Assignment 04
Processing point clouds

Dimitris Mantas (5836670)
Giorgos Iliopoulos (5842247)
Maria Luisa Tarozzo Kawasaki (5620341)

1. Introduction

The purpose of this report is to document the program developed by Team #4 and submitted as a solution to the fourth assignment of GEO1015 (Digital Terrain Modelling), titled “Processing Point Clouds”, for the academic year 2022-2023 at Delft University of Technology. This assignment entailed the appropriate preprocessing of a given region of interest (ROI) of the Actueel Hoogtebestand Nederland (AHN3) point cloud dataset and its automatic conversion to various terrain representations in the form of a raster digital terrain model (DTM) and vector contour map.

The ROI assigned to Team #4 was the one encompassed by tile 58EN2 of AHN3 ((205000, 368750)-(210000, 375000) in EPSG:28992). The LAZ file containing the point cloud corresponding to this area can be downloaded at https://download.pdok.nl/rws/ahn3/v1_0/laz/C_58EN2.LAZ.

The program developed for this assignment enables its end user to read LAS/Z files, process the point clouds defined by them using a custom toolset, and save their work to disk at any time. In addition, the aforementioned datasets can be generated and exported to separate files on demand, given the current state of the point cloud under consideration. At this point, it must be noted that this program works in a destructive manner, similar to scientific computing libraries, such as NumPy. This means that each operation performed directly on a certain point cloud permanently alters its state. For example, the crop filter overwrites any point records that are passed to it. In other words, there is no undo functionality offered.

2. Preparing Point Clouds

Once the above-mentioned LAZ file was downloaded from Publieke Dienstverlening Op de Kaart (PDOK), a 500x500 m area was selected such that it would contain a diverse mix of buildings, vegetation, and water bodies. The selected area was centered at (206593.9, 374018.2), and so its extents were defined by (206343.9, 373768.2)-(206843.9, 374268.2) (Figure 1).

Figure 1. Top view of the ROI.

In any case, the program can read point clouds into memory using its PointCloud class, which relies on laspy and lazrs for LAS and LAZ file support, respectively. All relevant functionality, except contour line extraction, is implemented as a method of this class. An instance of this class can be created using the following command:

```
pt_cloud = point_cloud.PointCloud("example.las")
```

where example.las is the relative or absolute system file path to an arbitrary LAS file saved somewhere on the end user's computer. In this case, the file is assumed to be present in the current working directory, but this is not mandatory. The pt_cloud object has a data attribute of type LasData and supports cropping, ground filtering, thinning, DTM generation, and saving capabilities.

Once read, the point cloud was cropped to the ROI presented in Figure 1 using:

```
# This is the coordinate pair representing the location of the lower left (i.e., southwest) corner of the ROI.
x =206343.9  #m
y = 373768.2  #m
# The dimensions of the ROI are in the same units as x and y.
pt_cloud.crop(geom.BoundingBox(geom.Point(x, y), 500))
```

where the geom module offers certain geometry-related utilities. Note that PointCloud.crop accepts the so-called origin of an axis-aligned rectangle and extends it along the positive X- and Y- semi-axes based on its desired length along them. If this rectangle is actually a square, the third argument can be omitted as demonstrated above. Note that cropping is

performed on a point-by-point basis by checking whether it lies inside a given bounding box or not.

The point cloud corresponding to the ROI is presented in Figure 2.

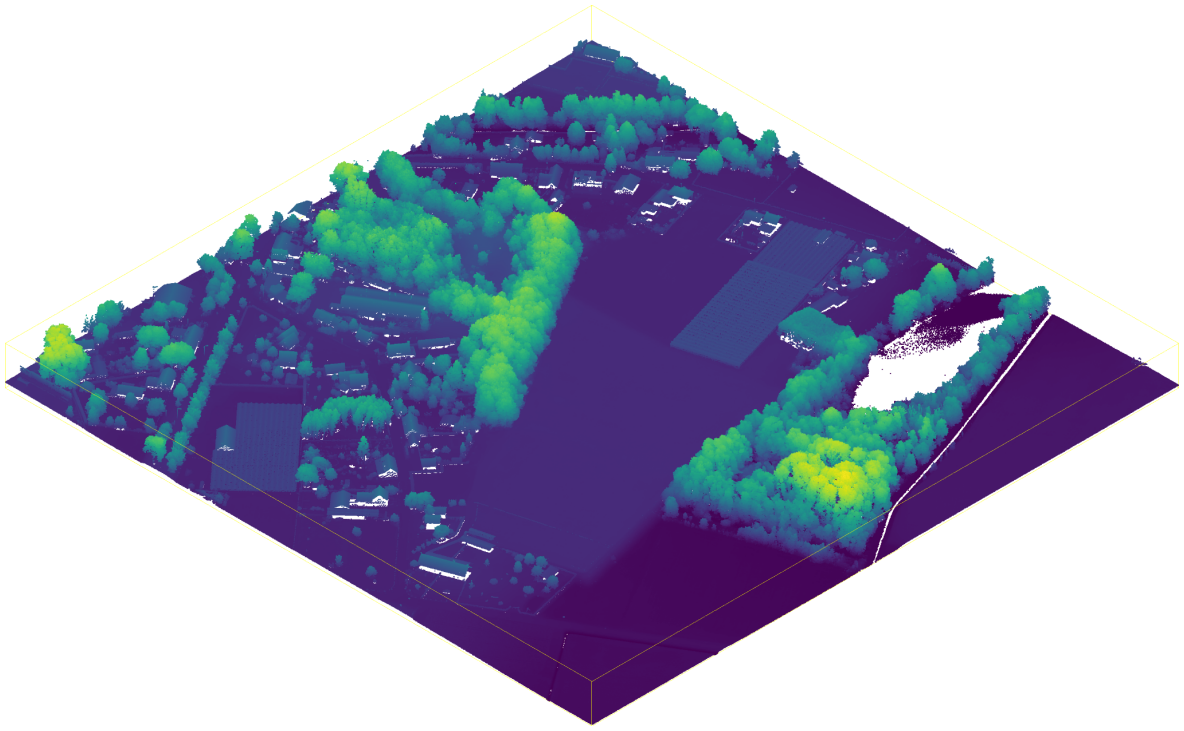


Figure 2. Front isometric view of the point cloud corresponding to the ROI.

Leastwise, the current state of a point cloud can be saved to a separate file using the following command:

```
pt_cloud.save("example_processed.las")
```

The resulting file contains not only the raw geometry of the point cloud but also all its original metadata, such as the classification, intensity, and number of returns. Consequently, the files produced by the program support the same file version and point format as the ones they are derived from and retain as much information from them as possible.

3. Filtering the Ground Points

The ground points of a point cloud can be classified using:

```
pt_cloud.csf()
```

The program uses a simplified version of the cloth simulation filter (CSF) for this purpose (Ledoux et. al., 2022), which is based on the algorithm proposed by Zhang et al. (2016). The main concept of this algorithm is based on that of laying a cloth on a surface while ignoring any air resistance. While a gravitational field pulls the cloth downwards, internal forces between its individual particles hold it together until it hits the surface and

ultimately reaches equilibrium. These forces are due to tension, shear, and torsion springs among the particles, but the implementation used in the program considers only the former ones. Note that the stiffness of these springs can be adjusted in order to accommodate different terrain types (i.e., increasingly rigid cloths are more suitable for flatter terrains).

Based on this idea, the point cloud is initially inverted (Figure 3). This is because the cloth must eventually settle on its ground points. The gradual process of then laying the cloth on the point cloud is briefly explained below.

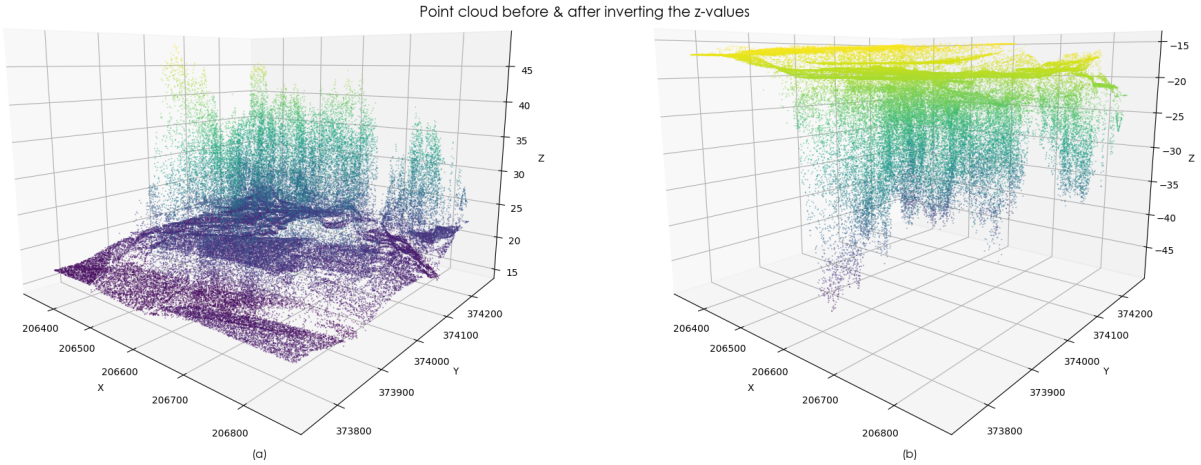


Fig. 3 Point cloud downsampled by 1.5%. In (a) the point cloud has its original Z-values. In (b) the Z-values are inverted. The axes are in scale.

Before the initialisation of the cloth (Figure 4) at an arbitrary height above the point cloud[1], the minimum elevation of each of its particles is taken equal to that of its nearest neighbor in it. After the cloth is "dropped" (Figure 4), its fall is simulated at certain time steps, during which any external or internal present forces acting on the cloth are progressively applied to it.

[1] In the program, the initial elevation of the cloth is computed as the sum of the maximum elevation of the inverted cloth and the vertical distance between its highest and lowest point, unless otherwise specified. However, it was found that this value would sometimes result in erroneous behavior and the simulation being terminated as soon as the cloth collided with the point cloud. For instance, the DTMs presented in Figures 03-04 were generated by setting this parameter to -15 m.

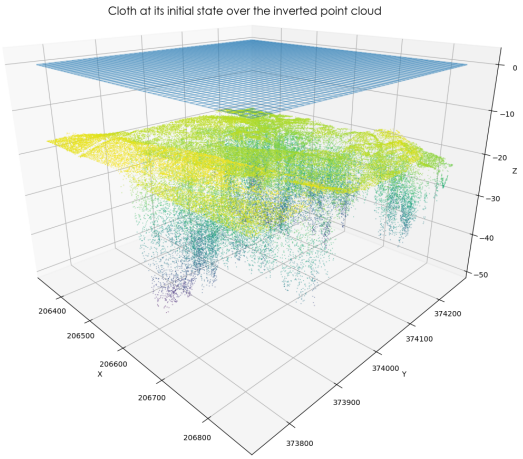


Fig. 4 The Cloth at its initial state over the inverted Point Cloud. The axes are in scale.

During the free-fall phase of the simulation, the elevation of the cloth is governed by gravity, and thus computed using Equation 1.

$$z_{previous} = z_{current} + 0.5 \cdot g \cdot timestep^2,\ g = 9.80665\, ms^{-2}\ eq.\ (1)$$

Once the external forces have been computed, the elevation of each particle is validated and possibly corrected in a way that the value of any is less than its assigned minimum. The particles for which this holds are deemed "unmovable" for the remainder of the simulation, meaning that they have successfully settled on the ground (Figure 5)[2].

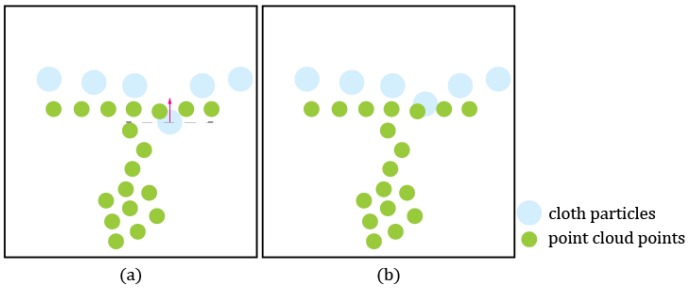


Fig. 5 (a) One cloth particle lays under the minimum allowed z-value. (b) Its z-value is updated to minimum acceptable and its status is set to unmovable.

This means that, when taking Equation 1 into consideration, the time step of the simulation is of utmost importance since, if it is too high, the cloth may collide with the ground at great speed, causing large particle patches to incorrectly become unmovable. This proved to be a crucial issue in this assignment and was only solved through trial and error.

In any case, the difference in elevation between each movable particle and its adjacent ones (i.e., the ones connected to it via the above-mentioned tension springs) is then calculated. As shown in Figure 6a, the four particles lying on each corner of the cloth have two such

[2] In the program, this process is performed after both the external and internal forces acting on the cloth have been applied to it.

neighbors, whereas particles along its side have three. The remaining particles are connected to four others each.

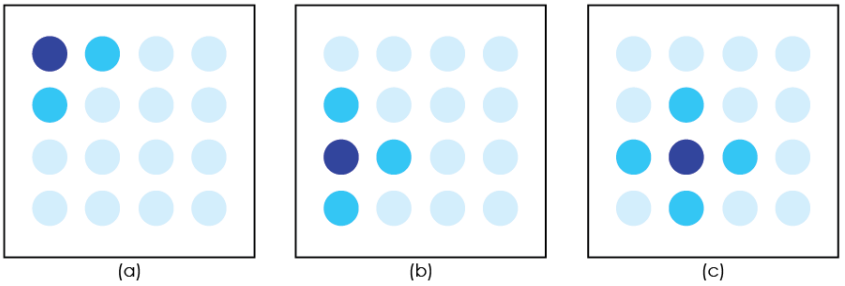


Fig. 6 Neighbours to consider in three different cases. The dark-blue particle is the one under scope (p0), the cyan particles the neighbours (pi) and the light-blue the rest of the grid.

From Figure 6, in case the particle is on the corner (a) only two neighbours are included, (b) when the particle is on the border of the cloth three while, when the particle is not on the border or corner (c) four particles are taken into account for the calculation.

After these differences have been calculated, the elevation of each particle is corrected according to Equation 2. This is effectively equivalent to applying the internal forces acting on the cloth.

$$z_{current} = z_{current} + stiffness \cdot dh \; eq. \; (2)$$

If the particle under consideration p0 is movable, but its neighbor p1 unmovable (Figure 8a) only p0 is moved towards p1. However, in the case where both p0 and p1 are movable, they are moved in opposite directions (Figure 8b).

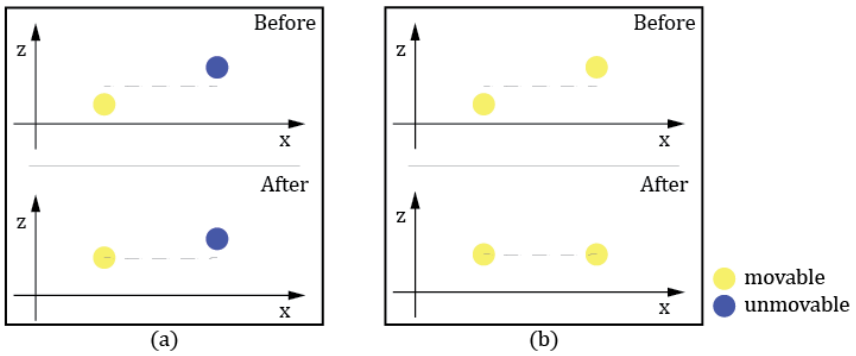


Fig. 8 (a) One particle is movable and the other is unmovable. (b) Both particles are movable.

That being said, this whole process is repeated until the maximum vertical displacement in the cloth between two consecutive time steps is found to be less than a given threshold, at

which point either the ground points of the point cloud are classified or the cloth is returned[3] (Figures 9-10).

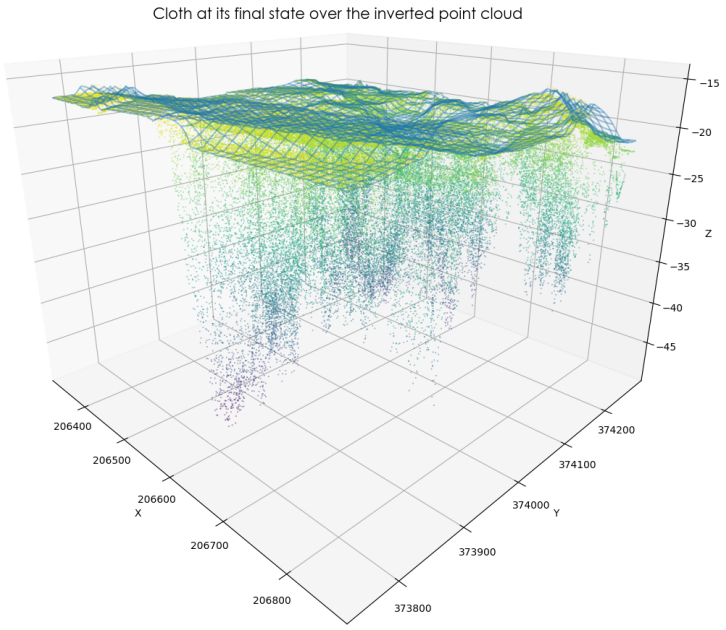


Fig. 9 The Cloth at its final state after the completion of the process over the inverted Point Cloud. (The axes are in scale)

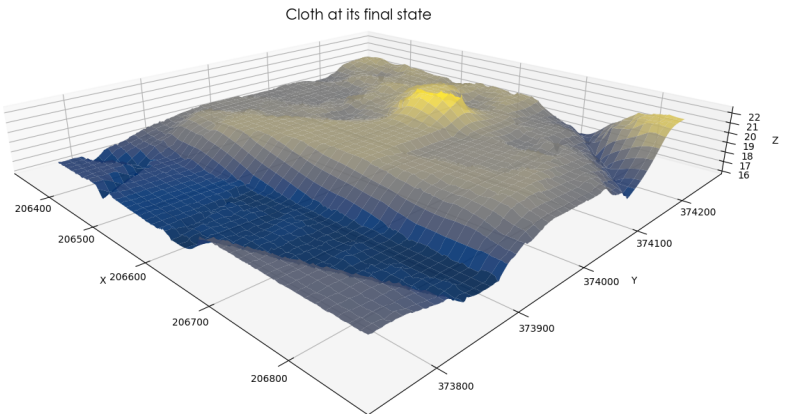


Fig. 10 The cloth at its final state. The axes are in scale.

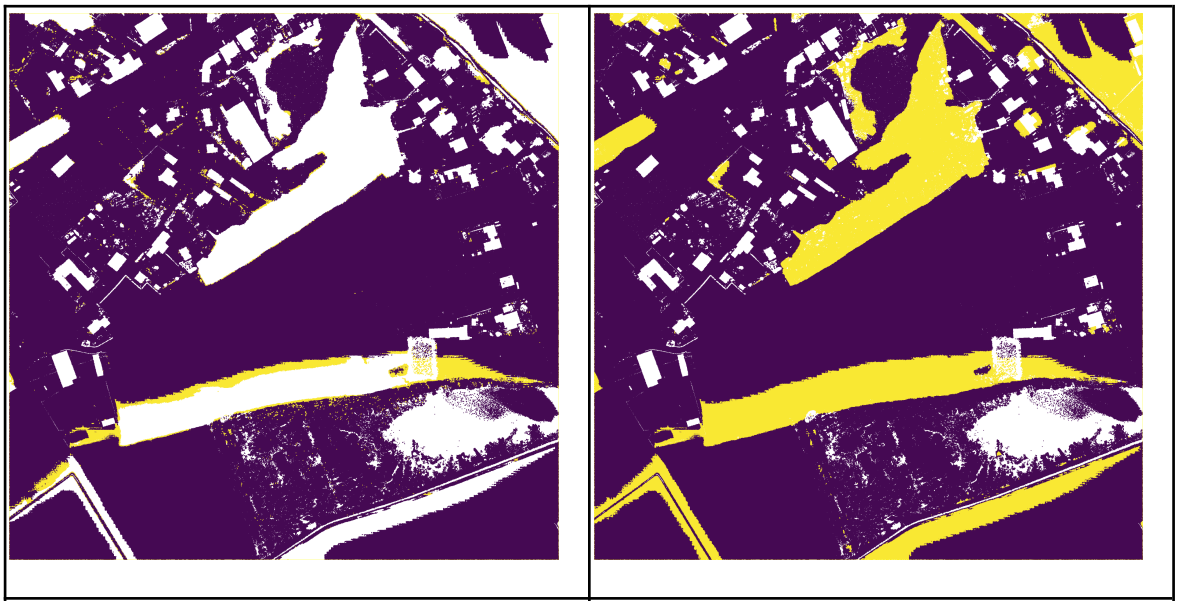
Leastwise, the point cloud is classified by looping over its points and checking if the difference in elevation between itself and its nearest neighbor in the cloth is less than or equal to a user-defined parameter. Points for which this hold is deemed as belonging to the ground. Furthermore, the cloth can also be saved to a separate file using the following command:

pt_cloud.csf("example_csf.npy")

[3] In the program, the point cloud is always classified automatically. Note that the metadata produced by PointCloud.csf overwrites any already existing, relevant records.

The resulting file is a serialized three-dimensional (3D) NumPy array which contains the final coordinates of each particle in the coordinate system used by the point cloud.

In general, it was found that the most challenging aspect of implementing the CSF was gaining a strong understanding of the structure of the individual processes involved. The paper by Zhang et al. (2016) was very helpful in resolving a great deal of lack of knowledge regarding the issue involved and allowed for results that closely resembled the state of the art (Figure 11). In the figure below, the arguments of PointCloud.csf were left at their default values, while the user-defined parameters of the plugin were set accordingly.



(a) (b)

Figure 11. Superimposition of the ground points of the point cloud corresponding to the ROI, as classified by the program (purple) and the official implementation of the CSF (yellow - a), which is offered as a free CloudCompare plugin, as well as AHN3 (yellow - b).

4. Digital Terrain Model Generation

4.1. Using the Dataset Produced during Ground Filtering

As mentioned in Chapter 3, one of the possible outputs of the CSF is the corresponding cloth in its steady state, when it is effectively equivalent to the ground surface of the point cloud the filter is applied to. At this point, it must be said that such an object is modeled in the program by a uniformly distributed particle grid (Figures 4, 9-10), but this is indeed a valid representation of a surface since this grid can, for example, be triangulated or resampled with minimal effort.

Leasewise, the DTM of a point cloud can be generated in this way using:

# Is in the same units as the coordinate reference system of the point cloud.
resolution = 0.5

```
pt_cloud.dtm("dtm.tif", resolution=resolution,
method=point_cloud.DTMGenerationMethod.CLOTH)
```

where the DTMGenerationMethod enumeration lists all the relevant capabilities of the program. Note that the resulting dataset is a GeoTIFF raster with a cell size of 0.5 units being saved to the current working directory under the name "dtm.tif".

The program uses an iterative algorithm for the creation of DTMs regardless of the selected generation method. Its main idea revolves around the instantiation of an appropriately shaped grid filled with NODATA values, which are read from a user-defined profile[4], and the subsequent population of each of its cells by interpolating the elevation of its center point. More precisely, the cloth, which must be made up of at least three particles, is first produced and triangulated. Afterwards, the nested for-loop found at https://gitlab.tudelft.nl/3d/geo1015.2022/-/blob/main/hw/01/geo1015_hw01.py#L61 is used to effectively resample it to the required resolution using the Laplace interpolant (Ledoux et. al., 2022). The choice of interpolation method was deliberate as, given ones offered by the triangulation library the program relies on[5], both linear and nearest-neighbor tend to visibly partition interpolation fields. This left the developers with a choice between natural nearest neighbor and non-Sibsnonian interpolation, of which the latter is automatic, almost continuously differentiable by default, exact and faster (Ledoux et. al., 2022).

Note that, in contrast to AHN3 DTMs, the program does not assign NODATA values to cells which cover areas where there are no ground points. Instead, it assumes that the ground is continuous everywhere, similar to Shuttle Radar Topography Mission datasets, except outside the convex hull of the above-mentioned triangulation. In other words, no extrapolation functionality is offered.

4.2. Using the Metadata Produced during Ground Filtering

As an alternative to what is mentioned in Chapter 4.1, the DTM of a point cloud can be generated based on its classification metadata and the following command:

```
# DTMGenerationMethod.CLASSIFICATION is the default method used by the program.
pt_cloud.dtm("dtm.tif", resolution=resolution)
```

In contrast, to DTMGenerationMethod.CLOTH, while this method also invokes PointCloud.csf, it does not require that the corresponding cloth be returned. Instead, the ground points are filtered, triangulated and then interpolated, as explained.

In general, it was found that the most challenging aspect of implementing the above-mentioned algorithm was the confusion caused by the unconventional use of

[4] In the case of the AHN3 dataset, the raster profile used corresponds to an LZW-compressed, 32-bit floating point GeoTIFF raster, with a NODATA value of exactly 3.4028234663852886e+38. The coordinate reference system (CRS) of the data produced using this profile is EPSG:28992. In any case, the program generally supports custom profiles, as explained at https://rasterio.readthedocs.io/en/latest/topics/profiles.html.
[5] https://pypi.org/project/startinpy/

NODATA cells in the AHN3 datasets and the consequent doubt as to whether this was the required approach or not. Once this issue had been resolved, work for this part of the program was relatively straightforward.

4.3. Comparison of Digital Terrain Model Generation Methods

The DTMs of the ROI, generated using DTMGenerationMethod.CLOTH and DTMGenerationMethod.CLASSIFICATION are presented in Figures 12-13, along with the corresponding AHN3 dataset (Figure 14).

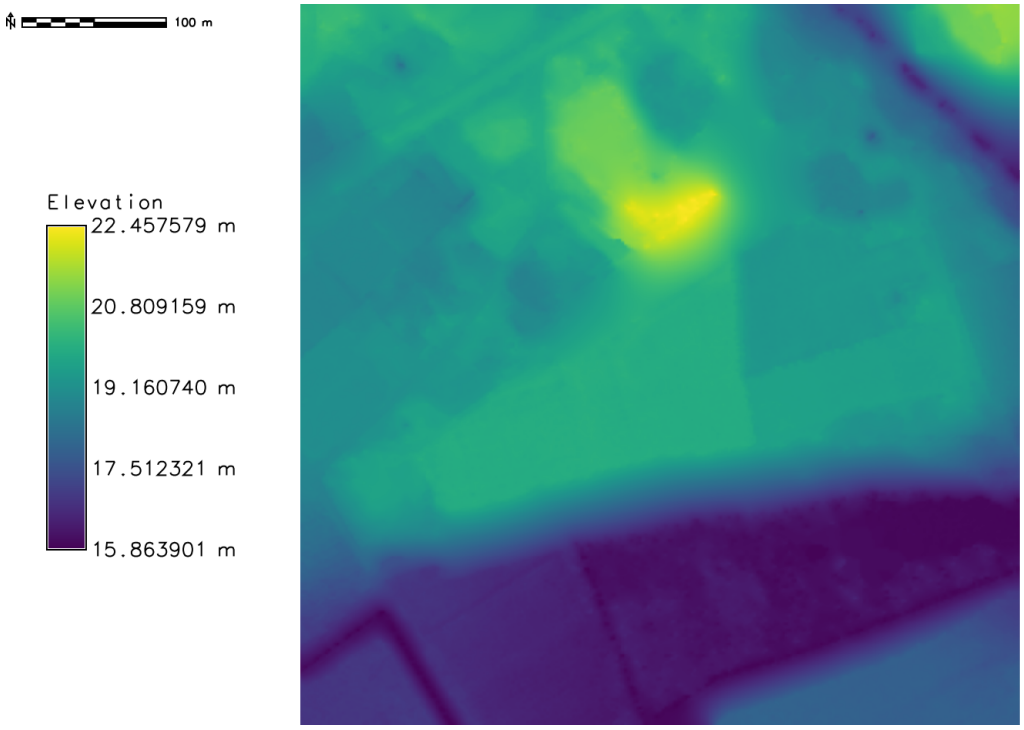


Figure 12. The DTM of the ROI, generated using DTMGenerationMethod.CLOTH at a resolution of 0.5 m.

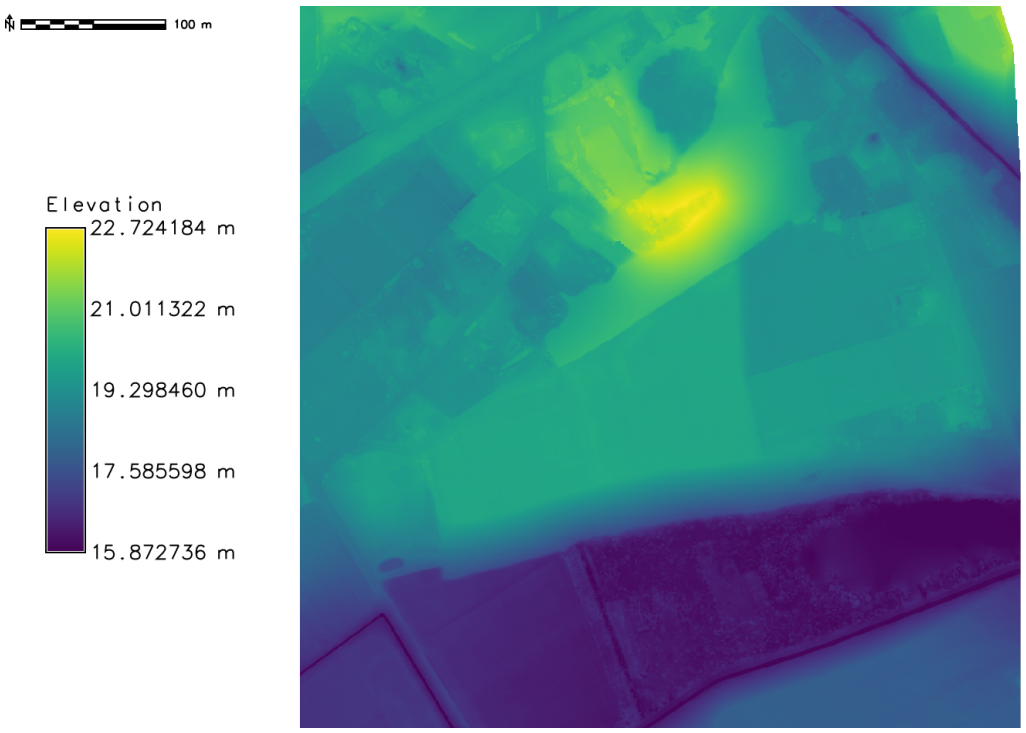


Figure 13. The DTM of the ROI, generated using DTMGenerationMethod.CLASSIFICATION at a resolution of 0.5 m.

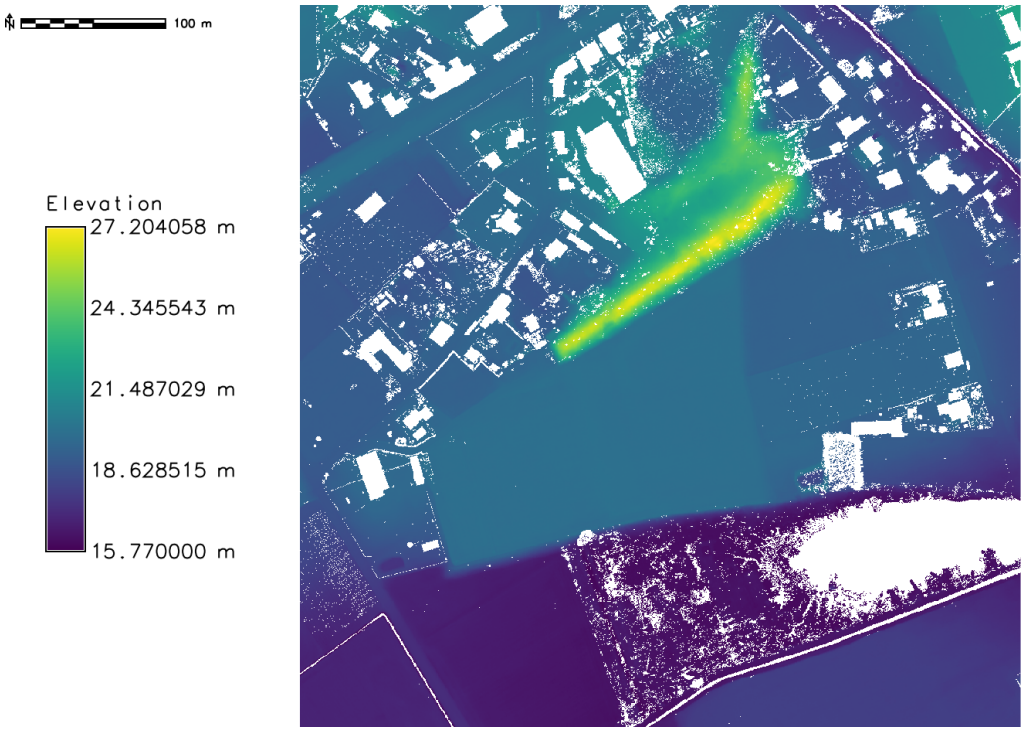


Figure 14. The official DTM of the ROI at a resolution of 0.5 m.

As a general remark the DTMs generated by the program are quite alike in terms of both data range and distribution. Nevertheless, it is clear that the cloth-based DTM is significantly softer than the classification-based one. This should not come as a surprise to the reader because, although both datasets have the same nominal ground resolution, the

former was generated by interpolating a particle grid with a cell size of 2 m[6], while the latter a point set with an almost guaranteed surface density of at least 6 points/m² or one every 40 cm[7]. This means that the cloth-based DTM has effectively been upsampled 16 times from its original resolution in order to achieve the required one, and thus appear blurry. At this point, it must be noted that this issue could be resolved just by generating the DTM using a higher resolution cloth, but this comes at the expense of exponentially longer waiting times and possible numerical instability issues.

Subtracting the aforementioned DTMs from each other (Figure 15) allows for some further observations.

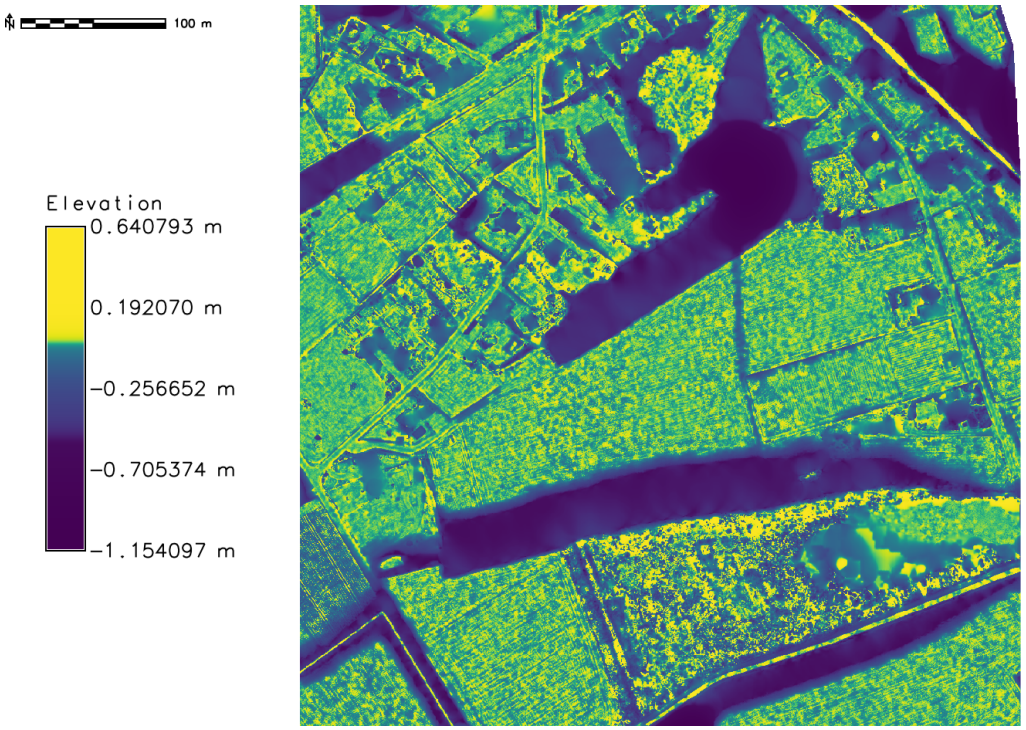
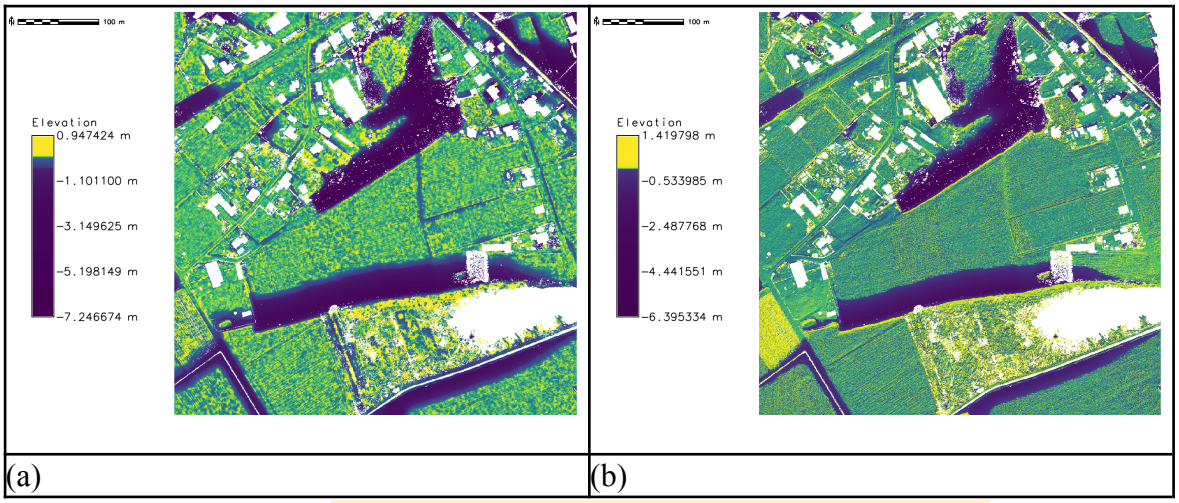


Figure 15. The result of the subtraction of the classification-based DTM from the cloth-based one.

It appears that the cloth- and classification-based DTMs are most similar in open areas, while the most significant disagreement between them manifests itself at areas surrounded by dense, relatively tall trees. This is most likely due to an issue with the former dataset rather than the latter, and it is assumed that the underlying cloth reached equilibrium at a locally incorrect elevation, possibly due to its simplified internal force model or a too fast time simulation time step.

[6] The default cloth resolution used by PointCloud.csf is 2 m, in accordance with the official implementation of the filter, which can be found at https://github.com/jianboqi/CSF.
[7] https://www.ahn.nl/kwaliteitsbeschrijving

In any case, when compared to the official DTM of the ROI (Figure 16), it is obvious that any significant differences between them are almost certainly due to the specific ground filtering method used[8]. This is also corroborated by Figure 11b.



(a) (b)

Figure 16. a:cloth-based-AHN3; a:classification-based-AHN3

5. Downsampling

5.1. Using nth-Point Sampling

A point cloud can be downsampled based on nth-point sampling using the following:

```
percentage = 0.1
pt_cloud.downsample(percentage=percentage,
method=point_cloud.DownsamplingMethod.NTH_POINT)
```

where the DownsamplingMethod enumeration lists all the relevant capabilities of the program. Note that the resulting dataset is thinned by 90% compared to the original.

In short, this method decreases the sample rate of the point cloud by keeping its first point and then every nth one after it. Since this can be achieved using a single for-loop, nth-point sampling is the fastest thinning method (Ledoux et. al., 2022), and was chosen based on this. However, it must be noted that $percentage^{-1}$ must evaluate to an integer in order for this method to work as expected. For instance, all values of percentage ranging from 0.4-0.5 invoke the same behavior from PointCloud.downsample due to technical limitations related to the operation of the built-in round function in Python.

In general, it was found that the most challenging aspect of implementing nth-point sampling was the fact that the underlying point record would become discontiguous in memory under certain circumstances. This would result in erroneous behavior and the end user not being able to save their work. Once this issue had been resolved, work for this part of the program was relatively straightforward.

[8] https://www.ahn.nl/ahn-the-making-of

5.2. Using Random Sampling

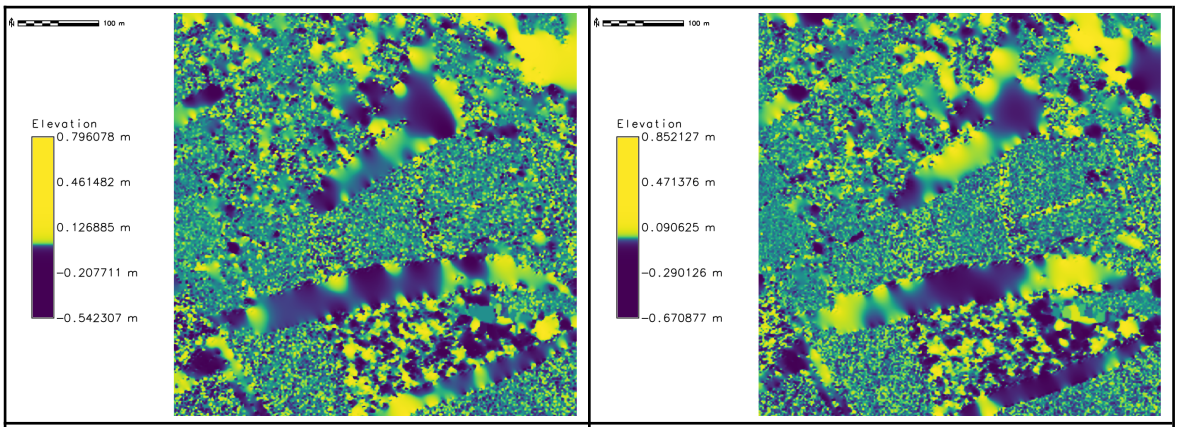
As an alternative to what is mentioned previously, the point cloud can be downsampled based on random sampling and the following command:

```
# DownsamplingMethod.RANDOM is the default method used by the program.
pt_cloud.downsample(percentage=percentage)
```

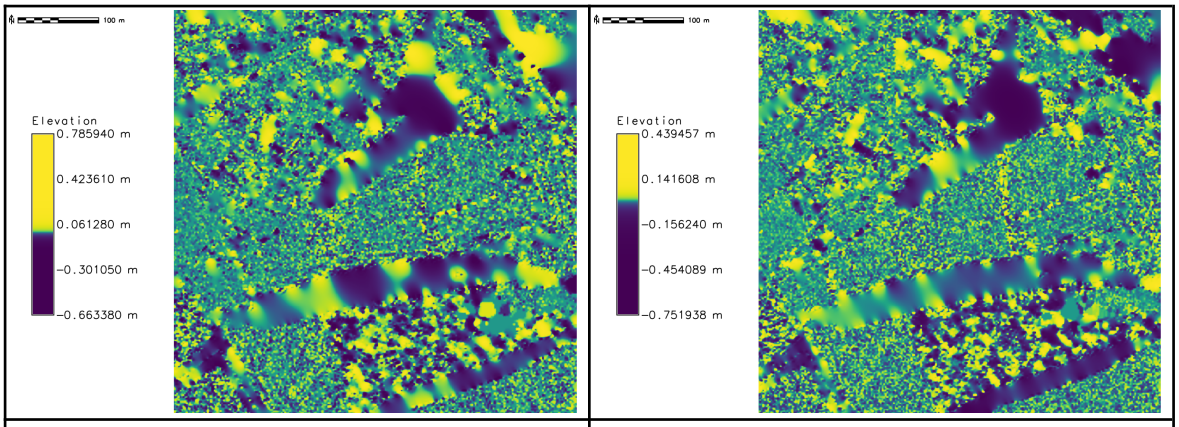
In short, random sampling first computes the number of points corresponding to the user-defined percentage, and then iteratively draws them from a random uniform distribution without replacement. While this method is appreciably slower than DownsamplingMethod.NTH_POINT, it is also significantly more robust as even the slightest changes in the value of percentage invoke a different behavior from PointCloud.downsample. At this point it must be noted that the program uses a specific seed for its random number generator, so datasets produced using this method are guaranteed to be from user to user.

5.3. Effect on Generated Digital Terrain Models

As shown in Figures 17-18, the general effect of point cloud downsampling on the corresponding DTM is significant. Specifically, it appears that a decrease in the sampling rate results in similarly less precise derivative datasets. This is because the interpolant used to generate them is fed with progressively spatially scarce data. Moreover, it is clear that DownsamplingMethod.RANDOM performs distinctly better than DownsamplingMethod.NTH_POINT. This is most likely because, while the former method is guaranteed to preserve the descriptive statistics of the point cloud it is applied to, the same cannot necessarily be said for the latter. For instance, it might be the case that points are not numbered in a grid-like manner in the underlying record, and thus certain areas are favored as more densely sampled. In other words, this method preserves the spatial distribution of not the point cloud, but the indices of its points, which might not be uniform.

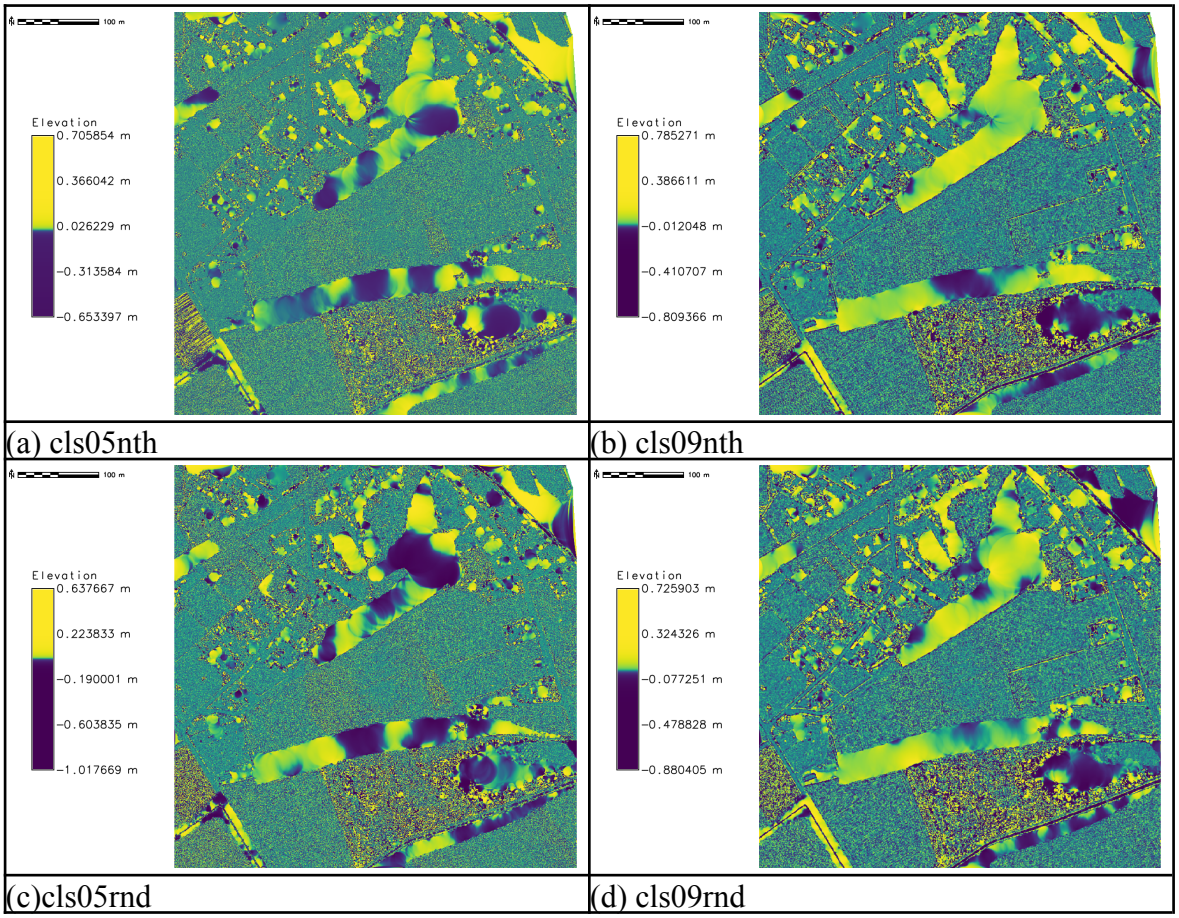


(a) clo05nth (b) clo09nth



(c)clo05rnd | (d) clo09rnd

Fig. 17 cloth-based DTM, from left to right and top to bottom: 50% downsampling nth point, 90 nth, 50 random, 90 random



(a) cls05nth | (b) cls09nth

(c)cls05rnd | (d) cls09rnd

Fig. 18 classification-based DTM, from left to right and top to bottom: 50% downsampling nth point, 90 nth, 50 random, 90 random

6. Extracting Contour Lines

6.1. The Meandering Triangles Algorithm

The approach used to draw isolines, at every two meters, was based on the Meandering Triangles algorithm. This algorithm is based on the concept of triangulation, which is the process of dividing a surface into triangles.

The Meandering Triangles algorithm begins by triangulating the terrain surface using a set of control points. The algorithm then traces the isolines by following the edges of the triangles, starting at a triangle corner and moving along the edges of the triangles until a triangle corner is reached. The process is repeated for each corner of the triangle until the entire isoline is traced.

To implement this algorithm in the program, an article[9] published by Bruce Hill in 2017 was particularly useful. The code used in the program is an adapted version of the code provided by him, with additions to it.

6.2. Extracting Isolines

The extracted isolines at every two meters from the gridded DTM of 0.5 m resolution can be observed in Figure 19.

[9] https://blog.bruce-hill.com/meandering-triangles

Fig. 19 Isolines corresponding to the DTM of the ROI, generated using DTMGenerationMethod.CLASSIFICATION at a resolution of 0.5 m. The isovalues are 16, 18, 20, and 22 m.

In general, it was found that the isolines shown in Figure 19 were in high agreement with those produced by both GRASS GIS and QGIS.

7. Author Contribution

Name	Contribution	Approximate Workload (%)
Dimitris Mantas	Crop, DTM, CSF, Readme, Report, Coordinator of the project	33%
Giorgos Iliopoulos	CSF, Classification, Report	33%
Maria Luisa Tarozzo Kawasaki	Preparing the dataset, Thinning, Isolines, Report	33%