Report: The Sinking Netherlands

Jeren Olsen Kailhan Hokstam Reinier van Elderen

1. INTRODUCTION

1.1 Problem Motivation & Definition

As about one third of the Netherlands lies below sea level[2] and climate change accelerates soil subsidence[5], it is important to monitor which specific areas of the Netherlands are currently sinking and potentially take additional preventive measures. These measures can, for example, be related to improving water management and helping with urban planning. In order to identify these areas, we utilize publicly available data of the Dutch terrain taken over two time periods: 2007-2012 and 2014-2019. More specifically, we analyze terabytes of point cloud data obtained using LiDAR sensors from the *Actueel Hoogbestand Nederland* (AHN[1])'s two datasets `AHN2` and `AHN3` to create two height maps, which can be interactively inspected in a 3D visualization. There are various challenges associated with the analysis of these large datasets that requires large scale data engineering techniques to be applied in order to process the input data and arrive at an output data product that can be used in a visualization. In the following sections, we will first describe related work, which allows us to determine our approach and formulate our research questions. Subsequently, we delve deeper into the input data, data pipeline, output data product and visualization. Finally, we answer our research question and list the contributions of our group members.

1.2 Related Work

1.2.1 Data Limitations and Previous Analyses of AHN2 and AHN3

As the measurements for the `AHN2` dataset have been completed relatively recently, namely in 2017, there is not an extensive body of previous works where `AHN2` and `AHN3` were used to indicate elevation changes over the entire Netherlands. However, a 2017 thesis project conducted at the University of Wageningen provides the most comprehensive assessment to date in the differences between the `AHN2` and `AHN2` datasets. The thesis project also provides several insights in the quality of the datasets that can help us interpret our results when answering our own research questions [6]. The key takeaways from this thesis project indicate that the `AHN2` dataset possesses considerable instrumental artifacts acquired during data collection, such as elevation change patterns corresponding to helicopter flight paths. The `AHN2` dataset was shown to deviate significantly from measurements of elevations of terrain gathered from particular study areas while `AHN3` was more reflective of actual measurements of elevations of terrain. While the elevation differences of many regions can be attributed to instrumental variation, there were also some notable changes in landscapes attributed to weathering and farming activities. These included sunken dunes in the West Frisian Islands and risen fields in Oostkapelle. We will use these findings as a basis of comparison in our own research. Overall, it was concluded that subtracting `AHN2` heights from `AHN3` in order to determine differences in elevation levels of terrain is unlikely to yield satisfactory results [6]. Additionally, per the `AHN2` Quality Document, AHN data carries a maximum systematic error and maximum random error of 5 centimeters, making it unreliable in determining elevation changes of only a few centimeters [4]. It is important to take these findings and limitations into account when interpreting our own results.

1.2.2 Height map

There are many height (or elevation) map visualizations of the `AHN2` and `AHN3` datasets available on the internet such as the point-cloud viewer at ahn2.pointclouds.nl. However, we were unable to find any examples of height maps comparing the two datasets or using a gridding pattern that would lend itself well to the point sampling approach we wanted to take. In addition, due to the resource constraints for our final visualizations, we needed to create something less detailed than the point cloud based height maps. Therefore, we moved forward with the intention of creating a novel height map tailored to our resource constraints and research question.

1.2.3 Visualization

As part of this project, we have created a visualization based on height maps that can be used to gain insight into the elevation of the terrain of The Netherlands. Various approaches exist to visualize height maps. A first simple approach is transforming the cells of the height map to a gray-scale image. In this case each pixel represents a cell and the intensity of a pixel can represent the height of the cell. A benefit of this approach is that an image can be generated ahead of time. Afterwards, this image only needs to be transmitted from the server to the client and displayed by the client therefore the client has a relatively small amount of work to do. In our case, the height map data comes from terrains with three spatial dimensions and we think

[1]https://www.ahn.nl/

it could be more intuitive and informative to also visualize this data in three spatial dimensions. A comparison of the user experience with respect to how intuitive and informative a visualization in two compared to three spatial dimensions is, is left outside of the scope of this project. Related work that deals with rendering large amounts of data in three dimensions is `Potree`. `Potree` is a WebGL based point cloud renderer for large point clouds[3]. WebGL allows web programmers to create interactive 3D graphics inside web browsers[1]. `Potree` specifically uses `Three.js` as the library to use WebGL to interact with the browser[2]. While we could represent the height map as individual points and use `Potree` to render these points, we decided against this which will be further in explained in Sec. 2.3. However, because of the level of detail that `Potree` is able to render with `Three.js`, we decided to also use this it to build a visualization.

1.3 Research Questions

This leads us to the following research questions:

1. Can we apply a point sampling pipeline to the `AHN2` and `AHN3` datasets to get a comprehensive overview of sinking regions of the Netherlands?

2. For which number of cells for a height map could an interactive 3D visualization while maintaining 60 frames per second be rendered?

2. PROJECT SETUP

In this section, the input data, the implemented data pipeline, output data and, lastly, the visualization are described and related decisions motivated.

2.1 Data Investigation

The two datasets, `AHN2` and `AHN3`, contain 3D points clouds in `LAZ` files, a compressed version of the readily readable `LAS` format which is typically 10x larger in storage size. The points were obtained using LiDAR technology, projecting lasers from helicopters onto the Dutch land. The measurements for `AHN2` were taken between 2007 and 2012 and are shown in Fig. 1. The measurements for `AHN3` were made between 2014 and 2019 and are shown in Fig. 2. The difference in time of measurement between a point from `AHN2` and a point from `AHN3` can vary between 5 and 10 years. The point clouds in each dataset have point densities between 6 and 10 points per square meter, with `AHN3` having a higher density than `AHN2`. Each `LAS` file contains millions of points, each with an X, Y, and Z coordinate, classification, scan angle, GPS time and other dimensions. It is important to note that all points in the `AHN2` dataset possess a classification of 0, making it impossible to filter the `AHN2` dataset based on point type (ie. ground, water, vegetation, building) without additional processing and potentially machine learning. In contrast, the majority of `AHN3` points possess classifications that are not all 0. Each file also contains a header with information such as total point count and coordinate ranges that can be read without reading all points in the file.

The `AHN3` dataset consists of 1375 files totalling 2568 GB of compressed point cloud data. Each file corresponds to a 5 x 6.25 km rectangle of land within the borders of the Netherlands. The `AHN2` dataset contains 41438 smaller files totalling 1747 GB of point cloud data. During initial data investigations, due to the larger size of the `AHN3` files, it was observed that reading in entire files would occasionally overload the memory. Upon further inspection of the documentation of the `laspy` Python library, we overcame this issue by reading each `LAZ` file in chunks of points using a chunk iterator.

Figure 1: `AHN2` Measurement years

Figure 2: `AHN3` Measurement years

2.2 Height Map Pipeline

The two data sets cannot be compared directly since for a point with specific X and Y coordinates in one dataset we cannot necessarily find a point with the same X and Y

[2]https://threejs.org/

coordinates in the other dataset. To be able to compare the two data sets the data has to be turned into a format that is equal for AHN2 and AHN3. To achieve this the map will be divided into grids that are equal for both datasets.

Our data pipeline is can be run in parallel for the files in each of the datasets. This is done since the files in each of the datasets already consist of equal sized areas. This means grid cells can be created within these files without misaligned borders as long as the grid size divides the length and width of the file area.

2.2.1 Sampling

The first step of our data pipeline is sampling the points. LAZ files can be quite large and are much larger decompressed. When reading some of the files this would fill the memory and crash the Python process. To prevent this the files are read in chunks using the `laspy chunk_iterator` function. With this function we read one million points at a time. The next step is to sample a number of points from a file. Since we know how large the area is of each file we divide the number of points by the area size to know how many points a file contains per square meter. With this we then sample approximately one point per hundred square meters from a file. This number is chosen to keep the time it takes to calculate the grid cell values reasonable.

2.2.2 Gridding

The AHN2 and AHN3 use a global coordinate system in meters which is equal for both data sets. Using these coordinates the two data sets can be mapped into grid cells. We have chosen to divide the map into fifty by fifty square meter grid cells. This would result in approximately seventeen million grid cells assuming each file has values for each grid cell. With the sampling of one point per hundred square meters this results in a twenty-five point sample per grid. To create the grid cells the coordinate values of a point are divided by the chosen grid size and floored to an integer value. This results in a list of height values with an X and Y grid cell coordinate.

The data sets contains some outliers of faulty data points. To remove these points at this stage for each grid cell sample the points the average is taken and the points that are larger than 3 times the standard deviation are removed. After this the mean is taken and stored as the height value for this grid cell. All files of a dataset are then combined into one large list of grid cell coordinates with height values.

Finally, the two lists of grid cells of the two datasets can be compared to get the difference between the two datasets.

2.2.3 Computation time

In the first few weeks a lot of computation time was used to analyse the data and test different methods for the pipeline. The pipeline took roughly 12 hours to run on the AHN2 dataset on the shared cluster overnight. For AHN3 this was a different story, with this dataset worker nodes crashed regularly. This made it difficult to exactly time the computation. It took roughly 3 days including some time between restarting the pipeline. These issues are most likely caused by the size of the AHN3 files. Finally, transferring the data to the format for the visualisation took 7 hours on the shared cluster.

2.3 Visualization

As previously discussed, we used `Three.js` as the library to use WebGL to interact with the browser. At a high-level, this library allows a developer to instantiate a camera that uses perspective projection, of which the position and rotation can be controlled by user input, and instantiate meshes in a scene that will be rendered. Specifically, as we are trying to draw as many vertices as possible, we make use of the `BufferGeometry`[3] class for defining what will be rendered. Using `BufferGeometries` we can store meshes and shaders within preallocated buffers and pass these to the GPU. This way of passing data to the GPU is more efficient compared to using higher-level `Three.js` API's like a `BoxGeometry` at the cost of having to calculate vertex positions for the meshes ourselves and manage the buffers.

The meshes rendered are cuboids. We decided on rendering cuboids instead of points as we expected these to look better at any mesh density and be more intuitive to interpret as an approximation of terrain certainly in conjunction with the sea level simulation which will be discussed later. Using cuboids means that we need a width, length and a height value for the dimensions of the cuboid and a X and Y coordinate for the position of the cuboid. These values are obtained from the height map. Ideally, a cuboid could be drawn for every cell in the height map, as this allows for the most detail in the visualization, however from experimentation this does not allow us to maintain 60 frames per second. Therefore, we sample the height map in a grid-like pattern, where we specify the number of columns and rows this grid has. The number of columns and rows combined with the bounds of the height map determines the width and length of the cuboids and the height value from the height map naturally translates to the height of the cuboid. The specific column and row of a grid point determines the X and Y coordinates of the cuboid. This number of columns and rows can be thought of as a sampling resolution. The higher the number of columns and columns, the higher resolution meshes we obtain and the more detail is visible.

After we know the height, width, length, X, and Y coordinates of the cuboids, we translate this information to vertex positions corresponding to triangles that a GPU can draw. Based on the grid we have created before, we group a set number of neightbouring grid points together. This group is called a 'chunk' and we give it an unique identifier based on the column and row number of the first grid point of this group. The vertex positions are then stored in `JSON` files. Additionally, as not all points in a chunk in the grid have data associated, e.g. if (all) points in the chunk are in the ocean, we keep track of the number of vertex positions and the minimum and maximum X, Y, and height values per chunk and overall. We store this data in a metadata `JSON` file. When a browser client visits our visualization it can therefore know which chunks it can and will need to fetch as well as adapt the default scaling of `Three.js` to that of the chunks. As fetching all chunks at once overloads the client from our experimentation, we only fetch at most ten chunks per frame.

The height value of a cuboid can be quite hard to understand if viewed from directly above the cuboid. To help the user's understanding of height values we implemented a simple shader that colors pixels using a mix of two colors with different ratios depending on the height of the pixel in

[3] https://threejs.org/docs/#api/en/core/BufferGeometry

the cuboid. The higher the pixel, the lighter the color is and we have picked contrasting colors for `AHN2` and `AHN3` to make it easier to distinguish between them.

Figure 3: Overview of visualization, top-down

Because we want a user of our visualization to be able to get an idea of changes in elevation for the whole of the Netherlands, but also be able to have a detailed view of a specific area if it is deemed interesting, while also maintaining 60 frames per second, we have implemented different levels of granularities per chunk. We also refer to this as dynamic zooming. This means that a part of the visualization, a set of chunks, has a higher sampling resolution than another part. In the current iteration of the project, the base resolutions is 2^{14} cuboids and the high resolution is 2^{20} cuboids. It is relevant to note here that we are always rendering two sets of cuboids: one for `AHN2` and one for `AHN3`. This difference in number of cuboids practically means that 1 cuboid in the high resolution is 64 times more detailed than a 1 cuboid in the base resolution. The other side to this is that 1 cuboids in the base resolution can be obtained by averaging 64 high resolution cuboids. Therefore, by choosing specific sampling resolutions and specific numbers of points that end up in a chunk, we can have the identifiers for the chunks be the same, which is relevant for dynamically switching between granularities per chunk.

Switching between the base resolution and higher resolution can be done by updating a simple array. As previously discussed, we use `BufferGeometries`. To obtain the most performance out of these, it is important to declare an array of a fixed size and pass it to a `BufferGeometry` and then only modify the values within the array e.g. not resizing it. An important observation here is that arrays in JavaScript get initialized to 0 and the number of 0's in the array seemed to have a negligible effect on performance compared to the number of draw-able triangles within the array. Following this, we can assign a fixed part of the simple array to a chunk identifier, based on the number of vertex positions for that chunk identifier for the higher granularity as stored in the previously mentioned metadata files. If we are rendering the higher granularity version of the chunk identifier we fill up the fixed part of the simple array with the corresponding values. When switching to the lower granularity version of the chunk identifier we fill the fixed part of the simple array up, up until we have have corresponding values, after which we can fill the rest of the part of the simple array assigned to the chunk identifier with 0's.

The chunks that have a higher sampling resolution are determined by a condition on their position relative to the current position of the camera, e.g. the user. This condition consists of two parts. First, we check if the camera is not too high, if so, no chunks are rendered in the high resolution. Second, we check for the distance in the X and Y plane. If the position of the chunk is within a certain radius, it will use the higher sampling resolution. This certain radius is primarily determined by the average number of frames over the last 50 frames. If the average is higher than 60 frames per second the radius will increase, if it is lower it will decrease. We also set a minimum and maximum range. This minimum ensures some detail can always be seen, while the maximum prevents the radius getting significantly too large, which can happen when focus is lost in which case performance will drop significantly when focus is regained.

Figure 4: High resolution overview of visualization, side-view

Finally, we implemented a slider that controls the height of a simple plane representing the sea level. The slider shows the height of the plane and changing the height shows which parts of the Netherlands would be under water given that height. No further 'flooding' logic has been implemented. This can be seen in Fig. 5.

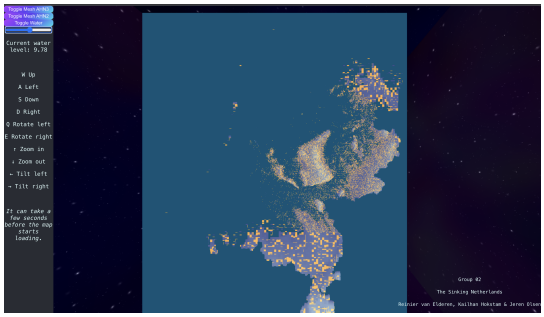

Figure 5: Sea level

2.4 Data Pipeline

Besides the visualization, the above has been implemented using Databricks notebooks with the 10.4 LTS runtime. In Fig. 6 a high-level overview of our data pipeline is shown.

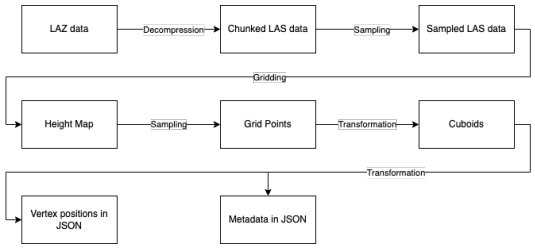

Figure 6: Overview of data pipeline

2.5 Output Data Product

The final data product consists of three datasets, two for the height maps and one containing the difference in height between the datasets. The `AHN3` grid list contains 14211438 points and the `AHN2` contains 14230067 points. The difference of height map contains 14026974 values meaning there where that many equal grids in `AHN2` and `AHN3`. The data still contains some inconsistencies since the difference of height values range from -114 to 368. It is relevant to note here that this these height values are already averages from multiple data points and the original data thus had more extreme values.

Fig. 7 illustrates the overall tiling results that are the output of our data pipeline. In general, it is noted that the colored indications of risen and sunken tiles are noisy. However, some areas do have distinct characteristics, such as the center of the Netherlands, the national forest, being particularly blue, while most coastal islands and dunes are red. The center of the Netherlands is likely related to either

growth of vegetation between the two time periods or that the AHN2 version of this was recorded during the winter when the trees had less volume.

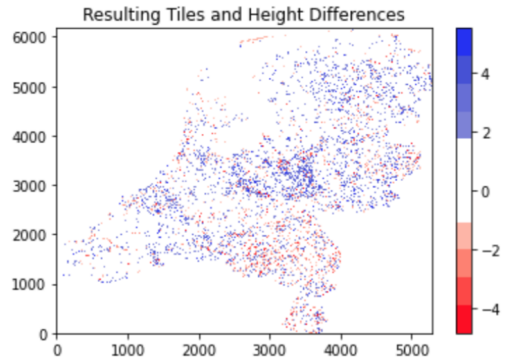

Figure 7: Overall Tiling Results. Height changes given in meters.

As can be seen in Fig. 8 and Fig. 9, our model was able to reflect the reality that the sand dunes in the West Frisian Islands have changed significantly between the collection dates of the AHN2 and AHN3 datasets. This phenomenon was also observed in the thesis project conducted at Wageningen University in 2017 [6].

Further, it appears that it is predominantly the northmost shores of these islands which have sunken. This can also be visualized using our 3D model in Fig. 10 where, saving for the outliers, the outer shores of the Islands (AHN2, blue) have sunken while inner sections (AHN3, orange) seem to have risen.

Another region of interest highlighted by Meijeren's work lies in Oostkapelle. Here it was speculated that farmers had been adding new soil to their land to even it out, causing their fields to have risen [6]. This also appears to be reflected by the results of our sampling pipeline and can be seen in Fig 11, where much of the land has risen by 0.25 to 1 meter. Both illustrations in this figure also highlight some of the height changes that are due to instrumental variation rather than changes in elevations of the terrain. This is apparent in the diagonal striations seen in both illustrations. When looking for these same height characteristics in the 3D visualization, it becomes more challenging to make a comparison as it's difficult to determine the exact regions present in Fig. 11.

3. CONCLUSIONS

With regard to our first research question, taking the difference between AHN2 and AHN3 datasets can indeed roughly reflect areas of the Netherlands that have experienced significant geographical changes. However, are large sections of the Netherlands sinking? It is not possible to make a comprehensive assessment of this with our approach, though it is apparent that particular regions have certainly experienced sinking such as the West Frisian Islands. To improve upon our model's ability to identify sinking regions, it would be beneficial to utilize points of the same class, such as ground, and to increase our point sampling density.

For our second research question, an answer is that it is possible on a 2021 MacBook Pro 14" M1 Pro with 16GB of RAM to render the high resolution map in full, while having more than 60 frames per second. In this case 458334 cuboids are rendered for AHN2 and 461433 for AHN3. Given that The Netherlands has a land area of approximately $33670km^2$[4], a single cuboids represent approximately $0.073km^2$. However, on a 2015 MacBook Pro 13" 2.7 GHz i5 with Intel Iris Graphics 1600 1536 MB GPU and 8GB of DDR3 RAM, 60 frames per second could not be maintained on the base resolution map. Additionally, on the 2021 MBP when rendering the visualization with dynamic zooming enabled and constantly moving around, leading to different chunks being rendered in higher resolution every frame, only approximately 19000 cuboids could be rendered, while having more than 60 frames per second.

The results for the second research question show that our implementation of dynamic zooming can introduce a performance hit so large that better performance can be obtained by rendering all of the high resolution chunks compared to rendering some base resolution chunks and some high resolution chunks when using dynamic zooming. Enabling dynamic zooming requires the browser client to perform extra work. This work can be separated in network-related and GPU-related. The network-related work involves fetching chunks (which are recognized as not modified and therefore do not need to be re-transmitted) and parsing text as JSON. It might be possible to transmit data in another format that can be more efficiently passed to the GPU. `Three.js` currently does not seem to support this in any of its high-level APIs and this would thus require future research to look into more directly interfacing with WebGL. The GPU-related work involves updating the specific buffers for AHN2 and AHN2 that are passed to the GPU and then rendering these buffers. It is potentially important to note that a specific `needsUpdate` flag for the buffer needs to be set for the updates to the buffer to be visible. We hypothesize that because there is only a single buffer for each of the datasets, every time there is an update to a part of this buffer, the whole buffer is passed to the GPU and the whole buffer is re-rendered. This re-rendering could be quite expensive and future research could look at amortizing the cost of re-rendering by not re-rendering every time there is a single update to the buffer. This would, as an example, be possible by only setting the `needsUpdate` flag after a certain number of updates to the buffer. Another option that can be looked into is creating separate buffers per chunk of the data instead of a single buffer for all data, so that only smaller meshes would need to be re-rendered.

Another observation with regards to the results for research question 2, is that the resolution a user can zoom again would need to be extended if the visualization is to render at 60 frames per second on more devices. The code for this project could be extended to support this.

[4]https://data.worldbank.org/indicator/AG.LND.TOTL.K2?locations=NL

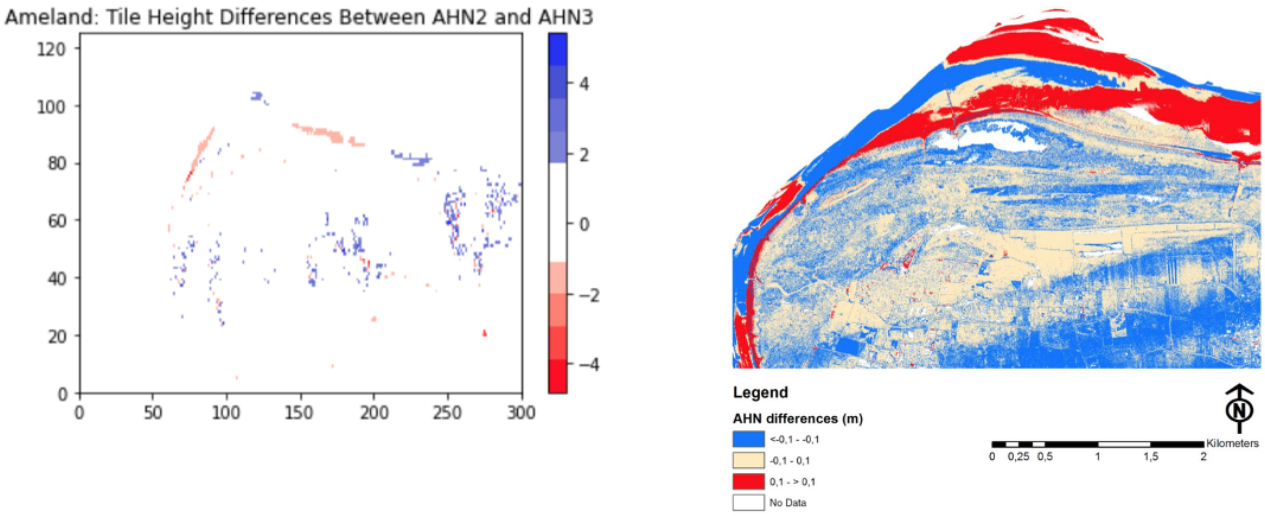

Figure 8: Ameland Region - Comparing our results to those of H. van Meijeren (right). Height difference results from our pipeline are visualized on the left.

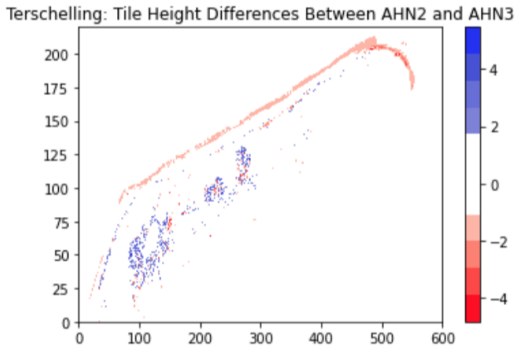

Figure 9: Height differences between AHN2 and AHN3 for Terschelling island region.

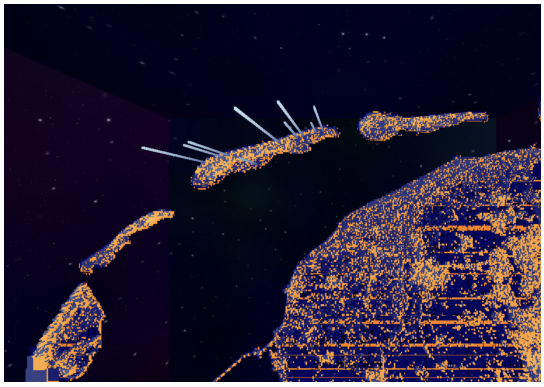

Figure 10: High resolution visualization of the West Frisian Islands using 3D model

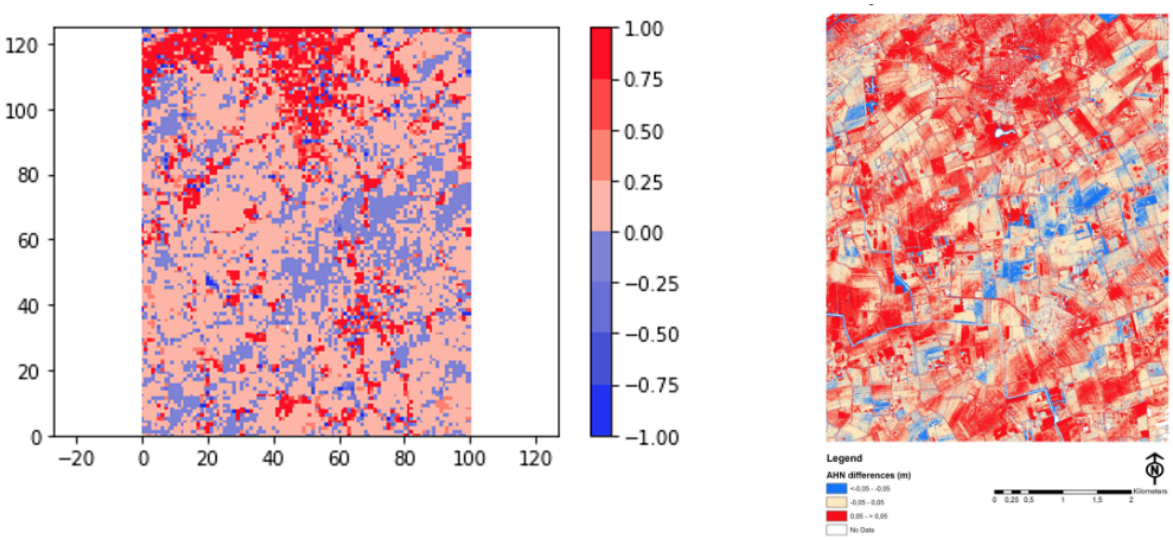

Figure 11: Oostkapelle Region - Comparing our results to those of H. van Meijeren (right). Height differences results from our pipeline are visualized on the left.

4. CONTRIBUTIONS

Task	Jeren	Kailhan	Reinier
Data investigation	33%	33%	33%
Data pipeline	45%	10%	45%
Visualisation	0%	100%	0%
Report	33%	33%	33%

Table 1: Contributions

5. REFERENCES

[1] K. Matsuda and R. Lea. *WebGL programming guide: interactive 3D graphics programming with WebGL*. Addison-Wesley, 2013.

[2] Q. Schiermeier. Few fishy facts found in climate report: Dutch investigation supports key warnings from the ipcc's most recent assessment. *Nature*, 466(7303):170–171, 2010.

[3] M. Schütz et al. Potree: Rendering large point clouds in web browsers. *Technische Universität Wien, Wiedeń*, 2016.

[4] N. Van der Zon. Kwaliteitsdocument ahn2. *Delft: Rijkswaterstaat*, 2013.

[5] T. van Dijk. The netherlands is sinking fast, Nov 2018.

[6] H. van Meijeren. *Assessing the differences between Dutch elevation datasets AHN2 and AHN3*. PhD thesis, MSc thesis, 2017.