# PROCESSING AHN4 POINT CLOUDS TO CREATE A CANOPY HEIGHT MODEL OF A REGION IN THE NETHERLANDS

## GEO1015 assignment 4


| Authors | Student number |
|---|---|
| Lars Boertjes | 4704541 |
| Noah Alting | 4828968 |
| Marieke van Arnhem | 4918738 |

**TU Delft**


January 18, 2024

# Contents

# 1 Introduction

This document provides an overview of the workflow for the fourth assignment of the Geomatics course Digital Terrain Modelling, focusing on the processing of a point cloud to a Canopy Height Model (CHM). The used point cloud source is a designated tile, specifically tile *69BZ2_13* from the AHN4 dataset. For a more manageable dataset, a 500x500 meter clip was extracted from this tile. The processing pipeline involved extracting ground points and generating a gridded Digital Terrain Model (DTM), identifying vegetation points and creating a grid, and ultimately producing a Canopy Height Model (CHM).

This report will be a guide through the steps taken to create a Canopy Height Model from the AHN4 point cloud dataset.

# 2 Pick a Team and Assign AHN4 Tile

This team consists of three enthusiastic students coming from different backgrounds. Lars has a BSc in Architecture, Noah has a BSc in Applied Physics and Marieke has a BSc in Civil Engineering. We try to use our differences to our advantage to make this project to a success. Our group picture is included in figure 1.

Figure 1: A group picture. From left to right: Lars Boertjes, Noah Alting and Marieke van Arnhem.

# 3 Download and Prepare the Dataset

For this assignment the focus was on processing a point cloud derived from the designated tile 69BZ2_13 obtained from GeoTiles in LAS format. This tile covers the small village of Scheulder in Limburg. Figure 2 displays the full tile seen from above using CloudCompare.

Figure 2: Full tile view in CloudCompare.

For more convenient processing and as stated in the assignment, the full tile was reduced to a 500x500 meter clipped tile, shown in Figure 3. This clipped tile features buildings, forests, water bodies, and also some elevation. In the figure, the clipped tile can be seen as all the points with RGB coloring. The start tile has been colored using a gray scale for the intensity values. This is done to give an indication of the placement of the clipped tile within its context.

Figure 3: Clipped tile (500x500 meters) shown in RGB within context of full tile

The full and clipped tile are located in the EPSG:28992 coordinate system. The full tile spans from (186980, 314980) in the lower-left corner to (188020, 316270) in the top-right corner. The bounding box of the clipped tile ranges from (187466, 315228) in the lower-left corner to (187966, 315728) in the top-right corner.

Figure 4 shows some of this elevation, when looking at the clipped tile from the side using CloudCompare.

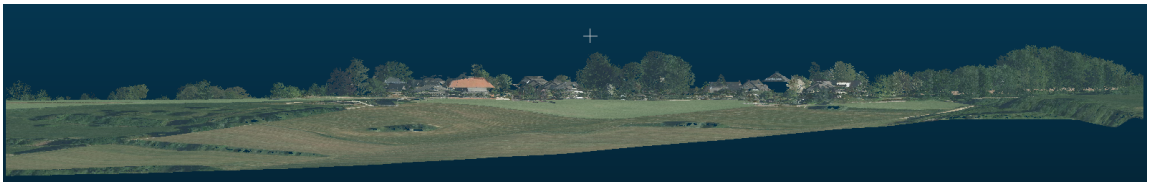


Figure 4: Side view of the cropped tile showcasing elevation

In the first part of our C++ program, the tile cropping process is executed. The LAS file is read using the 'lasreader.hpp' extension. The point cloud data is stored in a vector with the data type 'Point3', which is a CGAL data type. When reading the file, a thinning parameter is passed to the read function. This parameter takes every nth point from the dataset, making sure the size of the data will be manageable to work with. For now, the thinning parameter is set to 1000, dividing the number of points of the dataset by the same amount.

Next, we specify the minimum and maximum x and y values by using the lower-left corner of the bounding box, as shown in the added figures above. Additionally, 500 meters are added to obtain the maximum x and y values.

To crop the file in Python, the function pdal - filters.crop is used. This function can also save the cropped file as a LAS file. But, when exporting the cropped LAS file, four attributes per point are missing: infrared, amplitude, reflectance and deviation. Because infrared is required to extract the NDVI value per point (see Section 5.1.2), a CVS file is also exported to still work with these attributes. This is the reason why the input for vegetation extraction consists of two files: the LAS file and the CSV file. The LAS file is still created for easy visualization.

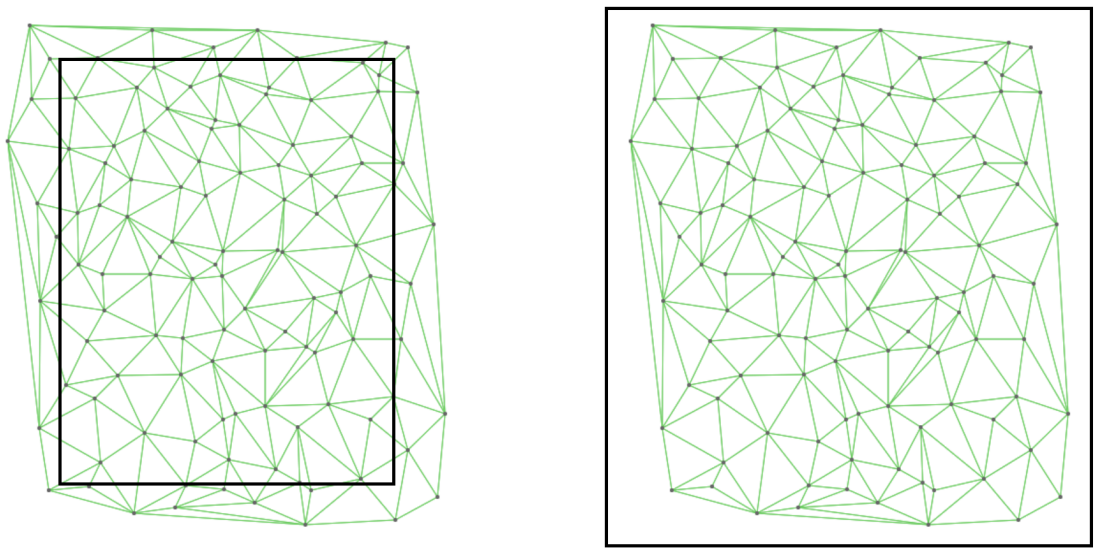
# 4 Extract Ground and Create Gridded DTM

In this step, we employ the Ground Filtering with TIN Refinement (GFTIN) algorithm to extract ground points for the cropped 500x500 tile and generate a gridded Digital Terrain Model (DTM) using Laplace interpolation. This DTM will be used later to create the Canopy Height Model in Step 5.

## 4.1 Ground filtering with TIN refinement (GFTIN)

In the C++ file, we imported the cropped 500x500 tile with a 25-meter buffer on each side. This is because we want the convex hull of the Delaunay triangulation, used with GFTIN, to be outside the 500x500 tile. Placing the convex hull outside the designated tile area helps avoid creating artefacts that may occur when the tile is triangulated.

The illustration in figure 5 shows the Delaunay triangulation with the convex hull positioned both within and outside the tile.



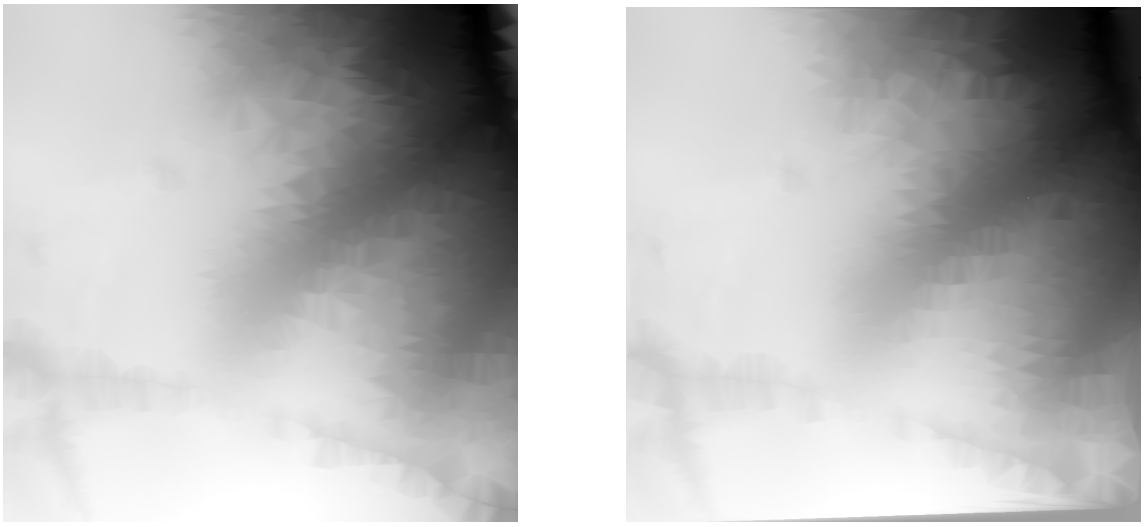
(a) Convex hull outside tile

(b) Convex hull within tile

Figure 5: Conceptual depiction of Delaunay triangulation within certain bounds

This leads to the following triangles after triangulation and interpolation seen in figure 6, in which an artefact can be seen in the lower right corner of the tile with a convex hull within the tile. Also, the terrain in this example slightly rises in the top right corner, resulting in a small bright area in figure a. This is not present in the figure on the right, probably because it falls outside of the

convex hull.



(a) Convex hull outside tile

(b) Convex hull within tile

Figure 6: Results when interpolating using both a buffer and no buffer

The input points for the GFTIN contain a 550x550-meter area. In the initial step of the Ground filtering with TIN refinement, a grid is superimposed on top of the point cloud, and for each cell, the lowest point in that cell is extracted. To make sure every lowest point is a ground point, the cell size must be larger than the tallest structure in the area. Otherwise, a cell may cover a structure entirely and therefore with certainty, no ground point is extracted for that cell. Employing a cell size larger than the tallest structure enhances the probability of capturing a true ground point in every cell. In our assigned tile, the tallest structure was approximately 23x23 meters. Therefore we chose to divide the 550x550 meter tile into 20 columns and 20 rows, resulting in a cell size of 27.5x27.5 meters. A grid with the same number of rows and columns is initialized and given a value of positive infinity for every cell, note this grid has a size of 500x500m since we only have to fill it and export it as the final DTM. Then we iterate over the points, check in which cell they belong and see if the z-value for the point is lower than the current lowest z-value. If this is the case, it replaces the z-value. The point is stored in a separate 2D array, where we keep track of all the lowest points using the CGAL Point3 class.

Having the Delaunay triangulation of the initial ground points, we take the rest of the cropped points and for every point check if two computations on the point fall below a certain threshold. The first computation is inserting the point into the Delaunay triangulation and calculating the height difference between the face it is located in and its height value. The second computation calculates the angle between all three vertices of the face the point falls into the point itself.

The thresholds we chose for extraction of the ground points are 1.5 meters for the height difference

computation and 7 degrees for the angle computation. At first, we picked slightly lower values, reasoning that a height difference of more than 1 meter also would include things like small benches, fences, etc. For the 7 degrees, we estimated the maximum gradient of the area to be around 5 degrees, so we initially picked a lower value. However, when running the algorithm and reviewing the output digital terrain model, some elements of the landscape were left behind and the overall result was too smooth. Thus, we opted for slightly higher values with 1.5 meters for the height difference and 7 degrees for the maximum angle, giving a better result.

After running the computations on all the points in the cropped tile, we added those falling within the threshold to a vector called groundPts and also inserted them into the Delaunay triangulation.

## 4.2 Creating of output dataset and Laplace interpolation to fill it

A DatasetASC object 'd' was created to make an empty grid, to fill with the values of the groundPts vector using Laplace interpolation. It was initialized with 1000x1000 cells having a size of 0.5 meters.

Then, the empty grid, groundPts vector, and the Delaunay triangulation were passed into the laplaceInterpolation function. The function iterates over all the cells individually, and inserts the cell center into the existing Delaunay triangulation, consisting of all the ground points.

Each cell center is interpolated by taking the weighted average of the adjacent vertices' heights in the Delaunay triangulation. The weight for each point is calculated by dividing the distance of the circumferences of the two incident triangles by the distance from the cell center to one of the vertices. This is done using a face circulator on the vertex cell center, to get the first triangle. To find the second triangle for the circumference calculation, we iterate over the face circulator until a face has both the cell center vertex and the adjacent vertex, for which we are currently computing the weight, in it.

Some vertices have a no-data value which would cause the weighted average of the vertices for the interpolation to have a no-data value, resulting in no-data values all over the output DTM. To tackle this problem, we check if the computed distance between the two circumcenters of the incident faces and the distances between the vertex cell center and adjacent vertices are not NaN and are bigger than 0. If this is not the case, the weighted height of that particular vertex is not taken into account for the final interpolation value. This gives an output DTM without any no-data values.

After the final calculation of the interpolation value for the cell center, we remove the cell center vertex from the Delaunay triangulation and assign its interpolated height value to the empty grid. For our designated tile, the output DTM can be seen in figure 7. Be aware, that the range in this figure legend is showing 129.15m - 185.37m, which is actually the range of the DTM plus the vegetation as seen in figure 15. The DTM itself does not span this whole range (129.15m - 167.42m), but this is done to keep the colors consistent between the figures

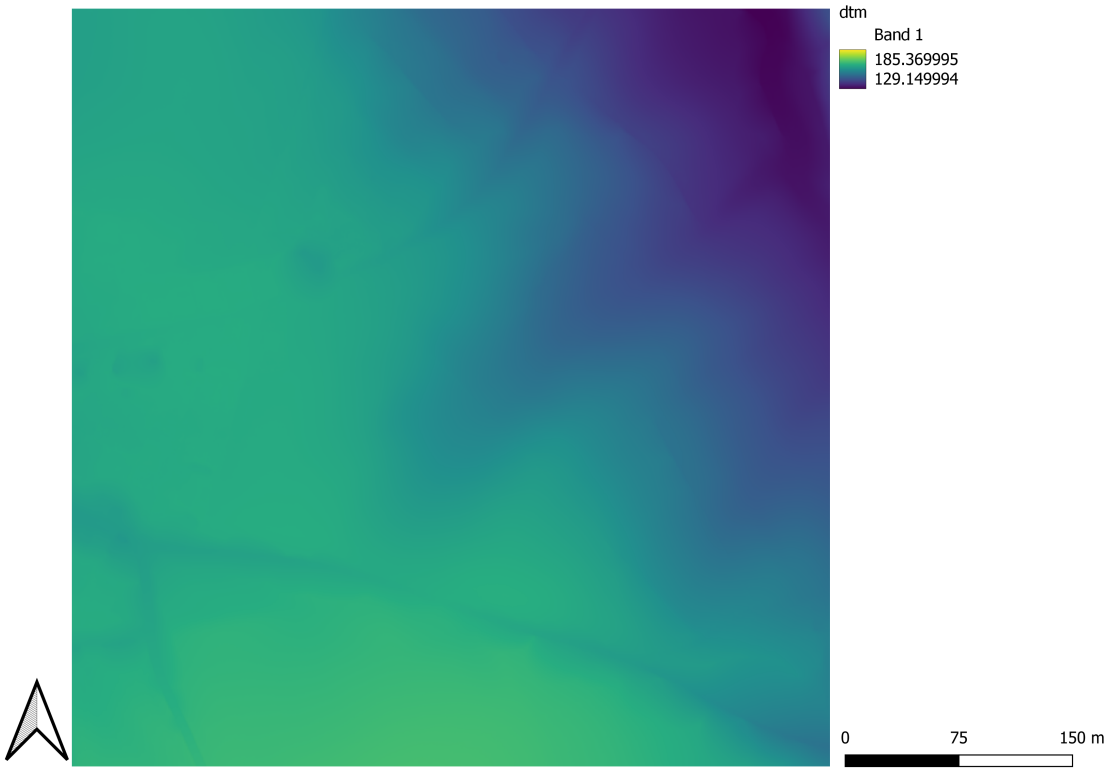



Figure 7: Output DTM after GFTIN and Laplace interpolation of designated tile

# 5 Extract Vegetation Points and Create Grid

## 5.1 Extracting vegetation

The points in the extracted AHN4 file have the following attributes, see Table 1.

| **Features** | | |
|---|---|---|
| X | Classification | Green |
| Y | Scan Angle Rank | Blue |
| Z | User Data | Infrared |
| Intensity | Point Source ID | Amplitude |
| Return Number | GPS time | Reflectance |
| Number of Returns | Scan channel | Deviation |
| Scan Direction Flag | Class Flags | |
| Edge of Flight Line | Red | |

Table 1: Features of the LAS points.

The classification of a point is determined by its function. A classification equal to 26 is a civil structure, 9 is water, 6 is a building, 2 is the ground, and 1 is unclassified. This means there is no class for vegetation; vegetation is classified as 1. However, road signs, cars, or benches in the park are also classified as 1. In this subchapter, the extraction of vegetation from the points classified as 1 will be discussed. In Figures Figure 8 the yellow dots show which points are for example classified as 1.

In this research, two methods are combined to classify vegetation. Subsection 5.1.1 describes method 1, which focuses on the number of returns. Subsection 5.1.2 describes method 2, which focuses on the NDVI.

Figure 8: Yellow dots represent points that are unclassified in the AHN4 dataset. The background is the visualization of the point cloud data.

### 5.1.1 Method 1: number of Returns

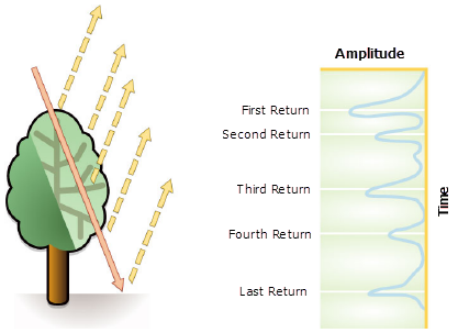



Figure 9: Visualization of the number of returns in a tree (Esri, 2020).

A pulse transmitted from the laser can be returned in one or multiple returns back to the scanner. If the pulse hits a hard object, there is just one return, and if the pulse hits a tree, there are at least two returns (de Groot, 2020). This makes it possible to see through the canopy of a forest (see Figure 9) and gives the ability to classify vegetation based on the number of returns.
This classification has been used for the used tile in this research. In Figure 10(a), the red dots show the points with a return number of more than 1. Yellow points are the unclassified points that have a single return. Tall trees are extracted as a result, but shrubs and small vegetation have a number of returns equal to 1. So, if only the number of returns is considered to extract vegetation, the small shrubs would not be included. Besides that, some single points or small groups of points are classified as vegetation. In some instances, small clusters of non-vegetation points in the point cloud data exhibit a number of returns equal to 2. This occurs, for instance, with a few points associated with a vehicle, whereas the vast majority of points related to cars have a single return.

(a) Red dots represent points with a return number of more than 1. Yellow dots represent points with a return number of 1. The background is the visualization of the point cloud data.

(b) Point cloud data, colors based on RGB bands.

Figure 10: Result of analyzing the number of returns for a specific area.

### 5.1.2 Method 2: NDVI

A widely used index when looking at vegetation is the normalized difference vegetation index (NDVI). NDVI is a key indicator for vegetation cover in a specific region derived from spaceborne sensor data (Hashim, Abd Latif, & Adnan, 2019). NDVI is calculated using the index of two spectral bands (B).

$$\text{NDVI} = \frac{B_{\text{NIR}} - B_{\text{RED}}}{B_{\text{NIR}} + B_{\text{RED}}}, \tag{1}$$

where $B_{\text{NIR}}$ represents the amount of near-infrared reflected by the object and $B_{\text{RED}}$ represents the amount of red light reflected by the object.
A point in the point cloud data can be classified as vegetation if its NDVI falls within a specific range. (Hashim et al., 2019) findings are presented in Table 2. This research states an NDVI between 0.2 and 1 can be classified as vegetation.

| Class | Description | NDVI Value |
|---|---|---|
| Non-Vegetation | Barren areas, built-up areas, road network | -1 to 0.199 |
| Low vegetation | Shrub and grassland | 0.2 to 0.5 |
| High vegetation | Temperate and Tropical urban forest | 0.501 to 1.0 |

Table 2: Thresholds for classes. Table copied from (Hashim et al., 2019).

The NDVI range [0.2, 1] has been applied to the AHN4 tile in this study, resulting in the classification of the blue points in Figure 11(a) as vegetation.

(a) Blue dots represent points with an NDVI between 0.2 and 1. Yellow dots represent points with a different NDVI value.

(b) Point cloud data, colors based on RGB bands.

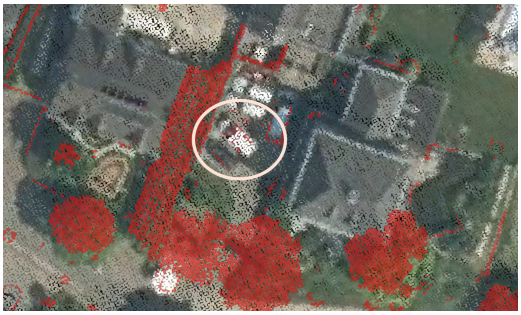
Figure 11: Result of analyzing NDVI for a specific area.

### 5.1.3 Combining Number of Returns and NDVI

In total method 1 (number of returns) classifies 585 287 points as vegetation. Method 2 (NDVI) classifies 608 808 points as vegetation. So with the NDVI almost 20 000 more points are classified as vegetation. Looking visually at the two methods (in QGIS, for example), it seems that the first method classifies more points as vegetation when it is not vegetation. See for example the points of

the car in Figure 12(a). The second method is accurate, but it also definitely misses some points and so that is where the first method complements the second method well. See the blue circle in Figure 12(b), not all vegetation is classified there as vegetation. Method 1 also struggles with this problem.

Thus, methods 1 and 2 can complement each other well. The result is still that not all vegetation points are classified as vegetation.



(a) The red dots represent the points that are classified as vegetation by Method 1. The red circle shows that even non-vegetation is sometimes classified as vegetation using this method.

(b) The blue dots represent the points that are classified as vegetation by Method 2. The blue circle shows that Method 2 also misses some points which are vegetation.

Figure 12: Two examples of missing elements in Method 1 and Method 2.

## 5.2 Rasterize

Now that the vegetation is filtered from the point-cloud the result can be rasterized for further use. To create a raster that contains the canopy height of an input area one starts by making a copy of the previously found DTM. Starting from an empty grid would be incorrect as the canopy height must be at least the ground height taken from the DTM, and not None or any other arbitrary no-data value.

Starting from the DTM, one now overwrites the values for each cell, if there are vegetation points. These should be higher than the DTM, as there are no sub-terrestrial bushes or trees. For every cell, the height of the points corresponding to this cell needs to be found. Going through each cell, finding the points that fall within its boundaries, and overwriting the cell value would be a time-consuming exercise, as one would need to iterate through all points for every cell. To reduce the time complexity of the code, the following approach was introduced.

### 5.2.1 Locating a point

From the point-cloud the X, Y and Z values of a point can be easily extracted. Because of unit conversion, these have to be divided by 100 before continuing. Taking the difference between the found X and the minimal X-value of the bounding box results in finding the length of the horizontal blue line in 13. dividing the length of the blue line by the cell-size will yield an x-index, i.e. the cell to be overwritten in the DTM. The same calculation can be repeated for the Y-values, with one extra step, the y-index value found through this method uses indexation from bottom to top, while

the code uses top to bottom. Therefore as the resulting grid will be 1000x1000, one subtracts the found y-index of 999 to adjust for this.

$$I_x = \frac{\left(\frac{X_{\text{point}}}{100} - X_{\text{min bbox}}\right) * 0.99999999}{\text{cell size}} \quad , \quad I_y = \text{grid size} - \frac{\left(\frac{Y_{\text{point}}}{100} - Y_{\text{min bbox}}\right) * 0.99999999}{\text{cell size}} \tag{2}$$

The nominators in both equations above are multiplied by $1 - 10^8 (= 0.99999999)$ to ensure that no value can ever exceed the maximum index. If a point is close to the maximum value within the bounding box, for the case of a 1000x1000 grid, the risk of an index of 1000 is possible. As the indexation starts at 0, this is out of bounds.
The results of 2 are floats and should be converted to integers using $int(I_x)$ and $int(I_y)$ respectively. Because of the multiplication of 0.99999999, results are automatically rounded down correctly. In the code, this is written to a virtual raster for further use in 5.2.2.

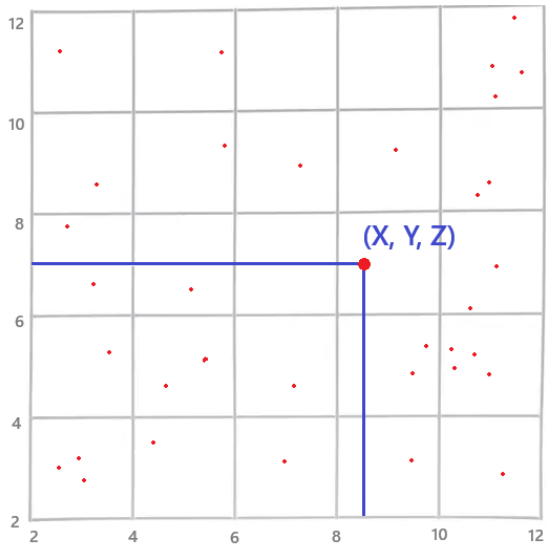
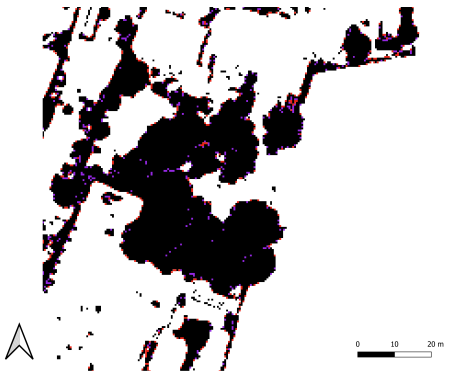



Figure 13: Finding the cell a point belongs to a point with values X, Y and Z. In this example, division by 100 is not necessary. The grid size is 5x5 with a cell size of 2x2m. Filling in equation 2 yields: $I_x = \text{int}((8.5 - 2) * 0.99999999/2) = 3$, $I_y = 5 - \text{int}((7 - 2) * 0.99999999/2) = 3$. The point belongs to cell (3, 2). (Starting the count at 0).

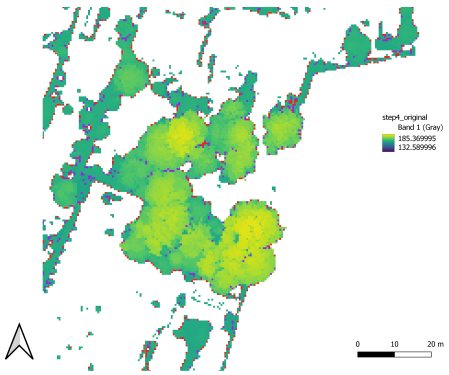
### 5.2.2 Smoothing the vegetation height values

The raster containing the maximum height of the vegetation points per pixel is rather discrete still. There are pixels within trees that are still NaN's, which will result in a coarse CHM. To counter this a mean filter with a 3x3 kernel was introduced, smoothing the vegetation raster. An unaltered 3x3 kernel would enlarge the vegetation areas and as this is an undesirable result, a parameter was included to decide how many neighbours must be also vegetation before a cell is smoothed. A high value will result in fewer cells considered for smoothing, and a low value will result in an increase of the vegetation area.

The difference between no filter, and a threshold of 4 or 5 neighboring pixels being also vegetation has been checked visually through QGIS. Both thresholds removed most NaN pixels within the regions of interest, but choosing for 4 neighbouring pixels did not fill more pits but mainly expanded the forest (see the red pixels in Figure 14). The team decided to go for smoothing if 5 or more of the neighbouring pixels are a vegetation cell.




(a) The black pixels are assigned as vegetation pixel, which mean they have a value higher than the DTM.




(b) The coloured pixels (range yellow to green) are assigned as vegetation pixels, which means they have a value higher than the DTM. In the legend, it is possible to see the height of the vegetation pixels.

Figure 14: These figures show what pixels (with a NaN value) change value by averaging over their neighbours. As a result, these pixels with a new value are considered part of the vegetation. If five or more of the eight neighbours do not have a NaN value, the pixel value of the purple pixels is changing. If four of the eight neighbours do not have a NaN value, the pixel value of the red pixels is changing.

### 5.2.3 Overwriting height value

The final step is to extract the height value from the smoothed vegetation raster and use the index to overwrite the height values of the DTM when the height value is in fact larger than the DTM height on that pixel. The latter part is to ensure erroneous values of Z will not be written to the canopy height grid.
Finally, the resulting canopy height grid is written to a .tiff file.

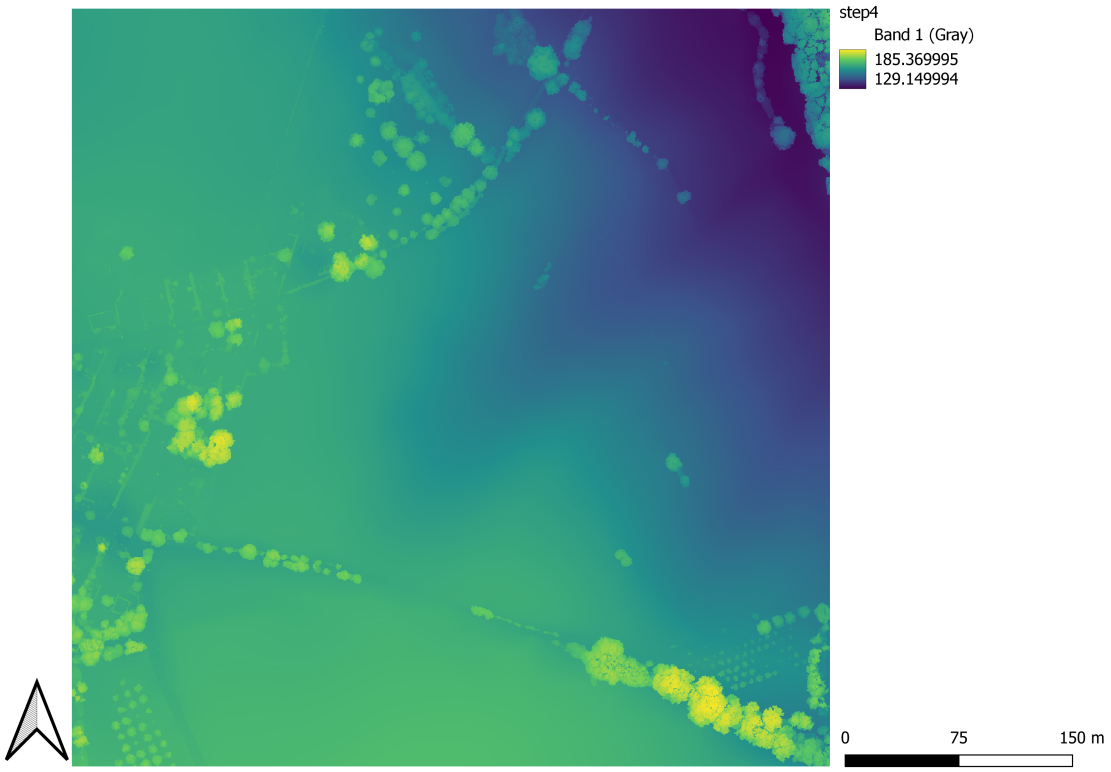



Figure 15: The calculated vegetation height of the area of interest.

# 6 Create the Canopy Height Model (CHM)

When all previous steps are performed, calculating the CHM is trivial. One subtracts the original DTM from chapter 4 of the canopy height grid calculated in chapter 5. The result of this subtraction is shown in figure 16.

When testing the generation of a CHM TIFF file using a different input file, an error was encountered indicating that the transformations of the two imported TIFF files were not equal. Table 3 illustrates the transformation matrices, with the left side representing the transformation of the "step4.tiff" file, and the right side representing the transformation of the "dtm.tiff" file. Despite visual similarity, a direct comparison using the '==' operator in Python is not advisable for floating-point numbers due to potential precision issues arising from floating-point arithmetic. Such direct equality checks may fail because of small differences in the internal representation of floating-point numbers. To address this, a small tolerance of $10^8$ was applied in the comparison process.

| | | | | | |
|---|---|---|---|---|---|
| 0.50 | 0.00 | 191072.99 | 0.50 | 0.00 | 191072.99 |
| 0.00 | -0.50 | 356817.93 | 0.00 | -0.50 | 356817.93 |
| 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 1.00 |

Table 3: Transformation matrices. On the left side the transformation matrix of "step4.tiff" and on the right side the transformation matrix of "dtm.tiff".

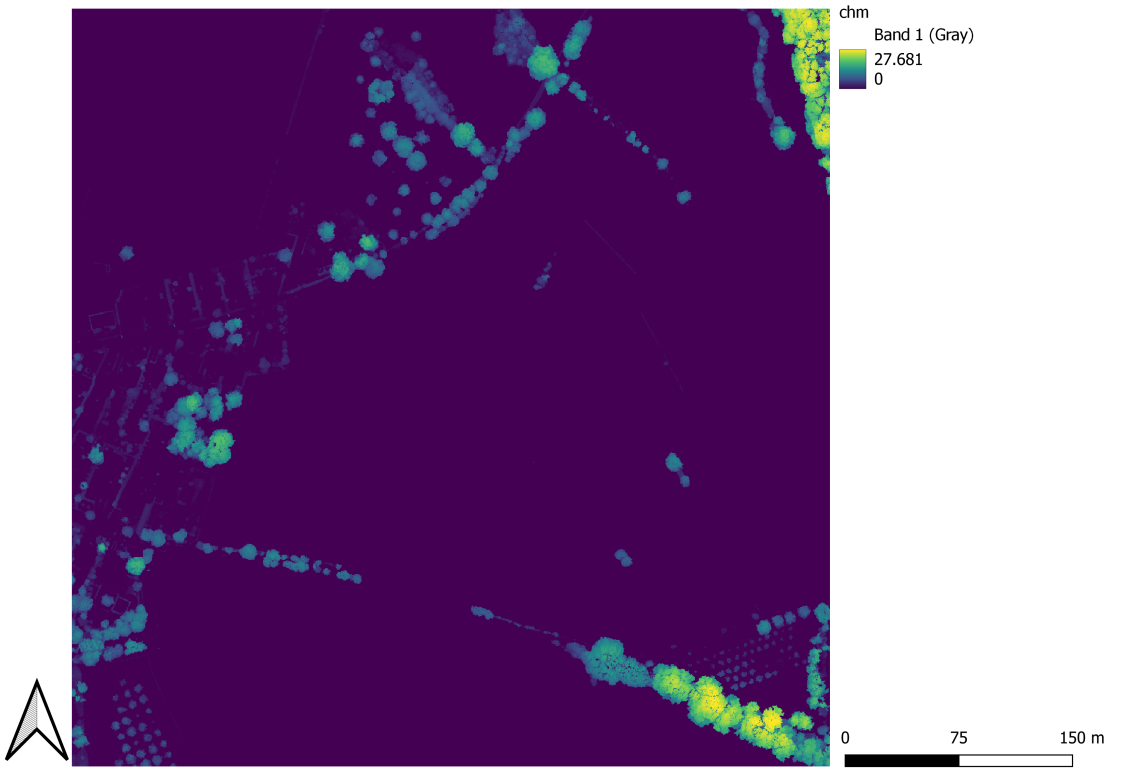



Figure 16: Canopy Height Model of the area of interest. Evidently, the areas that do not contain any vegetation show a value of zero. As seen from the legend, the highest tree has a height of 27.6 meters which is a reasonable height for a Dutch tree.

# 7 Discussion and Conclusions

In this report, a detailed process of creating a Canopy Height Model (CHM) from an AHN4 point cloud was described. The approach involved several steps, from ground extraction, gridding, vegetation point filtering, rasterization and lastly creation of the CHM image itself. Throughout this process, various challenges were faced and mostly overcome. A short reflection on these challenges, their solution and their downsides was considered appropriate. In Section 5.2.2 we discussed smoothing the vegetation raster to fill the pits. In Figure 16, it looks like there are still some single pixels with no value. That is not the case. These pixels have a low pixel value of, say, 0.4 meters above the ground.

The extracted CHM tiff file, of course, remains a model of reality. Many choices were made to create this file. This section critiques the decisions that were made.

Several improvements to the Laplace interpolation function could be made. However, since this assignment has already been quite a time-consuming learning process in C++, working with WSL and the cgal library we leave them for now as it is. To start off, iterating over all the faces of the cell center vertex to find the second incident triangle is not the most efficient, but the best that we could make it work for now. Secondly, ignoring the weighted value for points which have a no-data value works for the assigned area. However, there is a possibility that when working with different terrains, all the adjacent vertices for a certain cell center have no-data values and thus also the interpolated value will be no-data.

Additionally, the main.cpp file could benefit from the following improvements. When extracting the lowest points in the grid for the rudimentary Delaunay triangulation, two 2D vectors are created. One to store the Point3 objects and one for the height values. We think this can be improved by storing it in one 2D vector, storing the information of the Point3 object and the height value in the same vector, but for now, we were not able to make it work. For the size of this assignment, it was not a problem. Finally, when using the main.cpp file, you now need to know exactly where to change some variables for the bounding box, input dataset, output dataset and output name. This could be improved by encapsulating these parameters in a configuration file or through command-line arguments.

Two methods are used to extract the vegetation points from the point cloud data. As described in Section 5.1.3 not all points are classified correctly. Some vegetation points are not classified as vegetation and some non-vegetation points are classified as vegetation. The number of misclassified points is hard to say. Actually, this method should be applied to a dataset where the vegetation is already classified. Then the percentage of misclassification is easy to obtain.

In conclusion, the techniques and methods we employed demonstrated the effective use of point cloud data in terrain and vegetation analysis of a region. This report highlighted the potential for applying the described methods in numerous geomatics and environmental studies. Future work could focus on refining these techniques and algorithms for more diverse datasets and filtering the vegetation points to a more precise extent. This research provided a rather elaborate academic understanding of digital terrain modelling and provides a solid framework for similar future projects our team will encounter in future projects during the Geomatics Master program and beyond.

# 8 Workload division

| | Python: | C++: | Report |
|---|---|---|---|
| Step 2 | Marieke | Lars | Lars, Marieke |
| Step 3 | | Lars | Lars |
| Step 4 | Noah, Marieke | | Noah, Marieke |
| Step 5 | Noah | | Noah, Marieke |
| README.md | Noah, Marieke | Lars | |
| Introduction | | | Lars |
| Conclusion & Discussion | | | Noah, Marieke, Lars |

# References


de Groot, R. (2020). Automatic construction of 3d tree models in multiple levels of detail from airborne lidar data.

Esri. (2020). *What is lidar data?* Retrieved from `https://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/what-is-lidar-data-.htm`

Hashim, H., Abd Latif, Z., & Adnan, N. A. (2019). Urban vegetation classification with ndvi threshold value method with very high resolution (vhr) pleiades imagery. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *42*, 237–240.