French National School of
Geographic Sciences

Research Centre of the
Slovenian Academy
of Sciences and Arts

Summer internship

May-August 2019

Viewshed process documentation

Lucas Bres

December 2019

Ecole Nationale des Sciences Géographiques
6-8 Avenue Blaise Pascal - Cité Descartes - 77420 Champs-sur-Marne
Téléphone 01 64 15 31 00 Télécopie 01 64 15 31 07

Abstract

The Research Centre of the Slovenian Academy of Sciences and Arts (ZRC-SAZU) carried out in 2016 a LIDAR (LIght Detection And Ranging) campaign above the ancient city Maya Chactùn, located on the Yucatán peninsula in Mexico. This area is now covered by a tropical forest, and it is difficult to distinguish the architectural elements that made up the ancient Mayan city. A very precise DEM (Digital Elevation Model) has been created from these LIDAR data and has the advantage of not taking into consideration the vegetation. The training consists of analyzing this DEM to extract information that would further understand the spatial organization of Mayan cities and cultures.
The main objective of the project was to simulate the fields of view that had the Mayas from the main cities of the study area by calculating viewsheds from the top(s) of the ancient monuments.

Key words : Maya, viewshed, DEM, LIDAR, automation, spatial analysis

Contents

List of Figures

Glossary and used acronyms

Organism acronym :

ENSG *translated in English by :* National School of Geographic Sciences

ZRC-SAZU *translated in English by :* Research Centre of the Slovenian Academy of Sciences and Arts

Computer vocabulary :

GIS Geographic Information System: Software for visualization and processing of geographical data

QGIS Open-source GIS with a complete interface

GRASS GIS Open-source GIS with many geographical processes

Places of interest :

Yucatán Peninsula Peninsula located in southeastern `Mexico`, , on the border with `Guatemala` and separating the `Caribbean Sea` from the `Gulf of Mexico`.

Chactùn Ancient Mayan city located in the state of `Campeche`, near the modern city of `Xpujil`.

Geographical vocabulary :

Viewshed The area visible from an observation point on the Earth's surface.

Introduction

This paper explains the methodology of the viewsheds creating process for the 270 square kilometres around Chactùn, one of the ancient Maya cities in Yucatán peninsula. **The computation is dependant on QGIS** (version 3.4.9 or higher long term release) **and GRASS GIS** (version 7.6.1 or higher) both downloadable at this link. The input data used in this calculation are the **0.5 m DEM** (Digital Elevation Model) of the region in raster format, **buildings polygon layer** in vector format and **buildings point layer** in vector format as well.

These buildings layers describe the same area (the Southern part of the region around Chactùn) but in different ways. The polygon layer represents **the shape of 9303 buildings**, while the point layer describes **8794 points positioned roughly in the middle of buildings**. Moreover, these two layers complete each other. For example, 976 building contours haven't any centre points, and on the contrary, 177 centre points aren't in a building outline. The most interesting case is when a construction shape has more than one centre point in it. It happens because the point layer describes several buildings gathered in one outline. There are 218 shapes which include 2.3 buildings on average.

The objective of the process is **to simulate the views from the cities** in the region and to study settlement patterns focusing on the placement of the cities within the area.

Chapter 1 Determination of observation points

The goal of the internship consisted in calculating the fields of vision of the different buildings in a 270km² region around `Chactùn`, one of the ancient Mayan cities of the Yucatan peninsula. The GRASS GIS software (version 7,6,1 or higher) is required to calculate viewshed, the data used for this process are the DTM (Digital Terrain Model) with a planimetric resolution of 50cm in raster format, as well as the buildings in the area indicated as polygons and points.

These two last vector layers describe the same area (southern part of the region around `Chactùn`) but in different ways. The polygon layer represents the contour of the 9303 buildings, while the point layer describes 8794 points positioned approximately in the middle of the buildings. In addition, these two layers complement each other. For example, 976 building contours do not contain any points. On the contrary, 177 points indicate the constructions without contours. The most interesting case is when the silhouette of a construction has more than one point inside. This happens because the layer of points describes several frames gathered in a single contour. There are 218 contours in this case which include on average 2.3 constructions.

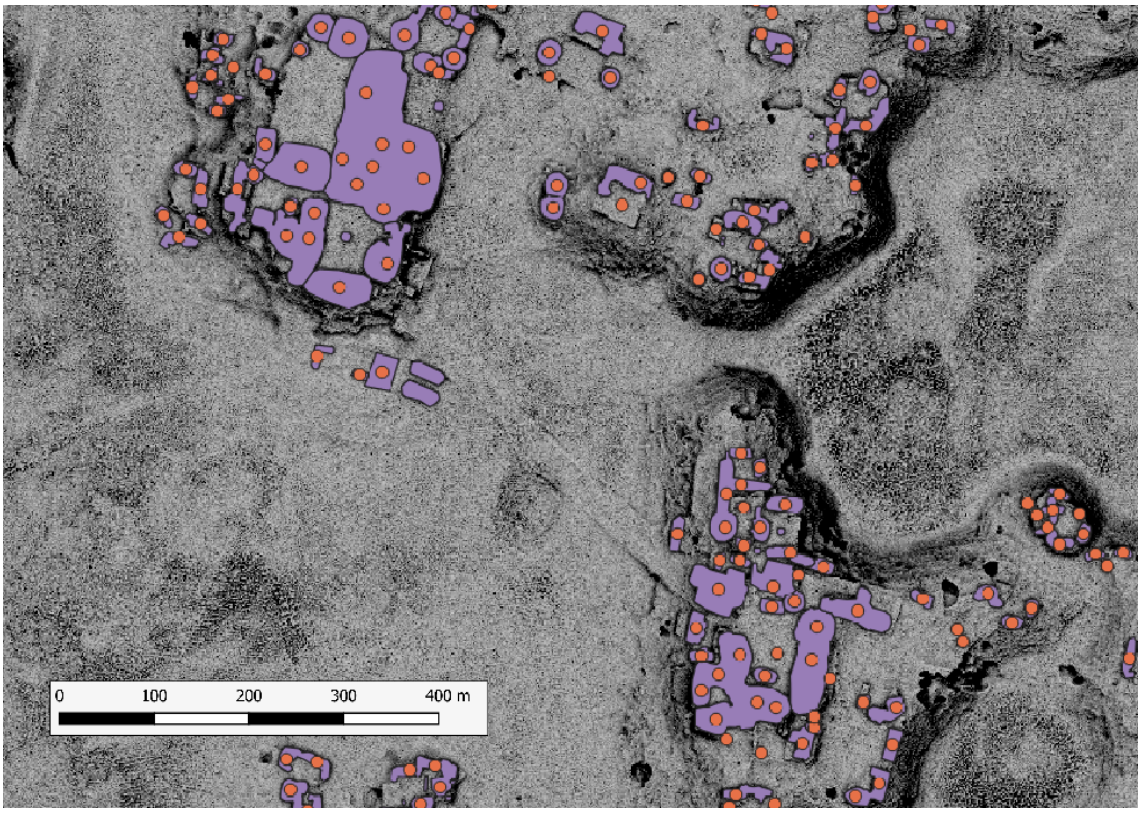

Figure 1.1 – Example of the contour and centre layers of buildings

The goal of this process is to simulate the fields of view of the Mayan cities in the region and to study the reasons for the implementation of the cities within the study area.

Viewsheds are created in two steps: Define the observation points, then calculate their field of view.

1.1 Defining the observer points

Before calculating the viewsheds of the region, we have to define the observer points. A viewpoint is a point **with planimetric coordinates** (X, Y) and **located inside the terrain**. It is the origin of a viewshed. The observer points must be in a vector layer in QGIS.

We are considering that the best observer point is the one which produces **the viewshed with maximum possible length**. For the purpose of our analysis, we are going to create viewsheds from the summits of all buildings. The summit of building as the observer point might not be optimal for various reasons:

— If the peak point is far from the edge, the building itself could obstruct the field of view.

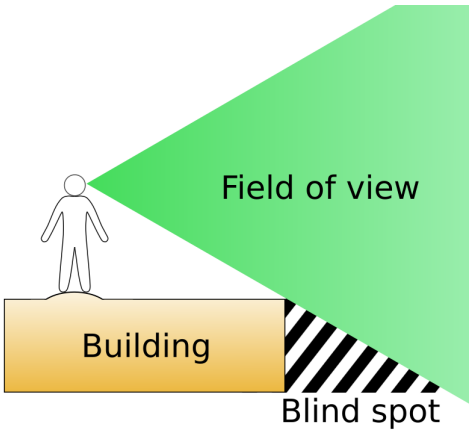

Figure 1.2 – Schema of the visibility of a man far from the edge

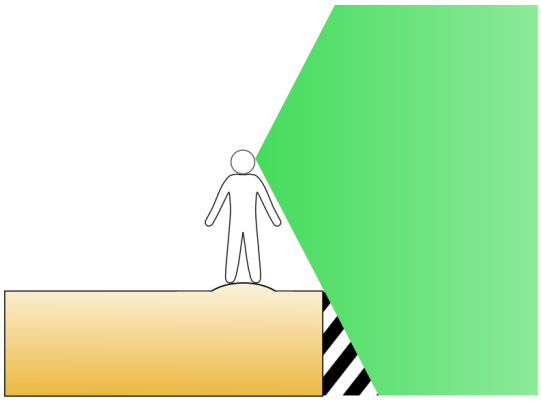

Figure 1.3 – Schema of the visibility of a man near the edge

— Buildings may also have several points with the maximum height.
In this case, **we take into consideration all of these summits**, although it may give more importance for this building than another one with a unique viewpoint. Taking into attention several points for one building is more realistic than one unique observer point because Mayan people could move and observe from different places of a house.

Determination of summits can be automated by locating the highest pixel value within a polygon or around a point. The automated process of selecting observer points does not necessary give the optimal results. The algorithm might create observer points that don't produce the optimal viewsheds. It might also create wrong observer points located outside an actual building.

In our example, it is possible to determine the summit of a building according to one of the two types of geometry - polygon or point- the building layer may have.

1.2 Summit calculation for polygons

The polygon layer represents the shape of each building in the region. Consequently, the summit of an edifice is inside the polygon. Fundamentally in the case using a polygon layer, the purpose of the summit calculation is to **find the peak point in each polygon**.

There are two main ways to retrieve the height of the pixel which correspond to the maximum height of a DEM within each polygon.

The first method is to create as many points as pixels included in the polygons, then compare these points to find the vertex(s) of each polygon.
The second method is to clip the DTM according to the footprint of each polygon and then locate the pixel(s) of higher values.

The first method was developed as a Python script running in QGIS only. However, the excessively long computation time (45 hours) and the creation of many temporary files for intermediate computation results led me to develop a second script. The latter implements the second method explained above. This Python script uses the rasterio, fiona and shapely libraries and works without the help of QGIS.

First, the algorithm reduces the DTM according to the footprint of each polygon in the form of a matrix. Only pixels whose center intersects the polygon are taken into account. However, if no pixel center intersects the polygon, the algorithm reduces the DTM where all pixels intersecting the polygon are taken into account. After this preparation step, the algorithm has a matrix containing the polygon altitude values. The script then locates the highest value point(s) and adds them to a new Point geometry layer.

This algorithm is considerably faster than the first method developed. Indeed, the script takes a few minutes to calculate all the vertices of the buildings.

1.3 Summit calculation for points

In addition to the layer of building contours, I had at my disposal a layer informing the centre of the buildings by a point. This layer has additional information compared to the silhouette layer of buildings, however it is more complex to find the top of a construction defined by a point than by its contour. Indeed, the method to find the building summit(s) from a point consists in creating a buffer around it, then using the previous algorithm to locate the apex(es) within the polygon of the buffer. Some buffers can nevertheless intersect. In this case, the top of one of the buffers may not match the building to which the buffer belongs.

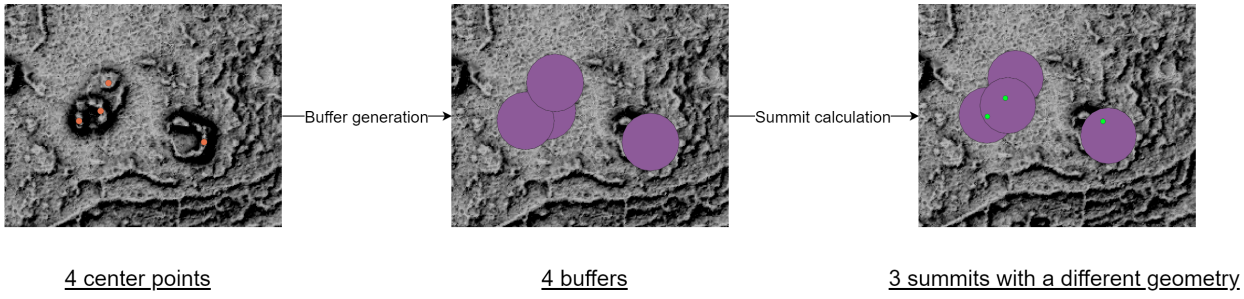

Figure 1.4 – Example of an intersection conflict between buffers

The algorithm solves this situation by recalculating the summits around too close points by reducing the size of the buffers.

The centre layer of buildings has additional information compared to the contour layer:
— 177 points are located outside the contours of the buildings.
— 218 building contours contain several points.

The location of the summits around the centres of the buildings was therefore only done for these almost 900 points, in order not to be redundant with the calculation of the apexes from the silhouettes of the buildings.

This algorithm requires a radius length as an input parameter in addition to the DTM and building centres. Choosing an adequate distance is essential because if the buffer radius is too small, the algorithm will not be able to detect the real top of the building but only the highest point at a certain distance from the center of the building. The resulting field of view would then present large areas of shadow.
On the other hand, if the buffer radius is too large, the summits may be located outside the buildings. In addition, the larger the radius, the more buffers will intersect and therefore the script will take longer to resolve these intersection conflicts.

So we have analyzed the size of the buildings to determine a relevant distance around the centres of the buildings. The procedure is explained in the following chapter.

1.4 Viewsheds use case

We will explain in this paragraph how we achieve to produce viewsheds. Note that we have only submitted algorithms to samples for now, so estimated process time may vary. The processing time mostly depends on the computer processor. We have used a *7th generation I5 processor* with 4 cores to estimate processing times. We had to our disposition three layers:
— the 0.5m resolution DEM: *Chactun_dem_05m.tif*
— the building shapes: *stavbe.shp*
— buildings described by a point: *corrIS_stavbe_center_points.shp*

First, we reprojected in QGIS the DEM from WGS84 (ESPG:4326) to WGS84 / UTM zone 16N (ESPG:32616) to avoid projection issues subsequently. We saved the DEM in a different file to set by default the new coordinate system.
Then we added an auto-incremented ID field to the two vector layers thanks to the tool *Add auto-incremental field* in QGIS.
Next, **we ran the summit calculation from polygon** with the new DEM and the building shapes layer as input data. It should take less than 3 hours to find the summits of the 9303 building shapes.

The Python program might have issues with very large DTM like the one of the Institute. In this case, the user can define a new working environment in order to run the script. We use the anaconda library to set up the python environment and then run the summit calculation from polygons as follows:

```
(base) C:\Users\lucas>conda create -n testgdal -c conda-forge gdal vs2015_runtime=14
(base) C:\Users\lucas>conda activate testgdal

(testgdal) C:\Users\lucas\Documents\Ljubljana>conda install rasterio numpy shapely fiona

(testgdal) C:\Users\lucas\Documents\Ljubljana>python
Python 3.7.3 | packaged by conda-forge | (default, Jul  1 2019, 22:01:29) [MSC v.1900 64 bit (AMD64)] ::
 Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from Summit_calculation_from_polygon import main
>>> main(building_path = 'QGIS_Project/Data_shp_Lucas/stavbe.shp', name_ID_field = 'OBJECTID', raster_pa
th = 'QGIS_Project/Data_shp_Lucas/DEM_BIG_UTM16.tif', output_path = "OUTPUT_SUMMITS_LIDAR.shp")
>>> quit()
```

Figure 1.5 – Deploying a Python environment and running the apex computation script

The centre points layer can complete the process because it has **extra information**:

— There are points located outside the structure shapes.

— There are multiple points in a single building outline.

We have decided to find a peak only near these points. So first, we selected by location the centre points intersecting the building contours, and then we inverted the selection. There are 177 points outside any building outlines.

Secondly, we counted the number of point in each building shape thanks to the *Join attribute by location (summaries)* algorithm. There are 218 structures with 2 points or more in it.

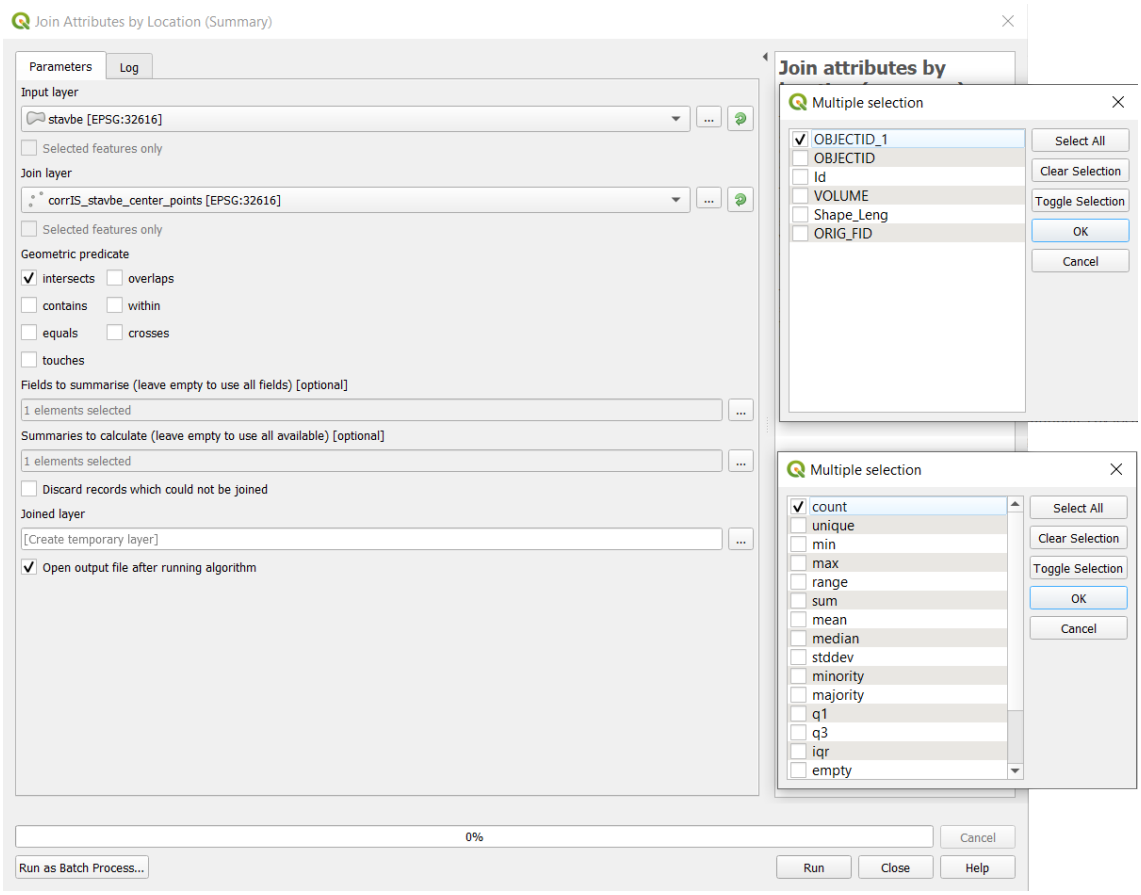

Figure 1.6 – Visual interface of the *Join attribute by location (summaries)* tool

To find the building summits thanks to these 395 points, the user must also give a distance as an input. First, we analysed the building shape layer to choose the right range because if the radius is too small, the algorithm might not find the true summit for each building. At the

contrary, if the length is too large, the algorithm will shrink the buffer around the centre point of the building to not intersect other buffer polygons. However, the more conflicts between buffer polygons there are, the more time the process takes. Moreover, if the buffer polygons are too wide, the algorithm might create point views outside of the actual building.

Consequently, we calculate for each structure the radius of the minimal circle, which entirely overlaps its shape. We used the field calculator to create a new field in the *stavbe.shp* layer with this expression:

sqrt(area(minimal_circle($geometry)) / pi())

Then we used the «Basic statistics for fields» to calculate its mean and median.

```
Processing algorithm...
Algorithm 'Basic statistics for fields' starting...
Input parameters:
{ 'FIELD_NAME' : 'radius', 'INPUT_LAYER' : 'C:/Users/lucas/Documents/Ljubljana/QGIS_Project/
Data_shp_Lucas/stavbe_id_autoincremental.shp', 'OUTPUT_HTML_FILE' : 'C:/Users/lucas/AppData/
Local/Temp/processing_c6ef613fba264dc59bf625016eae241d/0bf3159c47214a2eb4b75942f37241c8/
OUTPUT_HTML_FILE.html' }

Execution completed in 0.38 seconds
Results:
{'COUNT': 9303,
'CV': 0.6070624927000685,
'EMPTY': 0,
'FILLED': 9303,
'FIRSTQUARTILE': 6.612,
'IQR': 7.456999999999999,
'MAJORITY': 5.808,
'MAX': 93.389,
'MEAN': 11.435667311619936,
'MEDIAN': 9.87,
'MIN': 0.053,
'MINORITY': 0.053,
'OUTPUT_HTML_FILE': 'C:/Users/lucas/AppData/Local/Temp/
processing_c6ef613fba264dc59bf625016eae241d/0bf3159c47214a2eb4b75942f37241c8/
OUTPUT_HTML_FILE.html',
'RANGE': 93.336,
'STD_DEV': 6.94216470388069,
'SUM': 106386.01300000027,
'THIRDQUARTILE': 14.068999999999999,
'UNIQUE': 7287}

Loading resulting layers
Algorithm 'Basic statistics for fields' finished
HTML output has been generated by this algorithm.
Open the results dialog to check it.
```

Figure 1.7 – Statistic outputs of the building minimal circles

In the end, we have chosen the mean -**11.4 meters**- as a distance. We have calculated the nearest distance between the points outside buildings and the construction shapes layer to confirm our distance choice, and that there won't be any buffer polygon superposition with the building contours. We used the «Distance to nearest hub (points)» algorithm in QGIS to do so. The result shows that the minimal distance between a centre point and the nearest building contour is more than 13 meters. So there won't be any superposition between buffer polygons and structure outlines if the user uses 11.4 meters as a distance.
Therefore **we ran the summit calculation** with the DEM, the 395 points, and the range of 11.4 as inputs. It took 9 minutes and a half

We then merged the summits layers got from the two previous algorithms, deleted the points which have the same coordinates, an added an auto-incremented summit ID field.
All of what you have read so far is achievable in QGIS. After setting the points of view, we will calculate their viewsheds in GRASS GIS.

Chapter 2
Viewshed calculation

2.1 Setting up the GRASS GIS project

After calculating and merging the apexes obtained from the contours and centres of the buildings, we can calculate the fields of view of the 11000 observation points.
The user must use GRASS GIS, another software, to generate these viewsheds. It exists a GRASS plugin in QGIS, but we will use an addon command line which is currently not part of the standard distribution of GRASS GIS. This command line is available only in GRASS GIS, and **the user has to install it beforehand**.
After creating a new project and set the coordinate system (manually or thanks to a georeferenced file), the user has access to the GRASS GIS interface. He can now install *r.viewshed.cva* (not to be confused with *r.viewshed*) with the command line «g.extension - -ui». However, we have issues with this command because the download URL is incorrect. Here is how we manage to install it:

- First we downloaded the file *r.viewshed.cva.zip* at this URL.
- Then we copy-pasted the file r.viewshed.cva.zip > bin > r.viewshed.cva.bat in the same folder as the grass76.bat. You can locate this folder by right-clicking the GRASS GIS desktop icon and click on «Open file location».
- After that, we ran the command in GRASS GIS, like for example «r.viewshed.cva - -ui». The console displays an error message and says that it can't open the python file which should be located in a specific folder. The console shows the correct path.
- Finally, we copy-pasted the file r.viewshed.cva.zip > scripts > r.viewshed.cva.py in the right folder to match the previous path.

Next importing the DEM and the points of view, **the user has to set the region of study** thanks to the command line «g.region - -ui»[1]. This command sets the region resolution and extent and so the viewshed resolution and extent too.
The Institute's DTM has a high resolution of 50cm, but we have chosen to set the spatial resolution at 5m in order to accelerate the computation time. Indeed, a computer equipped with a 4 core processor takes about a minute to create one field of view with a resolution of 5m over a region 30km long by 9km wide. However, the human eye can only discern objects smaller than 5 metres from a distance of 17.1 km [2]. We therefore assume that the viewsheds are not sufficiently resolved over the first 17 kilometres around the observation points because a human eye could discern details of less than 5 metres in size.

Then, GRASS calculates viewsheds thanks to the command line «r.viewshed.cva - -ui»[3]. We will discuss the parameters in the next part.
The algorithm creates a bunch of viewsheds, but there are in a GRASS-specific format. So **we have to export them in an appropriate format** as GeoTIFF. Nonetheless, there is no way to export multiple layers in a single time. So we write a simple Python script to solve this minor issue. It just lists viewsheds thanks to their name beginning by «vshed» and then runs the export command line for each of them. The output folder is only settable in the Python code.

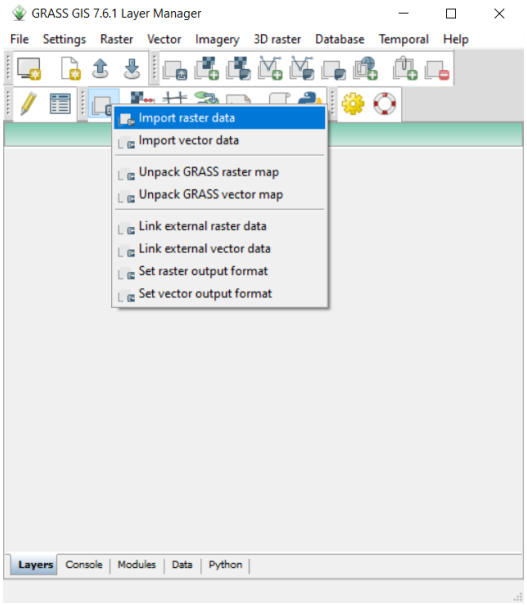

Figure 2.1 – GRASS GIS interface to import data

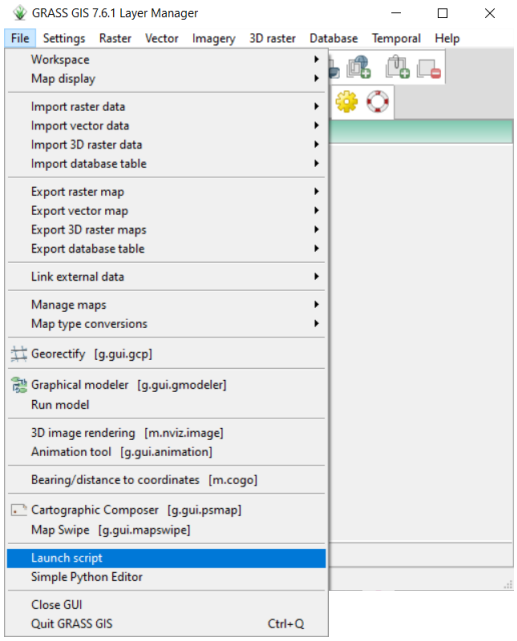

Figure 2.2 – GRASS GIS interface to run a Python script

2.2 Parameters for creating fields of view

The creation of viewsheds depends on many parameters. We will now list the ones we used, and explain why. The algorithm has two types of parameters called *Flags* and *Parameters*.

— Flags
  — - -verbose: the algorithm prints messages. This flag makes the user aware of the process progress.
  — -k: it saves calculated viewshed for each point
  — -b: the output format of the raster viewshed is 0: invisible by the observer point and 1: visible
  — -c: the algorithm takes into consideration the earth curvature. This parameter is very significant for visibility analysis in big distances because a 1.7-meter tall man can geometrically see the skyline around 5 km away. Our DEM is 30km long and 9km wide, so it is imperative to take earth curvature into account. The algorithm adjusts the altitude of a target point according to this expression [**?**]:

$$Z_{actual} = Z_{surface} - \frac{Dist^2}{2 \times a}$$

Where a is the semi-major axis of the ellipsoid used by the current projection.
  — -r: the algorithm takes into consideration the atmospheric refraction. The more a light ray travels through the atmosphere, the more its direction deviates because of atmospheric refraction. The algorithm adjusts the altitude of a target point by completing the previous expression [4]:

$$Z_{actual} = Z_{surface} - \frac{Dist^2}{Diam_{earth}} + R_{refr}\frac{Dist^2}{Diam_{earth}}$$

Where R_{refr} is the coefficient of refraction given by the user.

— Parameters:
  — input: The user indicates the DEM layer
  — vector: The user indicates the observer points layer
  — output: The user sets the name of the output cumulative viewshed layer
  — observer_elevation: 2 meters. We have chosen this height because it is a bit taller than the actual height of Maya people (around 1.5m for the descendants of the Mayans[5]). This increased height should compensate for the natural erosion of the buildings, the fact that the observer point isn't on the edge of the building, and the fact that Mayans could use some items to elevate themselves.
  — target_elevation: 0 meter. The objective of this visibility analysis is to know how far Maya cities could see. So if an observer can see the ground, it will also see a human being standing up.
  — max_distance: -1. This value represents an infinite distance. We chose an infinite range because of the purpose of this visibility study, which is to know how far Mayans could see.
  — memory: We have allocated 3000 MB of memory for the process. You should note that underestimating or overestimating the amount of memory would impact the processing time [6].
  — name_column: The user must indicate the ID field name of the points of view to link viewsheds and observer points. The output viewsheds will have a name like this: «vshed_IDValue_Auto-incrementedID».
  — refraction_coeff: The coefficient of refraction is directly related to the local vertical temperature gradient and the atmospheric temperature and pressure according to this expression [7]:

$$k = 503\frac{P}{T^2}\left(0.0343 + \frac{dT}{dh}\right)$$

where temperature T is given in kelvins, pressure P in millibars, and height h in meters. We have taken the average pressure and temperature between 2009 and 2019 of Xpujil [8], the nearest big city of the study area as the pressure and the temperature of the area. We have taken the tropospheric temperature gradient [9] (from sea level to 10km high) as the actual temperature gradient. Accordingly, the coefficient of refraction is equal to:

$$503 \times \frac{1013.3151}{(273.15 + 25.9206)^2} \times (0.0343 + 0.00649) = 0.2324$$

The viewshed calculation takes around **1 minute and a half** per point.

Finally, after the viewshed computation, we can export the viewsheds thanks to a python script to export all at the same time.

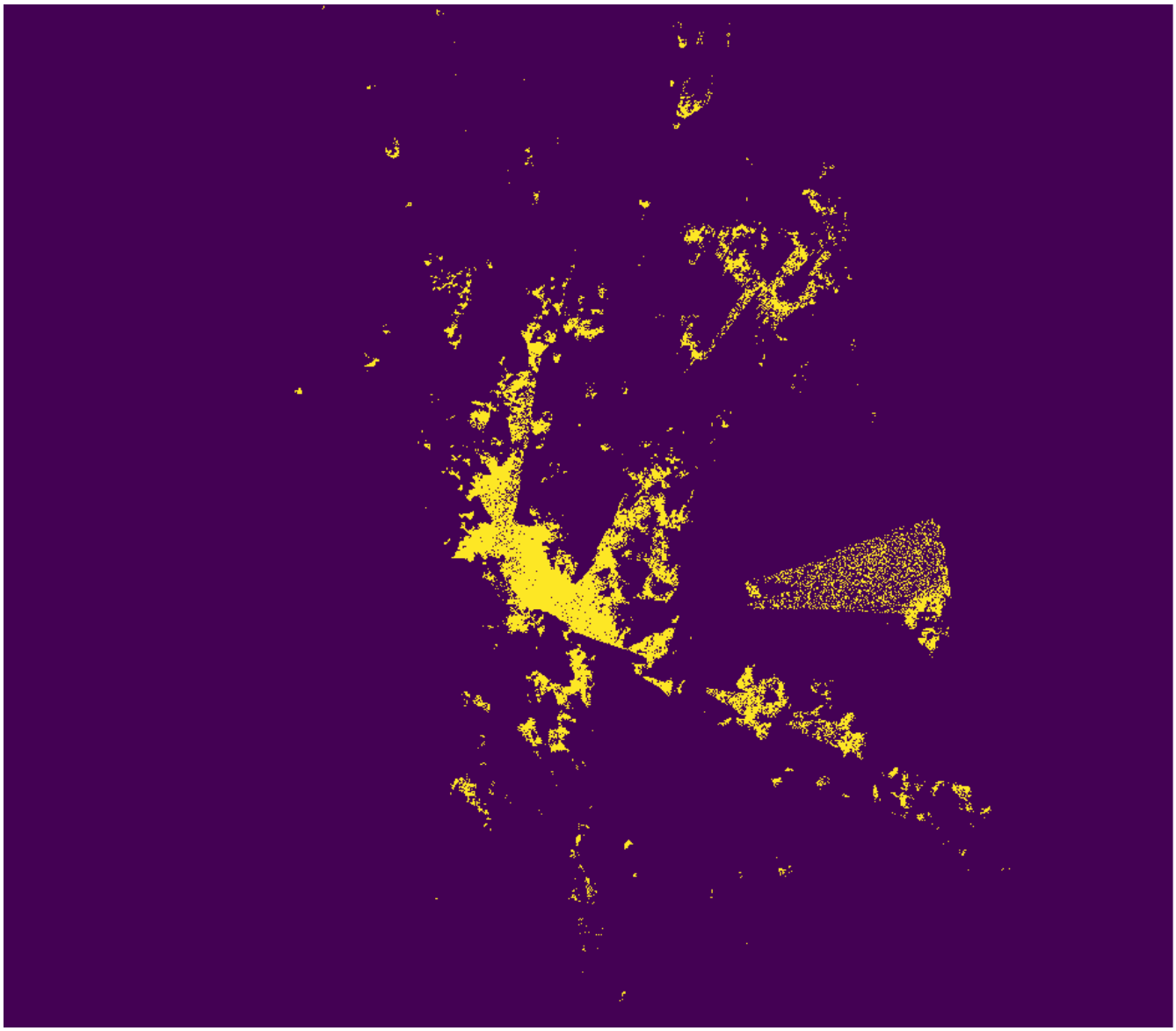

Figure 2.3 – Example of a viewshed

2.3 Independence between building volume and viewsheds

The spatial and anthropological study department wanted to verify if there was a link between the volume of the structures in the area, and the amount of land visible from these structures.

To perform this analysis, we calculated the viewsheds of the observation points in the `Chactùn` region on a wider DTM than that provided by the Institute to avoid edge effects. Which means viewpoints on the edge of the DTM had their field of view truncated by DTM boundaries. We therefore purchased a DTM from the SRTM (Shuttle Radar Topography Mission) of 2500km². We then counted the visible pixels of each viewshed to quantify the visibility of the buildings. We related the volume of the buildings to their quantity of visible pixels, the results of which are as follows:

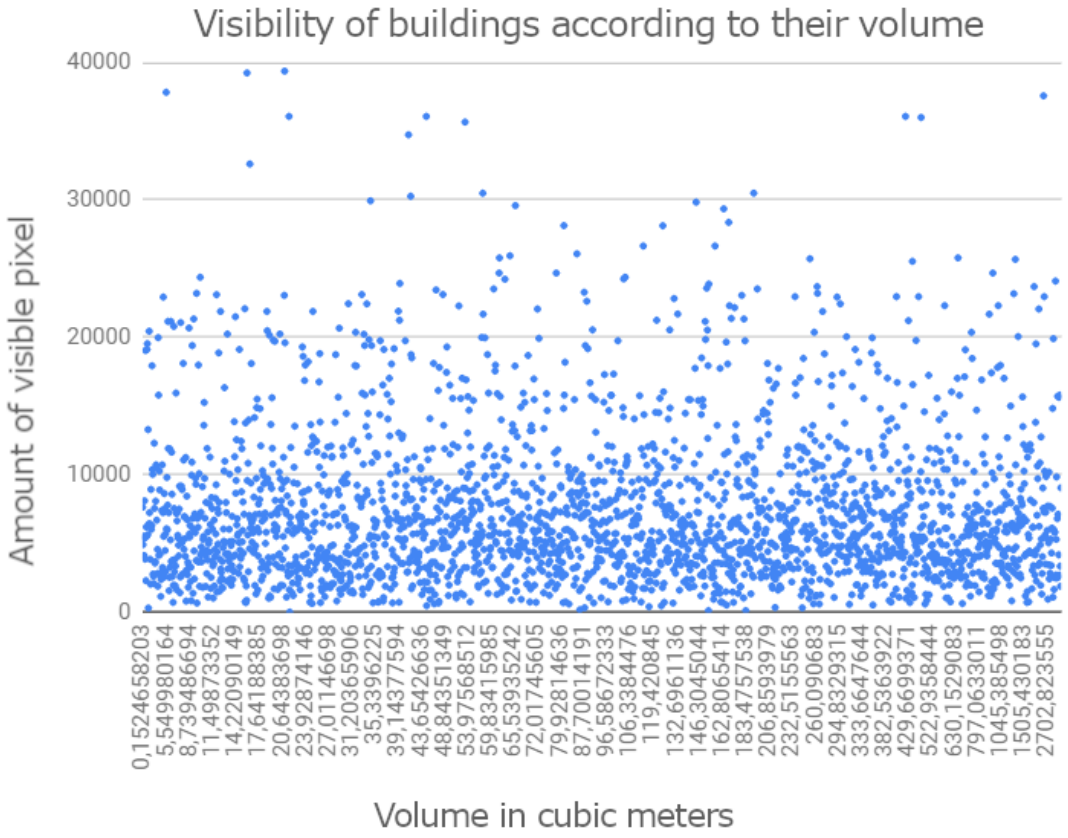

Figure 2.4 – Visibility of buildings according to their volume

It therefore appears from the results that there is no correlation between the volume and visibility of a building. The Bravais-Pearson linear correlation coefficient gives us a result of -0.00346 and therefore confirms this lack of a linear relationship between these two variables.

2.4 Visualizing viewsheds

After creating viewshed of all buildings in the area, we can start visualizing them. Because of their high number, it might be long and tedious to seek out viewsheds of specific observation points. Consequently, we have developed a QGIS plugin returning viewsheds according to input observation points. This python script also creates a new cumulative viewshed which is a merge of fields of view of input observation points. This cumulative viewshed is sightly different than a regular one because his values range between 0 and the number of input observation points and mean how many observation points can see the target pixel.

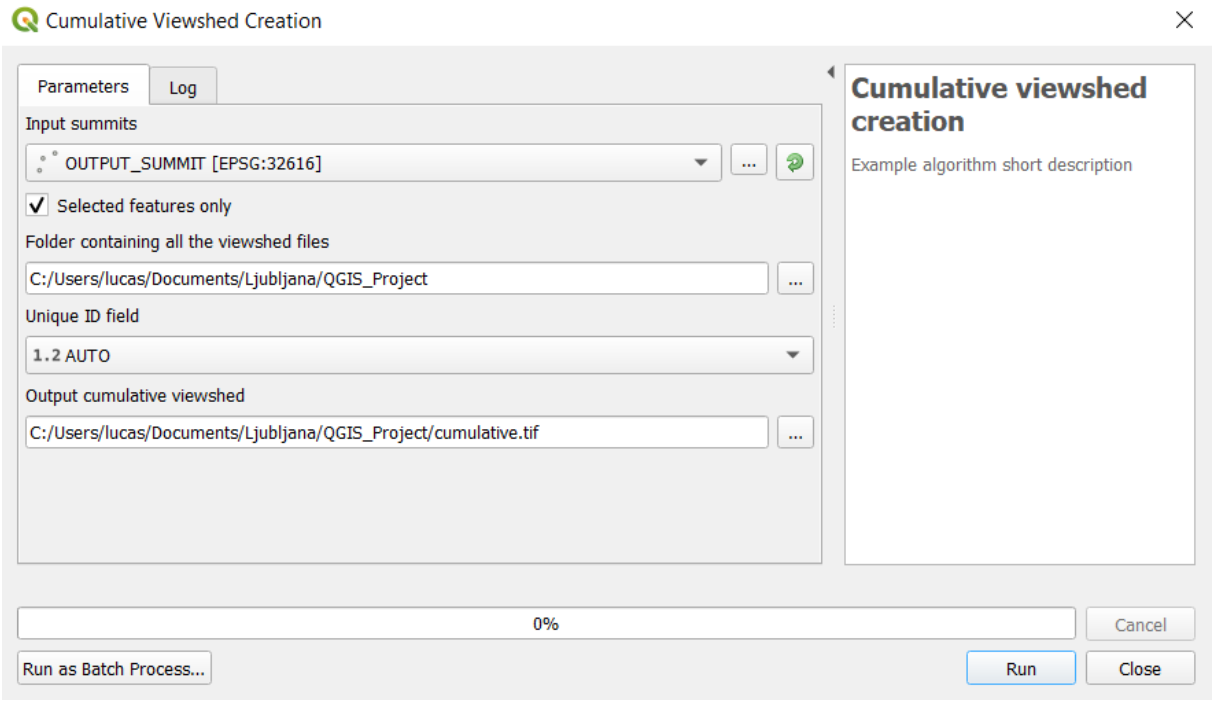

Figure 2.5 – Plugin interface

Conclusion

The most troublesome step of this study is the viewshed computation of plenty of building. Indeed, it takes a few days to get viewsheds of thousands of buildings. One solution should be to use most of the computation power of the computer. The official python library for GRASS GIS -*pygrass*- supports multiprocessing, so developing a multiprocessing python script which creates viewsheds is possible. The viewshed creation command uses only two processor cores. A four cores computer can run this command two times concurrently. So we have divided observation points and run simultaneously two commands *r.viewshed.cva*.

We choose to separate the viewshed calculation and visualization steps. Another solution to reduce the time to compute viewsheds could be to calculate only viewsheds needed. The QGIS plugin could be improved to create on the fly viewsheds of input observation points. The official python library for QGIS -*pyqgis*- supports GRASS algorithms, so this upgrade is achievable.

Bibliography

[1] GRASS Official Documentation. g.region.

[2] Myron Yanoff and Jay S. Duker. 2009. Angular resolution of human eye (p.54).

[3] GRASS Official Documentation. r.viewshed.cva.

[4] r.viewshed source code. Refraction adjustment [Line 84].

[5] David Bolles. Some Thoughts on Stature and Dimorphism Amongst the Yucatec Maya .

[6] GRASS Official Documentation. r.viewshed.

[7] Hirt Christian, Guillaume Sébastien, Wisbar Annemarie, Bürki Beat, and Sternberg Harald. 2010. Refraction coefficient [13].

[8] World Weather Online. Historical Xpujil weather data.

[9] International Civil Aviation Organization. Tropospheric thermal gradient.

Annexes

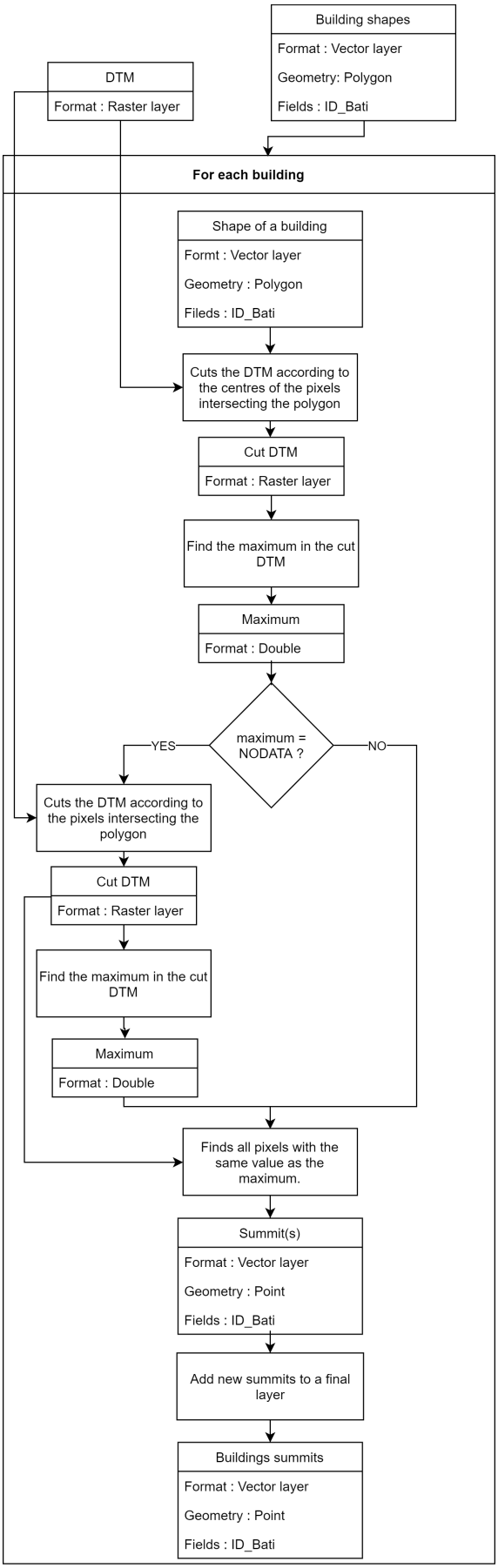

Figure 6 – Diagram explaining the process of the summit calculation algorithm from the contours of buildings

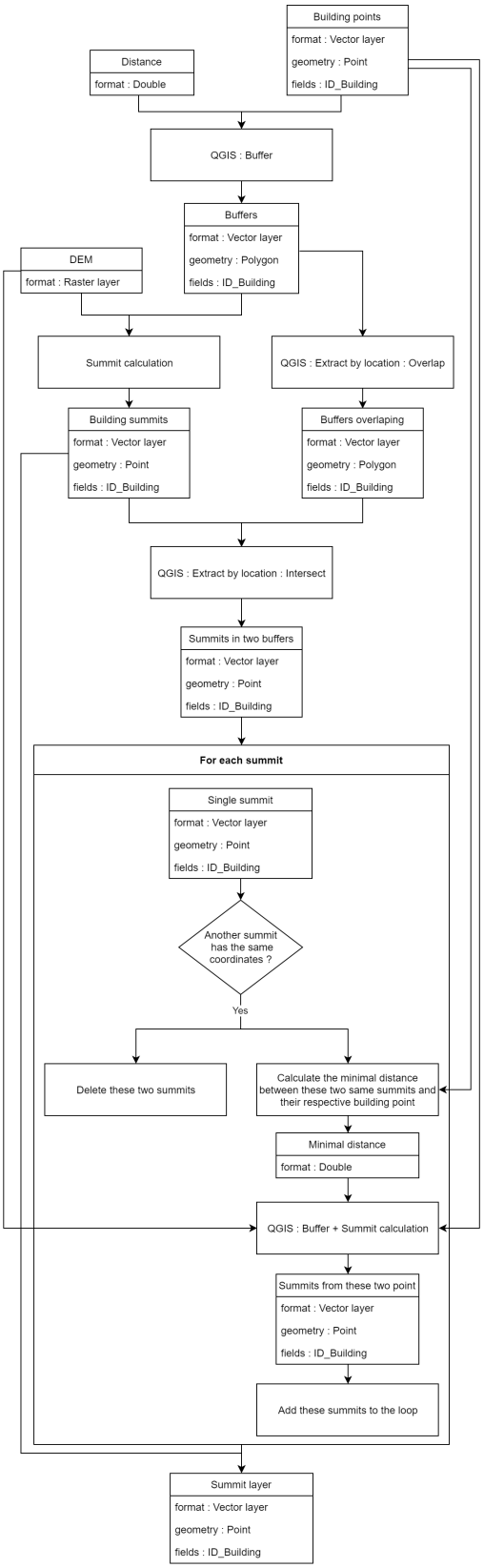

Figure 7 – Diagram explaining how to calculate summits thanks to centre points

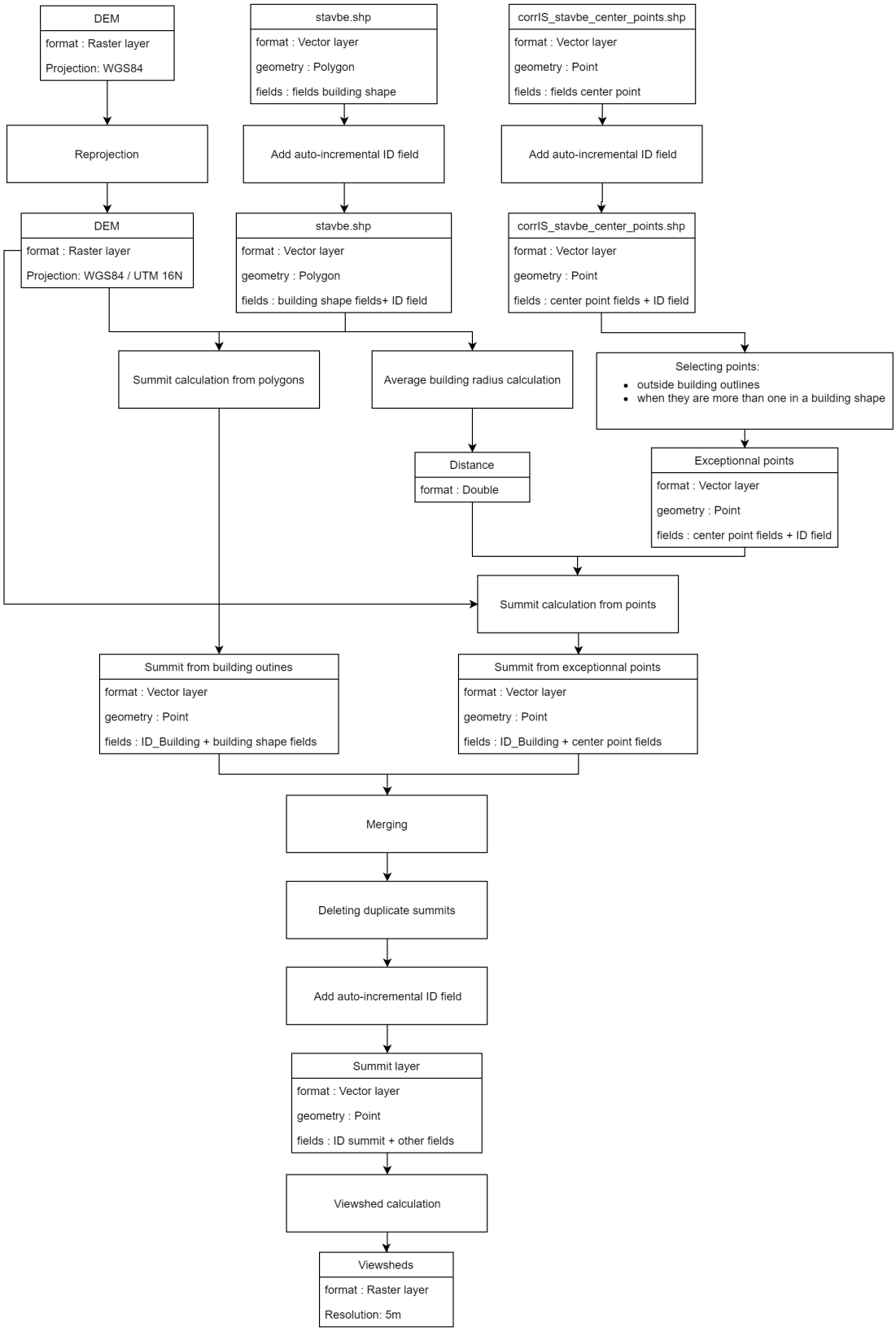

Figure 8 – Viewshed workflow

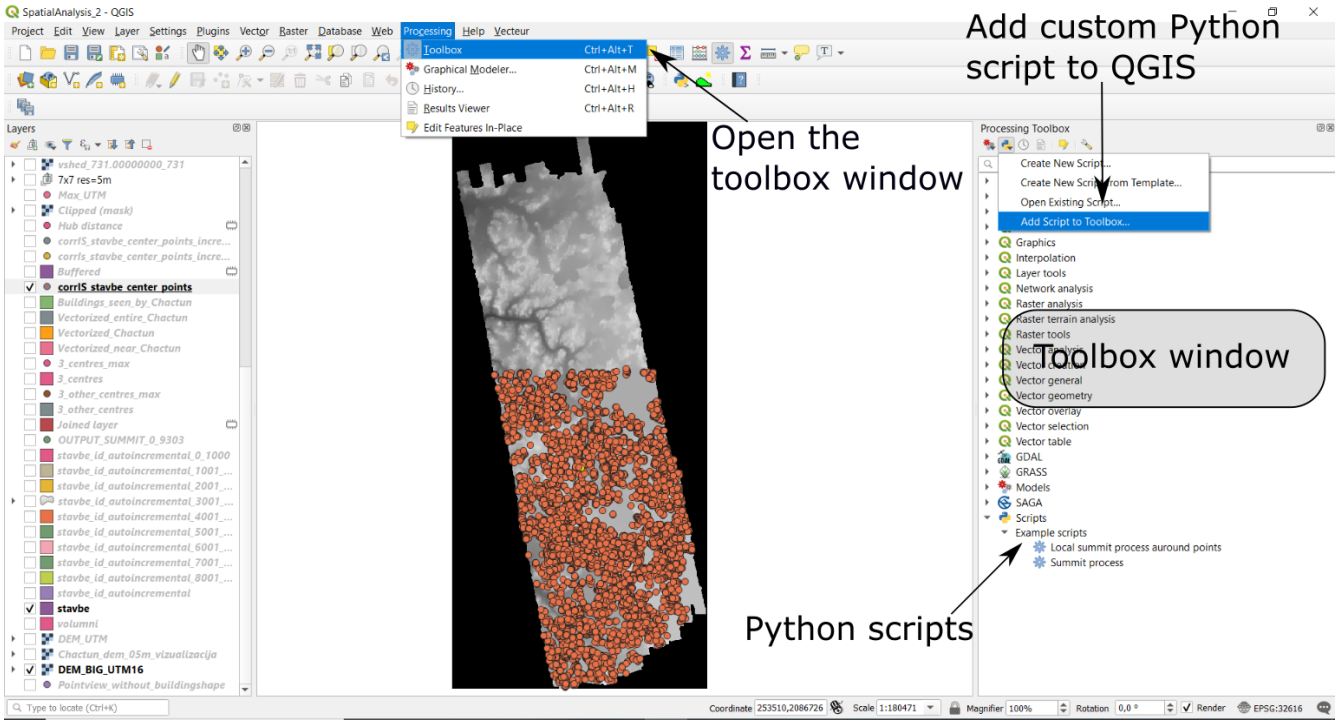

Figure 9 – QGIS interface presentation