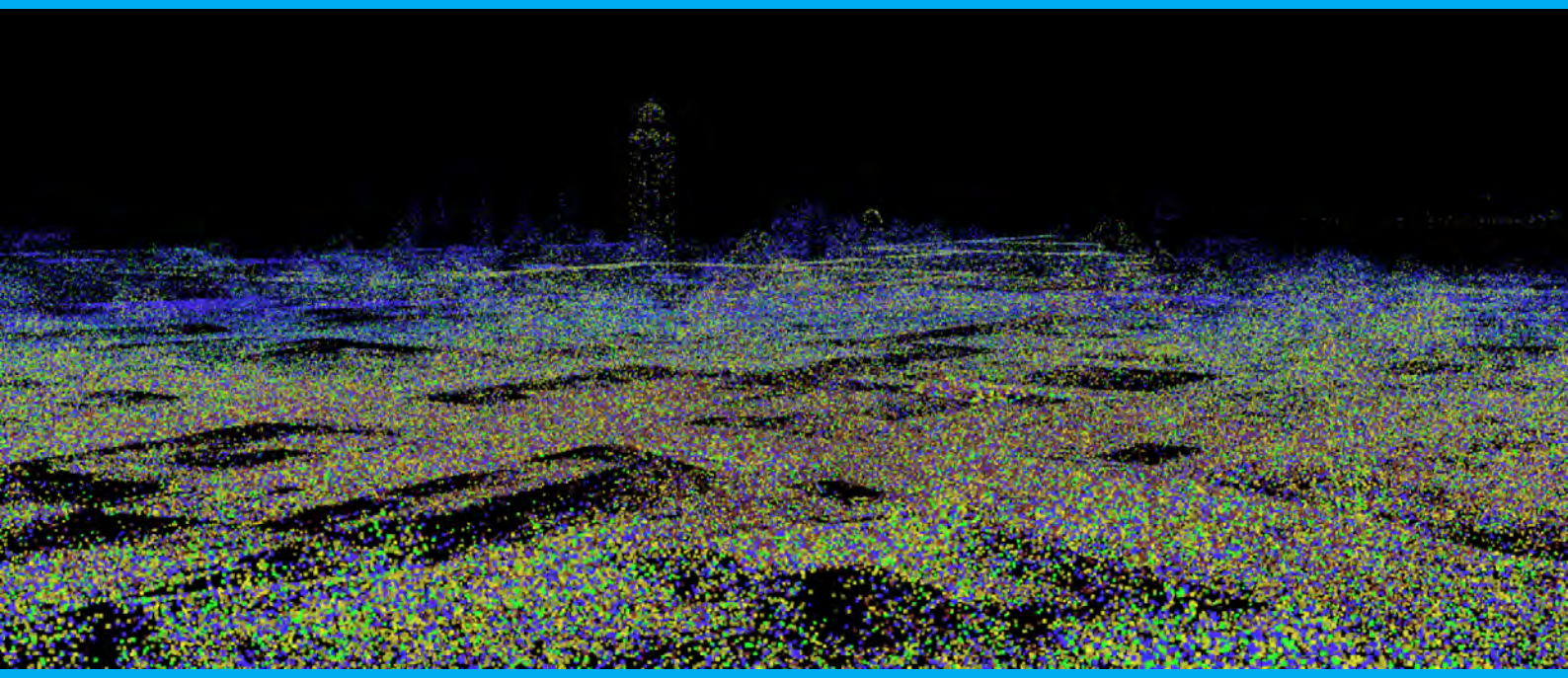
Technical Report

A vario-scale approach that improves integration of point clouds with different point densities

T. Hemmes
W. Li
J. van der Maaden
B. Olsen
M. Veenendaal



TU Delft

Technical Report

A vario-scale approach that improves integration of point clouds with different point densities

by

Tom Hemmes	4161726
Weiran Li	4575717
Jippe van der Maaden	4156617
Brenda Olsen	4149556
Marc-Julien Veenendaal	1241966

in partial fulfilment of the requirements for the degree of

Master of Science
in Geomatics

at the Delft University of Technology,
presented on Friday June 23th, 2017 at 09:20 AM.

Project committee:	Stefan van der Spek	TU Delft, course coordinator
	Peter van Oosterom	TU Delft, project coordinator
	Martijn Meijers	TU Delft, project coach
	Theo Tijssen	TU Delft, project coach

TU Delft Delft University of Technology

Preface

This report is written for the Geomatics Synthesis Project 2017, which is part of the Geomatics master at the Delft University of Technology. During this project, three teams worked on different topics in the field of point clouds. These topics included: scale, time and location. Our team, called GRIND (GRanular INtegration of Data), did research on the integration of point clouds with different scales and granularity.

We would like to thank our supervisors Martijn Meijers and Theo Tijssen for their support and guidance during this project. We would also like to thank our contact persons at Fugro, Martin Kodde and Stella Psomadaki, for sharing their expert knowledge and data with us. This project would not have succeeded without their support.

Tom Hemmes
Weiran Li
Jippe van der Maaden
Brenda Olsen
Marc-Julien Veenendaal
Delft, June 2017

Contents

List of Tables

List of Abbreviations

AHN	Actueel Hoogtebestand Nederland
BRIN	Block Range Index
CSS	Cascading Style Sheets
DBMS	Database Management System
fov	Field Of View
GiST	Generalised Search Tree
GRIND	Granular Integration of Data
HTML	Hypertext Markup Language
LOD	Level Of Detail
PDAL	Point Data Abstraction Library
PHP	Hypertext Preprocessor
pov	Point Of View
SSC	Space Scale Cube
XML	Extensible Markup Language

Executive Summary

The executive summary is written as a conference paper and can be found on the following pages.

A vario-scale approach that improves integration of point clouds with varying point densities

Tom Hemmes
T.Hemmes
@student.tudelft.nl

Weiran Li
W.Li-7
@student.tudelft.nl

Jippe van der Maaden
J.D.N.vanderMaaden
@student.tudelft.nl

Brenda Olsen
B.Olsen
@student.tudelft.nl

Marc-Julien Veenendaal
M.J.Veenendaal
@student.tudelft.nl

ABSTRACT

Point clouds are becoming one of the most common ways to represent geographical data. The scale of acquisition of point clouds is growing steadily. However, point clouds are often very large in storage size and require computationally intensive operations. The integration of point clouds nowadays still face a lot of challenges. This project focuses on one of these challenges; integrating point clouds of different scales and granularity. Solving this challenge enables appealing visualisation, usability for low and high computation powers and geometrical consistency for analysis. The following question is researched: 'To what extent can a vario-scale approach improve integration of point clouds with varying point densities?'. A data model is created that uses importance as an additional dimension. This dimension contains an importance value which is calculated using two methods. Firstly random assignment of values to the points and secondly exact computed values. To compute this value the smallest distances to its nearest neighbour is assigned as importance value. A web application shows the results. Both random and exact methods show an exponential decay in distribution of the importance value. Though the random methods run much faster, the exact methods preserve much more edges and other details.

KEYWORDS: Point cloud, vario-scale, point density, scale, granularity

1. INTRODUCTION

This paper is written for the Geomatics Synthesis Project 2017, which is part of the Geomatics master at Delft University of Technology. This project focuses on three different topics within the field of point clouds. In a joint effort, Fugro and TUDelft want to create an *Open Point Cloud Map*. Our group GRIND (GRanular INtegration of Data) researches the challenges related to scale & granularity of point clouds. Other groups focus on the topics time & location. The efforts of these three topics contribute to the implementation of this platform. A problem that arises with current massive point cloud viewers is the "block effect" where there is a clearly visible difference between LoD (Level of Detail). We research to what extent a vario-scale approach can improve integration of point clouds with varying point densities. The subtopics are the investigation of the best suited approach and the needed processing steps to create one overall structure. Furthermore we test the performance of our proposal. This is done within the time-span of 9 weeks of the Geomatics Synthesis project. Our proposal covers storage, processing, visualisation and recommendations on indexing. The results are shown in a web application, which can be found on `https://github.com/openpointcloudmap/GRIND`.

2. RELATED WORK

2.1 Point Cloud Server

Because of the storage size and simplicity most point cloud systems still rely on file based storage. In a benchmark, [6] shows not only the feasibility, but also the benefits regarding functionality of storing point clouds in a *Database Management System* (DBMS). Also [2] emphasises the added value of a flexible point cloud server, to realise such a system they use the point cloud extension for *PostgreSQL*. This extension, dubbed [5], proposes an approach of managing groups of points. Instead of managing individual points, this approach allows to profit more of the advantages a (Relational-) DBMS offers. Dealing with individual points would cause for great numbers of entries in the tables, which in turn would result in slow query results. [1] proposes in their paper a point cloud management system fully based on [5] and other open source tools.

2.2 Vario-Scale

The vario-scale structure with a non-discrete LoD (or importance) is proposed in the works of [6] and [3]. [3] introduces a new dimension in his paper to make a vario-scale point cloud. This dimension could either be *scale* or *importance*. These levels are inherent with the implementation of an *octree*, *kd-tree* or other common spatial indexing method. However, discrete levels will result in non-smooth zooming and when mixing scales in resolution bumps. [4] listed three methods to compute a continuous dimension, instead of a discrete one. These methods involve clustering, iterative

data set Name	Points	Format	Data size (GB)
Random Patch	10,653	las	0.0003
AHN3	11,417,422	las	0.38
Drivemap 2016	49,737,433	las	1.65
Integrated	61,154,855	las	2.03

Table 1: data set description

simplification and particle simulation though these methods are often computationally expensive and rather complex. Another way to add a continuous scale dimension is introduced by [7]. This method adds a random importance value between 0 and 1.

2.3 Point Cloud Viewers

Different libraries and framework already have an implementation to view point cloud data. Two examples of point cloud viewers are Potree and ThreeGeoJSON. Both viewers are open source and free for use. Also both viewer use the three.js library and WebGL for the creation and rendering of 3D graphics. ThreeGeoJSON (from `https://github.com/jdomingu/ThreeGeoJSON`) is a rather simple viewer that uses a geoJSON file as input. The UI (user interface) consists of a scene, where the user is able to zoom and rotate through the data. Potree on the other hand is a more comprehensive viewer. It first has to convert the point cloud data with the PotreeConverter.js. After this a data structure (octree) has been applied on the data and it's ready to be inserted in the viewer.

2.4 Server/Client transfer

Also the server-client aspects are examined. There are some front-end languages for web-applications for which there is no substitution: HTML, CSS and JavaScript. But in the back-end there are some alternatives that can be considered: JavaScript, the most used language generally on the web, due to the success of Node.js. PHP with it's possibility to be embedded in HTML and generating a dynamic webpage. Lastly we considered Python which can be considered a "Swiss knife" for connecting applications and libraries of different fields.

3. MATERIALS & METHODS

3.1 Data Analysis

The data used during this research consists of four different data sets. These are shown in Table 1 and visualised in Figure 1. The first data set is a randomly generated set of points, the small size makes it suitable to run preliminary tests allowing for on-the-fly adjustments. The second data set is the AHN3, a thinned .las file derived from AHN (Algmeen Hoogtebestand Nederland). The third data set is the Drivemap data provided to us by Fugro, which is acquired in 2016. The fourth data set is a data set of the AHN3 and Drivemap point clouds combined.

3.2 Importance value

The use case we focus on is a resolution output that depends on the distance to the point-of-view (PoV). So more detail close-by, and less further away. Another use case we have implemented is a selection with consistent point density for a region with two data sets from different sources

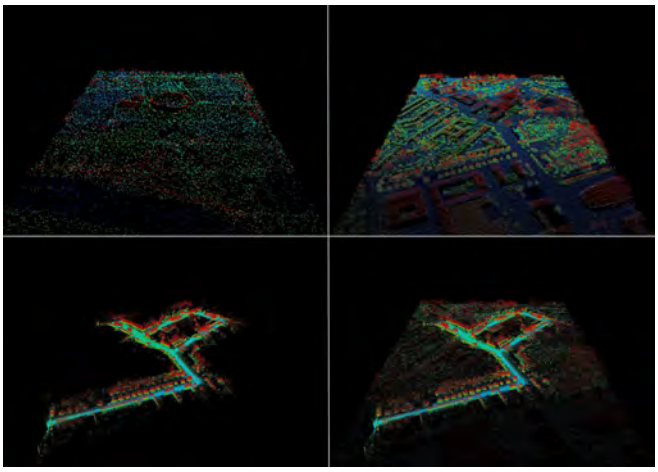


Figure 1: Raw point cloud data. Top left: Random Patch, Top right: AHN3, Bottom left: Drivemap, Bottom right: Integrated.

with originally different densities. For true vario-scale representation an additional dimension is added. This enables the ordering of every point for smooth zooming and progressive data transfer. Our research lists two methods for the computation of the importance value. The first method is based on *random assignment* and is called $FullRandom$. A variation on this first method is based on *local random assignment* and is called $LocalRandom$, which intents more consistency in point density. The second method is an *exact computation of importance*. This algorithm takes for every point the smallest distance to a neighbour. This distance will be added as importance value. After this the specific point will be deleted and the calculation to nearest neighbours will be repeated. This iteration is done until all the point are assigned an importance value. Two variations are used for this method, $ExactSlow$ and $ExactFast$, which are shown in respectively Algorithm 1 and Algorithm 2.

Algorithm 1 ExactSlow: Computation of importance value

```
make KDtree index of dataset
for all point in dataset do
    n = 1
    calculate distance to n nearest_neighboring_points
    while nearest_neighboring_point has importance value do
        n = n + 1
        Calculate distance to n nearest_neighboring_point
        if n is above threshold then
            Remove processed points from dataset
            Make KDtree index of dataset
        end if
    end while
    add distance as importance value to point
    add point to processed points
end for
```

Algorithm 2 ExactFast: Computation of importance value

```
make KDtree index of dataset
while dataset not empty do
  for all point in dataset do
    Calculate distance to point_nearest
    if point_nearest has importance value then
      continue
    end if
    Add distance as importance value to point
    Add points to points_processed
    if point is last point in dataset then
      Remove processed points from dataset
      make KDtree index of dataset
      break the for-loop
    end if
  end for
end while
```

3.3 Indexing

For some of the importance value computations a nearest neighbour search is needed. To optimise searching for neighbours a temporary index is constructed, specifically the *scipy.spatial.cKDTree*. This is a C implementation of the *KD Tree* for Python.

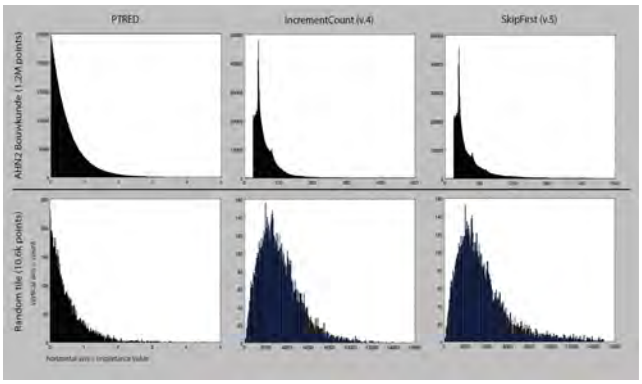


Figure 2: Histograms of importance values with at the top AHN2_BK data set (1.2M points), and the bottom two represent the random patch (10.6k points). The columns represent the 3 different approaches of algorithmic importance calculation: $LocalRandom$**,** $ExactSlow$ **and** $ExactFast$**.**

4. RESULTS

4.1 Importance Value

When we compare the implementations of the algorithms, we see there is not much difference in the importance values it returns. They all resulted in an exponential decay distribution, which is visualised in Figure 2. $ExactFast$ returns higher counts in the higher ranges of importance values. This can also be seen in image 01, where the points with an importance value of over 300 are clearly higher in number in the $ExactFast$ results. The AHN2_BK histograms in Figure 2 are also showing the minimum resolution of the AHN2 data set. There are no distances between points to be found below the value of 20.

The comparison of the distance based algorithm variations, $ExactSlow$ and $ExactFast$, shows differences on both large and small scale. The overview of the entire data set shows that $ExactFast$ compared to $ExactSlow$ contains more points in the higher value ranges. This is because $ExactFast$ skips points as soon as no near neighbours are found, this means distances increase more rapidly. In the detailed view, Figure 3 the $ExactSlow$ seems to preserve edges and other details better compared to $ExactFast$.

4.2 Performance Data Insertion

The data stored in the database includes three-dimensional location values for writing into a downloaded file according to a query, an importance value obtained from our methods, and a geometry code used for spatial queries. All the values are stored based on a patch of 0.1M points, based on the work of Project Pointless (repository `https://github.com/ivodeliefde/ProjectPointless`). This method speeds up the storage of one point after another by approximately 90%.

4.2.1 Data Insertion

The following Table 4 shows the results of the benchmark. When comparing to Potree, most differences pop up in the time it takes to run the .las files. In contrary to our data model, Potree transforms the input point cloud data set to a

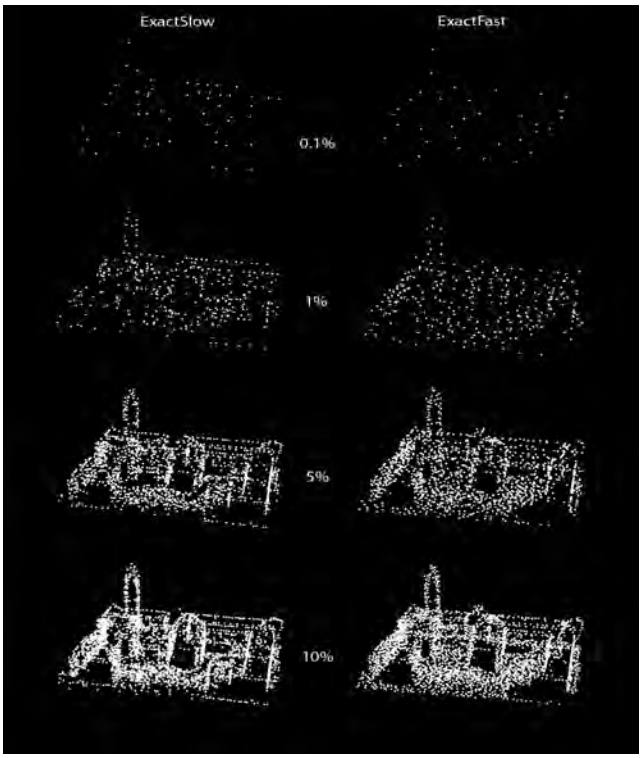


Figure 3: Detail view of AHN2_BK data set split up in points having the highest percentages of importance value calculated by the two approaches of the exact importance algorithm.

file based octree representation. The points are distributed over levels and not just stored in the leaf nodes.

Test	Potree	FullRandom	LocalRandom	ExactFast	ExactSlow
Random Patch					
Total time	0,957 s	0,05 s	0,20 s	0,70 s	0,64 s
Build-up time	0,957 s	0,008 s	0,16 s	0,65 s	0,54 s
Data size (GB)	0,00027	0,00034	0,00034	0,00034	0,00034
AHN3					
Total time	21,614 s	7,5 s	208 s	703 s	1428 s
Build-up time	19,258 s	1,5 s	208 s	688 s	1399 s
Data size (GB)	0,286	0,433	0,433	0,433	0,433
Drivemap					
Total time	88,811 s	28,8 s	363 s	3364 s	n/a
Build-up time	77,9 s	8,0 s	434 s	3289 s	n/a
Data size (GB)	1,21	1,89	1,89	1,89	n/a
Integrated					
Total time	129,603 s	42,5 s	1358 s	3906 s	n/a
Build-up time	124,402 s	8,4 s	1275 s	3818 s	n/a
Data size (GB)	1,49	2,32	2,32	2,32	n/a

Figure 4: Benchmark results

4.2.2 *Visual Inspection*

If we visualise the AHN and the Integrated data set that are processed with both PotreeConverter and our distance based algorithm, it results in the following images (see Figure 5 and Figure 6).

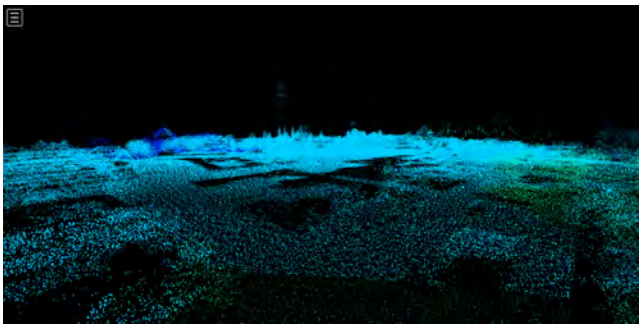


Figure 5: Visualisation of AHN in Potree

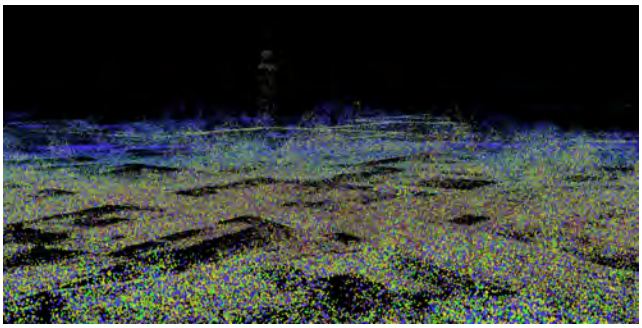


Figure 6: Visualisation of Integrated data set after *ExactSlow* computation

5. DISCUSSION

5.1 Conclusion

The integration, visualisation and distribution of point clouds involve many challenges. One of these challenges lies within the field of scale & granularity. To examine this topic the following research question was stated: 'To what extent can a vario-scale approach improve integration of point clouds with varying point densities?'. For the creation of a true vario-scale approach, an additional fourth dimension is proposed. This dimension is called the *importance value* and is computed using two methods with two variations: *FullRandom*, *LocalRandom*, *ExactSlow* and *ExactFast*. Results show both random and exact methods have an exponential distribution of importance values on the points. For the random algorithms this is due to the exponential distribution where the random values are drawn from. The importance values for the exact computations decay exponentially because of fairly consistent point density within the data set. This results in bisection of the amount of points for every range in importance value. It takes much longer for the exact methods to compute the importance values. This is party caused by the use of a (temporary) KD-tree and expensive computations. However, visualisation also shows that the exact methods results in a better preservation of edges, in contrary to the random method.

Reflecting on the two use cases we can state that the decreasing density distance from the viewpoint reduces clutter in the background of the visualisation. It also eliminates the artefacts that come with a discrete Level-of-Detail approach. The region selection use case on two different data

sets functioned as stepping stone because it was easier to implement.

It is important to state that when determining the importance value using the exact computation algorithm this would first assign importance values to the densest data set. Because of the iterative removal of point with an assigned importance value, this dense data set will be reduced to the same density as the less dense data set. The algorithm will then continue generalisation on both data sets, creating an integrated data set with a continuous importance dimension.

In conclusion there can be said that a vario-scale approach can help the integration of point clouds. Though it is limited by the fact that you have to compute neighbourhoods for each iteration. This can be improved by the creation of 'patches' as to limit the neighbourhoods to create, these patches will have to grow ahead of the neighbourhoods to prevent discrete levels. Other bottlenecks are performance and the visual factor.

5.2 Future Work

5.2.1 Curvature

The *ExactSlow* method clearly best preserves the edges of the building. However, this edge preserving property can be improved. By adding and computing curvature a factor could be added to the importance of certain points. This would reduce consistency in point density, however for many applications the detection of edges is more important.

5.2.2 PgPointCloud Integration

It could be recommended for future work to store the points as PcPoint objects in PcPatch objects. For now they are stored as point geometry in PostGIS.

5.2.3 PcPatch Optimalization

Points from a point cloud data set can be stored as PcPoint objects which can be stored in PcPatch objects (the equivalent of blocks in Oracle). While the current method of storing the points in PostGIS Point objects allows for four attributes (x,y,z,imp), storing points in this format allows for the creation of additional attributes. PcPoint objects are defined in an XML schema, this means they are customizable.

5.2.4 PDAL pipeline

PDAL allows for seamless integration of pgpointcloud in PostgreSQL. Pgpointcloud comes with a PDAL writer and reader; writers.pgpointcloud and readers.pgpointcloud. The PDAL pipelines allow modelling of the data from reading to processing and writing. Using a PDAL has multiple advantages

5.2.5 BRIN index

BRIN (Block Range Index) is a form of indexing introduced in PostgreSQL. This indexing takes far less storage space and could improve the speed. It also has the advantage of begin a part of the PostGIS extension that already supports indexing on four dimensions.

5.2.6 Interactive Viewer

The viewer is built with *three.js* (`https://threejs.org`) and provides the basic viewing tools such as rotating, zooming and panning. And the queries through which points are selected and retrieved are in a separated page. However, ideally, a user can selected points from the viewing page after seeing the data, but this is not yet within the scope of this project, and is placed within future work.

6. ACKNOWLEDGEMENTS

We would like to thank our supervisors Martijn Meijers and Theo Tijssen for their support and guidance during this project. We would also like to thank our contact persons at Fugro, Martin Kodde and Stella Psomadaki, for sharing their expert knowledge and data with us. This project would not have succeed without their support.

References

R. Cura, J. Perret, and N. Paparoditis. Point Cloud Server (Pcs) : Point Clouds in-Base Management and Processing. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W5:531–539, 2015.

R. Cura, J. Perret, and N. Paparoditis. A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing. {*ISPRS*} *Journal of Photogrammetry and Remote Sensing*, 127:–, 2016.

P. V. Oosterom, M. Meijers, and J. Stoter. *Abstracting Geographic Information in a Data Rich World*. 2014.

M. Pauly, M. Gross, and L. Kobbelt. Efficient simplification of point-sampled surfaces. *13th IEEE Visualization conference.*, (Section 4):163–170, 2002.

pgPointCloud. pgPointCloud, 2014.

P. van Oosterom, O. Martinez-Rubi, M. Ivanova, M. Horhammer, D. Geringer, S. Ravada, T. Tijssen, M. Kodde, and R. Gonçalves. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. *Computers & Graphics*, 49:92–125, 2015.

B. Wouda. Visualization on a Budget for Massive LiDAR Point Clouds. 2011.

Scientific Part

1 Introduction

Point clouds are becoming one of the most common ways to represent geographical data. Many acquisition techniques result in this type of data set, from photogrammetry with drone imagery to deep-sea sonar. The scale of acquisition of these point clouds is growing steadily. Governments, like the Netherlands and the United Kingdom, already provide airborne LIDAR rescans of their entire countries on a multi-annual basis. Institutions and companies are also collecting point clouds on a more regular basis. Not only the frequency of point cloud data is growing, coverage area and point density are growing as well. This makes data sets even larger and more difficult to process.

The recent development of efficient processing tools avoids the need for transformations of the representation (e.g. mesh). However, raw point clouds remain very large in storage size and require computationally intensive operations. For future development of these tools more data is needed to experiment with. Crucial factors in future development of point cloud tools are consistent quality and sources. Most of the governments, institutions or companies that acquire point clouds have their own solution for storing and distributing the point clouds. This means data sources are fragmented and quality varies dramatically.

Making these point clouds available for both visualisation and use from a single location, will increase the availability and could solve problems with interoperability and data quality. Integration of point clouds faces a lot of challenges within the *temporal aspect*, the *locational aspect* and the *Level of Detail* the point clouds are represented in. Combining multiple point clouds will increase difficulties already faced with visualising single point clouds.

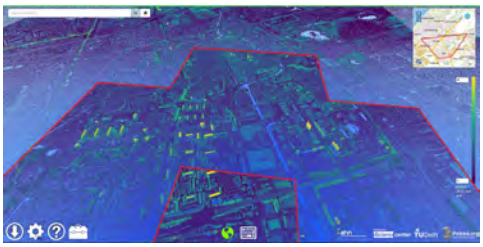


Figure 1.1: Discrete levels in point cloud visualisation

Visualising point cloud data that is stored in a DBMS with conventional tree structures results in discrete levels being visualised. An example of this, taken from the AHN2 point cloud viewer, is visible in Figure 1.1. The red lines represent the discrete levels that are visualised. A way to solve this visualisation issue is by using a vario-scale representation. In this report a vario-scale approach of solving the integration of point clouds with different scale and granularity is presented. The goal is to improve visualisation of integrated point cloud data by using this vario-scale approach.

1.1. Purpose

To create a collaborative solution for storing point clouds, *Fugro* and the *TU Delft* propose the idea of an *OpenPointcloudMap*. This idea entails a platform to which anyone could contribute by uploading any point cloud data set, much like *OpenStreetMap* is for 2D vector maps.

Making the OpenPointcloudMap is no trivial assignment. The process of enabling contributors to upload data, integrating these data sets, visualising and redistributing, involves many challenges. Some of these challenges are; firstly the alignment of the data sets and their *coordinate reference system*, secondly the updating of the map and dealing with the *time dimension* and thirdly handling different

scales and granularity of point clouds from different scanners and environments. Of these three, which are just some of the many challenges, our team focuses on the last; integrating point clouds of different scales and granularity. Solving this challenge enables appealing visualisation, usability for low and high computation powers and geometrical consistency for analysis. These are the aspects we think are needed to create a platform that is engaging for a community of users and contributors.

The following research question is stated.

- To what extent can a vario-scale approach improve integration of point clouds with varying point densities?

In order to frame this research, the following questions are stated to collectively provide a solution for the main research question.

- What kind of approach is best suited for creating a vario-scale representation of point cloud data?
- What processing steps are needed to transform different point cloud data sets into one integrated vario-scale point cloud?
- How does the proposed implementation perform compared to current solutions?

1.2. Methods

Over the course of 10 weeks this project aimed to create an implementation that integrates point clouds with multiple point densities with a vario-scale approach. Despite the limited project duration, multiple aspects of point cloud operations are worked on. Because of this, the first step in this project was to set up a project plan. This plan defines among other things the project phases, the organisation and the main work packages. In order to frame the project, a direction in the form of a specific use case is specified.

To gain information on the topic and to ultimately answer the research question, an initial literature study had to be performed. This study involves an exploration of the scientific body of knowledge pertaining all four main work packages: storage, indexing, processing and visualisation. As a result from this initial study, a data model is generated and implemented. This implementation is created in the phase following the mid-term presentation, marking the end of the exploratory phase.

Throughout the whole project, the project team members kept in touch with the experts and stakeholders. Weekly meetings were scheduled with the team mentors to discuss the progress. Also meetings with our stakeholder Fugro were made, to keep them informed and to exchange information and data. The GRIND implementation is accessible on `https://github.com/openpointcloudmap/GRIND`.

1.3. Boundaries

Multiple aspects bound this project. As mentioned above, the full project covers only 10 weeks. It is part of the Geomatics Synthesis Project, which ran from April 21 till June 23. Work on the project is carried out in the faculty of Architecture of TU Delft.

During this period, there was no time available for data acquisition. We depended on the point cloud data that was already acquired and provided to us by AHN and our stakeholder Fugro.

Due to the project being short-term, we made use of the knowledge gained throughout our studies. There was not sufficient time to learn a new programming language or software.

Legal restrictions could not be overlooked throughout this project. To implement new algorithms to integrate the point clouds, related research and previous work is needed. It is essential to pay attention to whom the work that is used belongs, which type of license the work is under, and to what extent we can use the work.

1.4. Reading Guide

For a quick read it is advised to read the executive summary, the content of this summary is intended to function as a scientific paper. The report consists of two parts, the first part (see Chapter 1 to 5) comprises the scientific works and the second part (see Chapter 6 to 9) comprises the organisational works.

2

Related work

This section elaborates on key theoretical aspects and other research relevant to the creation of an *OpenPointcloudMap*.

2.1. Point cloud server

Because of the storage size and simplicity most point cloud systems still rely on file based storage. In a benchmark van Oosterom et al. [12] shows not only the feasibility, but also the benefits regarding functionality of storing point clouds in a *Database Management System* (DBMS). It has the advantage over file-based processing is that it combines files from different sources in order to perform processing on them. Another distinct advantage is that it enables spatial queries for point cloud analysis. Cura et al. [3] also emphasises the added value of a flexible point cloud server. To realise such a system they use the point cloud extension for *PostgreSQL*. This extension, dubbed pgPointCloud [9], proposes an approach of managing groups of points. Instead of managing individual points, this approach allows to profit more of the advantages a (Relational-)DBMS offers. Dealing with individual points would cause for great numbers of entries in the tables, which in turn would result in slow query results. Cura et al. [2] proposes in their paper a point cloud management system fully based on pgPointCloud [9] and other open source tools.

2.2. Vario-scale

A vario-scale data structure uses scale as an additional dimension. Oosterom et al. [7] introduce the so-called *Smooth tGAP* data structure for smooth zoom in vector maps which is depicted in Figure 2.1. Adding *scale* dimension to 2D maps results in the concept of a *Space Scale Cube* (SSC) [6].

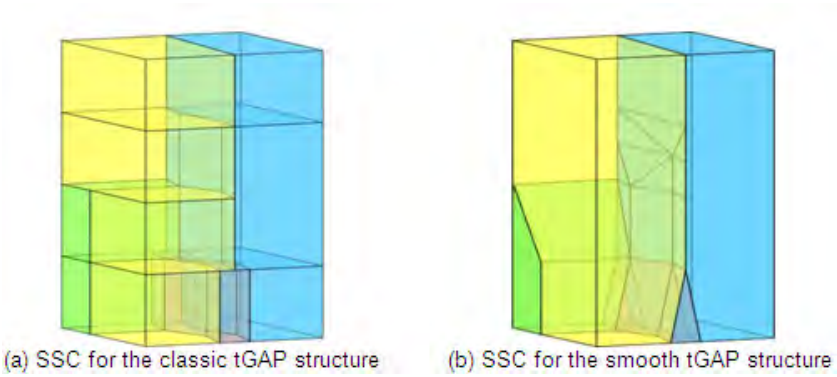


Figure 2.1: Classic and Smooth tGAP data structure resulting in Space Scale Cube

To make vario-scale point clouds, an additional attribute besides *XYZ-coordinates* is needed. This additional attribute could be *scale* or *importance* as Oosterom et al. [7] characterise this dimension. The

importance dimension enables specific resolution queries and even *mixed scale* queries (more on use cases in Section 8.2). This additional dimension can be discrete or continuous. Recent efforts focus on discrete *Levels of Detail* (LOD). Simply because these levels are inherent with the implementation of an *octree*, *kd-tree* or other common spatial indexing methods. However, discrete levels will result in non-smooth zooming, and when mixing scales in resolution bumps (see Figure 1.1 on page 1.1 and Figure 2.1 a) [5].

Computing a continuous importance dimension is similar to generalising point clouds. However, during the process we wish to save the order in which points are generalised. Many techniques exist for generalising point clouds; *point based*, *mesh decimation*, *grid based subsampling*, *Poisson subsampling* and *random generalisation*.

Pauly et al. [8] list three point based generalisation techniques; *Clustering methods*, *Iterative simplification* and *Particle simulation*. Clustering methods are often used as they simply pick a representative point for each node of the often already present hierarchy structure. Iterative simplification techniques are considered computationally expensive as they iterate through the entire point set at least once. Lastly, particle simulation uses the reconstruction of the implicit underlying surface for generalisation. This technique could also be used for interpolation, however these simulations are rather complex.

In their framework for mesh decimation Kobbelt et al. [4] list the following methods to generalise a point cloud data set; Vertex-removal, Edge-collapse, and Half-Edge-Collapse. Of these they state *Half-edge-collapse* is most suitable for mesh decimation because it constraints the decision for the next iteration.

Grid based generalisation of point clouds, even though these techniques are discrete, are often used for efficient rendering. This is illustrated in the implementation created by Schuetz [10] of octree based sub-sampling for the *Potree.js* point cloud viewer.

Cook [1] analysis of sampling techniques and human vision suggest preference for a non-uniform distribution of points. When the original point set is random this can be achieved using *Poisson disk sampling*. By sub-sampling based on distance to other points a consistent density is assured, but the representation remains non-uniform.

Alternatively, Wouda [13] introduces the concept of adding a random importance value to each point. This is a value between 0 and 2 that signifies the importance of the point to the human vision. To ensure an even representation of points through all *Levels of detail* (LOD), the importance value is in this case assigned by a weighted random shuffle. This is parameterised by Wouda [13] heuristically found mean value of 0.7, with a standard deviation of 0.2.

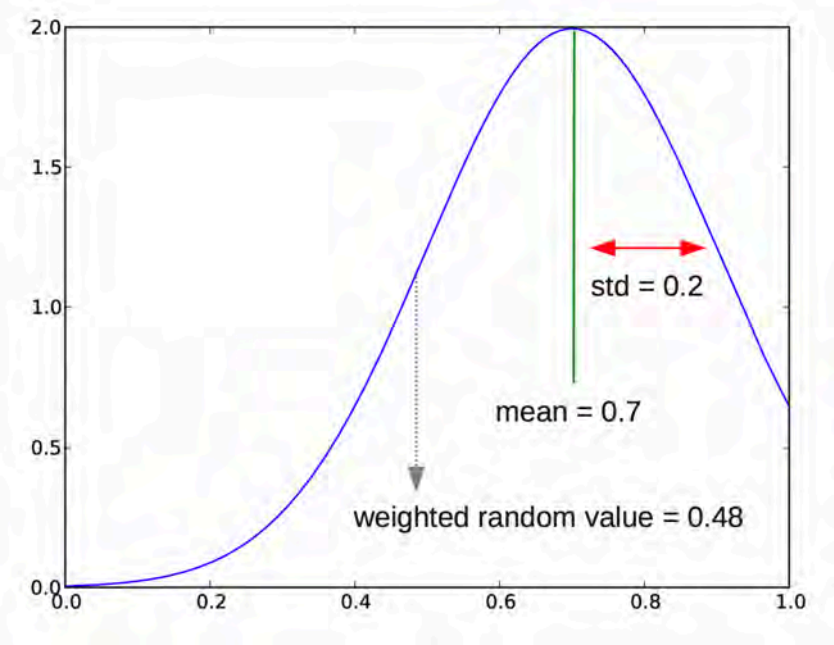


Figure 2.2: The weighed random shuffle as proposed by Wouda [13]

In the proposal by Wouda [13], there are still actual levels of detail in the data structure, and the importance value is used to leap the gaps in detail between the consecutive patches of points. When in an interactive environment (the viewer) the retrieval of these points will be controlled by an Proportional, Integral and Differential (PID) controller.

2.3. Point cloud viewers

The biggest challenge for this project lies in creating the vario-scale approach. However, it is also important to verify this structure through visualisation. There should be a way to show the results since this is an essential application of vario-scale representation for point clouds. For this reason, a web application will be made. This application will allow the user to upload, download and view point cloud data. Since the project is very short-term and we're limited by our (basic) programming knowledge, we'll look into existing frameworks and libraries that already have an implementation to view point cloud data. Potree.js and threeGeoJSON.js are examples of point cloud viewers. Both using a WebGL based viewer and the three.js library. Both viewers are open source and free of use as well. The drawing function that threeGeoJSON provides is suitable for drawing larger geometries such as lines, polygons and points with lower density. This functionality is less suitable for point cloud visualisation. Therefore our research focuses on Potree.js, which we will elaborate on bellow.

2.3.1. Three.js library

Three.js is a library written in JavaScript, used for creating and animating 3D computer graphics in a web browser. Three.js uses WebGL (Web Graphics Library) for this. WebGL does the rendering of the 3D graphics on a web browser. For the viewing of an object or data set, a scene has to be created. This can be done using three.js. Besides the scene, a camera is needed to define the viewpoint of the scene. With these basic elements and the objects that the web developers define for themselves, such as lights, geometries, textures, etc., three.js is useful for the development of different web applications. Good examples include animation of movements and panorama.

2.3.2. Potree

Potree is a web-based viewer that allows the user to display large point clouds. The viewer is constantly being improved by the Harvest4D project. Potree consists of a Potree script and a PotreeConverter script. Potree stores the point cloud data in an octree structure, which is done by the PotreeConverter. The converter takes any LAS, LAZ, PLY or PTX file as input. It converts the data to a BINARY, LAS or LAZ file format, which has the octree structure and different levels of detail. This data can be shown in the Potree viewer.

Schütz [11] describes in this paper the working of the PotreeConverter. The Potree viewer loads only the nodes (with its points) that are visible from the viewpoint. The former converter, written in Java, created an octree by randomly picking points for each node. This random point selection resulted in a non-uniform subset. In some areas holes can be found and in other areas clusters of points. The current converter is written in C++ and chooses points uniformly. new approach first specifies a minimum distance, then iterates through all the distances. All the points with a bigger distance than the minDistance will be stored in that node. Depending on their position, the remaining points will be split into nodes 0 to 7, which represent the 8 child nodes. This process is done until the desired level of detail is reached. This improves both visual and performance qualities, since fewer points are needed to achieve a better quality. This can be seen in Figure 2.3.

2.4. Server framework

The programming languages applied to the visualisation framework can be categorised into front-end languages and back-end languages. For front-end development, HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) and JavaScript are all mandatory and irreplaceable languages, for which there is no substitute.

For back-end development, however, multiple alternatives exist, and each one of them is widely used nowadays. JavaScript, an object-oriented programming language for web application development, has gained popularity also in software development due to the success of Node.js. In this entails that JavaScript not only has access to the server-side database, but also ensures the consistency of

programming language throughout the project.

PHP (Hypertext Preprocessor) is another programming language to be considered. In comparison to JavaScript, PHP is a popular server-side web development which enables the interaction between the front-end and the back-end. Various PHP frameworks such as Symfony, Laravel and CakePHP can be used to create web applications. The advantages of PHP include the possibility to be embedded in HTML to help generating a dynamic web page and an efficient management of the database.

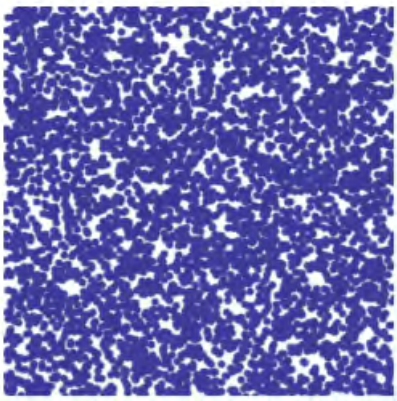
The third programming language assessed is Python. As a popular back-end language, Python is powerful in numerous aspects including scientific computation, data visualisation and being embedded in applications. Web frameworks for Python, such as Django, Flask and Tornado have also been developed to enable the connection to front-end. It has a distinct advantage of being simple, clean and explicit. Its broad range of use together with the web frameworks makes it a credible tool for web application development. A detailed description of the Python web frameworks is provided in Appendix B.

2.5. Data transfer

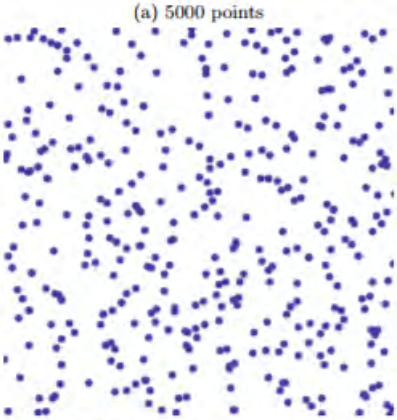
The data transfer from database to web client and vice-versa is accommodated by Django. An implementation of PDAL has been attempted because of its integration of multiple functions, the ability to record operations performed on the data and the ability to construct operators by modifying JSON in Python. Difficulties in the implementation have led to the use of Python for the data transfer.

A bulk loader is used which loads 10,000 points and inserts them into the PostgreSQL database, this significantly speeds up the uploading process compared to the single query uploading. Recommendations for the implementation of PDAL are made in Section 5.2.

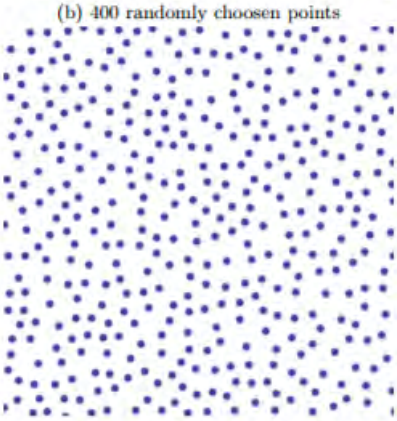
Retrieval of data can also be done with PDAL. The current implementation uses an SQL Query directly implemented from python to retrieve the selected data. Variables are defined by filling in a form on client side; then they are sent to the Python script via Django framework; and with GeoDjango statements, they can be used in the query to retrieve data from the database.



(a) 5000 points



(b) 400 randomly choosen points



(c) 403 points choosen with a minimum distance from each other

Figure 2.3: Different sampling strategies by Schütz [11]

3

Materials & Methods

The goal of this project is to create a vario-scale approach that improves the integration of multiple point clouds with varying point densities. Therefore, this project evolves around and is dependent on point cloud data. In this chapter the tools and data used for this research will be specified.

3.1. Data Analysis

3.1.1. Point Cloud Data

A point cloud is often described as a set of points in a three-dimensional coordinate system (usually defined by XYZ-coordinates), obtained by 3D scanners. The technology that provides this scanning nowadays is becoming more precise, cost effective and available. As a result point cloud data sets are increasing in size and availability. As (Cura et al. [3]) say in their paper, one of the downsides of this growth of massive point clouds, is their unorganised nature. A data hierarchy and index are needed to store these data sets. A good data structure also provides fast loading and easy access and retrieval of data. For integration of point clouds with different densities, providing a good data structure is even more challenging.

3.1.2. AHN data

The structure we propose for this integration, will be applied on four different data sets. One of these data sets is AHN (Actueel Hoogtebestand Nederland). AHN provides a digital height map that covers the Netherlands. The point clouds are obtained by airborne laser technologies. Because of the substantial datasize of almost 1 Terabyte the country is divided into units and sub units. This can be seen in Figure 3.1. The two units we need for this project are clipped and are shown in Figure 3.2. This pre-processing makes downloading and implementing the data faster.

Figure 3.1:
AHN Tiles

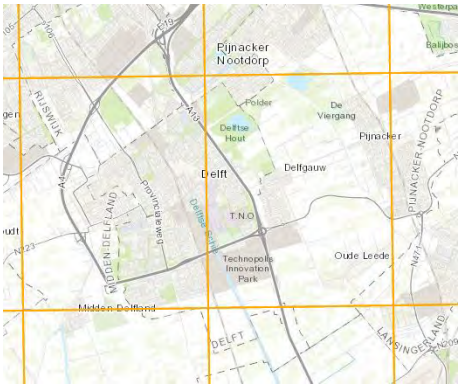


Figure 3.2:
AHN Clipped
tiles

There are three different AHN versions: AHN1, AHN2 and AHN3. For this project we have obtained the latest two. The AHN2 is collected and published between the years 2007 and 2012. Collection of AHN3 commenced in 2014 and is still in progress, full coverage of the Netherlands is planned to be completed in 2019. The average number of points per unit (square meter) is about 8. Further specifications on this data can be found in Figure A.1 in Appendix A.

3.1.3. Fugro data

The third and fourth data sets contain point clouds acquired by mobile laser scanning. This data is collected by Fugro and covers a part of Delft near the faculty of Architecture (see Figure 3.3). The first data set is collected in 2013 and second data set is collected in 2016. Further specifications on this data can be found in Figure A.2 in Appendix A.

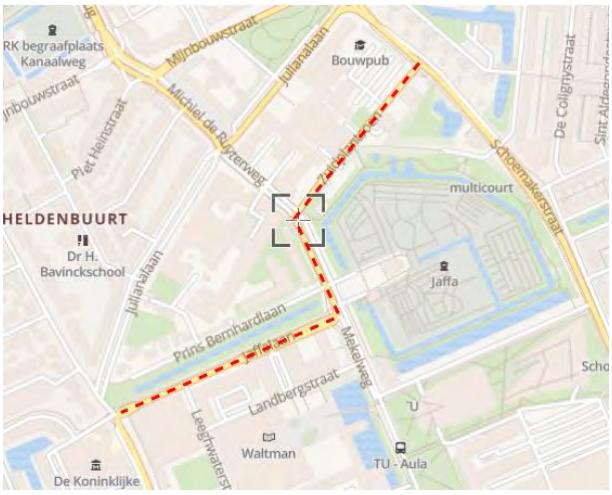


Figure 3.3: Location Fugro drive map 2013

3.1.4. Data sets used

In this project a selection of four data sets is used for the performance analysis. These are shown in Table 3.1. A visual comparison of these data sets is made in Figure 3.4. All four data sets are visualised using the FugroViewer. Both the Drivemap and Integrated data set have been decimated because of the FugroViewer's limitation in loading massive point clouds. For reference, the AHN3, Drivemap and Integrated data sets are visualised from the south, meaning the images are facing north.

The first data set is a randomly generated set of points, the small size makes it suitable to run preliminary tests allowing for on-the-fly adjustments. The second data set is the AHN3, as described in Section 3.1.2. The third data set is the drivemap data from Fugro collected in 2016, which is described in Section 3.1.3. The fourth data set is a data set of the AHN3 and Drivemap 2016 point clouds combined. Combining these data sets is done using FME. In Section 5.2 uploading of multiple data sets is discussed, eliminating the need to integrate the data set with 3rd party software before processing.

data set Name	Points	Format	Data size (GB)	Description
Random Patch	10,653	las	0.0003	Custom generated data set
AHN3	11,417,422	las	0.38	AHN3 tiles clipped to fit the Fugro Drivemap
Drivemap	49,737,433	las	1.65	Fugro Drivemap 2016 data set
Integrated	61,154,855	las	2.03	AHN3 and Fugro Drivemap 2016 combined

Table 3.1: data set description

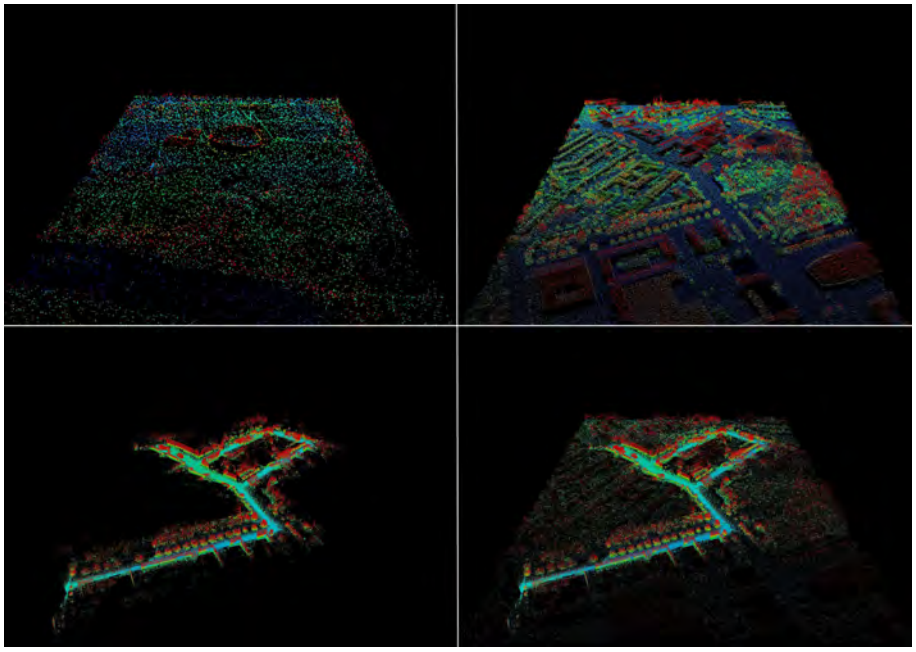


Figure 3.4: Raw point cloud data. Top left: Random Patch, Top right: AHN3, Bottom left: Drivemap, Bottom right: Integrated.

3.2. Data model

For true vario-scale representation an additional dimension is added. This enables the ordering of every point for smooth zooming and progressive data transfer.

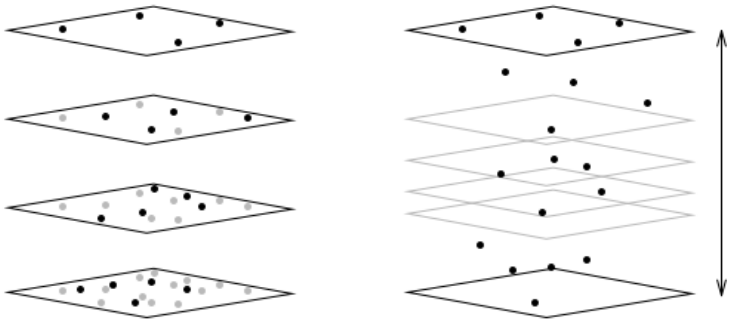


Figure 3.5: Multi-resolution and vario-scale point cloud

This point ordering is shown in Figure 3.5. It shows the difference between discrete levels and a continuous dimension. By computing an importance value based on distance or point density the value can be used to order points and maintain global point density when integrating point clouds of different densities.

3.2.1. Computation of importance value

Our research implements four methods for the computation of importance value. The $FullRandom$, $LocalRandom$ and two types of $Exact$ method. These methods are explained below.

FullRandom Create an array with as many random *float* values as there are points in the data set. Then assign those values to the importance dimension of the points.

LocalRandom This method assigns random values similar to the $FullRandom$. However, for this method the random values are generated per neighbourhood. The *Poisson disk sampling* as described by Kobbelt et al. [4] iteratively discards points within a certain radius. In the $LocalRandom$ method points are assigned a random importance value within a certain radius. The assignment of a distribution of random values per neighbourhood intents to ensure greater consistency in point density.

Exact Algorithm 1 and 2 assign an importance value to each point in the data set. This value is computed based on the distance to its nearest neighbour. The distances from every point to its nearest neighbour are sorted and the shortest is assigned as importance value to the point. When a point has an importance value it is removed from the queue and distances in this area are recomputed. Finally, when all points have an importance value the queue is empty.

The implementation of the importance value algorithm into Python code is done using two approaches. This is because the KD-Tree index that is used to speed up the calculations needs regular updating. The removal of a point creates a level of uncertainty as to what point is nearest to the other, the remaining source data set will be indexed again.

The first approach is based on an incremented number of neighbours search when the returned distance belongs to a point that is already appended with an importance value:

Algorithm 1 ExactSlow: Computation of importance value

```
make KDtree index of dataset
for all point in dataset do
    n = 1
    calculate distance to n nearest_neighboring_points
    while nearest_neighboring_point has importance value do
        n = n + 1
        Calculate distance to n nearest_neighboring_point
        if n is above threshold then
            Remove processed points from dataset
            Make KDtree index of dataset
        end if
    end while
    add distance as importance value to point
    add point to processed points
end for=0
```

This approach is dubbed $ExactSlow$. This approach ensures the same sequence of points in the output as there is in the input. The incremented number of neighbours searched per point is a costly operation. When the number n gets bigger than the threshold, the data set will be updated and the KD-tree rebuilt. The threshold is calculated with a heuristically found equation using the number points (np) left in the data set:

$$Threshold = (np/9000) + 40$$

The second approach is based on the principle that when for a given point i the nearest neighbouring point is already processed, point i will be skipped. Subsequently the next point in the data-set will be queried:

As illustrated in the pseudo-code above, the KD-tree will only be rebuilt after the algorithm has scanned through all points. It will proceed with the updated data set until all points from the initial data set are removed as they are processed. This approach is named $ExactFast$. The performance of the second approach is much faster in terms of processing time (-60%) for files containing more than 50k points, hence the name.

3.2.2. Indexing

For some of the importance value computations a nearest neighbour search is needed. To optimise searching for neighbours an index is constructed, specifically the *scipy.spatial.cKDTree*. This is a C

Algorithm 2 ExactFast: Computation of importance value

```
make KDtree index of dataset
while dataset not empty do
  for all point in dataset do
    Calculate distance to point_nearest
    if point_nearest has importance value then
      continue
    end if
    Add distance as importance value to point
    Add points to points_processed
    if point is last point in dataset then
      Remove processed points from dataset
      make KDtree index of dataset
      break the for-loop
    end if
  end for
end while=0
```

implementation of the *KD Tree* for Python.

The implementation does not allow for updating. Therefore a list of remaining points is kept and the KD tree is regularly recomputed.

3.3. Client/Server framework

The client/server framework serves as the communication platform between the front-end, where a user can send requests, and the back-end, where data management happens after receiving the requests, and various responses are generated. The framework, in essence, passes the request from the front-end to the back-end, and returns the responses from the back-end to the front-end.

Because of the detailed documentation, simple connection with PostgreSQL, and support of point geometry and PostGIS, Django is used as the framework for client/server interaction in this project.

This section explains how the web application works. The application includes three functional modules: *uploading*, *downloading* and *viewing*; an extra button for transforming is also on the page (see Figure 3.7). The complete structure of the web application developed for this project is shown in Figure 3.6.

3.3.1. Uploading

The upload module consists of an HTML input of files, where a button on the website enables the user to select a file from the local directories, and another HTML input to ensure submission (Figure 3.8). Multiple data sets can be uploaded. However, only .LAS format will be recognised and processed. After this the files are stored in the Django framework and ready for transformation. This transformation will calculate the importance values, add them to the point cloud and store it in a database. This will be further explained in Section 4.2.

3.3.2. Downloading

After the transformation and storing, the data is ready to be downloaded (Figure 3.9). After clicking the download button, the requested data will be queried from the database. This querying is implemented in two different ways:

1. The first query is based on so called *envelope*, a rectangular surface, with a specified total amount of points. The client has to specify the rectangle by defining the two corner points. This will create a rectangle geometry and all the points within this area (with the selected point density) will be added to the requested download.

```
1    DROP VIEW thisgeom;
2
3    CREATE VIEW thisgeom AS SELECT ST_MakePolygon(ST_GeomFromText('LINESTRING(point0,
         point1, point2, point3, point0)'));
```

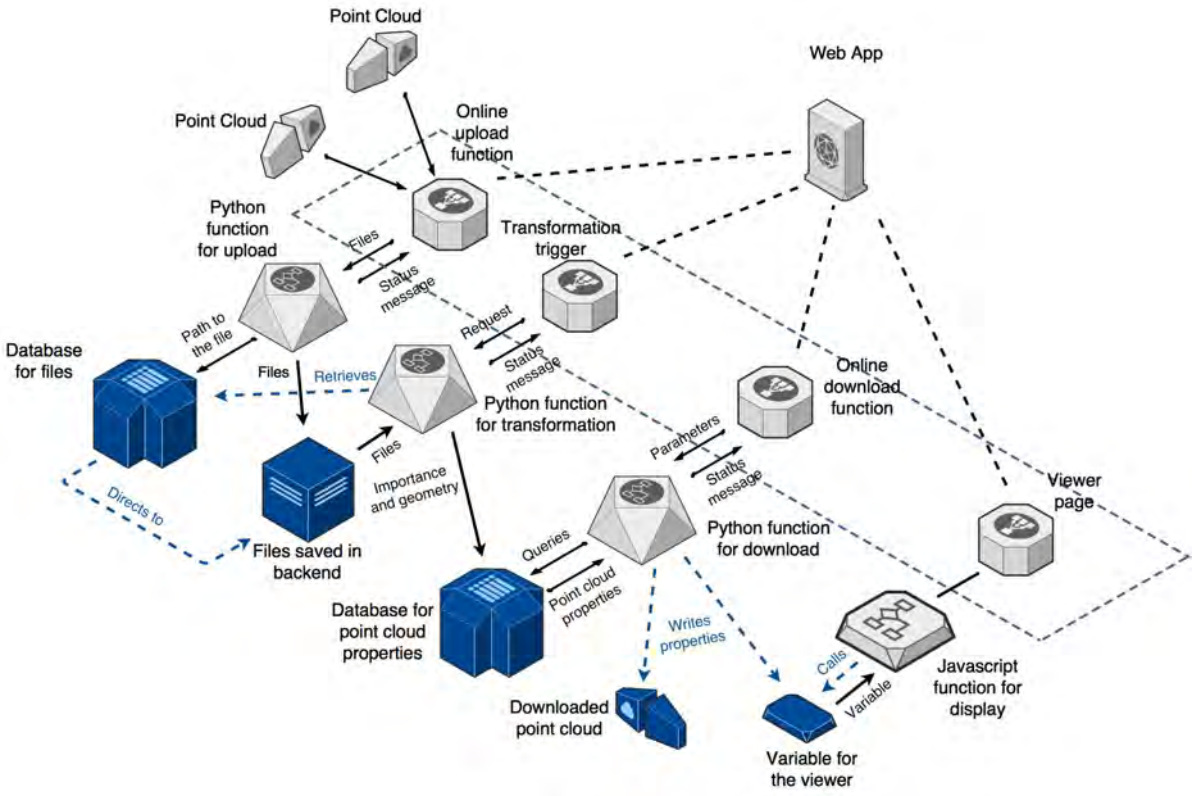


Figure 3.6: Django web application

```
4
5       SELECT ST_M(pointcloud.the_geom), pointcloud.the_geom, thisgeom.st_makepolygon
6       FROM pointcloud, thisgeom
7       WHERE ST_Within(pointcloud.the_geom, thisgeom.st_makepolygon)
8       ORDER BY ST_M(the_geom) DESC
9       LIMIT 10;
```

2. The second query is based on a radius from a viewpoint with decreasing point density relative to the distance from the viewpoint. The client is able to select this viewpoint as a location based on x and y values. Also a value for the radius has to be specified. A geometry is made by getting a buffer around the point. All point within this region will be selected. The density around the point will be high and decreases as the distance grows. A function determines whether or not a point is taken, depending on the distance to the viewpoint and its importance value. This function is made so that it has a fairly even distribution of points.

```
1       DROP VIEW circle_geom;
2
3       CREATE VIEW circle_geom AS SELECT ST_Buffer(ST_GeomFromText('POINT(location)', SRID)
            , radius);
4
5       SELECT p_id, ST_AsText(point), imp
6       FROM pointcloud, circle_geom
7       WHERE ST_Within(pointcloud.point, circle_geom.st_buffer)
8       AND imp > ST_Distance(ST_GeomFromText('POINT(location)', SRID), pointcloud.point)/
            radius;
```

The data query will be carried out using GeoDjango, a Django module of PostGIS queries in Python language. With this module, the input parameters defined on the front-end can be sent to the PostgreSQL database where data can be retrieved, and the data can be processed with a Python script where the output file is written in a .LAS format. Finally, a response will be sent back to the front-end informing the user that the download is completed.

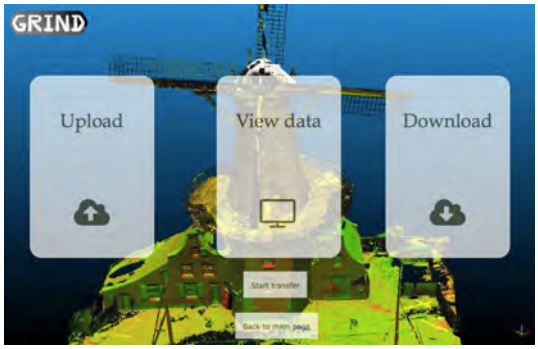


Figure 3.7: Functions of the web application

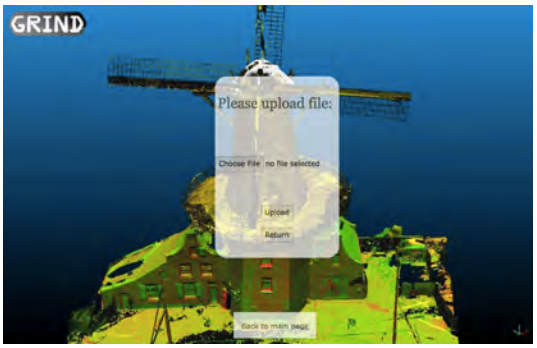


Figure 3.8: Upload module

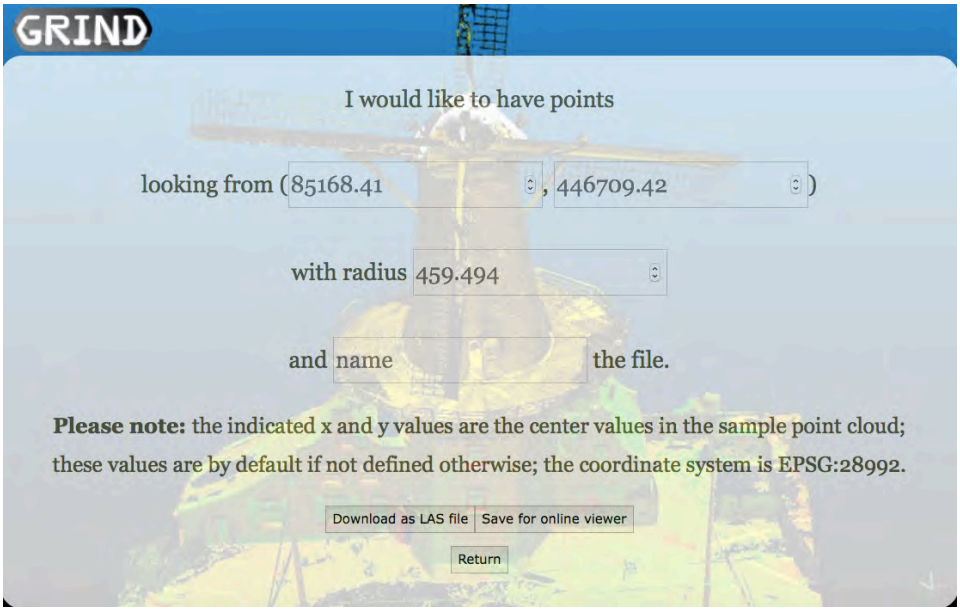


Figure 3.9: Download module

The second button on this page is to send the query to the online viewer. The data retrieval is the same as above, the difference being that instead of writing a .LAS file, this function writes the data into a JavaScript variable for the viewer to call and use. This will be introduced in Section 3.3.3.

3.3.3. Viewing

The online viewing is realised by using three.js (`https://threejs.org`) functions. Within the function, three important components are defined: *the scene*, *the camera* and *the controller*.

The scene acts as the canvas where the point cloud is drawn. The controller is a three.js class named TrackballControls. With this, the user can drag the mouse to control the rotation in all directions of pitch, roll and yaw, and can zoom in and out with mouse controls as well.

The initial viewing point is defined by the camera. This is a Perspective Camera which simulates the vision of the human eye. The mandatory parameters to define the camera are introduced in Figure 3.10.

Three parameters are defined when creating a perspective view; *field of view (fov)*, *aspect* and *near and far planes*. The first parameter, fov, is defined as twice the angle of BAC, i.e. the angle between the horizontal plane and the upper plane of vision. Aspect is the canvas width divided by height and projects . Its tangent value is symbolised by angle CAD. Near and far planes are the boundaries within which the object can be shown. These are defined via distances.

This idea of visualising the vario-scale point cloud requires the camera to see from the central point defined by the user. To set the position of the camera, the concept provided by Figure 3.11 is applied

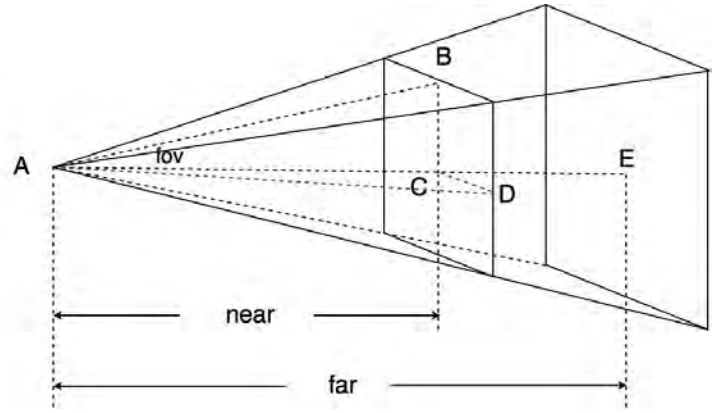


Figure 3.10: Perspective camera

(https://www.udacity.com/course/cs291).

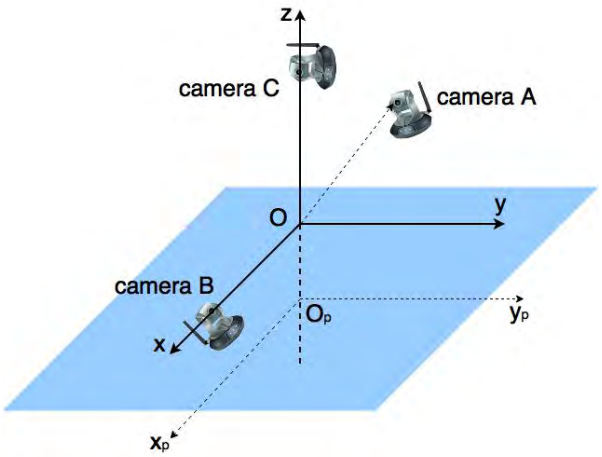


Figure 3.11: The position of the camera

Camera A is a camera set in the random position of a 3D Cartesian space. The position is defined with vector (x, y, z), and the camera looks towards $(0, 0, 0)$. However, the camera is still free to rotate within the plane perpendicular to the vector, which may make the view upside-down or pointing to a random direction instead of upwards. In this case, the upper direction of the camera is also defined, in order to fix the position of the camera. Our viewer sets the camera on the x-axis, and the upper direction is the z-axis. This position functions as the position of camera B. The point cloud is moved towards the negative direction of the z-axis for a partial bird-view, as is defined by the plane $x_p - O_p - y_p$. The result within the viewer is shown in Figure 3.12.

This is ideal when the observing point is located within the central zone of the point cloud. However, if the user selects a point near the border from which only one direction is populated with retrieved points, it is likely that the camera by default faces a direction where there are no points. Theoretically, the direction can still be defined by finding the nearest point to the observing point, and setting the displacement vector between the two points as target direction, and the points in this direction will be shown. But this target setting is violated by the Trackball controller, and no documentation is found to fix this. In this case, the user can choose to 'Switch aspect' and locate the point cloud from the above. The position of the camera is shown as camera C in Figure 3.11, and the resulting view is visualised in Figure 3.13.

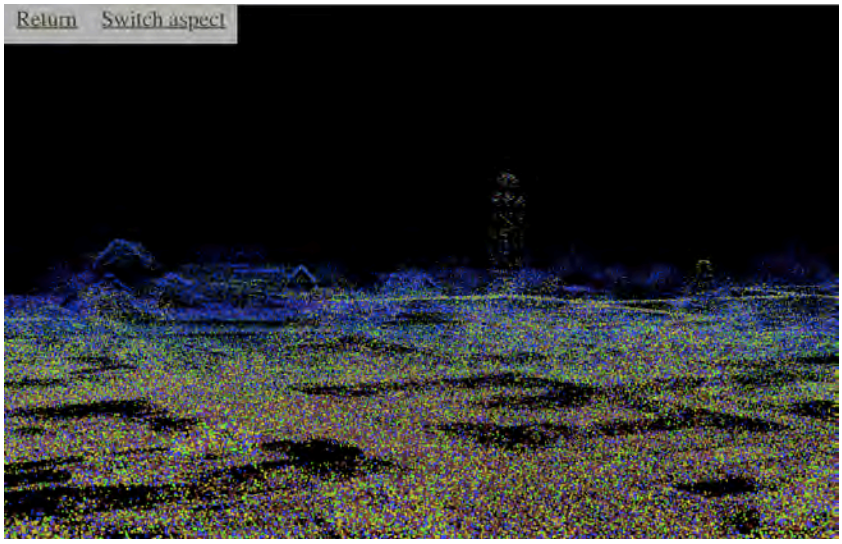


Figure 3.12: The point cloud viewed from the centre

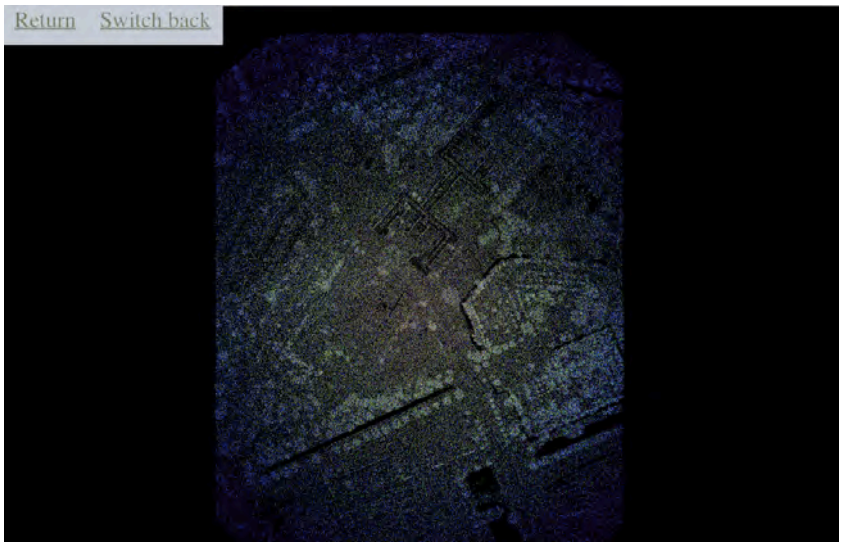


Figure 3.13: View from the above

3.4. Performance analysis

In the set up of the performance test is discussed. The performance analysis step of this methodology compares our implementation with an existing benchmark. For this, *Potree* is used (`www.potree.org`). Potree is an existing web viewer for point clouds which performs and octree indexing on the data set when converted.

In the performance analysis four data sets will be used:

1. A generated patch of uniformly dispersed points
2. A clipped AHN3 patch
3. The Drivemap 2016 data set as provided by Fugro
4. The combined Drivemap 2016 and clipped AHN3 data set

More information on the AHN and Fugro data sets can be found in respectively Subsection 3.1.2 and Subsection 3.1.3. The performance will be tested on the following quantitative indicators:

- *total time*, the complete time the process takes including reading the data and converting it.

- *build-up time*, the time it takes to build up the data structure and index.
- *data size*, how much disk-space does the resulting structure take.

3.4.1. Hardware

For the performance analysis a laptop is used with the following hardware specifications:

- ***OS***, Windows 10 Home
- ***Processor***, Intel Core i7-6700HQ CPU @ 2.60GHz
- ***RAM***, 16GB
- ***System***, 64 bit, x64 processor

4

Results

The previous chapter explains the methodology for this research. This methodology resulted in a point cloud application that handles uploading, downloading and viewing of vario-scale point clouds. In this chapter the functionality of this application and its results are shown.

4.1. Importance value

For computing the new dimension for each point, the importance value, multiple methods have been implemented. The distributions of these values determine the amount of points that are retrieved for specific spatial queries. To analyse these distributions the following histograms are plotted.

When we compare the implementations of the algorithm, as explained in Subsection 3.2.1, in Figure 4.1, we see there is not much difference in the distribution of importance values it returns. The $LocalRandom$ as well as both the $ExactSlow$ and $ExactFast$ implementations result in an exponential decay distribution of point distribution. For the random assignment this is logical as the random samples are drawn from an exponential distribution.

The $ExactSlow$ and $ExactFast$ computation algorithms result in an importance value based on the distance. These values range from 20 to 450 for the AHN2 data set and 0 to 15.000 for the random patch. This means the nearest points are circa 20 centimetres apart, which is therefore the maximum resolution. For the $LocalRandom$ computation algorithm the value ranges from 0 to 5. The random values can not directly be related to a specific point density.

The comparison of the distance based computations, $ExactSlow$ and $ExactFast$, shows differences on both large and small scale. The overview of the entire data set in Figures 4.2 shows that $ExactFast$ compared to $ExactSlow$ contains more points in the higher value ranges. The points with an importance value of over 300 are clearly higher in number in the $ExactFast$ visualisation compared to the $ExactSlow$. This is because $ExactFast$ skips points as soon as no new near neighbours are found, whereas $ExactSlow$ increases the number of neighbours to look for. This means distances in $ExactFast$ increase more rapidly. In the detailed view, Figure 4.3 the $ExactSlow$ seems to preserve edges and other details better compared to $ExactFast$.

The $FullRandom$ and $LocalRandom$ approaches on large scale data sets are visualised in Figure 4.4. This figure shows that the $LocalRandom$ method overall improves consistency in point density. The small scale in Figure 4.5, however, shows that both methods result in holes and as expected do not guarantee qualitative coverage. It is hard to visually determine whether the $LocalRandom$ method actually has an impact or that it is just as random as the $FullRandom$ method.

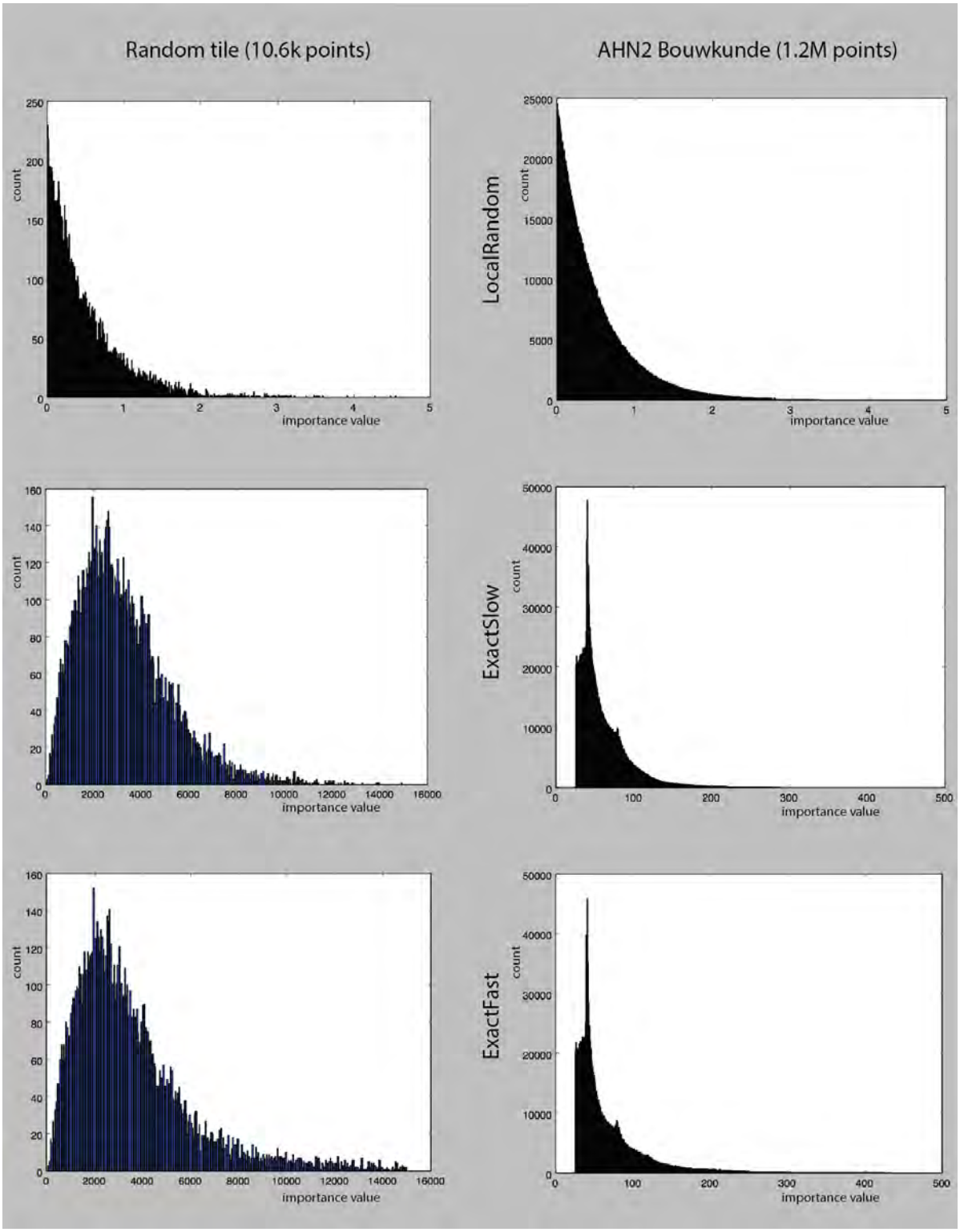


Figure 4.1: Histograms of importance values with at the top AHN2_BK data set (1.2M points), and the bottom two represent the random patch (10.6k points). The columns represent the 3 different approaches of algorithmic importance calculation: $LocalRandom$, $ExactSlow$ and $ExactFast$.

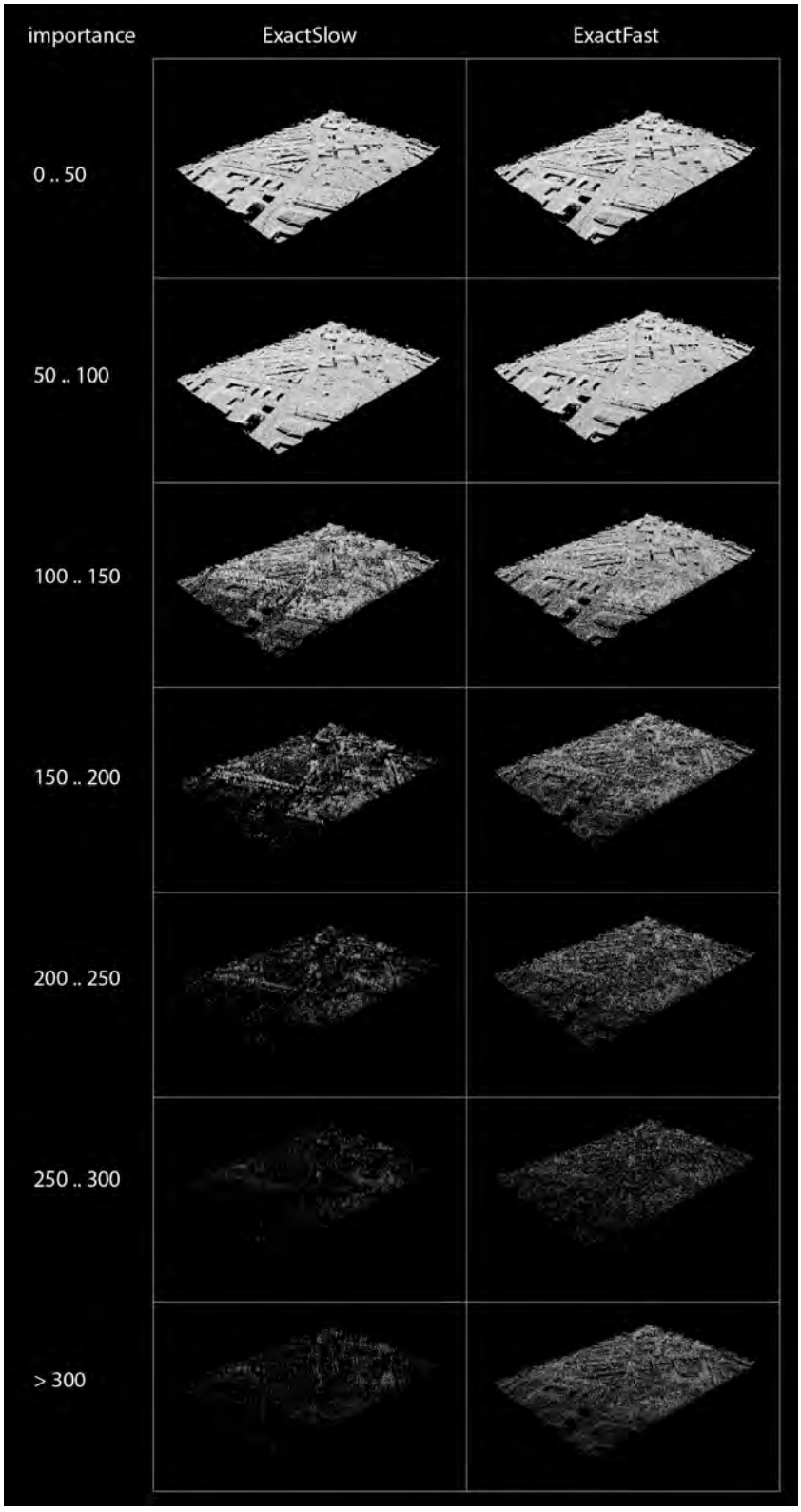


Figure 4.2: Visual representation of AHN2_BK data set split up in different ranges of importance value calculated by the IncrementSearch and SkipFirst algorithms.

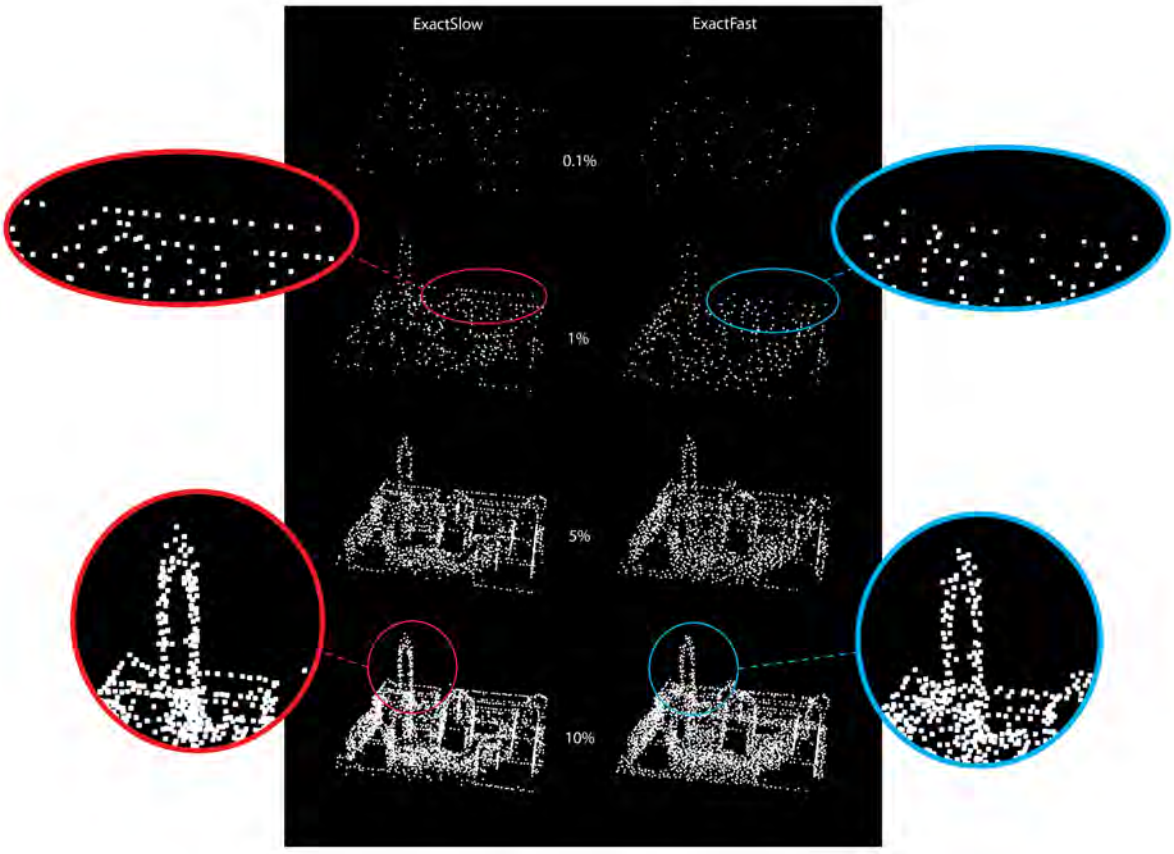


Figure 4.3: Detail view of AHN2_BK data set split up in points having the highest percentages of importance value calculated by the two approaches of the exact importance algorithm.

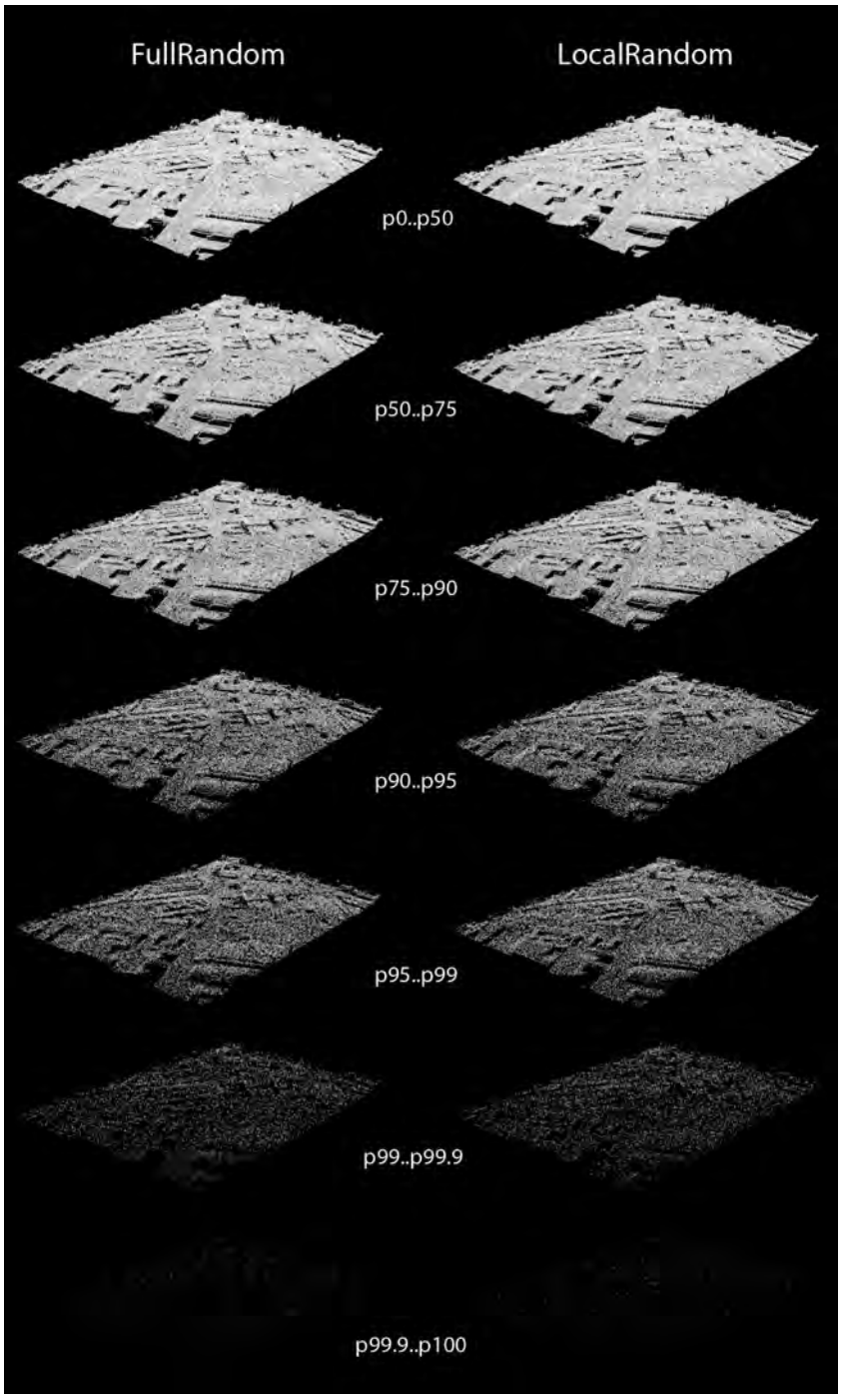


Figure 4.4: Visual representation of AHN2_BK data set split up in percentiles on importance value calculated by the two approaches of the random importance calculation.

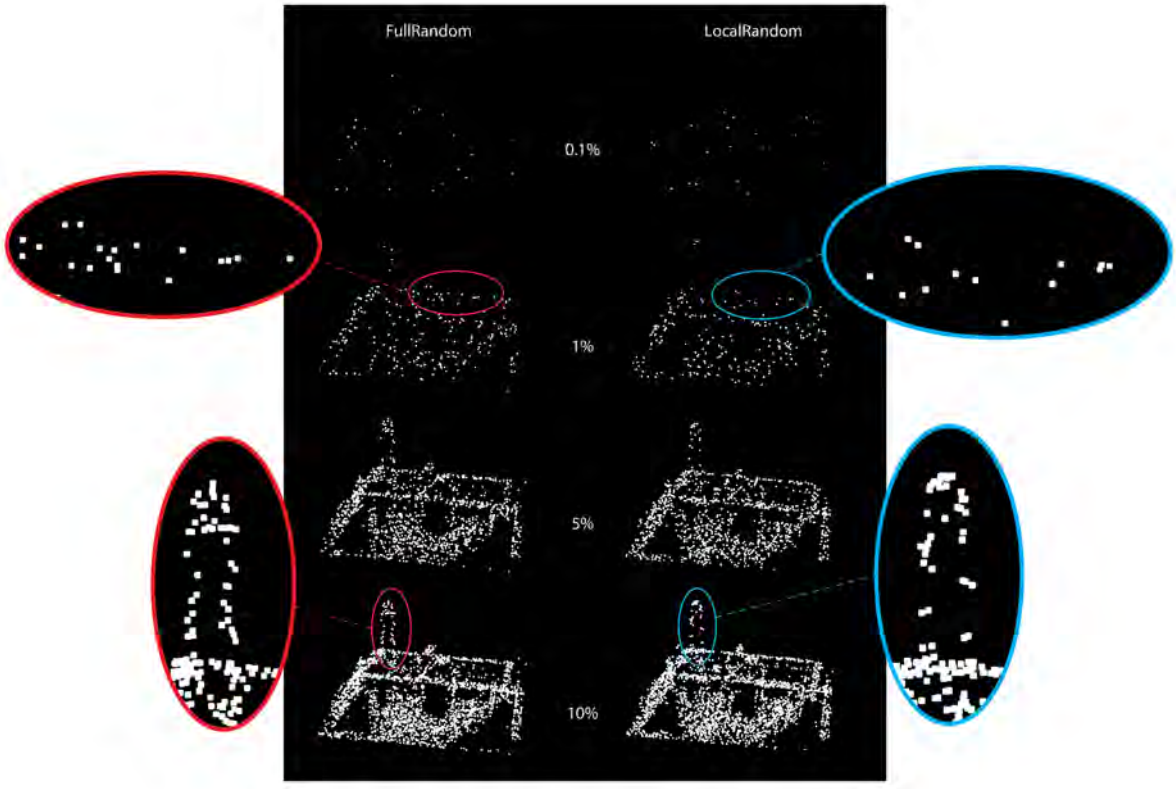


Figure 4.5: Detail view of AHN2_BK data set split up in points having the highest percentages of importance value calculated by the two approaches of the random importance calculation.

4.2. Performance analysis

The point cloud table inside PostgreSQL contains the columns and value types as can be seen in Table 4.1;

id (integer)	x (double)	y (double)	z (double)	imp (double)	xypoint (geometry)
0	84897.55	446362.99	3.86	0.291204	...
1	84896.98	446363.43	3.84	0.063986	...
2	84896.79	446363.63	3.87	0.826300	...
3	84897.09	446363.67	3.84	1.082571	...
4	84897.16	446363.95	3.8	0.988383	...
...	...	...	...	...	...

Table 4.1: PostgreSQL table

Initially, the locations of each point are retrieved from the .LAS file, the importance value is retrieved from the implementation result, and the geometry is defined as 2-dimensional point geometry under coordinate reference system EPSG:28992. Next the table is filled in row by row, one at a time. This whole process takes approximately 100 minutes for 1.2M points, with randomly generated importance values. This is a costly process for the storage of a point cloud, therefore an optimisation where multiple records can be saved at one time. However, thanks to the experiment by Stefano Dissegna (`http://stefano.dissegna.me/django-pg-bulk-insert.html`) and the method provided by Project Pointless (`http://projectpointless.bitballoon.com`), the data is stored after every 0.1M points. This reduced the storage time to 10 minutes for 1.2M points. This method first writes every 0.1M records to a CSV file-like object, and then imports the records to the database. This method first ensures that the records are in batches, and CSV format is used for storing tabular data, which can be directly input to the database without processing overhead.

During retrieval, the geometry column is used to perform spatial query during data retrieval, thus saving the downloading time, and only the x, y, z and importance values are written to the output file. The process from defining the geometry to carrying out the spatial query can be realised by *GeoDjango* functions. A generic index structure (GiST) is also created on the geometry column to simplify thus speed up the querying process.

Interesting to note is the difficulty of installing the Potree converter. On both the Linux with *GCC 4.9* and the Windows with *Microsoft Visual Studio 2015* build problems arose. The hardware of the machine that the Potree converter is built on can be found in Subsection 3.4.1. When built, the Potree converter is easy to use and did not require any adjustments to run the performance analysis. The difficulty with building the Potree converter pleads for the development of an easy to use web platform like OpenPointcloudMap.

4.2.1. Data insertion

In Table 4.2 the results from the benchmark are shown. When comparing to Potree, the most notable difference is the time it takes to run the .LAS files. Contrary to our data model, Potree transforms the input point cloud data set to a file based octree representation. The points are distributed over levels and not just stored in the leaf nodes. The GRIND implementation uses a temporary KD-tree and has a more expensive computation. This results in higher calculation times compared to Potree. For the integrated data set the computation using the $ExactFast$ method takes about 3906 seconds, while Potree's converter takes only 129 seconds.

The Build-up time for $FullRandom$, $LocalRandom$, $ExactFast$ and $ExactSlow$ is the time spent on calculating and appointing the importance value itself. So without the time it takes reading and writing the LAS-file, and loading the initial data in the memory. For $FullRandom$ this simply consist of generating the random values for the number of points. For $LocalRandom$ this is building the KD-tree, querying the points for other points within range. Finally for $ExactFast$ and $ExactSlow$ this is multiple calculations of the KD-tree, querying for the nearest neighbours and calculating the distance.

The ExactSlow code is not optimised for Windows and works only on MacOS for data-sets over 1GB. Therefore Table 4.2 shows no answer for this algorithm for the Drivemap and Integrated data.

Test	Potree	FullRandom	LocalRandom	ExactFast	ExactSlow
Random Patch					
Total time	0.957 s	0.05 s	0.20 s	0.70 s	0.64 s
Build-up time	0.957 s	0.008 s	0.16 s	0.65 s	0.54 s
Data size (GB)	0.00027	0.00034	0.00034	0.00034	0.00034
AHN3					
Total time	21.614 s	7.5 s	208 s	703 s	1,428 s
Build-up time	19.258 s	1.5 s	208 s	688 s	1,399 s
Data size (GB)	0.286	0.433	0.433	0.433	0.433
Drivemap					
Total time	88.811 s	28.8 s	363 s	3,364 s	n/a
Build-up time	77.9 s	8.0 s	434 s	3,289 s	n/a
Data size (GB)	1.21	1.89	1.89	1.89	n/a
Integrated					
Total time	129.603 s	42.5 s	1,358 s	3,906 s	n/a
Build-up time	124.402 s	8.4 s	1,275 s	3,818 s	n/a
Data size (GB)	1.49	2.32	2.32	2.32	n/a

Table 4.2: Benchmark results

4.2.2. Visual inspection

If we visualise the AHN and the Integrated data set that are processed with both PotreeConverter and our $ExactFast$ algorithm, it results in the following images (see respectively Figure 4.6 and Figure 4.7). On the Potree image, different Levels of Details are visible through colouring. The further away from the centre view, the less dense the point cloud gets. The biggest difference between both is seen in far of this centre. The Potree points here get all clustered up, which blocks the sight. This is not the case for our result, due to the continuity of the model.

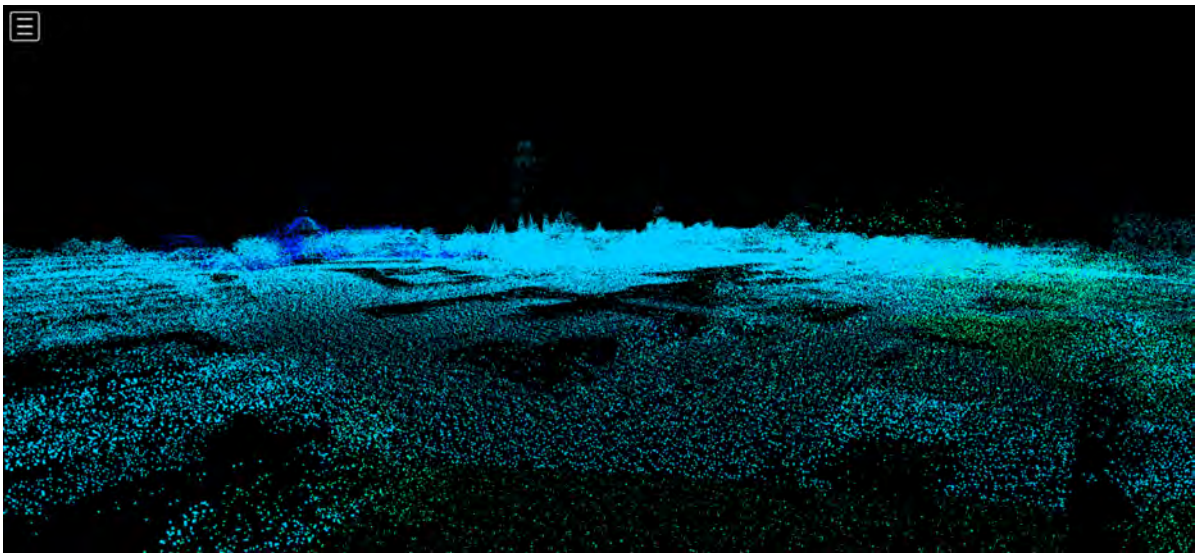


Figure 4.6: AHN processed with PotreeConverter from a center view

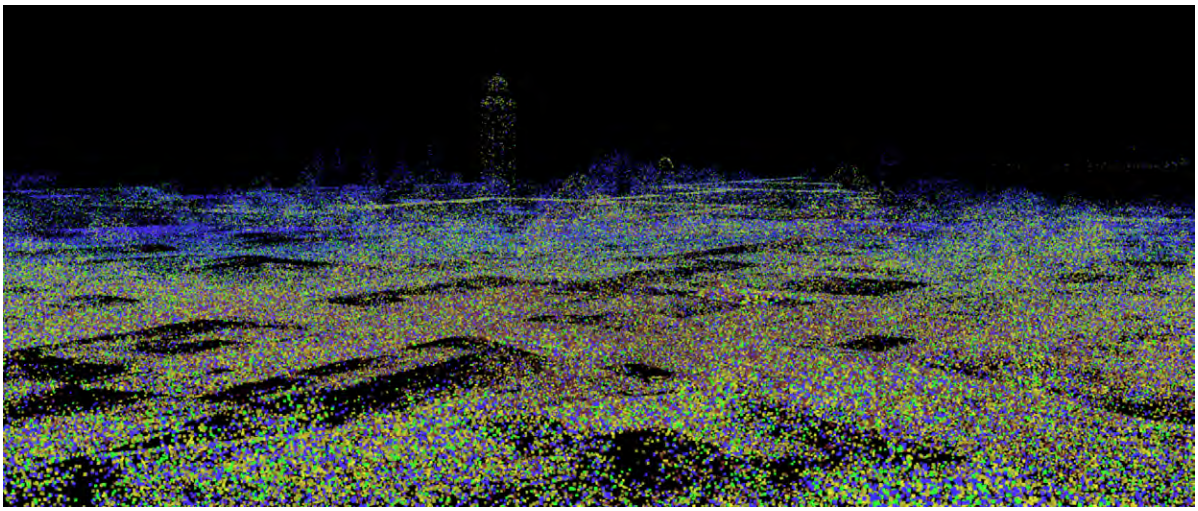


Figure 4.7: Integrated data set processed by distance based algorithm from a centre view

The following images show the same data set from a bird's eye view. In Figure 4.8 the blocks with different LOD's generated by Potree are visible. Figure 4.9 visualised the vario-scale model.

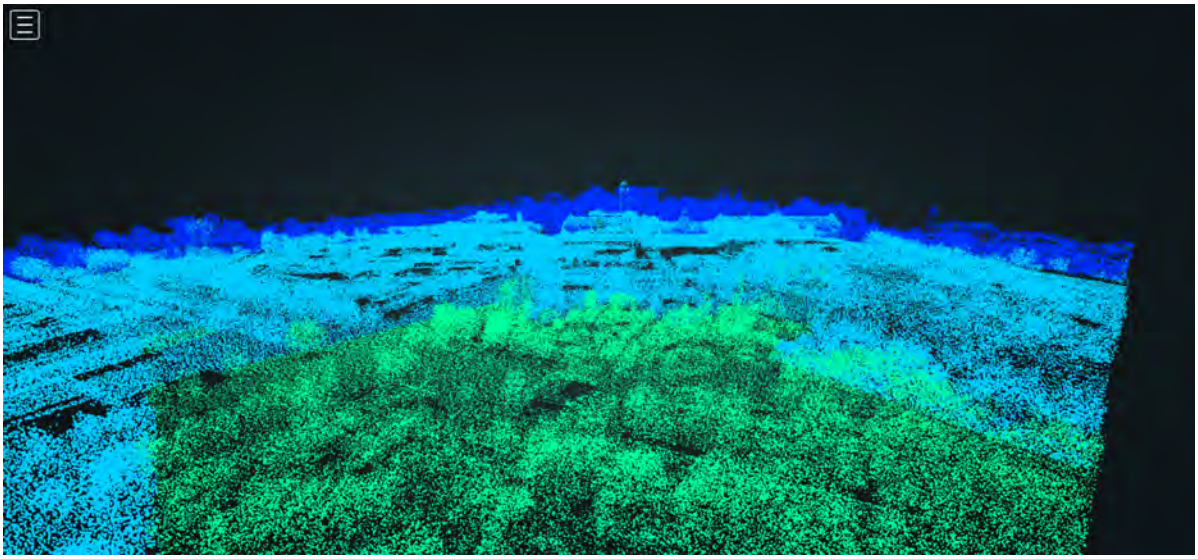


Figure 4.8: AHN processed with PotreeConverter from a bird's eye view

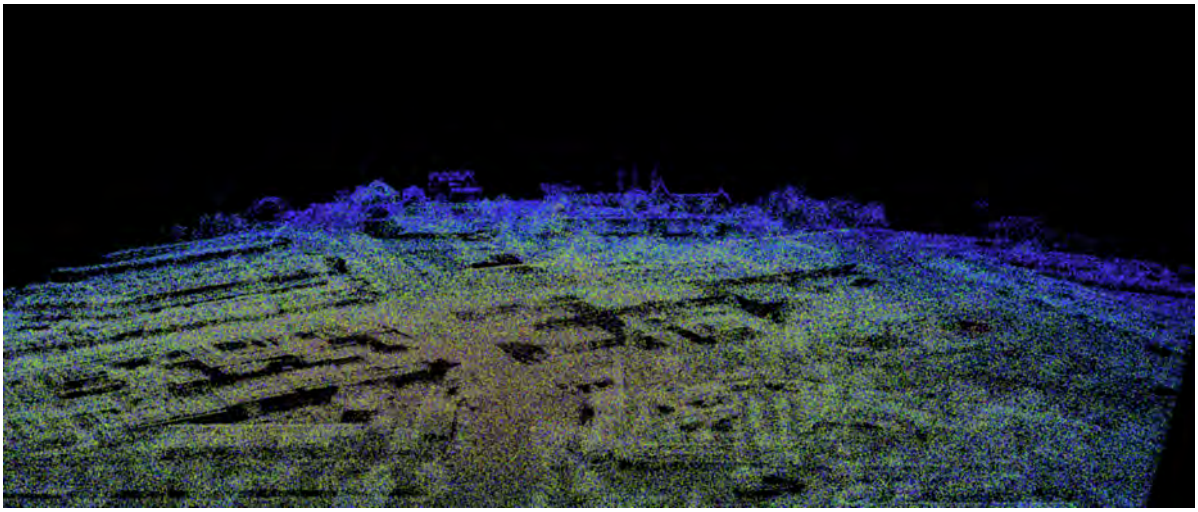


Figure 4.9: Integrated data set processed by distance based algorithm from a bird's eye view

If we zoom in a little more, Figure 4.10 shows the differences in detail between the Potree converted file and the results of the $ExactFast$ computation. The quality of the visualisation has improved using this implementation. Edges are even more preserved, which makes the tower visibly better.

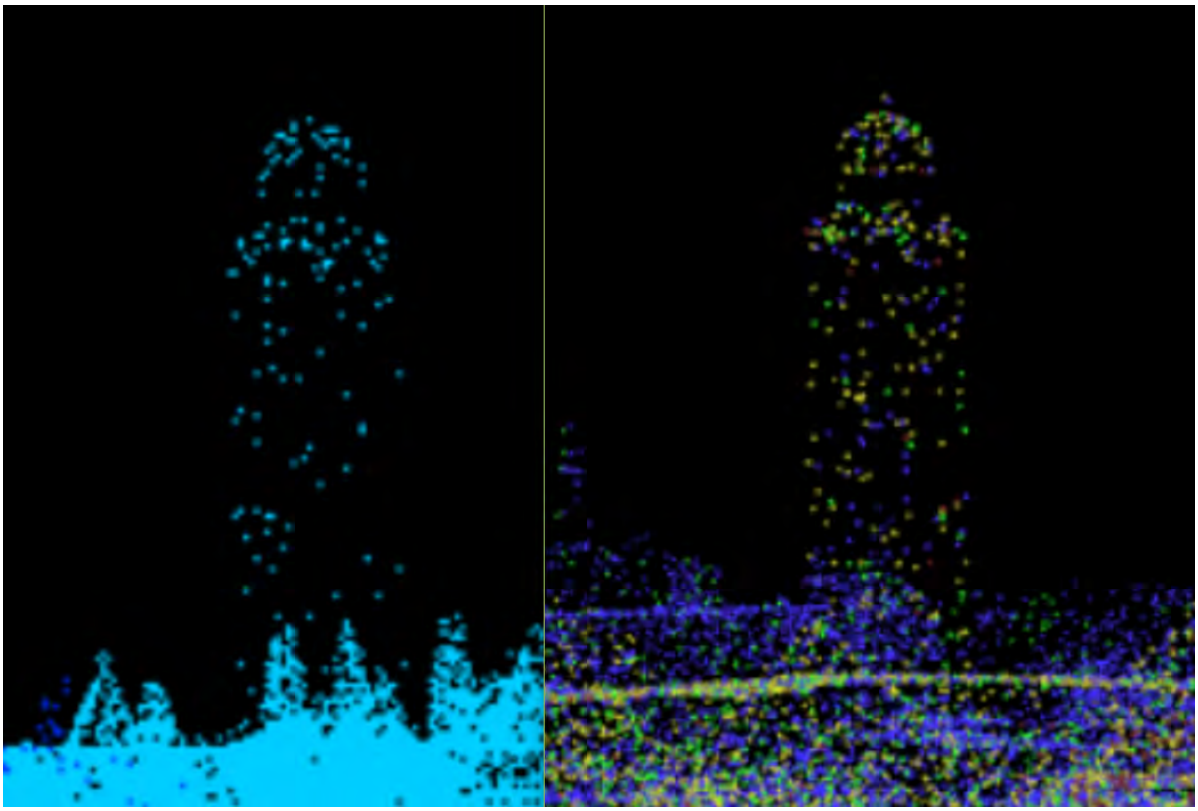


Figure 4.10: Difference in detail between Potree (left) and our implementation (right)

5 Discussion

This chapter discusses the results from the previous chapter. Based on the conclusions, recommendations for future work will be suggested.

5.1. Conclusion

Integration of point clouds can be challenging. These challenges can lie within the *locational aspect*, the *temporal aspect* or the *Level of Detail* the point clouds are presented in. This research focuses on the last one; the integration of point clouds with different scales and granularity. To examine this topic, the following research question is stated: *'To what extend can a vario-scale approach improve integration of point clouds with different point densities?'*. Three main subtopics within this research were specified. Firstly the best suited approach is to be examined. Secondly the processing steps which are needed to transform different point cloud data sets into one point cloud with vario-scale implementation are identified. The third subtopic consists of a performance comparison, to test the proposed implementation with current solutions. To do this performance analyses and to visualise the results of our data model, a web application has been build. This application can be found on `https://github.com/openpointcloudmap/GRIND`.

Research has shown the value of having the points stored in a Database Management System (DBMS), instead of file based storage. This is emphasised in the works of van Oosterom et al. [12] and Cura et al. [3]. There is a point cloud extension for PostgreSQL which handles groups of point, which results in faster query results.

The vario-scale structure with a non-discrete LoD (or importance) is proposed in the works of [12] and Oosterom et al. [7]. Oosterom et al. [7] introduces a new dimension in his paper in order to create a vario-scale point cloud. This dimension could either be *scale* or *importance*. These levels are inherent with the implementation of an *octree*, *kd-tree* or other common spatial indexing method. However, discrete levels will result in non-smooth zooming and when mixing scales in resolution bumps.

Pauly et al. [8] listed three methods to compute a continuous dimension, instead of a discrete dimension. These methods involve clustering, iterative simplification and particle simulation. These methods are often computationally expensive and rather complex. Another way to add a continuous scale dimension is introduced by Wouda [13]. This method adds a random importance value between 0 and 1.

To create the vario-scale approach a fourth dimension (importance) is added to each point in the point cloud. The use case the project focuses on is a continuous LoD output that depends on the distance to the point-of-view (PoV). This implies more detail close-by, and less further away. The added fourth dimension allows for a true vario-scale representation. This enables the ordering of every point for smooth zooming and progressive data transfer. Our research lists four methods for the computation of the importance value. The first method is based on random assignment of importance values; $FullRandom$. The second method is based on $FullRandom$, and modified to cluster the random assignment; $LocalRandom$. There is no visible difference in the vario-scale representation of point clouds when using the $FullRandom$ method or the $LocalRandom$ method. The main differnce between both methods is the total time for which both methods run. The $FullRandom$ is consistently faster compared

to the $LocalRandom$ method and therefore is the preferred method when randomly assigning the importance values. The third and fourth method are exact computations, named $ExactFast$ and $ExactSlow$. These exact algorithms take for every point the smallest distance to a neighbour. This distance is added as importance value. After this the specific point is deleted and the calculation to nearest neighbours is repeated. This iteration is done until all the point are assigned an importance value. To optimise searching for neighbours a temporary index is constructed, specifically the *scipy.spatial.cKDTree*. This is a C implementation of the *KD Tree* for Python.

The proposed data model is implemented on four different data sets. The first data set is the Random Patch consisting of 10.653 points. The second data set is derived from AHN3, consisting of 11 million points. The third data set is a Drivemap provided by Fugro, consisting of almost 50 million point. For the last data set, the AHN3 and the Drivemap are combined into one, consisting of about 61 million points. We refer to this data set as the Integrated data set.

When looking at the results of the created importance values, both the local random computations and the exact computations lead to an exponential distribution of points. The random assigned values vary from 0 to 5. The exact importance values are different for each data set, depending on their point densities. For the AHN2 the values range from 20 to 450 cm. This means the nearest points are circa 20 cm apart, which is therefore the maximum resolution. The exponential distribution for the random algorithm is expected because the distribution of random values is chosen to be exponential. The distribution for the exact algorithms is due to the relatively consistent point density within the dataset. In case of a perfect equally dense dataset the first range of importance value in the varioscale distribution would contain about half of the points. The second range would contain half of that, and so.

Edges of buildings are noticeably better preserved using the exact methods rather than of the random methods. This is shown in Figure 4.5.

When comparing the GRIND implementation to Potree, the most noticable differences is the time it takes to run the .LAS files. Contrary to the GRIND implementation, Potree transforms the input point cloud data set to a file based octree representation. The points are distributed over levels and not just stored in the leaf nodes. Both the $ExactSlow$ and $ExactFast$ methods use a temporary KD-tree which results in a more expensive computation. This results in longer calculation times for our implementation. The longer duration in computation times is to be expected because a fourth dimension is added to the data set, something which Potree does not do. Recommendations on how to speed up the GRIND implementation can be found in Section 5.2.

5.2. Future work

5.2.1. Curvature

In the results chapter the detail views (see Figure 4.3) of both the $ExactFast$ and $ExactSlow$ methods are shown. The $ExactSlow$ method best preserves the edges of the building. However, this edge preserving property can be improved. In the project plan it is suggested that *curvature* could be added to the importance value computation. By computing curvature a factor could be added to the importance of certain points. This would reduce consistency in point density, however for many applications the detection of edges is more important.

5.2.2. pgpointcloud integration

Points from a point cloud data set can be stored as PcPoint objects which can be stored in PcPatch objects (the equivalent of blocks in Oracle). While the current method of storing the points in PostGIS Point objects allows for four attributes (x,y,z,imp), storing points in this format allows for the creation of additional attributes. PcPoint objects are defined in an XML schema, this means they are customisable. An XML schema defining a PcPoint with four attributes (X, Y, Z, Imp) is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<pc:PointCloudSchema xmlns:pc="http://pointcloud.org/schemas/PC/1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <pc:dimension>
    <pc:position>1</pc:position>
    <pc:size>4</pc:size>
    <pc:description>X coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>X</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>2</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Y coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>Y</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>3</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Z coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>Z</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>4</pc:position>
    <pc:size>8</pc:size>
    <pc:description>The importance value is the integer representation
                    of the varioscale indexing. This value is computed
                    to allow for varioscale representation of the
                    point cloud data.</pc:description>
    <pc:name>Imp</pc:name>
    <pc:interpretation>uint16_t</pc:interpretation>
    <pc:scale>1</pc:scale>
  </pc:dimension>
  <pc:metadata>
    <Metadata name="compression">dimensional</Metadata>
  </pc:metadata>
</pc:PointCloudSchema>
```

The points can be loaded into the database in one of two ways. By making WKB objects that adhere to the XML schema that defines the points or by using a PDAL pipeline. The XML schema is referred to with a pcid. The XML schemes are stored in the pointcloud_formats table that holds all the pcid entries and schema documents. Because of this seamless integration with PDAL it is advised to used PDAL for both the storage and retrieval of points from the PostgreSQL database.

5.2.3. PDAL pipeline

Using PDAL for the loading and retrieval of points in PostgreSQL has multiple advantages. The compatibility with pgpointcloud allows for the use of PcPoint and PcPatch objects for the storage and indexing of the point cloud. A PDAL pipeline has multiple advantages in the way it handles complex requests and the fact it is easily to customise. Using the PDAL pipeline creates a record of the applied operation by creating a skeleton which can be created and modified through the json library available for python.

PDAL allows for seamless integration of pgpointcloud in PostgreSQL. pgpointcloud comes with a PDAL writer and reader; writers.pgpointcloud and readers.pgpointcloud. The PDAL pipelines allow modelling of the data from reading to processing and writing. Using a PDAL has multiple advantages

1. Access to the record of operation(s) applied to the data
2. Construct the pipeline skeleton with operations and substitue specific options such as file name, EPSG or boundary.
3. Construction of the PDAL pipeline can be done within any language with JSON manipulation facilities.

An example reader pipeline that reads an uploaded file, assigns each point an importance value and stores them in the database as well as an example writer pipeline which retrieves these points and outputs a .LAS file are shown below.

PDAL reader pipeline

```
{
    "pipeline":[
        {
                "type":"readers.las",
                "filename":"\file.laz",
                "spatialreference":"EPSG:28992"
        },
        {
            "type":"filters.programmable",
            "function":"importance",
            "module":"anything",
                "script":"\importance_pdal.py"
        },
        {
                "type":"writers.pgpointcloud",
                "connection":"host='localhost' dbname='database' user='user' password='
                    password' port='5432'",
                "table":"table",
                "column":"column",
                "compression":"none",
                "overwrite":"true",
                "srid":"28992",
                "pcid":"XML Schema id"
        }
    ]
}
```

PDAL writer pipeline

```
{
    "pipeline":[
        {
                "type":"readers.pgpointcloud",
                "connection":"host='localhost' dbname='database' user='user' password='
                    password' port='5432'",
                "table":"table",
                "column":"column",
                "spatialreference":"EPSG:28992",
                "where":"SQL statement"
        },
        {
                "type":"writers.las",
                "filename":"\file_out.las",
```

```
                "a_srs":"EPSG:28892"
            }
        ]
}
```

Using a PDAL pipeline has clear advantages in terms of ease-of-use. As future work researching the use of the PDAL pipeline for (possibly) optimising both the reading and writing operations is essential. This research would show whether PDAL as a pipeline is beneficial in terms other than ease-of-use such as implementation time and pgpointcloud compatibility.

5.2.4. Clustering of points

The database in its current structure stores individual points. Storage can be improved by storing these individual points in block objects. To improve the query response time pgPointcloud uses $PcPatch$ objects. Since point cloud data typically contains millions of points, collecting these points into a PcPatch object will reduce processing times of, for example, linear scans.

Generally assigning points to patches is dependent on what choices are to be made in terms of indexing and data structure. Index libraries and extensions that support indexing on 4 dimensions of spatial data are limited. Regarding the available research a promising follow-up would be any of the space filling curves in 4D.

These space filling curves will result in a sequence of points that relates to the spatial ordering of all points. This sequence of points can then be split up in patches of each 400 points, the recommended amount of points in a PcPatch. This is a lossy way of splitting up the data into chunks and resulted in our preliminary explorations in overlapping bounding boxes.

Research has been done on collecting points into one patch based on $XYZ - coordinates$ and $importance$ attributes. The most feasible approach in the near future of this project is collecting the points in a manner based on the $BlockRangeIndex$ further explained in Subsection 5.2.5. So instead of building the index, each block is represented by a minimum and maximum value for each of the 4 dimensions. Especially when we consider scaling up the project to billions of points, patches will then still result in millions of rows. Then on top of these millions of patches an index can be built.

An $OpenPointCloudMap$ will be updated and data will be added multiple times. In order to minimise the time of restructuring and recalculating the data, a clear bounding hyper-box is preferable. To make this approach workable, the data-structure has to be modified. Each entry will have at least have a key value, and a PcPatch geometry written as WKB (Well Known Binary) or Extended-WKB. To eliminate redundancy no individual points will be stored in the database. An added advantage of this approach is that spatial queries in PostGIS are still possible with these geometry types, the query response time will be much quicker, and lastly the disk space used for the data will be reduced.

5.2.5. BRIN index

BRIN stands for "Block Range Index" and is a generic form of indexing that has been introduced by Alvaro Herrera in 2013, then dubbed Minmax index [1]. It is first implemented in PostgreSQL version 9.5. BRIN is a lossy index and is mainly used to handle large tables for which some of the columns have some natural correlation with their physical location within the table. This means that it is especially suited for time stamps, but also geographical data. Since an integrated point cloud table will most likely exceed millions of rows, building an index to speed up spatial requests of the data is essential. Compared to the alternative of GiST indexes, BRIN indexes are a less optimal solution but require less RAM, and use less disk-space. This makes BRIN a suitable indexing solution for the current GRIND implementation, which runs on a laptop. An added advantage is that BRIN in the spatial extension PostGIS already supports indexing in 4 dimensions. As the 4th dimension the importance value is used, which will be stored as M in the point geometry object POINT(XYZM). [2] The 4D BRIN index stores the minimum and maximum values of the hyperblock bounding the geometries. In the database these are the rows in a set of table blocks. When applied to millions of rows, a BRIN index increase the query response time by up to 5 times. [3] The use of a BRIN index has no consequences for the logic of the query itself, although it does have consequences for the storage of the geometry type. The

[1] https://www.postgresql.org/message-id/20130614222805.GZ5491@eldon.alvh.no-ip.org
[2] https://postgis.net/docs/using_postgis_dbmanagement.html#brin_indexes
[3] https://wiki.postgresql.org/wiki/What's_new_in_PostgreSQL_9.5#BRIN_Indexes

PostGIS application demands a specific 4d point object in the table. This is an *pointXYZM* object as demonstrated in the query hereunder: a proposal of how a BRIN index can be made from our table.

```
1 CREATE INDEX 4d_brin ON points_table USING BRIN (point_XYZM brin_geometry_inclusion_ops_4d);
```

5.2.6. Interactive viewer

The functions provided by the GRIND web application include the viewing and downloading modules separately. This is mainly because using three.js, the online visualisation of a complete point cloud containing 1.2M points is not well-supported by the browser on a laptop with an average performance, where not only the browser, but also other programmes are slowed down. This limits further operations including zooming, selecting, querying and downloading. For further improvement, it is essential to have the download functions integrated into the viewer, where the user can zoom, rotate, select points, put these in queries and download from a single page.

On the other hand, as is described in Section 3.3.3, the vector from the camera to the target point is limited to one direction due to the controller. Online documentations regarding this is absent, and StackOverflow discussions are also limited. Our solution to find a view point is to switch to a view from the above and locate the observation point manually, whenever the default view is not ideal. However, it would be better to have a more flexible camera thus a better view using three.js.

Organisational Part

6

Introduction

The second part of this report contains the organisational aspects of the project. This organisational aspect is defined in the early stages of the project to frame the project and define the parameters that are crucial for the success or failure of the project. Using this document as a foundation, the research question 'To what extent can a vario-scale approach improve integration of point clouds with varying point densities?' has been answered. The organisational part is organised as follows; Chapter 7 contains the project plan. The project plan consists of an organisational breakdown structure in which the actors and their roles are defined, a work breakdown structure and a work package descriptions and the project schedule in which the different phases and timeline are presented. In conclusion a project logic diagram and rich picture visualise the project, its phases and actors. Chapter 8 identifies the project experts and stakeholders during the Synthesis Project. Furthermore it defines multiple possible uses cases and defines the project use case. Chapter 8 conlcudes with an analysis of the project goals using the MoSCoW framework as well as the functional, non-functional requirements and killer requirements. Chapter 9 elaborates on the critical path. Both the resource allocation as well as the risk map are defined which will allow the project to stay on schedule and prioritise critical tasks.

7

Project Plan

This chapter provides an overview of the project. It will define the organisational structure of the project as well as frame the work structure and the planning aspect.

7.1. Organisational Breakdown Structure

Organisationally two groups can be identified; the project team and the project supervisors. The project team consists of the five team members with their assigned roles and the project mentors. Overseeing the entire project are the project supervisors; the project coordinator, the stakeholder and the master coordinator. The project team works towards delivering a successful product in the form of research supporting the creation of an OpenPointcloudMap. The project supervisors aim to create a successful project for all the actors involved. If successful the overall project will contribute to the OpenPointcloudMap. This organisational structure is visualised in Figure 7.2.

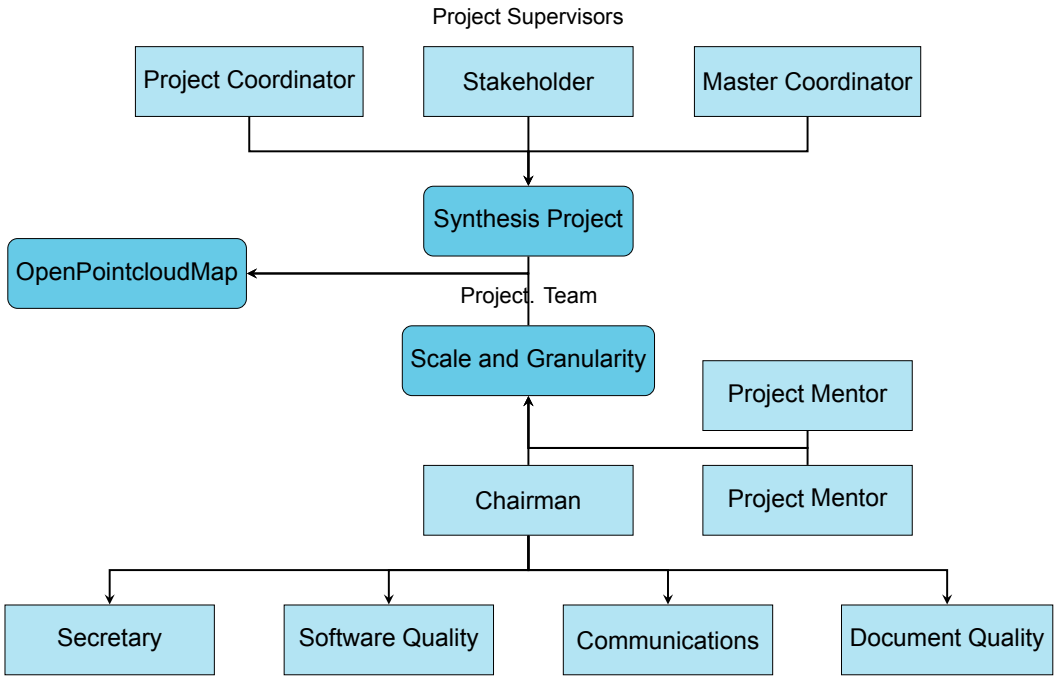


Figure 7.1: The breakdown of the organisational structure

7.1.1. Role definition

Chairman *Tom Hemmes*
The chairman has responsibility and overview of the entire project. To ensure this overview at all times the following responsibilities come with this role. The responsibility for planning and running of team meetings. Managing the distribution of workload among the team members and their involvement in the project. The setup of collaboration tools and encourage usage by the team. Representation of the team and project at events or presentations.

Secretary *Marc-Julien Veenendaal*
The secretary assists the chairman with its organisational tasks, and presides when the chairman is not available. The secretary tracks the progress, the planning, agenda and the schedule. He takes meeting minutes and books rooms if necessary.

Software Quality *Weiran Li*
The team member responsible for software quality integrates the codes and ensures that the codes work properly and function on all the computers. The member is not responsible for writing all the codes, but will mainly help with debugging and making sure that the codes are coherent and comprehensible to the team members and the others.

Communications *Jippe van der Maaden*
The responsibilities of Communications is to communicate with the different actors involved in the Synthesis Project. This includes setting up meetings with the project mentors and communicate with the client. Because three groups are simultaneously working on a complex project involving point clouds he will also keep up communication with the other two groups. Communications will also arrange workshops from third parties and maintain communication with other third parties involved.

Documentation Quality *Brenda Olsen*
This team member is responsible for the quality of all the written documentation. This includes creating a set up for all documents and making sure everyone is able to participate in writing these documents. During the project and before submitting, the member will check whether the documents are conformable and make changes if needed, after consultation with the team. Fixing content related errors is not included in this members responsibility.

Project mentors *Martijn Meijers, Theo Tijssen*
The project mentors will supervise the progress and attend weekly meetings. The project team expects the mentors to assist the team with question regarding handling large data sets, management of massive point cloud data, geo-database management systems and other fields of expertise that apply to the project scope.

Client *Fugro*
The client, Fugro, provides the team with data sets and expert knowledge to support the team. If needed the project team will discuss with Fugro the use of their resources such as the Amazon Web Service (AWS). Supervisor Stella Psomodaki will occasionally attend group meetings allowing the group to inform Fugro of their progress.

7.2. Work Breakdown Structure

All tasks for the project team are determined and organised in the Work Breakdown Structure, as shown in Figure 7.2. The project work is divided in six main tasks: research, reporting, storage, indexing, processing and visualisation.

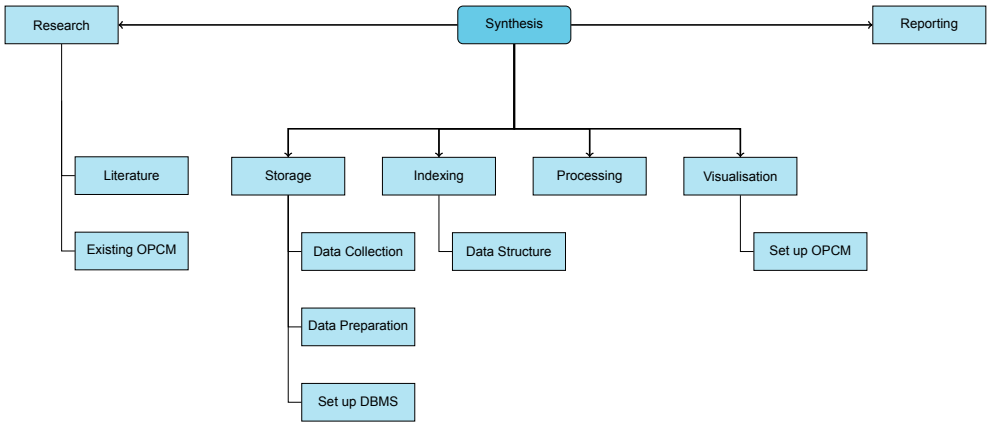


Figure 7.2: The breakdown of the work structure

Research will be done throughout the project as it is needed in all phases as shown in Figure 7.3. It will be done mostly in Phase 2, Research, since this phase researches the possible research directions.

Throughout the project Reporting will be an important aspect since research is meaningless without proper documentation. The document quality will be protected by Brenda Olsen during the whole of the project.

The four remaining tasks, storage, indexing, processing and visualisation are what make up the main focus of our research. These tasks and their sub-tasks will be explained in more detail in the Work Package Description.

7.3. Work Package Descriptions

This section elaborates on the work packages as presented in Figure 7.2 and their corresponding tasks. The explicit specification of these tasks is necessary in order to perform efficient work distribution and a continuous work flow.

Storage When storing data sets from multiple sources multiple operations are needed to efficiently store the cumulative data set. The data should be collected, prepared and stored. The work package therefore includes the data acquisition. Because the scans are acquired already, this is a simple matter of requesting the correct data sets (which has already happened at the moment of writing). Next is the decision for what type of database to use and running the setup. Setting up the database should be done in a location with future processing in mind. This means, possible growth in storage size after processing and connection to the server that will perform processing. Besides determining the optimal data structure (see **Index**) the data also may need some additional processing.

Index A structure for the data sets would be beneficial for future processing. The vario-scale data structure should enable the integration of multiple data sets. This work package contains work on the implementation of a data model for vario-scale representation. Such a model would incorporate a hierarchy based on importance for points in the data set; A spatial index for fast retrieving of data. The current implementation will remove all attributes that are stored, apart from the x,y,z coordinates and the importance value. Future work should include compatibility for multiple and varying attributes since most Lidar data contains much more attributes than just x,y,z coordinates.

Processing The processing work package consists of tasks that implement the created approach of creating an integrated vario-scale point cloud. Most data sets will either be raw point clouds or, like the AHN, already have a data structure. To cover both aspects, a server and software environment is needed, where functions for adding or removing attributes that are specified in the data model should be implemented, and the hierarchy and spatial index are computed. A script that handles all the functionality and executes the whole process is generated.

Visualisation Visual inspection is one of the aspects for final evaluation of the methodology. To implement the option for visualisation, a download option will be added to the web client. This means it should have the functionality to select a region and define a resolution parameter in a web client. This enables the user to view the integrated data set in any viewer. The implementation of a viewer within the web client is beyond the scope of this research and a possible future improvement.

7.4. Project Schedule

7.4.1. Meetings

Weekly team meetings are on Monday, Wednesday and Friday. The stand-up meeting on Wednesday is attended by mentors Martijn Meijers and Theo Tijssen. The team meets with Stella Psomodaki (supervisor from Fugro) on a bi-weekly basis. During these meetings the daily tasks will be discussed and the team will be informed on the progress made by other team members.

7.4.2. Phases

The project is divided in three phases. These phases are defined in the Project Guide for the synthesis project and illustrated in Figure 7.3.

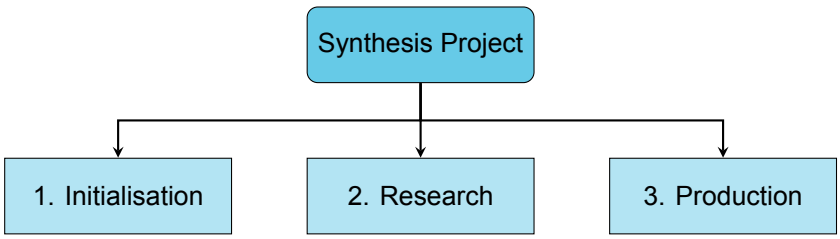


Figure 7.3: Project phases

The Initiation phase entails the creation of a project plan, this plan can be seen as a contract for all actors. Other deliverables for the first phase include the requirement specification and the rich picture. Both of these deliverables provide the team with more insight in the process of working towards the project goal.

After these deliverables are created and discussed with the mentors the Research phase will start. This phase will focus on determining the possible direction based on existing research and the stated research goals. To do this the team will identify multiple methods to work towards these research goals. This process will lead to trade-offs and prioritisation. The result of this phase will be presented at the Mid-term Review.

Finally, the team will work on implementation of the theory and create a proof of concept in the Production phase. This extent of the implementation will be determined during the review of the deliverables and a discussion with the mentors.

These phase are within the time frame of the study project. However, in the past it has occurred that projects continued due to high interest of involved actors. If so, a new time line will be created.

7.4.3. Time line

The following chart (Figure C.1) contains a time line for this project. The overview includes the larger tasks that are to be done.

7.4.4. Planning

The sequential order of the time line indicates dependence of successive tasks on previous work. Whenever a task shows not feasible, the research tree provides an alternative.

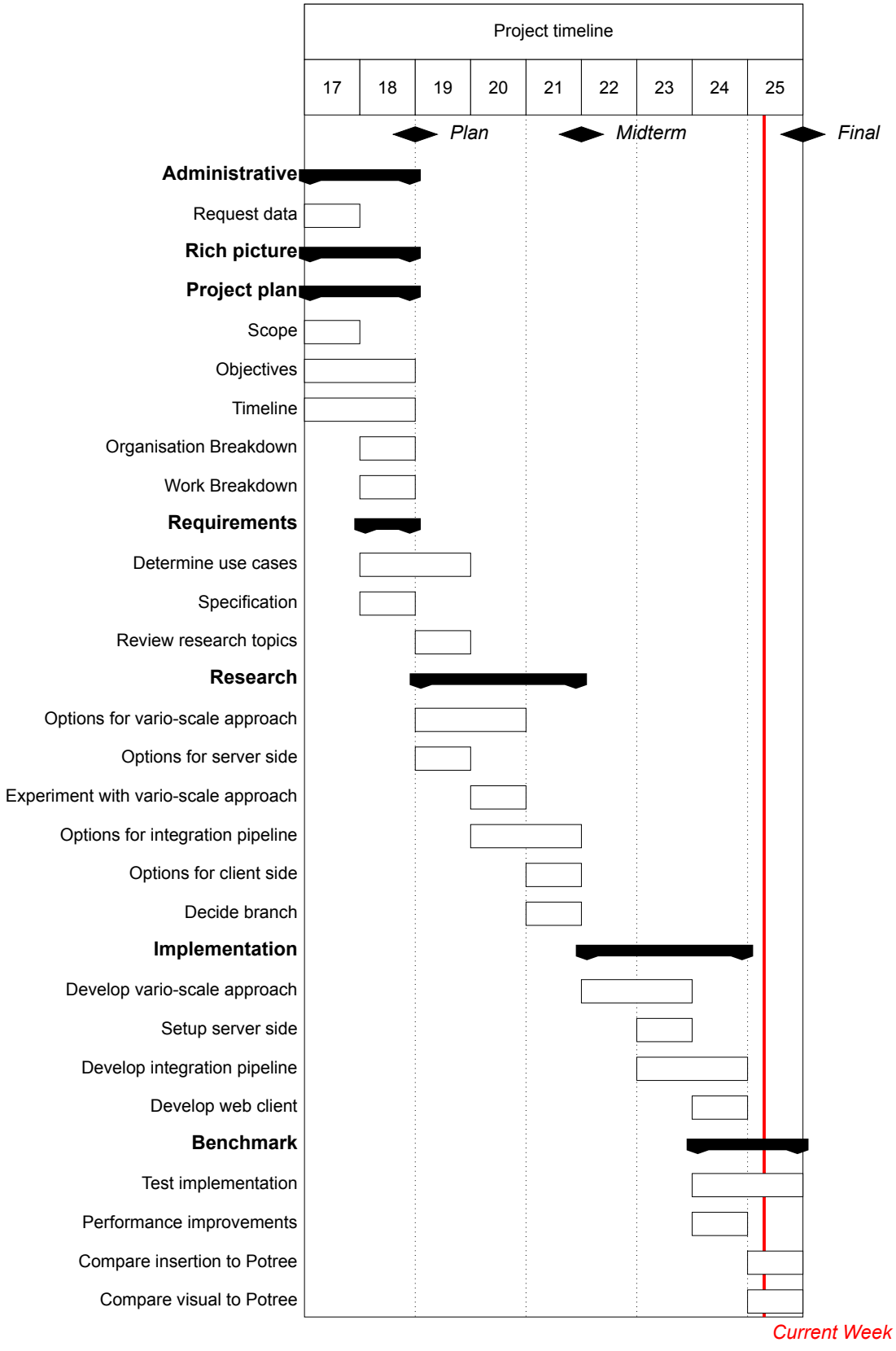


Figure 7.4: Gantt chart of project timeline (Week 17 - 25)

7.5. Project Logic Diagram

In this section the Project Logic Diagram is shown in Figure 7.5. The diagram gives an overview of the activity flow, phases, milestones and schedule. It is shown what data comes in in what phase during the project.

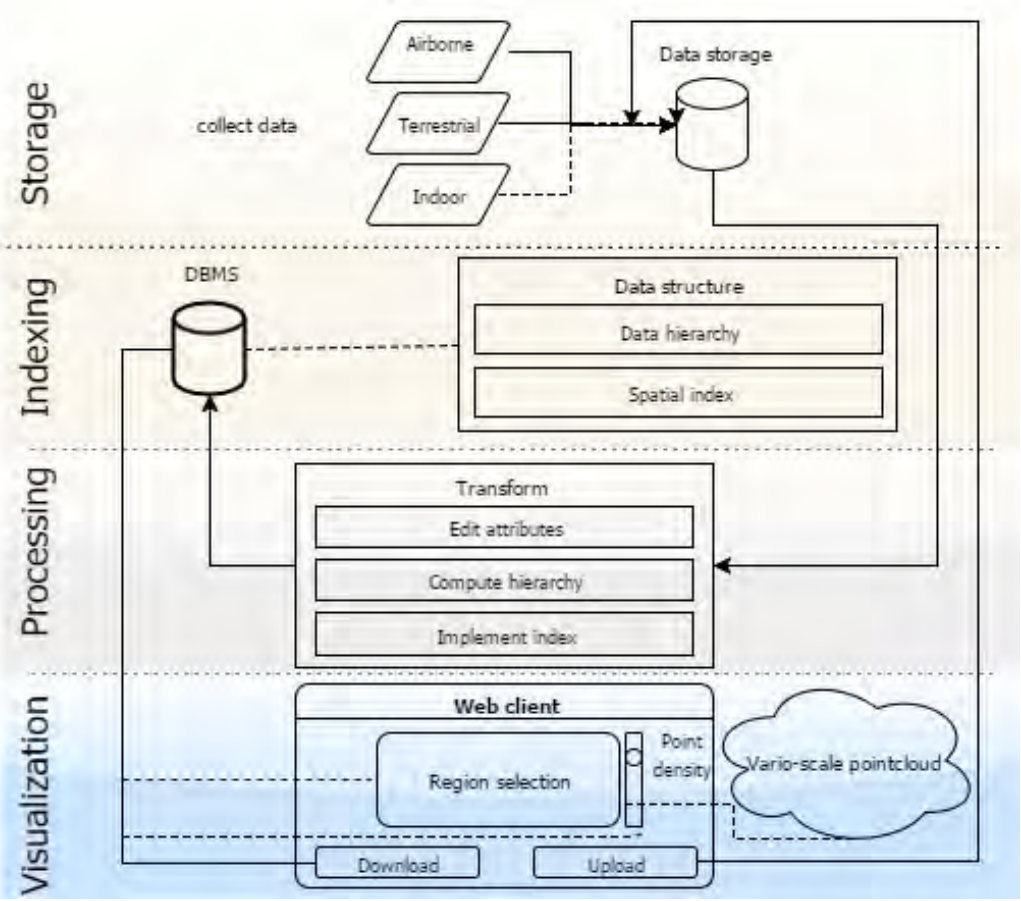


Figure 7.5: Project Logic Diagram

7.6. Rich Picture

The rich picture that is created for this project can be seen in Appendix D. The picture gives an overview of all actors and stakeholders involved in the project.

8

Requirements

The requirements are documented to give a clear picture of the aim of this project. This aim is focused on the wishes of the project team, the specified stakeholders and other potential users. In the end a mutual vision will be created, which meets both our expectations and those of the stakeholders.

To be able to do this, first the stakeholders and experts will have to be identified. Then possible use cases are described. After this the goals of this project are projected in a MoSCoW table, which divides them into must, should, could and will not. From thereon the requirements can be specified and divided into functional and non-functional requirements. These requirements, together with the killer requirements, can be found in the third section. The project, as a part of the Geomatics Synthesis Project, is bounded by multiple aspects, like time or place related issues. Those boundary conditions are defined in the last section.

8.1. Identify Experts & Stakeholders

8.1.1. Experts

The complete synthesis project is managed by the course coordinator. The synthesis project is divided into two subjects: Internet of Things and Point Clouds. This last one is subdivided over three project groups. Those three groups, including us, are supervised by the project coordinator. Every group has two project coaches linked to them, who will guide the team throughout the whole project.

Course coordinator: *Stefan van der Spek*
Project coordinator: *Peter van Oosterom*
Project coaches: *Martijn Meijers $ Theo Tijssen*

8.1.2. Stakeholders

The main stakeholder of this project is Fugro. They provide the team with datasets and expert knowledge. Meetings will be scheduled if needed to achieve an agreeable vision on the purpose of this project.

Stakeholder: *Fugro*
Contacts: *Stella Psomadaki & Martin Kodde*

8.2. Use Cases

During this project research will be done on a vario-scale implementation that improves the integration of point clouds with varying point densities. A true vario-scale structure should provide a continuous and smooth way of zooming through all wanted scales. However, a vario-scale point cloud platform can have different use cases. First some possible use cases will be briefly described in section 8.2.1. In section 8.2.2 our chosen use case will be described. Following, section 8.2.3 will elaborate on the usability.

8.2.1. Possible Use Cases

- *Consistent point density cloud*
  The approach for integration of point clouds will strive for a consistent point density. To achieve this a vario-scale structure is needed. After implementing this structure it is possible to generate a view of the combined data sets in any point density. In this way, the point cloud will be vario-scale and is able to provide an equal density on every scaling level. This will enable users to download the point cloud in any file size suiting their computational resources. Creating an equal density throughout the point cloud, can also ensure faster rendering, which will be good for visualisation purposes.

- *Multiple point densities cloud (depending on data set)*
  A downside of the above mentioned use case is the loss of data. Keeping an equal density means a lot of points from for instance terrestrial or indoor point clouds will have to be removed. This can be prevented by applying the vario-scale approach on every data set separately and then merging the data. In this way, all the data is vario-scalable and still has its own density (and thus level of detail). This method most probably ensures faster uploading and integration of new data sets.

- *Multiple point densities cloud (depending on region)*
  When downloading point cloud data for a specific region the selected area might not be equally important. In this case the user should be able to highlight specific areas for higher point density and other areas for only sparse coverage. Supporting multiple point densities in one point cloud would mean the server should have an interpretation to select multiple regions.

8.2.2. Our Use Case

- *Variable density cloud projection*
  For viewing point clouds it is hard to create a visually appealing representation. The ability to derive data with variable point density from the database enables atmospheric perspective. This means more points close to the point of view and gradually less points for increasing distance.

8.2.3. Usability

Having a variable point density cloud, means the user of the platform has to specify a location (viewpoint). The input for this location is a point, consisting of X,Y,Z coordinates. From this point, the viewer looks over the point cloud. The density of this point cloud decreases as the distance to the viewpoint increases.

8.3. Requirements

8.3.1. MoSCoW

A project can consist of a long list of goals. In able to specify the most and the least important ones, they will have to be prioritized using the MoSCoW method. This method distinguishes the things that must, should, could and will not be done during the project. The MoSCoW table is shown in Figure 8.1 .

Must	**Should**
• One vario-scale approach • Injection of data • Input data transformation module • Consistent point density • Variable granularity support • User Interface • Performance comparison to Potree • Download button	• Web visualization through a platform • Granularity slider
Could	**Will not**
• Distributed storage • Distributed processing • Integration of indoor point clouds • Improve processing time • Improve GPU load • Improve storage size • File size indication for download	• Specific structure for every type of dataset • Support for multiple region selection • Remote integration of multiple point cloud datasets

Figure 8.1: MoSCoW method

8.3.2. Functional Requirements

This section consists of the requirements that will make up this project. These requirements are subdivided in functional and non-functional. Functional requirements include calculations, technical details, data manipulation and processing. A functional requirement specifies what a system should do. A non-functional requirement on the other hand, specifies how a system should perform a certain function. These requirements include for example performance, security, reliability, availability and accessibility. (Eriksson, 2015)

1. Access to multiple point clouds with different scale and granularity.
2. Creation of a DBMS.
3. Determination of importance value per point for varioscale classification.
4. The efficient storage and indexing of point clouds in a varioscale storage structure.
5. Ability to specify the density of the desired point cloud.
6. Download functionality through efficient querying.
7. Connection between the DBMS and the user interface through which the desired point cloud can be downloaded.

8.3.3. Non-Functional Requirements

1. The goals as created in collaboration with the stakeholders shall be met.
2. The data-structure shall be populated with data in a specified data-format from different sources with a limited amount of errors and correct error handling.
3. The environment shall provide a retrieval time of point cloud data that is comparable with current benchmarks.
4. The environment shall run in a stable manner when performing queries.
5. Point clouds of different scales shall be integrated into a uniform point cloud.
6. The product and its functionalities can be implemented and integrated with current products/tools of Fugro and TU Delft.
7. The resulting product can be combined with the products of both of the other two teams. Preferably in one framework, resulting in a platform with functionalities resembling an OpenPointCloudMap.

8.3.4. Killer Requirements

Some of the above listed requirements are essential for a sufficient outcome of this project and could be a risk for the completion of the project. These are defined as 'killer requirements' and are listed below.

- **Data source**
  To successfully implement the data structure and indexing, the point cloud should contain basic properties such as coordinates and metadata. Therefore, the data source is essential for the completion of this project. For this project, the data is acquired from Fugro because of a time limit.
- **Research**
  In the phase of research, it is important that sufficient studies are found and the methods are understood. The fulfillment of this requirement will lead the project to a correct direction.
- **Performance comparison**
  To assess the performance of the outcome of this project, parameters such as time of insertion and time of retraction are needed. A self-defined criterion is hardly objective or reliable, so a comparison with the existing platform, Potree, is needed.

8.3.5. Boundary Conditions

The project GRIND is within the generic topic Smart(er) Environments of GEO1011. It runs from April 21 to June 23, when the final outcome will be presented. Working activities will be mainly carried out in the faculty of architecture of TU Delft. In addition, a workshop will be organised in the Fugro branch in Leidschendam.

The data used for this project includes point clouds collected by airborne laser scanning and terrestrial laser scanning, and an optional indoor point cloud. Fugro, as the sponsor of this project, will provide the data. They will also offer this project technical support.

The goal of GRIND is to research on the possibility to establish an Open Point Cloud Map in the aspect of vario-scale point cloud integration. Together with other projects covering the aspects of different time-series and multiple locations, this project will in return help Fugro create their product in the future. The project will be conducted with the help of the project coordinator and coaches from Geomatics programm of TU Delft.

The legal aspect cannot be overlooked throughout this project. To implement an anticipated new algorithm to integrate the point clouds, related research and previous work will be needed. It is essential to pay attention to whose work is used, which type of license the work is under, and to what extent we can use the work.

9 Critical path

This chapter will elaborate on the critical path to follow.

9.1. Timeline

In the appendix the Gantt chart (Appendix C.1) contains a timeline for this project. The overview includes tasks that have been done (~~strikethrough~~) and tasks that are still to be done.

9.2. Resource allocation

The sequential order of the timeline indicates dependence of successive tasks on previous work. Currently the project is in its research phase, which is projected to be finished by the start of week 22. This leaves four weeks to finish the production phase of the project. In this phase three tasks will be performed simultaneously, in the Gantt chart (Appendix C.1) referred to as Data model, Client server and Data insertion. The fourth task is the Benchmark, this task relies heavily on the previous three.

In this section the resource allocation for each task will be explained. This means the dependencies will be elaborated on as well as the designated executor and its responsibilities.

9.2.1. Research

In the Gantt chart (Appendix C.1) the research tasks that have been performed are filled in with gray. This leaves just the experimenting with the vario-scale approach to be done. Options for the vario-scale approach, server side development, client side development as well as the integration pipeline have been elaborated upon in this report. Finishing the Research phase will require two team members. This leaves three team members to start working on the production phase of the project.

9.2.2. Data model

Implementation of the data model is the main focus of the project and should be treated as such. This means this task will require constant evaluation by the team and should be monitored closely for any problems that might cause delays or otherwise impede the project. Progress has been made during the research phase in defining the data model, from week 22 onwards this data model definition will be translated into a working data model.

Because this task is critical for the project it is split up in two phases. The first phase is to create an algorithm that randomly assigns importance values to the individual points. This algorithm will give the project a working data model and allows other tasks to advance regardless of the other developments or complications in the data model task.

The second phase is to create an algorithm that assigns importance values not randomly, but based on the neighbours. This takes into account the scale & granularity of the dataset. The final data model which will be worked towards is shown in section 3.2.

9.2.3. Client server

Significant progress on the development of the client server has been made by moving his task forward, this is indicated by the Gantt chart (Appendix C.1). In week 19 the development has commenced by assigning the task to one team member. The progress that has been made indicates that this task does not require extra attention and has a low risk factor of being critical for the projected progress. Development of the client server is explained in section 3.3.

9.2.4. Data insertion

Research into the data insertion has been done during the Research phase and is documented in subsection 5.2.3. The progress that has been made on the creation of this pipeline suggest that this task has a low risk factor of being critical for the projected progress. When this task is at risk the designated team member will reach out to either a mentor or the project coordinator who have experience with PDAL and will thus be able to assist in the creation of the pipeline. Contact information is listed in section 8.1.

9.2.5. Benchmark

The benchmark will take place in week 23 since it is highly dependant of all previously mentioned tasks. This means these tasks will have to have working product by week 24. This gives all tasks two weeks, with additional adjustments to be made during week 24 since it is reserved for Performance Improvements. Week 25 is set up for the benchmark testing, at this time all other tasks are done and comparison to the current standard is the only task left. All team members will assist in the performance benchmark since errors that may arise will require specialty knowledge. Extra time will be allocated to documentation and presentation of the results

9.3. Risk map

The following risks were identified during the project plan phase.

1. **No data is provided by Fugro in time.**
   Making the request for data first priority we assured to have the data at the start of the project. Alternatively, we would have used an academic dataset such as the *Paris RueMadame database* `http://cmm.ensmp.fr/~serna/rueMadameDataset.html`

2. **Upload and download speeds of the point clouds are extremely slow.**
   This risk is inevitable and optimization is of low priority. However, this could be solved by using either *chunk* uploading or to *thin* the point cloud at the moment of uploading. A simple file size limitation would also suffice for this experimental implementation.

3. **The visual impact of the variable density is high, the point cloud becomes unrecognizable.**
   In this case we can scale the points according to their distance to the viewpoint. This will improve visual appearance but not the data load.

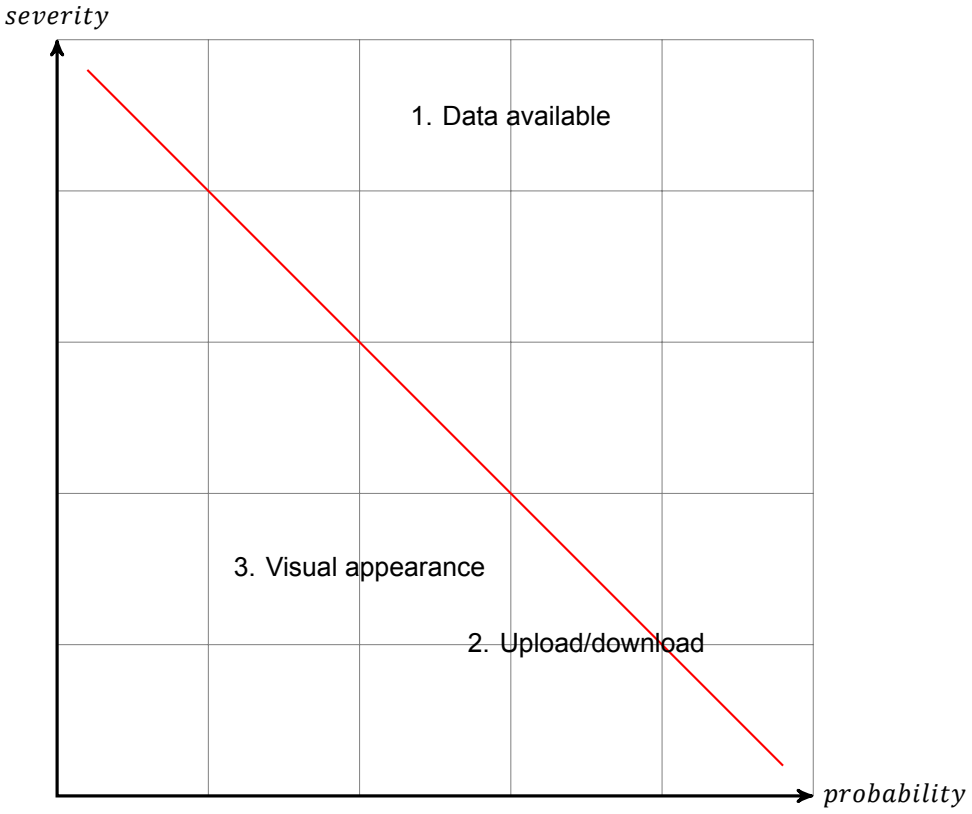


Figure 9.1: Risk map of project tasks

A

Data Specifications

name	AHN2 Clipped	AHN3 Clipped
file format	.las	.las
file size	30,7 MB	38 MB
points	1.239.907	1.141.742
min xyz	84896.13 446362.67 -7.89	84896.133 446362.666 -10.671
max xyz	85433.72 447079.95 104.08	85433.719 447079.950 109.192
covered area (km)	62,5	62,5
Attributes X,Y,Z	+	+
intensity	-	+
number_of_returns	-	+
return_number	-	+
classification	-	+
scan_angle_rank	-	+
point_source_ID	-	+
gps_time	-	+
R,G,B	-	+
density p/m2	8	8
file creation d/y	167/2015	63/2014
scale factor xyz	0.01 0.01 0.01	0.001 0.001 0.001

Figure A.1: Data Specifications AHN

name	2013_TUDelft	2016_TUDelft
file format	.las	.las
file size	1,68 GB	1,57 GB
points	53.273.939	49.737.433
min xyz	84942.304 446394.125 -17.298	85186.740 446855.489 -0.768
max xyz	85433.718 446985.093 47.687	85432.128 447062.925 57.353
covered area (km)	9	9
Attributes X,Y,Z	+	+
intensity	+	+
number_of_returns	-	-
return_number	-	-
classification	-	+
scan_angle_rank	-	-
point_source_ID	-	-
gps_time	+	+
R,G,B	+	+
density p/m2	?	?
file creation d/y	200/2015	120/2016
scale factor xyz	0.00000049 0.00000059 0.000000065	0.001 0.001 0.001

Figure A.2: Data Specifications FUGRO

B

Python web frameworks

Name	Latest update	Documentation	Community (questions on StackOverflow)	Database	Configuration
Django	04-04-2017	Very detailed and clear documentation; a step-by-step tutorial is provided and further documentations are introduced.	143,211	SQLite 3, PostgreSQL, MySQL, Oracle and other databases.	The configuration on Windows can be a little complicated; the settings in general are very simple.
web2py	10-05-2016	The documentation is too simple; insufficient tutorial provided.	4,708	SQLite	The type of supported databases depends on operating systems, otherwise extra configuration according to the specific database is needed.
TurboGears	04-12-2016	Relatively complete documentation; example tutorials can be more clear.	726	SQLAlchemy and MongoDB	To keep the original Python packages in order, a virtual environment needs to be installed specially for TurboGears.
Tornado	16-04-2017	Focuses on functions and integration with other services; the navigation within the website is not clear.	8,745	N/A	To connect to databases, external libraries always need to be configured.
Flask	31-03-2017	Neat documentation including a tutorial, templates, debugging instructions etc. Every section is simple. Switch between sections is not trivial, with only previous section and next section on the corner of the sidebar.	42,779	SQLite 3 and SQLAlchemy	The configuration of Flask is simple; external libraries Jinja2 and Werkzeug can be required, but the configuration is simple as well; instructions of configuring PostgreSQL on Windows are insufficient.
Bottle	09-01-2017	A very simple documentation introducing only the basic contexts of the framework.	8,373	SQLite 3 and SQLAlchemy	An virtual environment may be needed.

Figure B.1: Comparison of Python web framework

C

GANTT

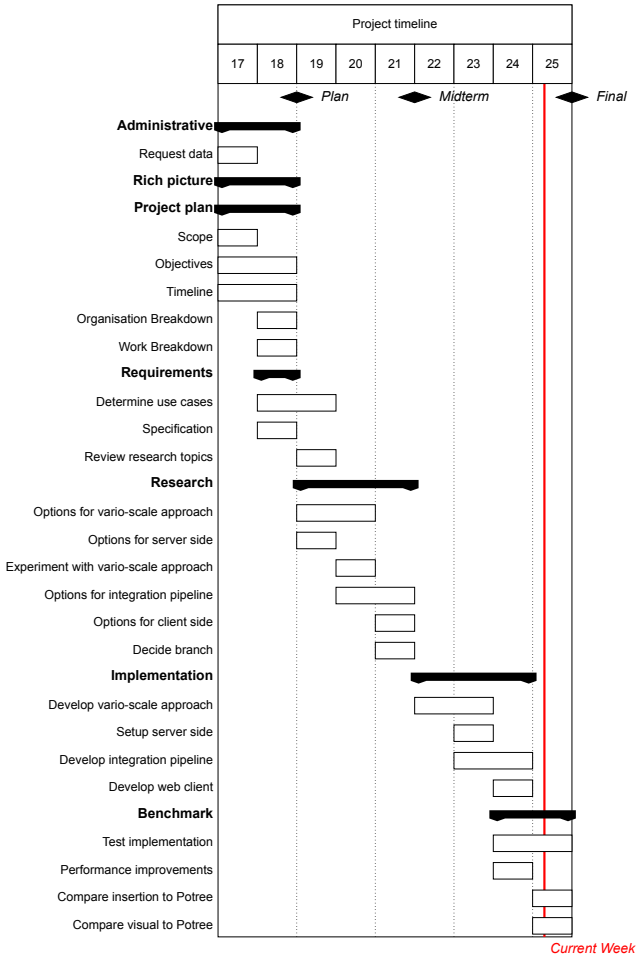


Figure C.1: Gannt chart of project timeline (Week 17-25)

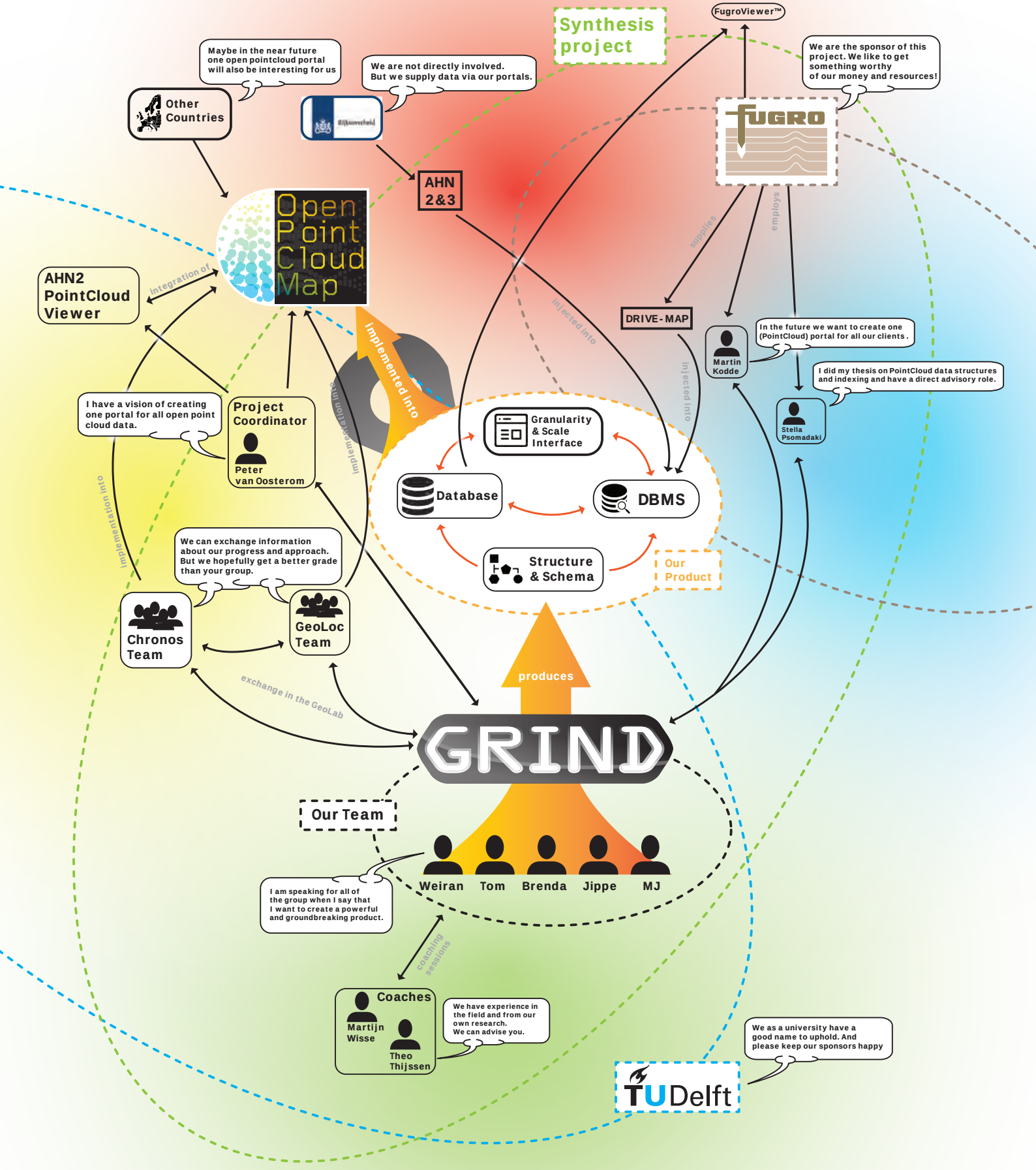
D

Rich Picture

DID - B3: Rich Picture

Tom Hemmes, Weiran Li, Jippe van der Maaden, Brenda Olsen, Marc-Julien Veenendaal

GRIND



Bibliography

[1] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1): 51–72, 1986. ISSN 07300301. doi: 10.1145/7529.8927.

[2] R. Cura, J. Perret, and N. Paparoditis. Point Cloud Server (Pcs) : Point Clouds in-Base Management and Processing. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W5:531–539, 2015. ISSN 2194-9050. doi: 10.5194/isprsannals-II-3-W5-531-2015. URL http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/II-3-W5/531/2015/.

[3] Rémi Cura, Julien Perret, and Nicolas Paparoditis. A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing. *{ISPRS} Journal of Photogrammetry and Remote Sensing*, 127:–, 2016. ISSN 0924-2716. doi: http://dx.doi.org/10.1016/j.isprsjprs.2016.06.012. URL http://www.sciencedirect.com/science/article/pii/S092427161630123X.

[4] L Kobbelt, L Kobbelt, S Campagna, S Campagna, H.-P. Seidel, and H.-P. Seidel. A General Framework for Mesh Decimation. *Graphics Interface*, pages 43–50, 1998. ISSN 07135424. URL http://visinfo.zib.de/EVlib/Show?EVL-1998-250.

[5] Oscar Martinez-rubi, Stefan Verhoeven, Maarten Van Meersbergen, and Markus Sch. Taming the beast : Free and open-source massive point cloud web visualization Taming the beast : Free and open-source massive point cloud web visualization. *Capturing Reality Forum 2015*, (March 2016): 23–25, 2015. doi: 10.13140/RG.2.1.1731.4326.

[6] B. M. Meijers and P. J. M. van Oosterom. the Space-Scale Cube: an Integrated Model for 2D Polygonal Areas and Scale. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-4/(Udms):95–102, 2011. ISSN 16821750. doi: 10.5194/isprsarchives-XXXVIII-4-C21-95-2011.

[7] Peter Van Oosterom, Martijn Meijers, and Jantien Stoter. *Abstracting Geographic Information in a Data Rich World*. 2014. ISBN 978-3-319-00202-6. doi: 10.1007/978-3-319-00203-3. URL http://link.springer.com/10.1007/978-3-319-00203-3.

[8] M. Pauly, M. Gross, and L.P. Kobbelt. Efficient simplification of point-sampled surfaces. *13th IEEE Visualization conference.*, (Section 4):163–170, 2002. ISSN 10702385. doi: 10.1109/VISUAL.2002.1183771.

[9] pgPointCloud. pgPointCloud, 2014. URL https://github.com/pgpointcloud/pointcloud.

[10] Markus Schuetz. Potree : Rendering Large Point Clouds in Web Browsers by. 2016.

[11] Markus Schütz. PotreeConverter -Uniform Partitioning of Point Cloud Data into an Octree. URL http://potree.org/downloads/converter_documentation.pdf.

[12] Peter van Oosterom, Oscar Martinez-Rubi, Milena Ivanova, Mike Horhammer, Daniel Geringer, Siva Ravada, Theo Tijssen, Martin Kodde, and Romulo Gonçalves. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. *Computers & Graphics*, 49:92–125, 2015. ISSN 00978493. doi: 10.1016/j.cag.2015.01.007. URL http://dx.doi.org/10.1016/j.cag.2015.01.007.

[13] Berend Wouda. Visualization on a Budget for Massive LiDAR Point Clouds. 2011.