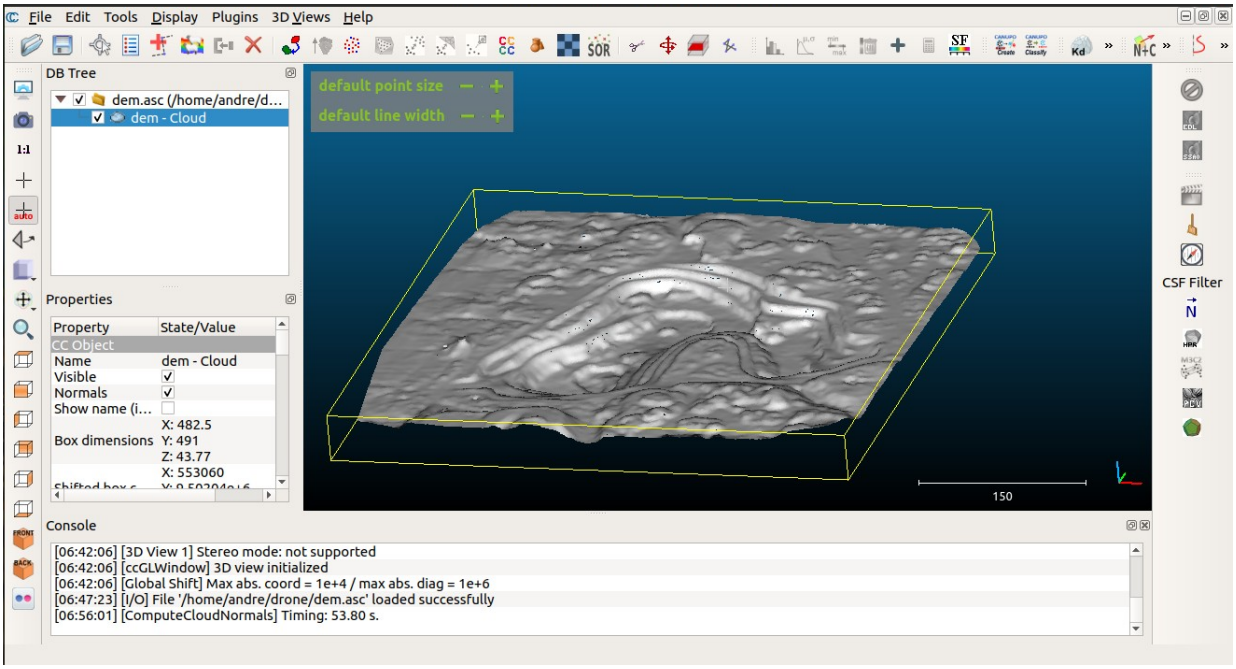
Select the dem-Cloud item and select menu **Edit → Normals → Compute**



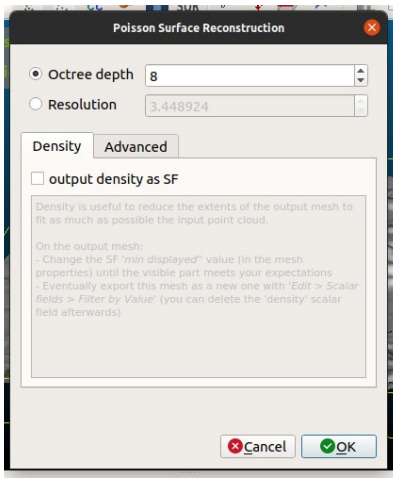
The dialog should look like this (Use preferred orientation +Z selected) before clicking ‘OK’

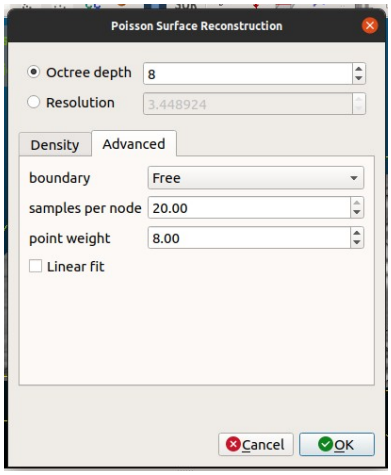


After computing the normals the point cloud object will look like:

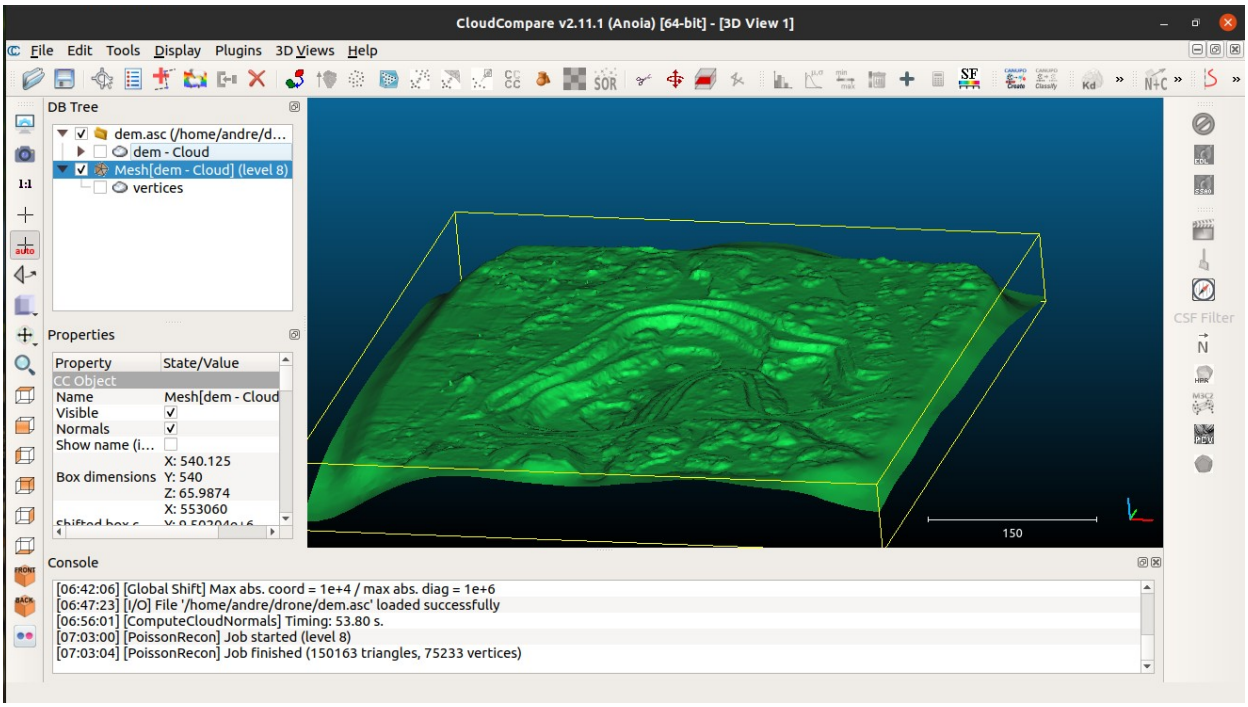


Select menu **Plugin → PoissonRecon** and the following dialog will appear. Select the advanced tab and select ‘Free’ boundary, 20 samples per node and point weight of 8. The octree depth should be 8. Click ‘OK’. (if no satisfactory result is found, try it with different values)





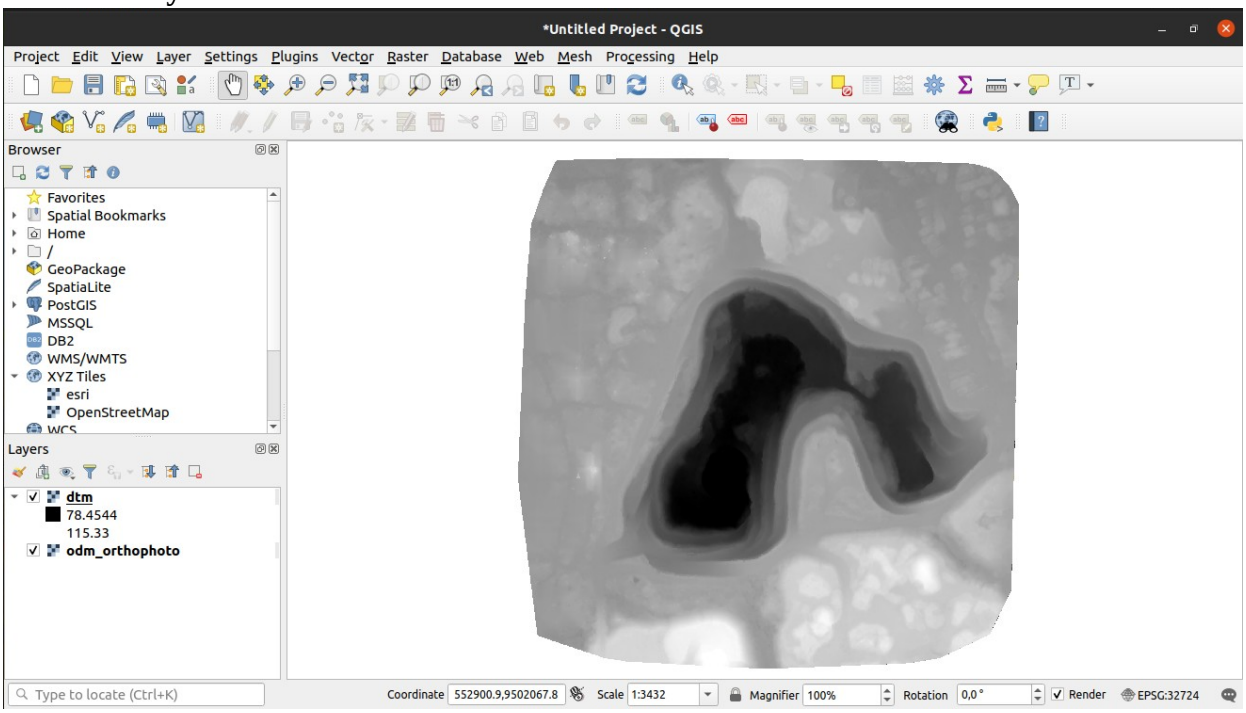
The resulting mesh should look like:



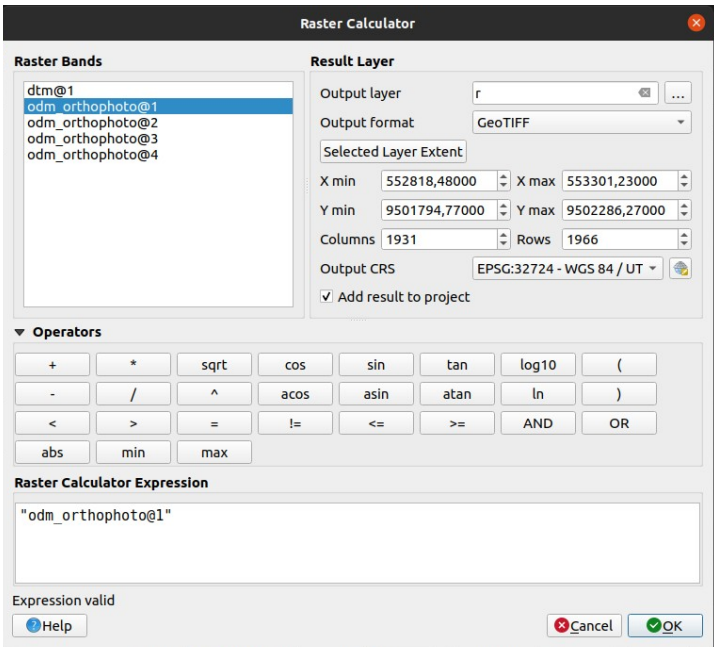
Select the mesh item and save it as STL mesh with the filename of **clipsurf.stl**
Close CloudCompare.

The vtk file

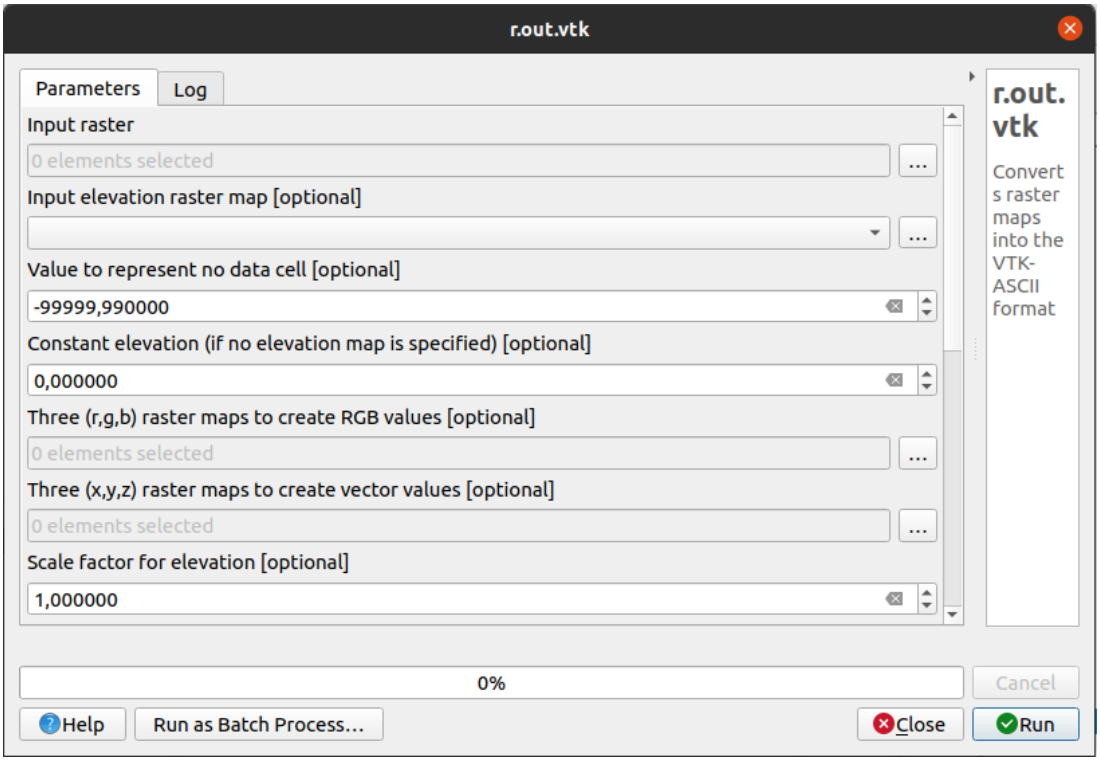
Start QGIS and load the files **dtm.tif** and **odm_orthophoto.tif** using menu **Layer→Add Layer → Add Raster Layer**.



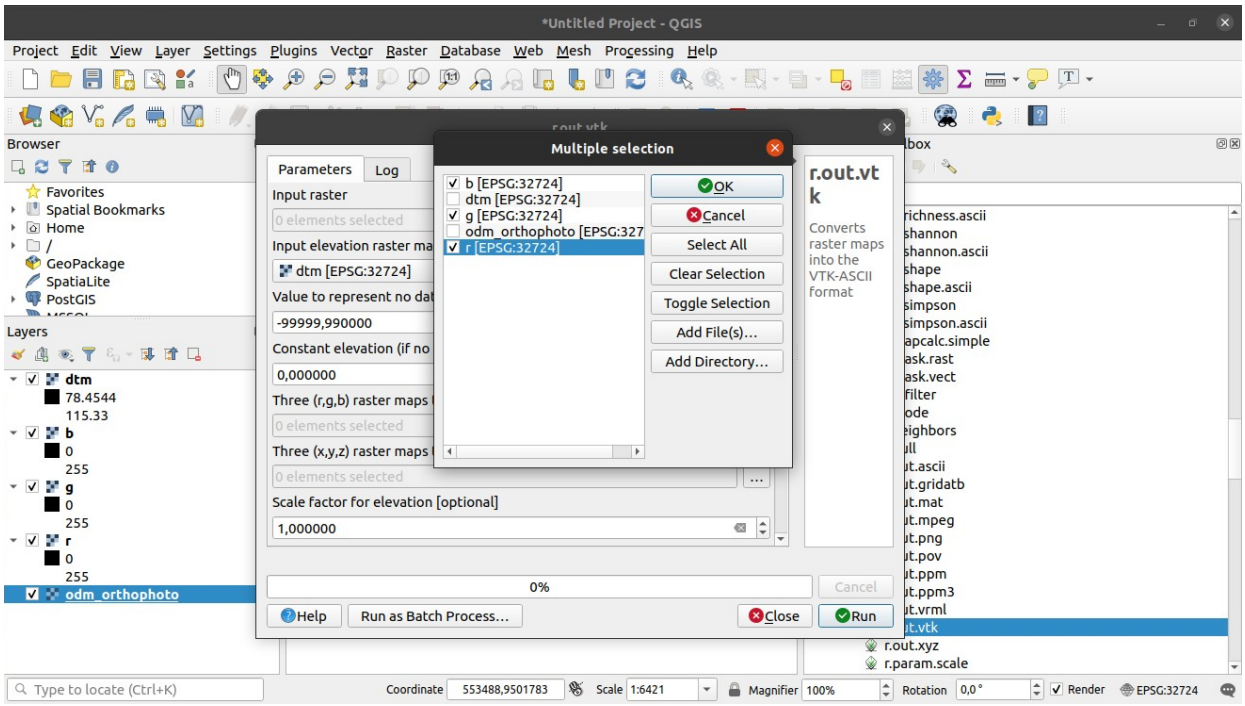
It is necessary to split the bands r, g and b from the odm_orthophoto layer. This is done by selecting the menu **Raster → Raster Calculator**. Double click on odm_orthophoto@1 band and input 'r' into Output layer. Click 'OK'. Repeat the same using odm_orthophoto@2 input 'g' into Output layer and odm_orthophoto@3 and input 'b' into Output layer.



Once the layers r, g and b are created. Select ***Processing → Toolbox*** and select and run ***GRASS → Raster → r.out.vtk*** by double clicking on it. The following window will appear:



Select the dtm layer as elevation input and the three (r, g and b) bands raster maps as the color to be added into the VTK file.

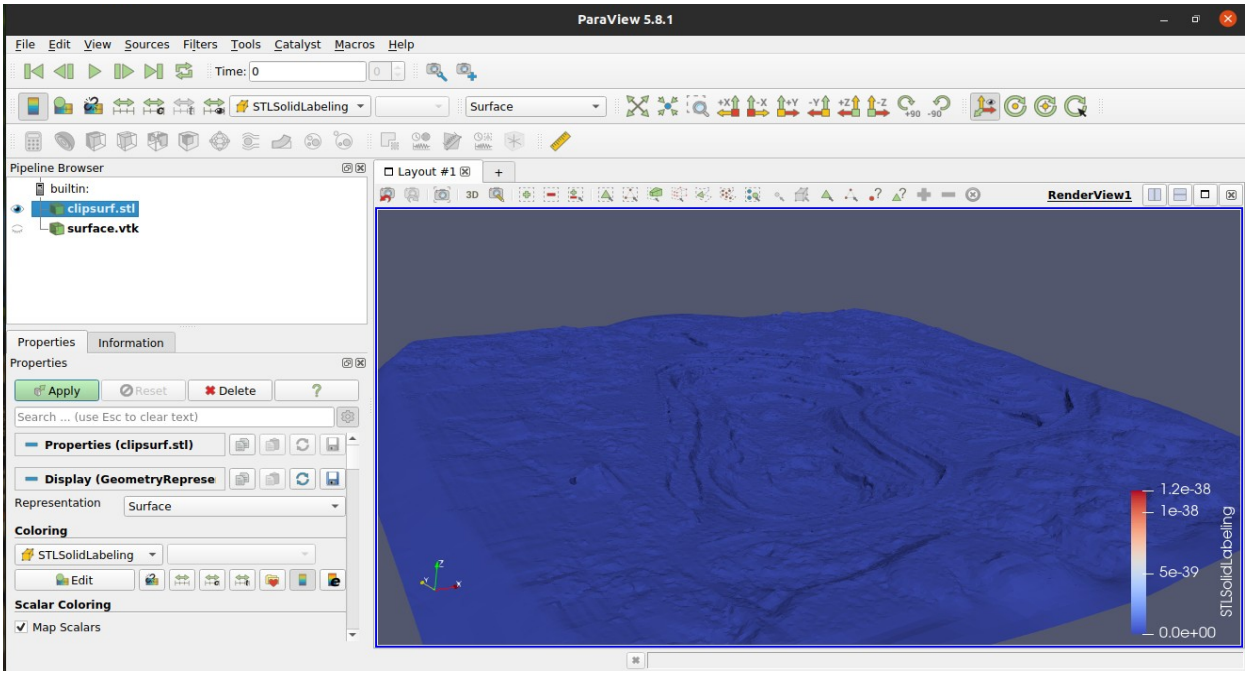


Replace the values to represent ‘no data cell’ and ‘constant elevation value’ (**use the maximum elevation of your area here for best results or the border elevation of the pit**). Add an outfile name (**surface.vtk**) of your VTK file and run the tool. After creating it you can close QGIS. A VTK file should be created with the draped values of the R, G and B bands composite.

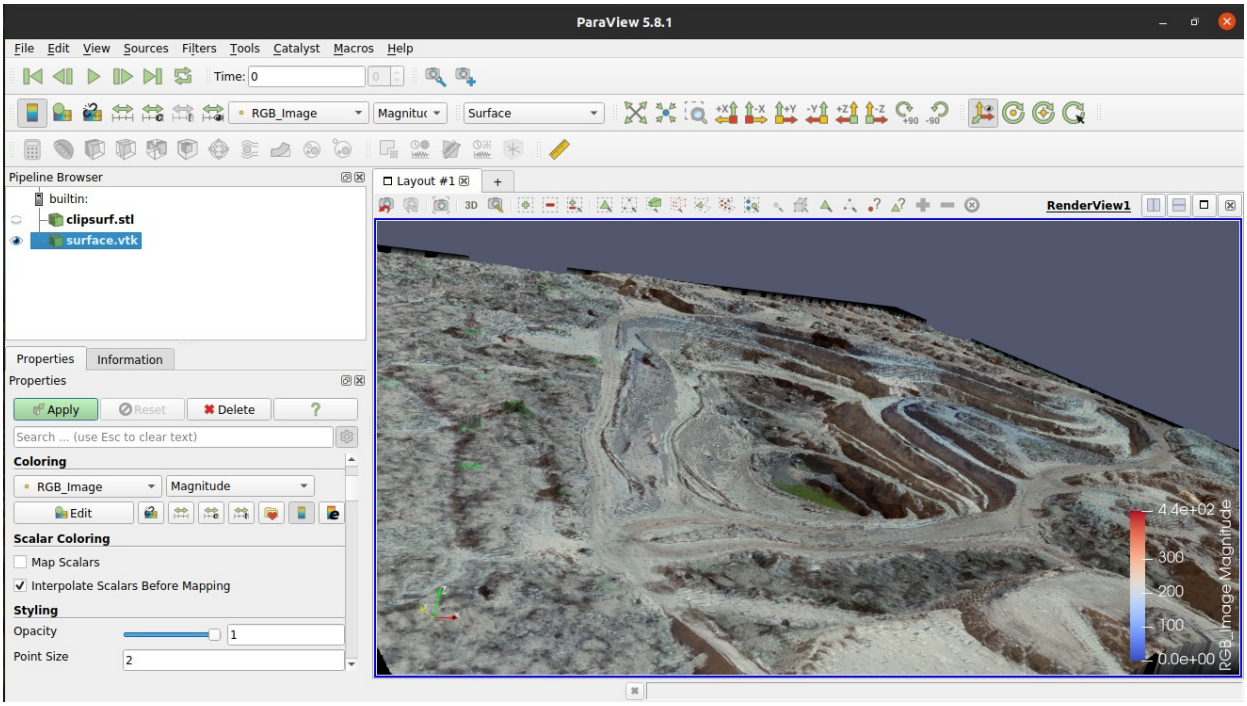
The Final 3D Surface

Use **Paraview** to check the files created above. Open them using the menu **File→ open** (uncheck the Map scalar checkbox for the vtk file to see the RGB color composite).

These files should look like:
clipsurf.stl



surface.vtk



This reflects the actual pit surface. The next part will cover the loading of drillhole associated with its assay data using different methodologies.

PART 2
Attaching Assay results to Drillholes in 3D

Creating a PostgreSQL database to load drill hole and assay results

A minimal set of tables will be created now using a PostgreSQL / Postgis database. A real world geodatabase is more complete with several tables and lookup tables. The bare minimal will be created here to continue with our tutorial.

**It is assumed that this database target area is on UTM 24S WGS-84 datum (SRID 32724) and with 1 element assayed. Adjust the database in accordance with your data.*

The drillhole data must be within the area covered by the Part images of course.

As a database user, create a database named 'geofoss' and create the postgis extension on it using:

```
createdb geofoss --encoding=utf-8
psql -d geofoss -c "CREATE EXTENSION postgis"
```

Create the necessary tables by creating a file **create.sql** with the content below.

```
CREATE TABLE collar(
bhid        varchar(50) NOT NULL,
xcollar     numeric(12,4) NOT NULL,
ycollar     numeric(12,4) NOT NULL,
zcollar     numeric(8,4) NOT NULL,
geom        geometry(PointZ,32724),
UNIQUE(bhid));

CREATE TABLE survey(
bhid        varchar(50) NOT NULL,
_at         numeric(8,4) NOT NULL,
az          numeric(7,4) NOT NULL,
dip         numeric(6,4)NOT NULL,
UNIQUE(bhid,_at));

CREATE TABLE assay(
bhid       varchar(50) NOT NULL,
_from      numeric(8,4) NOT NULL,
_to        numeric(8,4) NOT NULL,
el1        numeric(9,4),
sample     varchar(50) NOT NULL,
UNIQUE(sample));
```

Save the file and run it using:

```
psql -d geofoss -f create.sql
```

Check the database by running psql geofoss on your system and type `\d` to see the tables. You should see something like this:

```
psql geofoss
psql (11.5)
Type "help" for help.
geofoss=# \d
              List of relations
 Schema |       Name        | Type  | Owner
--------+-------------------+-------+--------
 public | assay             | table | you
 public | collar            | table | you
 public | geography_columns | view  | you
 public | geometry_columns  | view  | you
 public | raster_columns    | view  | you
 public | raster_overviews  | view  | you
 public | spatial_ref_sys   | table | you
 public | survey            | table | you
(8 rows)
geofoss=#
```

Type `\q` to exit. If you are running Postgis 3 you will not see the raster table listed, If you want to work with raster on your database and you have postgis 3 use the command below:

```
psql -d geofoss -c "CREATE EXTENSION postgis_raster"
```

Now it is time to load the data into the tables. See the steps below to format your data and feed the database.

Put the collar data into a **collar.csv** file using the format as showed in this example

```
BHID,XCOLLAR,YCOLLAR,ZCOLLAR
hol_1,345678,9876543,78.6476
…
hol_N,354876,9876644,87.5644
```

The survey (**survey.csv**) data has to have at least a collar and a bottom entry for each hole entry using the format showed in the example below:

```
BHID,AT,AZ,DIP
hol_1,0,0,90
…
hol_N,0,180,60
hol_N,34.80,180,60
```

Finally enter the sample assay (**assay.csv**) data in the following format (adjust elements if needed):

```
BHID,FROM,TO,EL1,SAMPLE
hol_1,20,21,23.4,A_001
…
hol_N,14,15.5,34.8,A_067
```

Use the R script below to load the drillhole data into the geofoss database (replace user and password accordingly):

```
library(RPostgreSQL)
col<-read.csv('collar.csv',header=TRUE)
sur<-read.csv('survey.csv',header=TRUE)
asy<-read.csv('assay.csv',header=TRUE,na.strings = "NULL")
asy[is.na(asy)] <- 'NULL'
con<-dbConnect(PostgreSQL(),host="127.0.0.1",dbname='geofoss',user="username,
password="*")
for(i in 1:nrow(col)){
 sql<-paste0("INSERT INTO collar (bhid,xcollar,ycollar,zcollar)
VALUES('",col$BHID[i],"',",col$XCOLLAR[i],",",col$YCOLLAR[i],",",col$ZCOLLAR[i],
");")
 dbGetQuery(con,sql)
}
for(i in 1:nrow(sur)){
 sql<-paste0("INSERT INTO survey (bhid,_at,az,dip)
VALUES('",sur$BHID[i],"',",sur$AT[i],",",sur$AZ[i],",",sur$DIP[i],");")
 dbGetQuery(con,sql)
}
for(i in 1:nrow(asy)){
 sql<-paste0("INSERT INTO assay (bhid,_from,_to,el1,sample)
VALUES('",asy$BHID[i],"',",asy$FROM[i],",",asy$TO[i],",",asy$MN[i],",'",asy$SAMP
LE[i],"');")
 dbGetQuery(con,sql)
}
sql<-"UPDATE collar SET geom=ST_SetSRID(ST_MakePoint(xcollar,ycollar,zcollar),
32724);"
dbGetQuery(con,sql)
```

The drillhole data and assay data are now properly loaded into the database*. we will create a block model for the element el1 in Part 4 but in this part we will cover several options to desurvey and visualize the drillhole data. **In this tutorial it is assumed that the ore has a constant density for the sake of simplicity but if the sample density is known you can add a column density to your assay table creation step.*

```
CREATE TABLE assay(bhid varchar(50) NOT NULL, _from numeric(8,4) NOT NULL, _to
numeric(8,4) NOT NULL,el1     numeric(9,4),density     numeric(9,4),     sample
varchar(50) NOT NULL, UNIQUE(sample));
```

*and add the density value column for each sample into the **assay.csv** file.*

Drillhole desurveying and visualization

We will use pyGSLIB, R and Julia to create a drillhole file for visualization based on one element assay result by generating a file with the drillhole data which can be visualized using Paraview.

Option 1 - Using pyGSLIB

PyGSLIB is an open source python package designed to do Mineral Resource Estimations with scripts. It was inspired by Datamine Studio macros. Its philosophy is reproducibility and auditability. With PyGSLIB you can write a script to do the entire resource estimate process, from reading drillhole tables to estimation and validation. You can rerun the entire script with different parameters until you get the desired output, share it with your colleagues or send it to a peer reviewer for auditing. PyGSLIB may be also useful to automatize an estimation process such as grade control model updates, or to do loops required by some nonconventional applications such as drillhole spacing studies with conditional simulations. (more about pyGSLIB can be found at: https://opengeostat.github.io/pygslib/ .

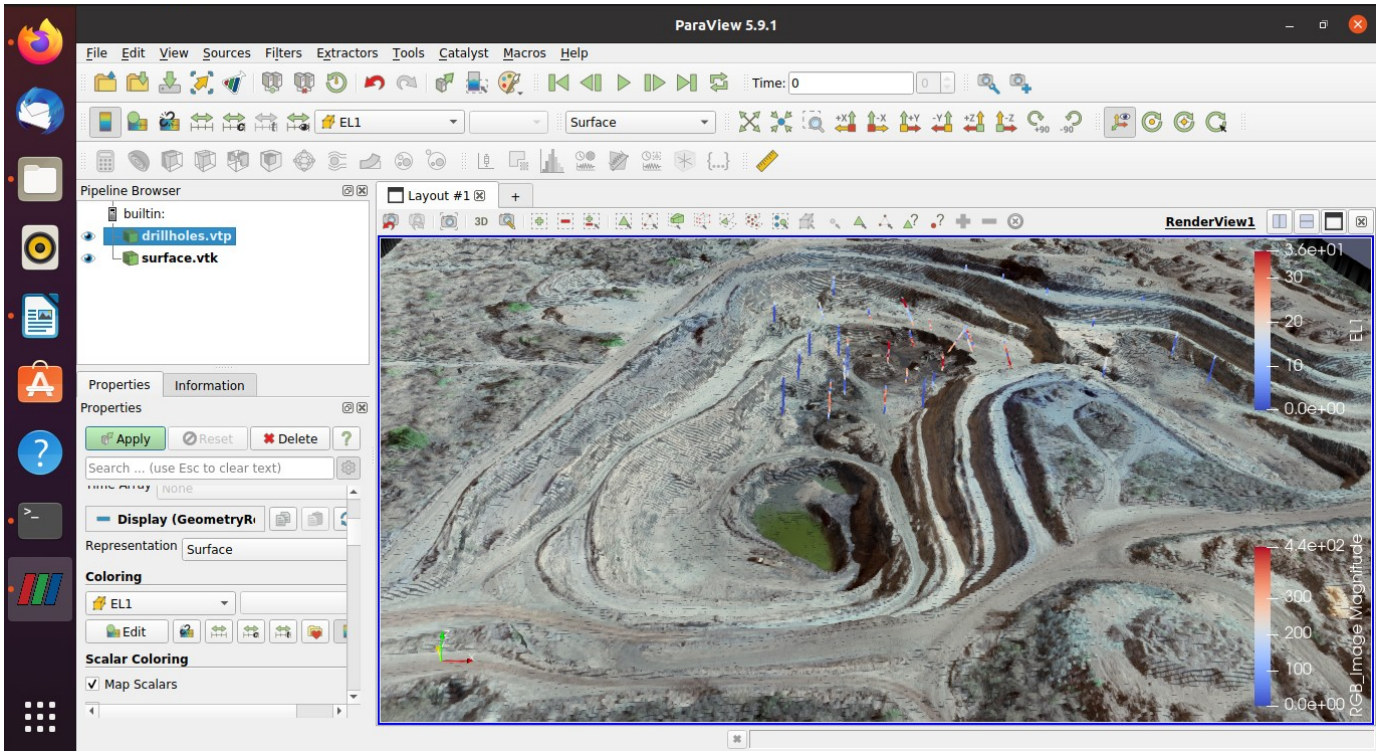
Load conda environment for python 3.6 already created to run pyGSLIB under python 3.6 I named mine env36 which may differ from the one you created):

```
conda activate env36
python
```

Now run the script below:

```
#loading the libraries needed
import pandas as pd
import matplotlib.pylab as plt
import numpy as np
import psycopg2 as pg
import pygslib
# connecting with the geofoss databese and retrieving
# the collar,survey and assay data
connection=pg.connect("host=127.0.0.1 dbname=geofoss user=username password=**")
collar = pd.read_sql('SELECT bhid as "BHID", xcollar as "XCOLLAR", ycollar as
"YCOLLAR", zcollar as "ZCOLLAR" FROM collar', connection)
survey = pd.read_sql('SELECT bhid as "BHID", _at as "AT", az as "AZ", dip as
"DIP" FROM survey', connection)
assay = pd.read_sql('SELECT bhid as "BHID", _from as "FROM", _to as "TO", _to-
_from  as "Length", el1 as "EL1" FROM assay ', connection)
# Creating our object drill hole integrated with
# the survey info and adding the assay result table to it
dh=pygslib.drillhole.Drillhole(collar=collar, survey=survey)
dh.addtable(assay, 'assay', overwrite = False)
# Desurveying each individual composited sample
dh.desurvey('assay',warns=False, endpoints=True)
# GSLIB (fortran) uses integer values for index, converting holeid to integer
dh.txt2intID('assay')
# writing our drillholes object as vtk for visualization with paraview
dh.intervals2vtk('assay', 'drillholes')
```

A file **drillholes.vtp** was created and represents the drillholes associated with the El1 assay results on it. Start Paraview and load the **drillholes.vtp** file we just created and the **surface.vtk** file created previously on Part1. Select the field EL1 in the coloring and the result will look similar to the image below:



Lets check the drill hole object structure. The desurveyed assay table inside our drill hole object is composed by 21 columns that will be used as basis to create our Block Model.

The columns are:

```
BHID          → Drill hole name
FROM          → Depth From
TO            → Depth To
length        → The interval length
EL1           → Sample grade
azmm--------    intervals contributing to the composite interval)
dipm         |
xm           |
ym           |
zm           |
azmb         |
dipb         |
xb           | → Desurveyed composited sample AZ,DIP,X,Y,Z at middle, begin and
yb           |   end of the sample
zb           |
azme         |
dipe         |
xe           |
ye           |
ze ---------
BHIDint       → Integer drillhole id
```

Option 2 - Using Julia

DrillHoles.jl is a package written in Julia for drill holes desurvey and compositing.

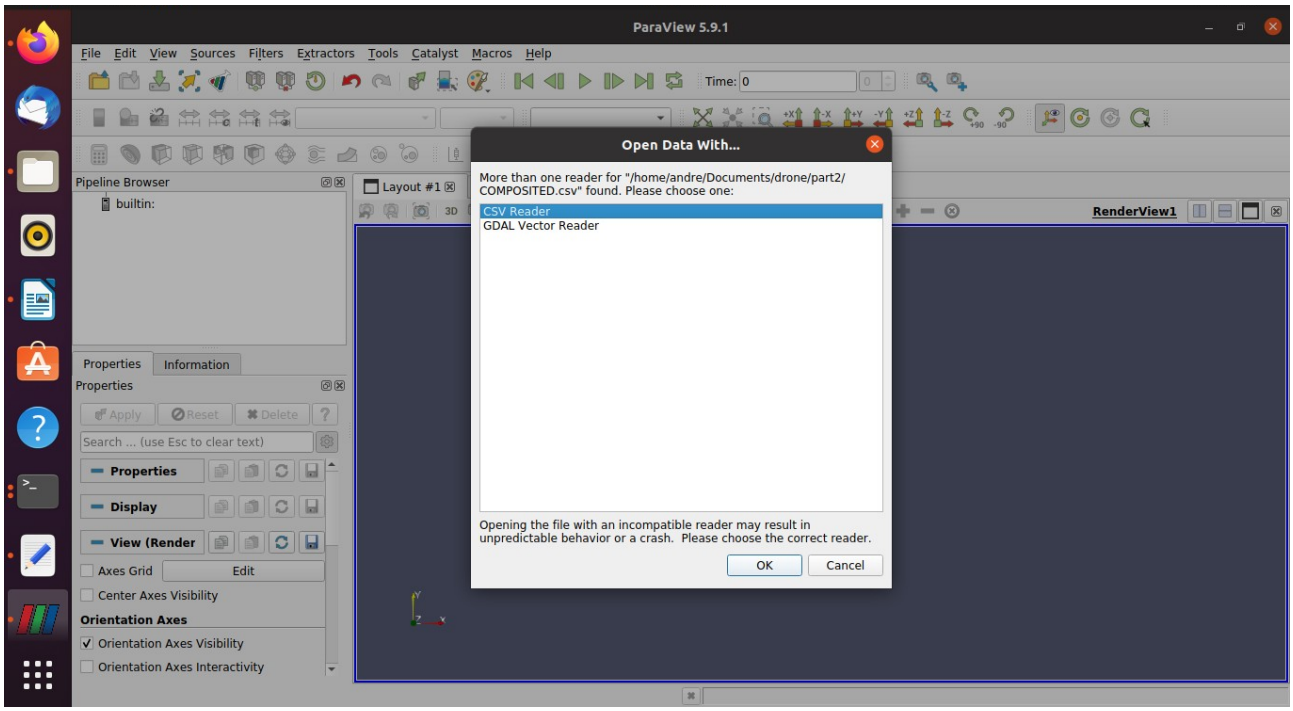
Use the code below to generate a desurveyed object (or file).

```
using LibPQ, DataFrames, DrillHoles, Plots,CSV

conn=LibPQ.Connection("host=127.0.0.1 dbname=geofoss user=username password=*")

collar=DataFrame(execute(conn, "select c.bhid as \"HOLEID\", c.xcollar as \"X\",
c.ycollar as \"Y\",c.zcollar as \"Z\",max(s._at) as \"ENDDEPTH\"from collar as
c,      survey      as      s      where      c.bhid=s.bhid      group      by
c.bhid,c.xcollar,c.ycollar,c.zcollar"))
collar.HOLEID=convert(Vector{String}, collar.HOLEID)
collar.ENDDEPTH=convert(Vector{Float64}, collar.ENDDEPTH)
collar.X=convert(Vector{Float64}, collar.X)
collar.Y=convert(Vector{Float64}, collar.Y)
collar.Z=convert(Vector{Float64}, collar.Z)
collar
# ------use *-1 if dip angle is positive ------
survey=DataFrame(execute(conn,  "SELECT  bhid  as  \"HOLEID\",_at  as  \"AT\",az
as \"AZM\",dip*-1 as \"DIP\" FROM survey"))
survey.HOLEID=convert(Vector{String}, survey.HOLEID)
survey.AT=convert(Vector{Float64}, survey.AT)
survey.AZM=convert(Vector{Float64}, survey.AZM)
survey.DIP=convert(Vector{Float64}, survey.DIP)
survey

assay=DataFrame(execute(conn, "SELECT bhid as \"HOLEID\",_from as \"FROM\", _to
as \"TO\",el1 as \"EL1\" FROM assay WHERE el1 IS NOT NULL"))
assay.HOLEID=convert(Vector{String}, assay.HOLEID)
assay.FROM=convert(Vector{Float64}, assay.FROM)
assay.TO=convert(Vector{Float64}, assay.TO)
assay.EL1=convert(Vector{Float64}, assay.EL1)
assay

collarIn = Collar(file=collar, holeid=:HOLEID, x=:X, y=:Y, z=:Z)
surveyIn = Survey(file=survey, holeid=:HOLEID, at=:AT, azm=:AZM, dip=:DIP)
assayIn  = Interval(file=assay, holeid=:HOLEID, from=:FROM, to=:TO)

# Desurvey the drillhole data objects
dh = drillhole(collarIn, surveyIn, assayIn)
CSV.write("DESURVEYED.csv",dh.table)
```

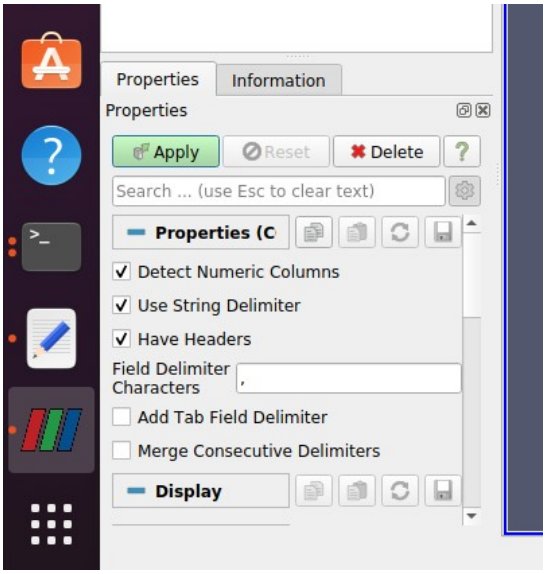
The structure of the desurveyed object (or file) is:

```
HOLEID      drillhole ID
FROM        top of   interval
TO          base of interval
LENGTH      Interval length
EL1         Element grade value or interval value
X           interval X
Y           interval Y
Z           Interval Z
```

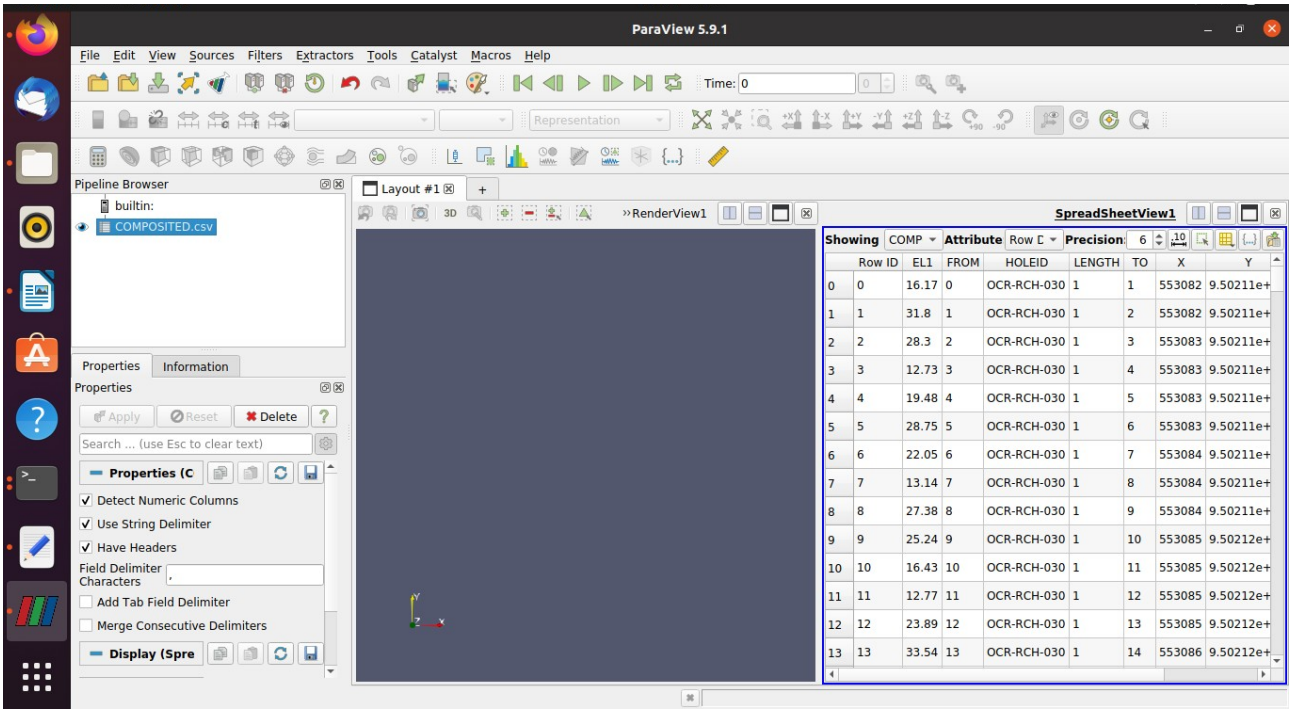
It is possible to visualize the data above using paraview following the steps below:
Open the DESURVEYED.CSV with the normal CSV filter.



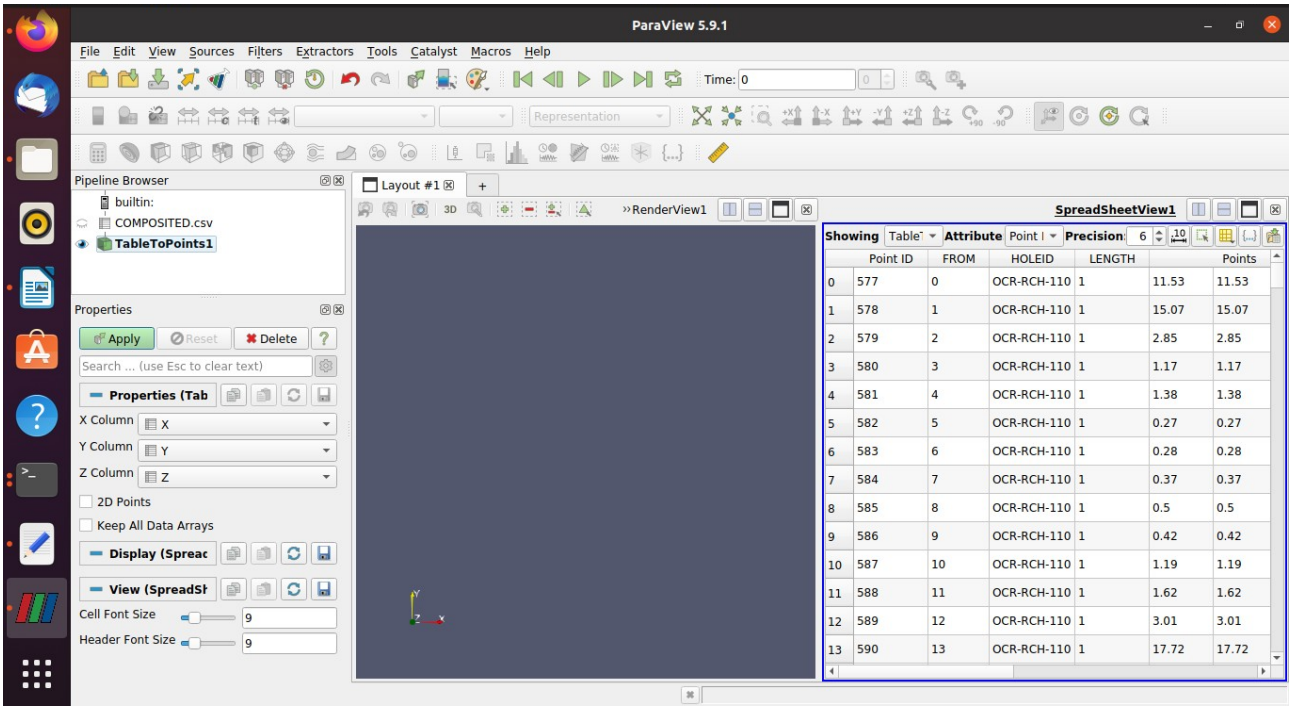
Click Apply in properties



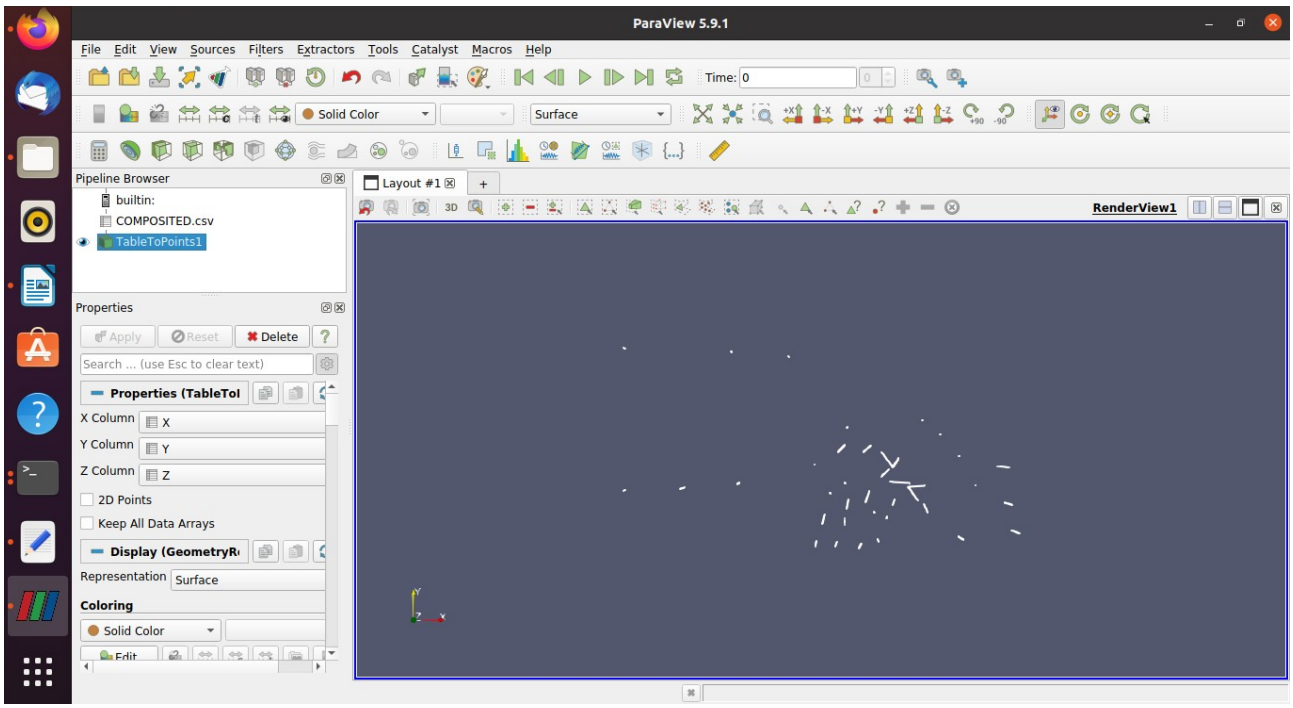
and the screen should look like:



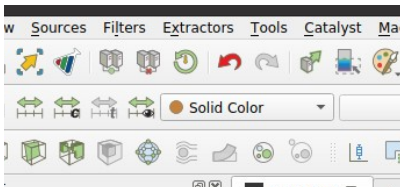
Right click the object and select **Add Filter → Alphabetical → Table to Points** and select the X, Y and Z and Apply:



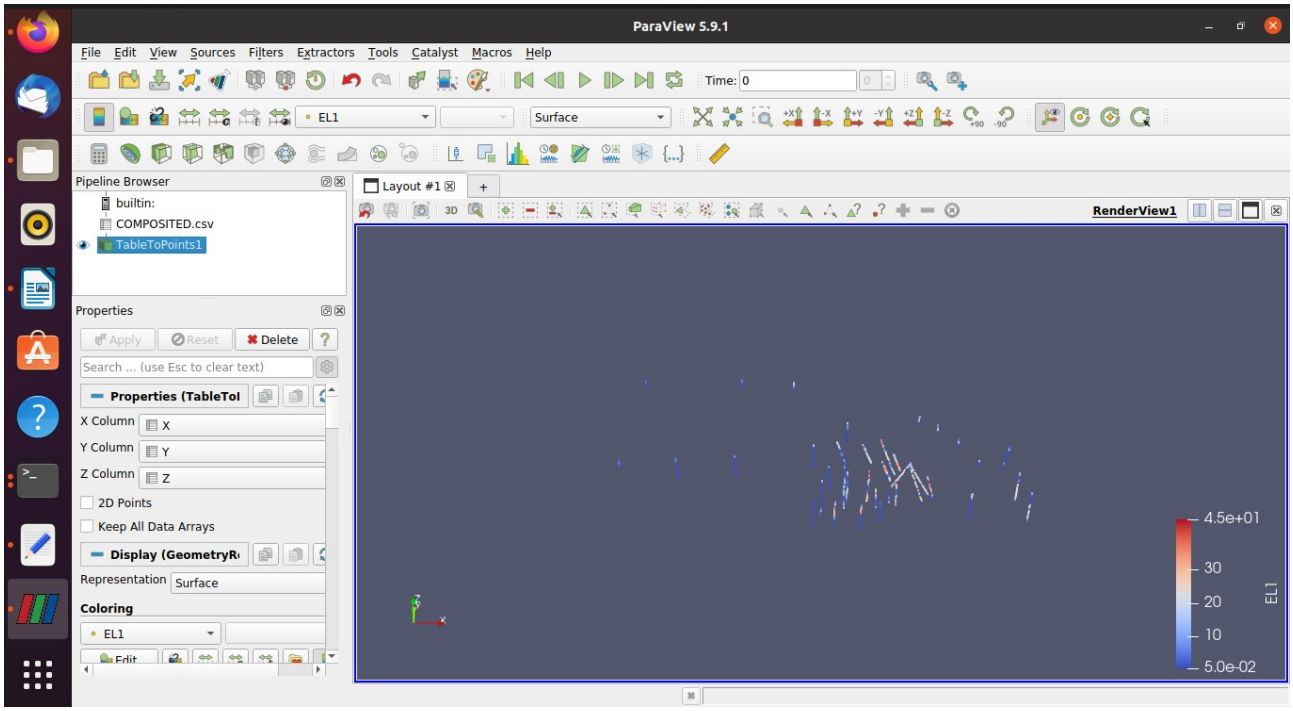
Resulting in:



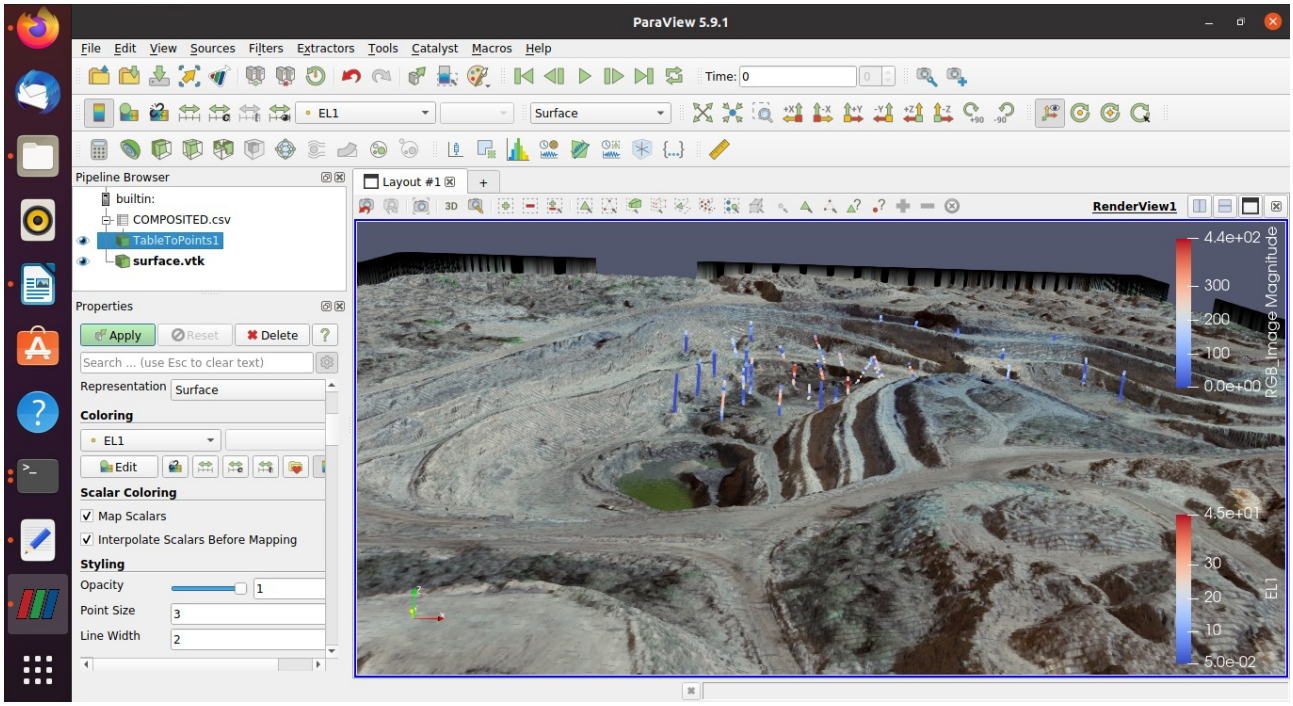
Change to EL1 in



Resulting in:



Add the VTK surface created in Part 1 and the result will be:



Option 3 - Using R

Below it is a “raw” R script to desurvey the drillhole data.

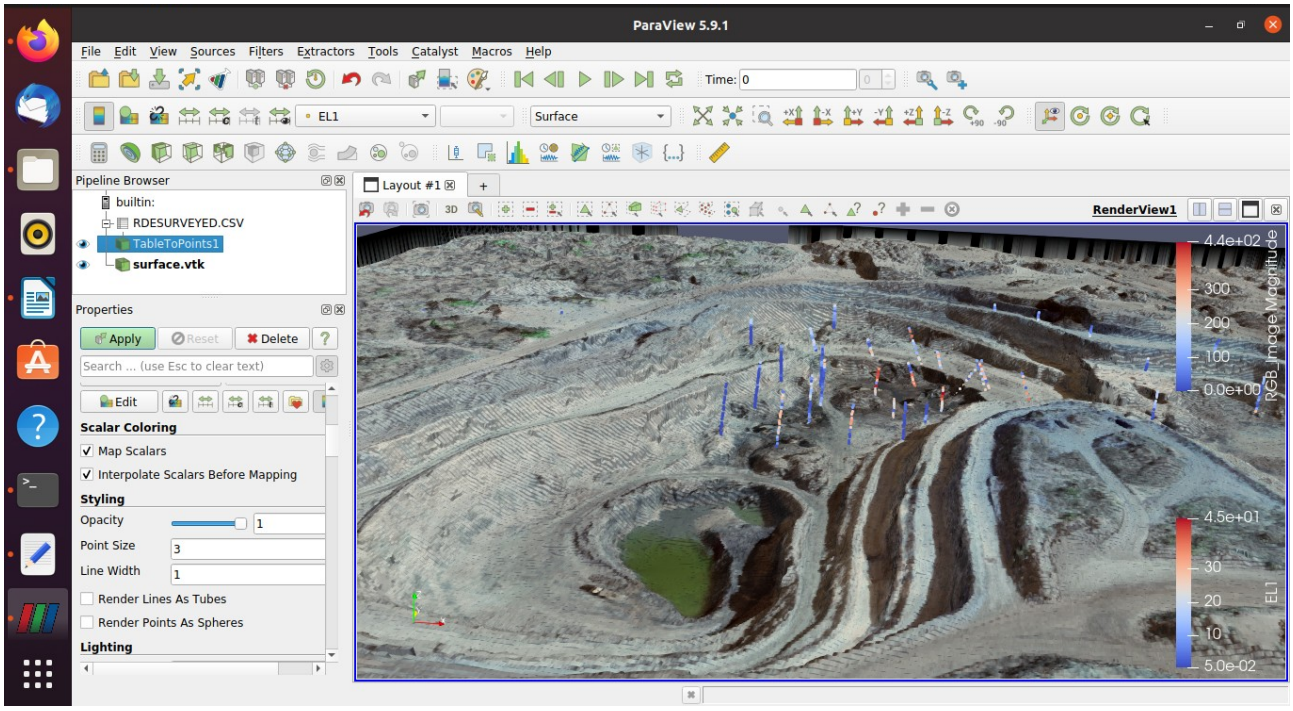
```
library(RPostgreSQL)
con<-dbConnect(PostgreSQL(),host='127.0.0.1',user='username',password='*',
dbname='geofoss')

dados<-dbGetQuery(con, "select c.bhid as furo, c.xcollar as X, c.ycollar as
Y,c.zcollar as Z, a.el1 as EL1, a._from as prof, a._to as to, (a._to-a._from) as
len from collar as c, assay as a where c.bhid = a.bhid and a.el1>0 order by
furo,prof")

colar<-dbGetQuery(con, "select c.bhid as furo,s._at as prof, c.xcollar as x,
c.ycollar as y, c.zcollar as z, s.dip as mergulho, s.az as az from collar as c,
survey as s where c.bhid = s.bhid and s._at=0")

# if dip is positive uncoment this line------------------------------
#colar$mergulho<-colar$mergulho*-1
#-------------------------------------------------------------------

estacoes<-dbGetQuery(con, "SELECT bhid as furo, _at as prof,null AS x, null AS
y,null as z, dip as mergulho, az FROM survey")

# if dip is positive uncoment this line------------------------------
estacoes$mergulho<-estacoes$mergulho*-1
#-------------------------------------------------------------------

tresD<-function(da,es){
      linha<-data.frame()
      for(i in 1:nrow(da)){
            x<-da[i,3]
            y<-da[i,4]
            z<-da[i,5]
            di<-0
            fur<-es[es[[1]]==da$furo[[i]],]
            fur<-fur[order(fur$prof),]
            for(j in 1:nrow(fur)){
                  angulo<-fur[j,7]
                  d<-fur[j,2]-di
                  di<-d+di
                  deltaz<-abs(d*sin(fur[j,6]*pi/180))
                  raio<-d*cos(fur[j,6]*pi/180)
                  x<-round(x+raio*sin(angulo*pi/180))
                  y<-round(y+raio*cos(angulo*pi/180))
                  z<-round(z-deltaz)
                  linha<-
rbind(linha,data.frame(furo=fur[j,1] ,prof=fur[j,2],x=x ,y=y ,z=z,mergulho=fur[j
,6],az=fur[j,7]))
                  z<-z
            }
      }
      lin<-rbind(linha,da)
      lin<-lin[order(lin$furo,lin$prof),]
      return(lin)
}

pts.XYZ<-tresD(colar,estacoes)
obterXYZ<-function(pontos,furo,profu){
      sete<-pontos[as.character(pontos$furo)==furo,]
      refa<-sete[sete$prof==max(sete[sete$prof<=profu,]$prof),]
      ref<-refa[1,]
      d<-profu-ref$prof
      deltaz<-abs(d*sin(ref$mergulho*pi/180))
      raio<-d*cos(ref$mergulho*pi/180)
      z<-ref$z-deltaz
      angulo<-ref$az
      x<-ref$x+raio*sin(angulo*pi/180)
      y<-ref$y+raio*cos(angulo*pi/180)
      return(c(x,y,z))
}
```

```
for(i in 1 : dim(dados)[1]){
      dado<-obterXYZ(pts.XYZ,dados$furo[i],dados$prof[i])
      dados$x[i]<-dado[1]
      dados$y[i]<-dado[2]
      dados$z[i]<-dado[3]
}
colnames(dados)<-c("BHID","X","Y","Z","EL1","FROM","TO","LENGTH")
write.csv(dados,"RDESURVEYED.CSV",row.names = FALSE)
```

The structure of the desurveyed file created (RDESURVEYED.CSV) is:

```
BHID
X
Y
Z
EL1
FROM
TO
LENGTH
```

Use Paraview to visualize the data in the same way as described in Julia section



So far we got the surface surveyed by Drone generating the orthophoto and DTM (Part 1) and also we generated the drillhole and their associated assay result in 3D space using three different methodologies (Part 2).

On Part 3 we will cover the data preparation and parameters definition to build a block model using GSLIB (original fortran program), pyGSLIB (python interface to GSLIB), R and GeoStats (Julia).

On Part 4 the block model will be created.

References

Caixeta, M, R. (2021) - DrillHoles.jl - A package written in Julia for drill holes desurvey and compositing. Retrieved from https://github.com/JuliaEarth/DrillHoles.jl

Deutsch, C.V. and A.G. Journel. (1998). GSLIB: Geostatistical Software Library: and User's Guide, 2nd Ed. New York: Oxford University Press.

Hoffimann, (2018). GeoStats.jl -- High-performance geostatistics in Julia. Journal of Open Source Software, 3(24), 692, https://doi.org/10.21105/joss.00692

Vargas, A. M. (2016). PyGSLIB documentation. Retrieved from https://opengeostat.github.io/pygslib/index.html