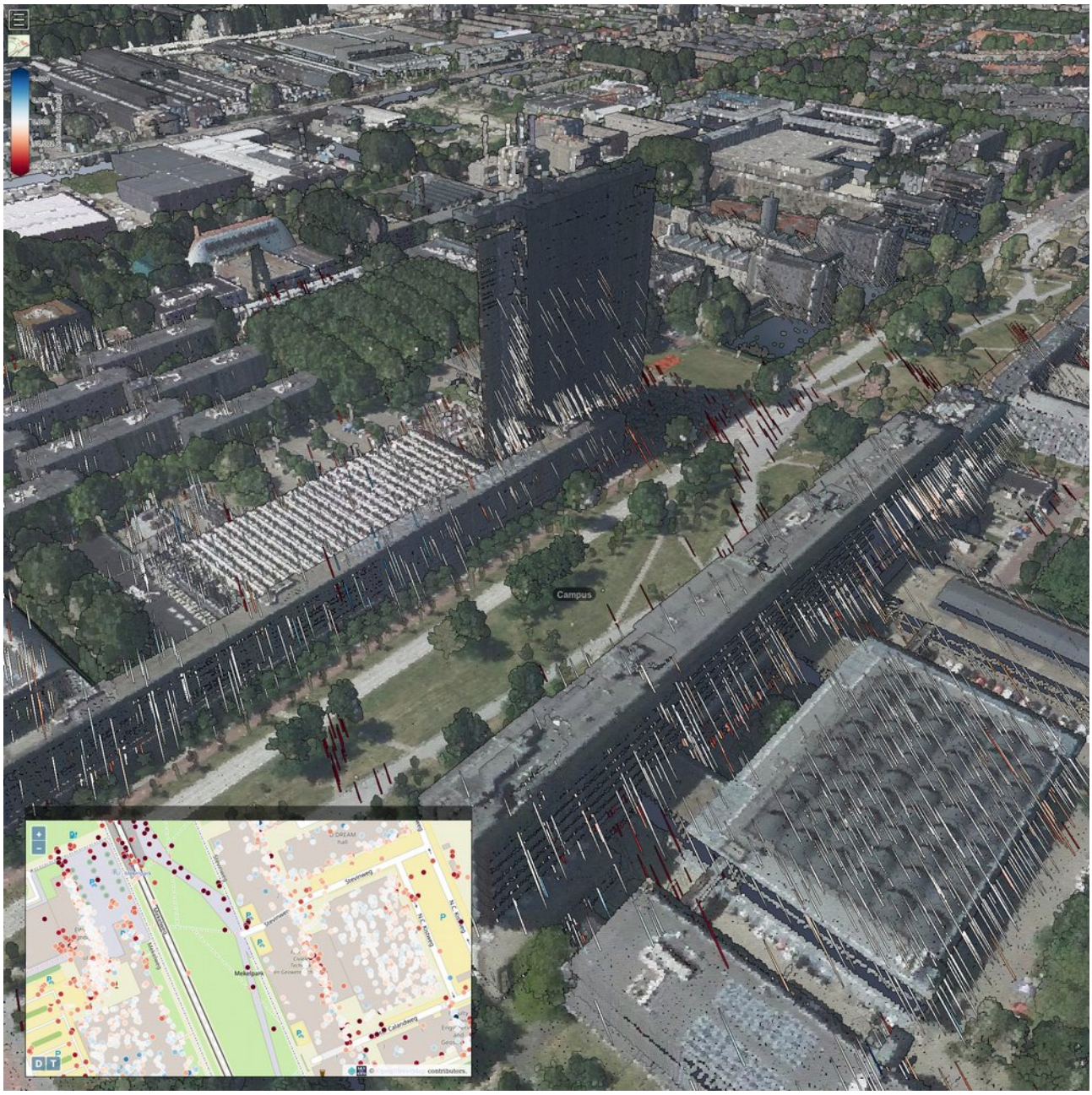
Web based visualisation of 3D radar and LiDAR data



Additional Thesis (`AES4011-10`)

Adriaan van Natijne <A.L.vanNatijne@student.tudelft.nl>
Supervisors: Prof. dr. ir. R.F. Hanssen
Dr. R.C. Lindenbergh

Assessment committee:
Ramon Hanssen & Roderik Lindenbergh

As this report is intended as a manual too, footnotes are used as links to online resources, as those resources are more spread out than conventional literature.

Figure 1 (on the cover): the completed application showing radar observations with their error ellipsoids. An explanation of the features shown can be found in chapter 6.A (page 43).

Abstract

Laser scanners and Interferometric SAR both create point clouds. Their density, position accuracy, varying on the sensor. To improve the interpretation of the low density InSAR data a combination is made with high density airborne and terrestrial laser scanner data. Allowing for improvement of the positioning of the scatterers and the understanding of their behaviour.

This combination is implemented as a web application, suitable for researchers to create 3D visualisations for the greater public. Challenges are: the high volume nature of point clouds; inhomogeneous coverage of laser measurements; different coordinate systems and limited processing power of web-browsers.

All datasets were brought to the same coordinate system and where possible enriched with other sources such as aerial photographs and maps. Tiling was applied to limit downloads and processing requirements at the web-browser. Clustering of InSAR data may be applied to group points with similar behaviour while preserving unique features in the data.

Results of this work are a demo application, this report and a manual on how to make a similar application based on a combination of yet existing tools. These visualisations will allow for a new approach in InSAR analysis, integrating measurements with their surroundings.

Contents

1. Introduction

Interferometric Synthetic Aperature Radar (InSAR) can be used to monitor deformation from satellites. Millimetre (per year) accuracy can be achieved in deformation trend estimations. Unfortunately the source of the deformation signal is less accurately known. The location of the scatterer can be of great importance to understanding and valuing the deformation behaviour: a subsiding garden house or street will require different precautions than a subsiding bridge pillar.

Radar measurements are often dominated by a single scatterer. In those cases the reflection may be attributed to a distinct feature. Unfortunately this relation is not always clear. To find and demonstrate the dominant scatterer it is beneficial to combine radar measurements with a (high resolution) point cloud. This will allow for linking scattering behaviour to a geometric feature in the scene.

To illustrate this, the effects of the error in the position estimate are shown in Figure 2. The estimated position of the scatterer is in mid-air. As such scattering behaviour in air is unlikely at best, there are three candidates for the (dominant) scatterer.

To decide which of them is most likely the estimates of the positioning error may be used, indicated here as a blue ellipsoid. Option 3 is the most likely (maximum likelihood) option, as positioning the scatterer at this point will require the least possible error (compared to the estimated position) of the three options. Estimates for the size of the errors are provided by P. Dheenathayalan et al.[33].

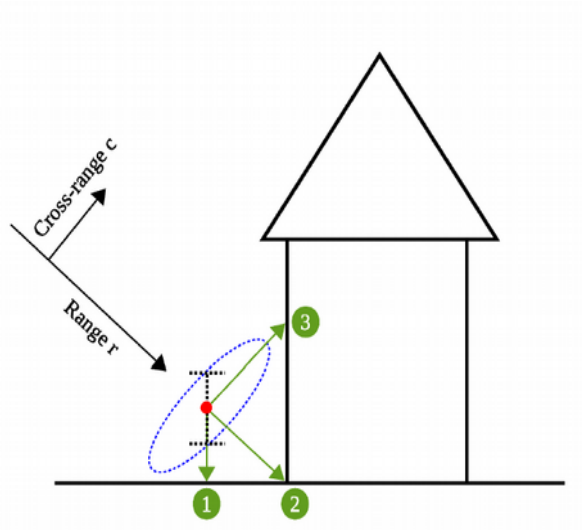


Figure 2: Example error ellipsoid, showing the position ambiguity. (P. Dheenathayalan et al.[33])

The introduction of web-based 3D techniques paved the road for online point cloud viewers. In recent years multiple solutions have been launched for showing large (billions of points) point clouds in the web-browser. Unfortunately no combination between point clouds and radar scatterer information has yet been made public.

This project was aimed at creating this missing link and integrating both data sources. Given the rise of web based point cloud viewers and the availability of a nationwide airborne LiDAR dataset (AHN) The Netherlands forms a perfect test bed for this integration of datasets in an online application.

This additional thesis has three final products: this report, a short manual on how to create a similar application and a demo application. The requirements for the demo application are set out in paragraph 1.A (Terms of reference), the associated research questions are stated in paragraph 1.B of this introduction.

First the currently available solutions for point clouds will be discussed in chapter 2. The data available to this study is discussed in chapter 3. In chapter 4 an outline of the steps necessary to implement such a combined solution is shown. This chapter may serve as a manual for creating a similar web-application.

In chapter 6 the demo application will be shown. Finally in chapter 7 the terms of reference and research questions will be discussed. Those interested in creating a similar application could consult Appendix A for a head start.

1.A. Terms of reference

The following requirements formed the basis of the demo application. In short: an application aimed at a broad audience of professionals, skilled producers but novice users. The concept will be demonstrated on the university campus, based on provided radar data.

Purpose of the tool	To visualise the error ellipsoid of the radar measurements in their environment (e.g. point cloud data), accessible for large(r) audiences. The resulting map will be 'write once, read many'.
Target audience	Focus is on internal use by researchers. Future use may include usage in publications. The creator will be skilled, but users should not require training.
Initial study area	Two areas were marked for initial testing: the campus of the Delft University of Technology and a reflector test site in Wassenaar (NL). The campus was selected for initial testing, as (processed) radar data for this location is available as well as high resolution scans from ground level.
Available data	Initial radar data will be provided by SkyGeo and will consist of a TerraSAR-X time series. A constant error (ellipse) is assumed for all persistent scatterers, later usage may include datasets with error estimates per scatterer. Various point clouds from airborne LiDAR are available (Actueel Hoogtebestand Nederland, AHN), several terrestrial scans and data from a

mobile laser scanner were available.

Aerial photographs were used to colour the airborne point clouds to ease navigation.

1.B. Research questions

The following research questions were formed to accompany the development of the demo application.

1. What software and applications are already available?
2. What information is required to position SAR data?
    1. How are SAR signals and their uncertainties represented?
    2. Which coordinate systems and (file) formats are involved?
3. Which combination of data is effective?
    1. How is effectiveness assessed?
    2. How does this combination help in finding the (dominant) scatterer?
4. What additional features can be foreseen and/or recommended?

2. Available solutions for point clouds

The introduction of the WebGL standard early 2011 allowed for a unified method of rendering complex 3D images in a web-browser. WebGL defines a connection between scripts ('programs') in the browser and the underlying graphics capabilities of the computer, without the further need for plug-ins such as Java or Adobe Flash[1]. WebGL is currently supported by most desktop and mobile browsers: an estimated 93% of the users is able to use WebGL in their browser[2]. In this chapter existing solutions for rendering point clouds in the browser will be discussed, all are based on WebGL.

All solutions consist of three distinct steps: converting (pre-processing), serving and viewing. Of which only the latter is visible to the client. As it is (currently) unfeasible to load more than two million points on the client side[3], preprocessing (e.g. sorting, tiling) of the data is required. This will allow the browser to load only the required points, thus reducing the workload to a workable number of points. PoTree, plas.io and Cesium will be discussed, of which PoTree and plas.io stand out as they are specifically designed for viewing point clouds in the browser. This process is shown in Figure 3, the steps (converter/server/viewer) will be elaborated on in this chapter.

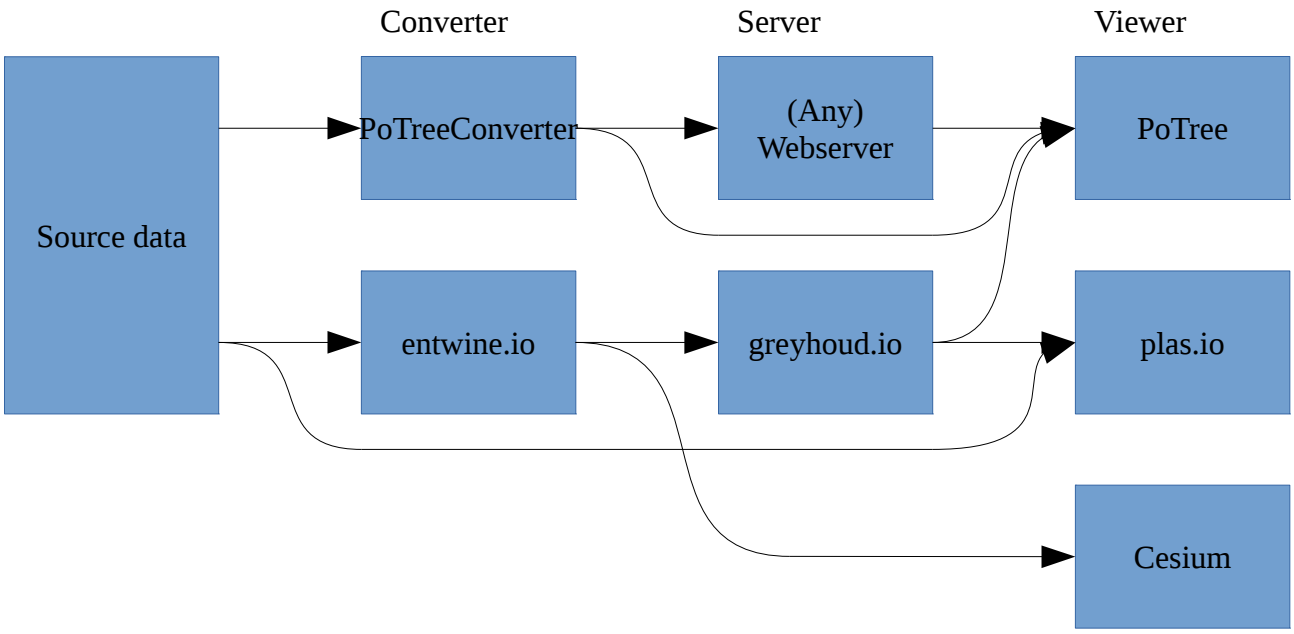


Figure 3: Processing workflow, from input to viewer. Any path from left to right will result in a working point cloud viewer.

2.A. Converter

A converter reads input data and outputs a file structure readable by either the server or directly by the viewer. This step is only performed once (per input point cloud). The file structures generated

1 "WebGL Overview", Khronos Group, https://www.khronos.org/webgl/. Retrieved 2017-11-06.
2 Statistics on WebGL by Can I Use (http://caniuse.com/#search=WebGL). Retrieved 2017-10-23.
3 On average consumer hardware. This figure is dependant on both hardware (CPU, GPU, RAM) and software (OS (drivers), browser) and no single number exists. Automatic detection is possible, but is outside the scope of this work.

resembles an Octree. (Readers unfamiliar with *Octrees* could refer to chapter 4.B for a short introduction.) The following two converters create file structures are supported by PoTree, Entwinte.io structures are supported by plas.io and Cesium:

- PoTreeConverter

  Written by the authors of PoTree, the Computer Graphics group at the TU Wien. PoTreeConverter[4] can read LAZ/LAS, PTX, (binary) PLY and ASCII (XYZ). Converts to an Octree structure[5] of an internal binary format, LAS or LAZ. Files can be used by the viewer (PoTree) directly from the filesystem, or served (as static files) by any webserver to the viewer.

  The Netherlands eScience Center created `Massive-PoTreeConverter` as part of their efforts to convert the full AHN2 point cloud to the PoTree format[18]. This converter is capable of running many `PoTreeConverter` jobs in parallel[60] and adds extends of the created tiles to a database (PostgreSQL, PostGIS)[6].

- Entwine.io

  Made by Iowa City (Iowa, United States) based collective Hobu[7], maintainers of point cloud related software like PDAL. PDAL, or the “Point Data Abstraction Library”, is a C++ library for “translating and manipulating point cloud data”. It exposes a series of “stages” (readers, writers and filters) that can be chained using pipelines. The filters include a wide variety of options, such as transformations, tiling, colourisation, etc.[8].

  Their converter, Entwine.io[9], is based on PDAL and reads all file formats PDAL supports[10]. By default Entwine.io is distributed as a `docker` container[11]. The output is either a 'sorted structure' (Octree) for their Greyhound.io server and/or “3D Tiles”[12,17] for Cesium.

2.B. Server

After processing the data has to be fed to the client. No data is written in this step, only transferred to the client. Depending on the requirements by the client this step is either implemented as static (all calculations done by the converter) or dynamic (final processing done in the server step).

- Static files

4 “PoTreeConverter”, https://github.com/potree/PotreeConverter. Retrieved 2017-10-30.
5 “PoTree Data Format”, https://github.com/potree/potree/blob/develop/docs/potree-file-format.md. Retrieved 2017-10-23.
6 “Netherlands eScience Center, Massive-PotreeConverter”, https://github.com/NleSC/Massive-PotreeConverter. Retrieved 2017-10-24.
7 “Hobu”, https://hobu.co/. Retrieved 2017-10-24.
8 “PDAL – Filters”, https://www.pdal.io/stages/filters.html. Retrieved 2017-11-06.
9 “Entwine.io”, https://entwine.io/. Retrieved 2017-10-25.
10 “[PDAL] Readers”, https://www.pdal.io/stages/readers.html. Retrieved 2017-10-24.
11 Docker containers allow running a dedicated 'operating system' alongside the current operating system. Creating unified environments for application developers and separating the application from the coordinating operating system.
12 “Static website to view Entwine.io's 3D Tiles in Cesium”, https://github.com/connormanning/entwine-cesium-pages. Retrieved 2017-10-25.

Can be used directly from the filesystem[13] (eg. an external harddrive shared between colleagues) or in combination with a webserver. The files can be read by the viewer immediately, without intervention of another program. “Static” is commonly used to refer to files that are constant in time and require no further processing before being served to the client (viewer).

In case of the PoTree Octree structure this structure is traversed by the client, requiring frequent requests to either the webserver or filesystem. Those requests are relatively fast per transfer as they are simple (read file, output file over network) and can be cached (each request to the same file is equal). A drawback is that the full data of an only partially visible cell will have to be loaded. Furthermore all filtering will be done on the client side.

- Greyhound.io (dynamic)

  Streaming server, for files prepared by Entwine.io. Serves parts of the point cloud on request by the client[14].

  Only points within a region requested by the client are sent, limited by the requested depth of the Octree. Filtering can be applied on the server side, sending only points that will be shown. This limits the amount of requests and data necessary to load the point cloud. Nevertheless tiled requests may be employed to parallelise the loading process on the client side[15]. (Parts of) requests may be cached, to allow faster responses.

2.C. Viewer

Three major web-based point cloud viewers exist. PoTree and Plas.io are especially written for point clouds. Cesium has a broader focus as a virtual Earth, comparable to Google Earth. All three will be introduced here:

- PoTree

  PoTree is built as a plugin on the Three.js (3D) library and is built to shown point clouds from either its own Octree structure or a Greyhound.io server. Built as an extension (to the Three.js library) the viewer can easily be extended to show radar ellipsoids or other geometries together with the point cloud.

  The default PoTree interface has tools for various geometric measurements (distance, surface area, profile).

- Plas.io

13 Results may vary, depending on how the browser handles the `file://` protocol. Firefox, for example, will not compute a mime-type for files. As a result JSON-files may be identified as XML-files. Some JavaScript libraries, eg. OpenLayers4, will refuse to read those files.

14 “Client development”, https://github.com/hobu/greyhound/blob/master/doc/clientDevelopment.rst. Retrieved 2017-10-24.

15 “Client development: Progressive Querying”, https://github.com/hobu/greyhound/blob/master/doc/clientDevelopment.rst#progressive-querying. Retrieved 2017-10-24.

'Trendy' alternative to PoTree, built by the creators of Entwine.io and Greyhound.io (Hobu). Allows live overlay of satellite photographs if points are reprojected as Web-Mercator.

Capable of loading small point clouds directly from the filesystem, larger point clouds can be loaded from a Greyhound.io server.

- Cesium

  Focussed on "3D globes and maps"[16], but point cloud integration is possible. The experimental "3D Tiles" output Entwine can be used as data source for Cesium[17].

2.D. Online examples

Major examples of those viewers are Speck.ly (run by Hobu[7]) and the AHN2 viewer by the Massive Point Clouds for eSciences project[18].

In Figure 4 the AHN viewer by the Massive Point Clouds for eSciences project is demonstrated on the town of Willemstad (Noord-Brabant). This viewer is based on PoTree and is publicly available at http://ahn2.pointclouds.nl and contains a heavily processed version of AHN2, including removing duplicate points[60]. Data is served from a PoTree file structure.

Figure 5 shows the same town in Speck.ly (http://speck.ly), an implementation of the Plas.io viewer. Demonstrated is the feature to use imagery from a third source to (live) overlay the point cloud served by a Greyhound.io server.

Further examples on combinations of the software discussed in this chapter are available online, a selection:

- PoTree: http://potree.org and http://ahn2.pointclouds.nl/;
- Entwine.io, Greyhound.io and PoTree: http://potree.entwine.io/;
- Entwine.io, Greyhound.io and Plas.io: http://speck.ly/;
- Entwine.io + Cesium: http://cesium.entwine.io/.

16 "About Cesium", https://cesiumjs.org/about/. Retrieved 2017-10-13.
17 "Add initial 3D Tiles output prototype", https://github.com/connormanning/entwine/pull/12. Retrieved 2017-10-23.
18 "Massive Point Clouds for eSciences", http://www.gdmc.nl:8080/mpc (redirected from http://pointclouds.nl). Retrieved 2017-10-24.

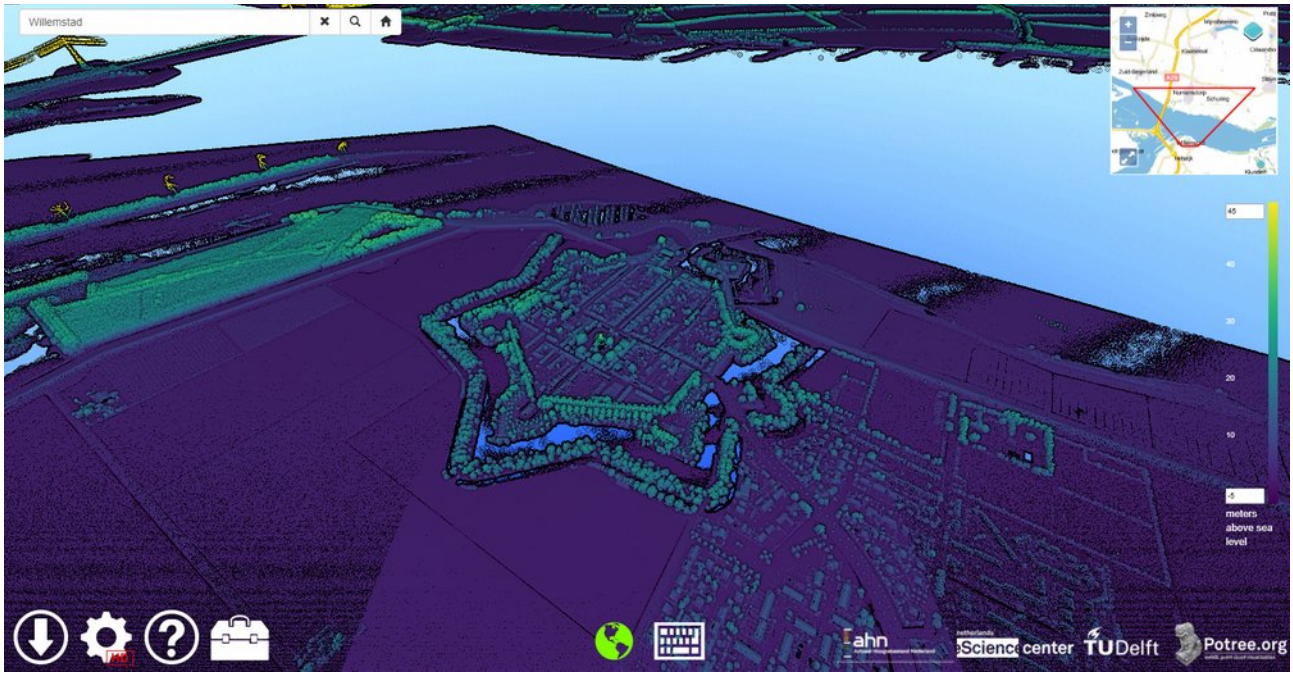


Figure 4: AHN2 as shown on ahn2.pointclouds.nl. (Source: M. van Meersbergen, Netherlands eScience Center)

Figure 5: Speck.ly, showing AHN2 overlayed with ArcGIS satellite imagery.

2.E. Demo application

Based on the possibilities discussed in this chapter an important subdivision can be made based on the serving structure: static versus dynamic solutions.

Static solutions are processed once and the output will not change in time. After initial processing it is left to the client (visitor) to request the correct files and do final processing on the points. Major advantage is that no complex (web) server is needed. A file service, like Amazon S3[19], will be able to serve the files at request. As a consequence this solution is (very) low in maintenance.

Dynamic solutions pre-process data that is later served by a (web-)service, sending the data on request to the client after some final, *on the fly*, processing. Processing is shared between the server and client. The advantage of reduced data transfer (only requested points are sent to the client) comes at the price of the (relatively) high maintenance cost of maintaining a (complex) server.

Given the terms of reference of a *write once, read many* application that is focussed on researchers rather than an internet company a static solution is preferred. Therefore PoTree with a backend of static files (generated by `PoTreeConvert`) was chosen as basis for the application.

Due to the experimental nature of the Entwine.io – Cesium integration this solution was not considered as a basis for this demo application.

19 "Amazon S3 (Simple cloud Storage Service)", https://aws.amazon.com/s3. Retrieved 2017-11-08.

3. Available data

Data search was aimed at getting a high resolution view of the possible scatterers in the area. Scatters are likely man made structures[33] and will be in the slant viewing angle of the satellite. Thus coverage of those areas is essential.

Given the initial study area, Delft University of Technology campus, the following datasets were selected to be included in the application. Four types of data were used: LiDAR, radar, optical and map (vector) data. For each dataset used the coverage, file format, coordinate system, (expected) resolution and accuracy will be discussed.

3.A. Actueel Hoogtebestand Nederland 1/2/3

Three iterations of the country wide 'Actueel Hoogtebestand Nederland' exist, recorded over the last 20 years. Data was acquired via airborne LiDAR with the main purpose of creating a digital terrain model. Therefore coverage is focussed on nadir measurements, rather than facades. The different properties of the versions/years are shown in Table 1.

		Error (vertical[20], 1σ)		
	Recording	Systematic	Stochastic	Density
AHN1[21]	1996 – 2003[22]	5 cm	15 cm	1 $^{pt}/_{16\ m^2}$ – 1 $^{pt}/_{m^2}$
AHN2[21]	2008 – 2012[23]	5 cm	5 cm	6 – 10 $^{pt}/_{m^2}$
AHN3	2014 – 2018[23]	5 cm	5 cm	~ 16 $^{pt}/_{m^2}$[24]

Table 1: Properties of different iterations of the AHN product.

Data is provided as tiled LAZ-files (5 km × 6.25 km) and is available for public download through PDOK (Publieke Dienstverlening op de Kaart)[25]. Coordinates are expressed as RD-NAP coordinates (EPSG:7415)[27].

The selection of tiles used is refered to as “Groot Delft” and consists of the following AHN tiles: `30DZ2`; `30GZ1`; `30GZ2`; `37BN1`; `37EN1` (University campus); `37EN2` (University campus); `37BZ2`; `37EZ1` and `37EZ2`. Their extend is shown in Figure 6. Together they cover the extends of the provided InSAR dataset (see paragraph 3.E).

20 AHN2 has a maximum horizontal error of 50 cm. See footnote 21, paragraph 3.1.2.
AHN3 has a identical constraint. See footnote 27.

21 “Kwaliteitsdocument AHN2”, N. van der Zon, May 2013, http://www.ahn.nl/binaries/content/assets/hwh---ahn/common/wat+is+het+ahn/kwaliteitsdocument_ahn_versie_1_3.pdf. Retrieved 2017-10-24.

22 Swartvast: http://www.swartvast.nl/ahn_1_vs_2.php#actualiteit

23 Waterschappen: http://www.ahn.nl/common-nlm/inwinjaren-ahn2--ahn3.html

24 Estimate based on the $^{points}/_{area}$ for tiles `37BN1` and `37EN1`.

25 AHN1 (Atom feed): http://geodata.nationaalgeoregister.nl/ahn1/atom/ahn1_gefilterd.xml (ground) and http://geodata.nationaalgeoregister.nl/ahn1/atom/ahn1_uitgefilterd.xml (other points).
AHN2 (Atom feed): http://geodata.nationaalgeoregister.nl/ahn2/atom/ahn2_gefilterd.xml (ground) and http://geodata.nationaalgeoregister.nl/ahn2/atom/ahn2_uitgefilterd.xml (other points).
“AHN3 downloads”, PDOK, https://www.pdok.nl/nl/ahn3-downloads. Retrieved 2017-10-24.

Classification data is provided with the point clouds. For AHN1 and AHN2 data is split in ground/non-ground LAZ-files. With AHN3 further classes are available, stored in a single LAZ-file. Classification follows the classifications used in ASPRS LAS-files[26,27]. The points are divided in the following classes: unclassified (1), ground (2), building (6), water (9) and civil structure (26). Where the numbers in brackets correspond the class numbers (identifiers) used in the LAS-file. The code for civil structure (26) is a custom code and is not in the ASPRS LAS standard.

AHN1 and AHN2 only provide coordinates, AHN3 contains per-point intensity information. All three versions are included in the demo application. For recent InSAR data AHN3 is best suitable as it has the highest point density of the three iterations, coincidently increasing the point coverage on facades. For older data, older iterations of AHN may describe the ground truth at that time better.

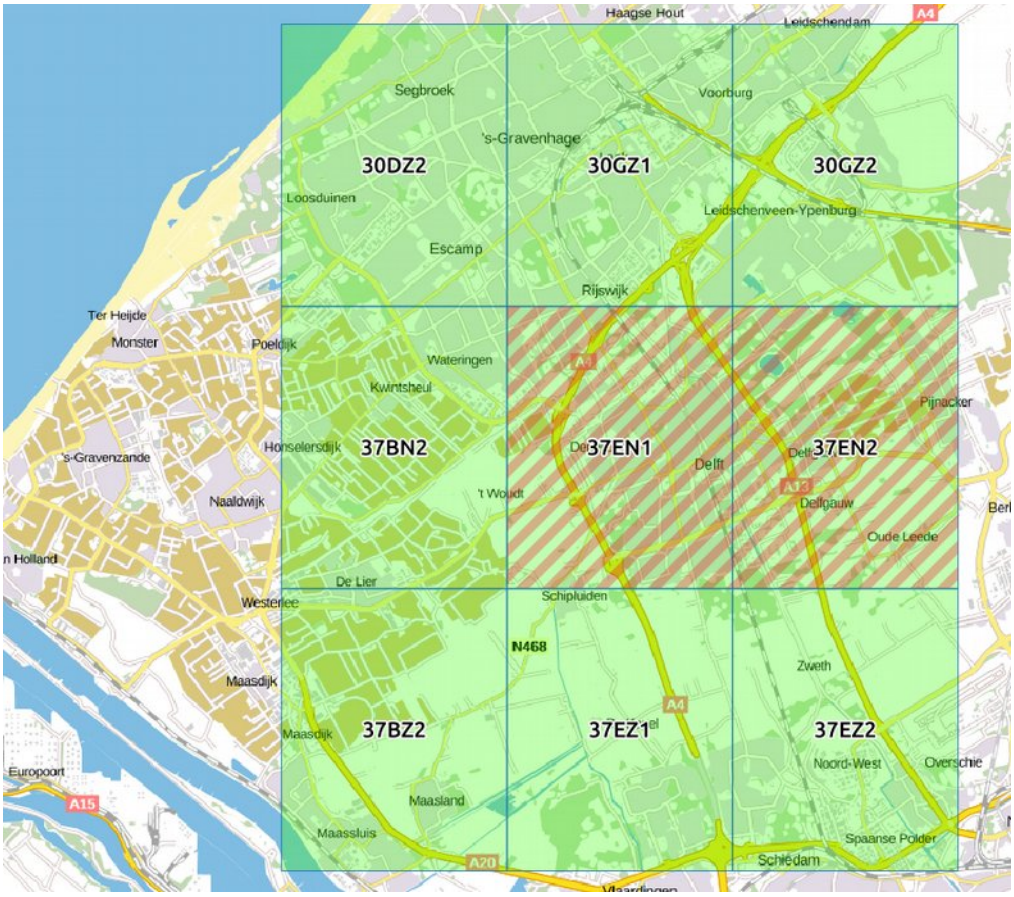


Figure 6: Coverage of AHN tiles. Green: "Groot Delft"; red: "Campus". (Background: PDOK BRT Achtergrondkaart.)

3.B. Terrestial Laser Scanners

Over the years parts of Delft were scanned using the department's Leica C10 Terrestrial Laser Scanner. Unlike with down looking airborne AHN these scans were (mostly) focussed on facades

26 "LAS Specification, version 1.4", https://www.asprs.org/a/society/committees/standards/LAS_1_4_r13.pdf. Retrieved 2017-10-23.

27 See "Bestekvoorwaarden" as supplied with the tender for "inwinning en controle AHN 2018 – 2019, Rijkswaterstaat Centrale Informatievoorziening", March 2017, https://www.tenderned.nl/tenderned-web/aankondiging/detail/samenvatting/akid/d4ccd8312612ae997b3cdf85cd2caba8/pageId/D909C/huidigemenu/aankondigingen/cid/1929040/cvp/join . Retrieved 2017-10-24.

and trees, rather than the ground. They provide a dense cover in those areas, where AHN is lacking information. The benefit of using multiple sources can be seen in Figure 7, where the full facade of the Faculty of Civil Engineering and Geosciences (Delft University of Technology) is in the point cloud, rather than only the roof in AHN3 (Figure 8).

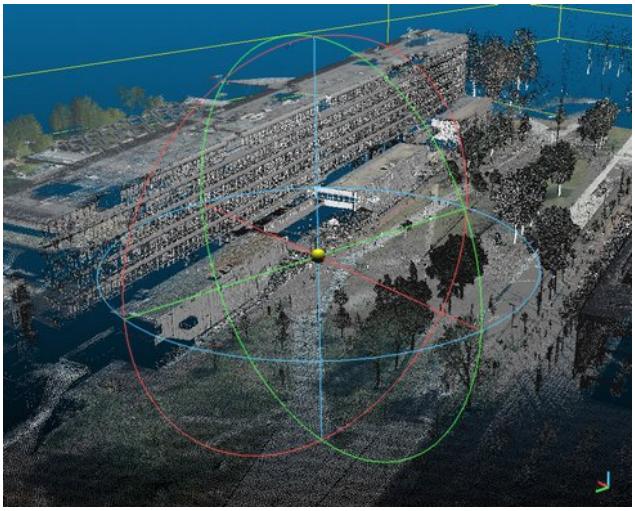


Figure 7: AHN3, Mekelpark DTM and Mekelpark Gras.

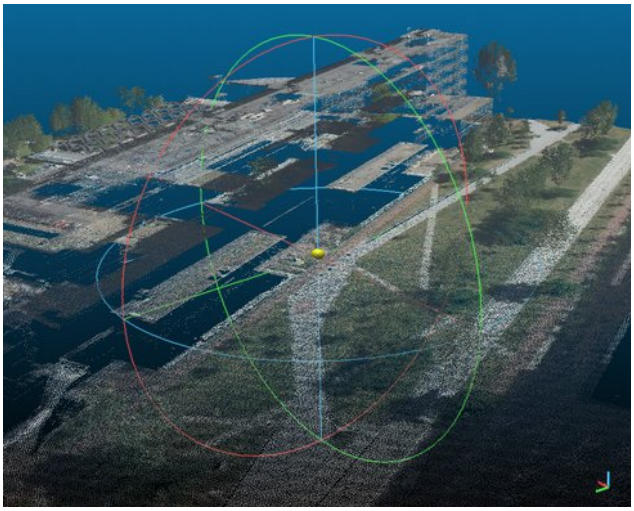


Figure 8: AHN3, coloured with PDOK Luchtfoto.

The following scans were available to this study:

- Mekelpark Tram

  A (very) high resolution scan of the Tram tracks at the crossing of the Balthasar van der Polweg, Berlageweg and Mekelpark in Delft. Recorded March 2015.

- Mekelpark DTM

  Scan of the central part of the Mekelpark (Delft University of Technology campus). Recorded March 2015.

- Mekelpark Trees

  Scan at the south side of the CEG-building. Recorded May 2017.

- Mekelpark Gras

  Scan of the central part of the Mekelpark. Contains large detail on the facades of EEMCS and CEG. Recorded March 2017.

- Faculty of Architecture

  A facade scan of the Faculty of Architecture at the inner court adjacent to the Michiel de Ruyterweg (Delft). Recorded April 2015.

- Wassenaar (radar reflector)

  Scan of an experimental radar reflector, the surrounding field (including one house) and the transponders present on the field. Recorded August 2017, using a Leica P40.

Unfortunately all scans are in local coordinates, without references to a national/global coordinate system. Therefore matching is part of the processing workflow. Within a scan (local coordinates) the accuracy is 6 mm (1σ)[28], an improvement over the AHN resolution. N.B. the transformation to the national/global coordinate system will introduce extra errors, decreasing the overall position accuracy of the data.

To illustrate the extends of the dataset after alignment, the point density is shown in Figure 9 - 12 in square cells of 0.25×0.25 m (≈ 0.6 m²). These images were generated using the Rasterize module in CloudCompare and overlayed on OpenStreetMap in QGIS.

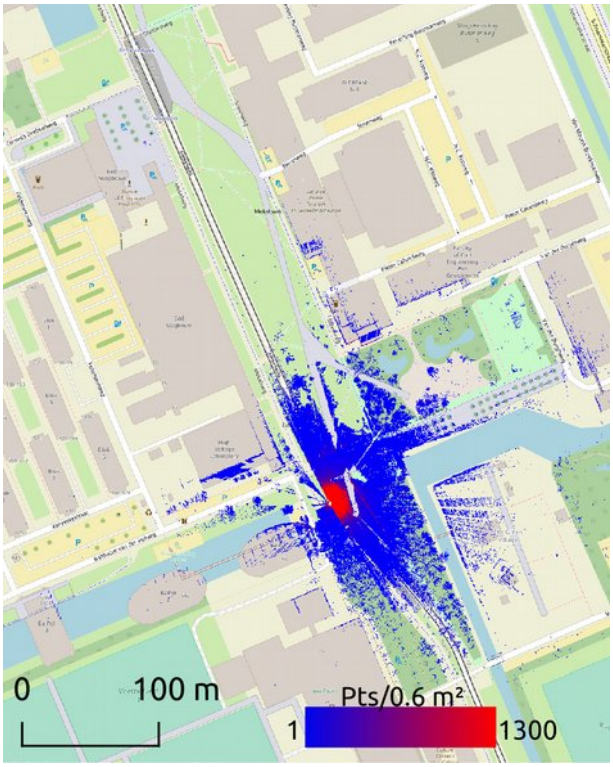


Figure 9: Point density of "Mekelpark Tram".

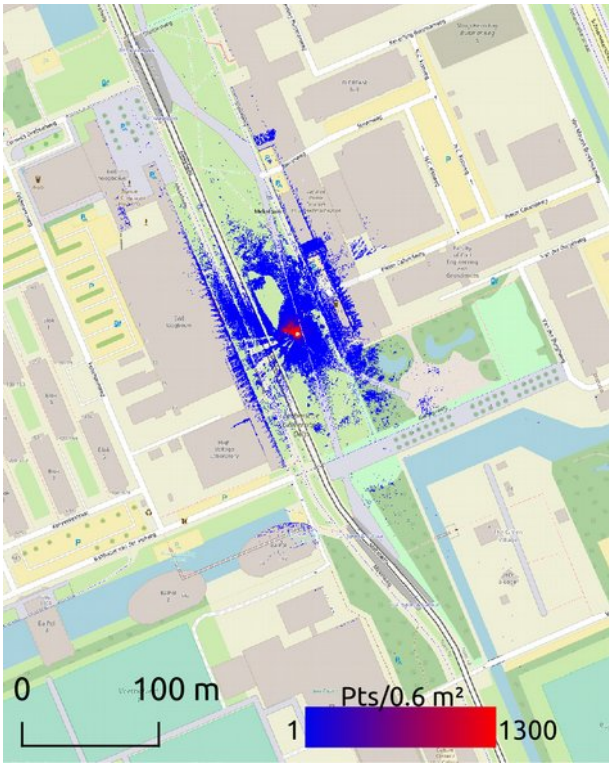


Figure 10: Point density of "Mekelpark DTM".

28 "Leica ScanStation C10, product specifications", https://hds.leica-geosystems.com/downloads123/hds/hds/ScanStation%20C10/brochures-datasheet/Leica_ScanStation_C10_DS_en.pdf. Retrieved 2017-10-30.

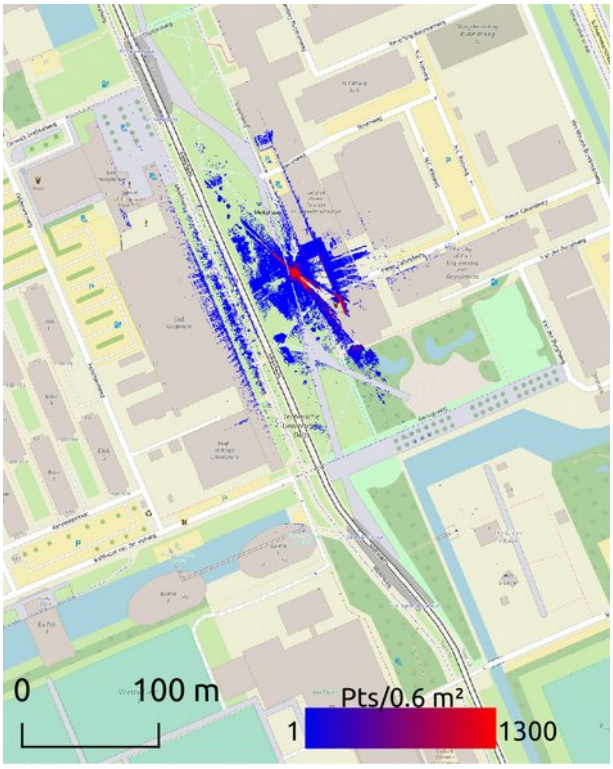


Figure 11: Point density of "Mekelpark Trees".

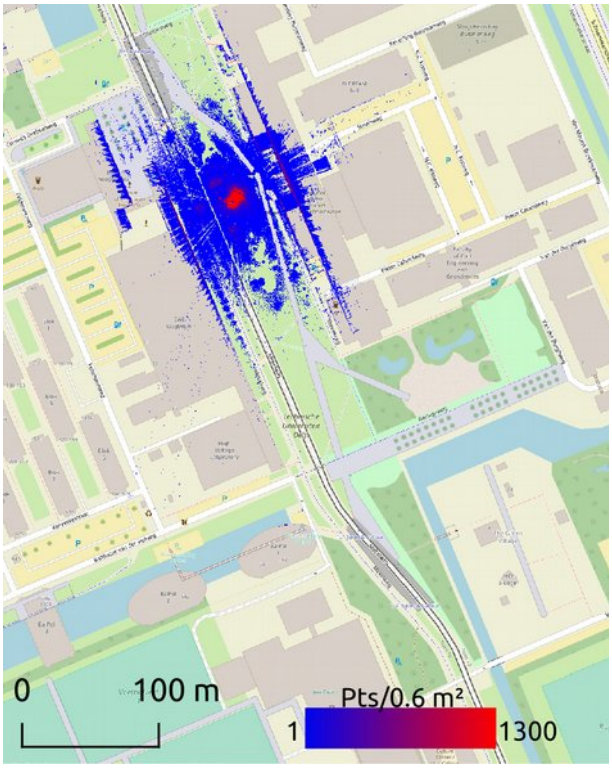


Figure 12: Point density of "Mekelpark Gras".

3.C. Mobile laser scanner

Two point clouds from a Mobile Laser Scanner (MLS) were provided by Jinhu Wang. Data was recorded by Fugro Geoservices B.V. using their Fugro Drive-Map system for the SigVox project[29]. Both point clouds were recorded on the northern part of the university campus, roughly between the Faculty of Technology, Policy and Management and the Faculty of Architecture. The first point cloud was recorded in 2013 (Figure 13), the second in 2016 (Figure 14). The density figures were produced using the same procedure as with the TLS data.

29 "A 3D feature matching algorithm for automatic street object recognition in mobile laser scanning point clouds", J. Wang et al., 2017, ISPRS Journal of Photogrammetry and Remote Sensing. doi:10.1016/j.isprsjprs.2017.03.012

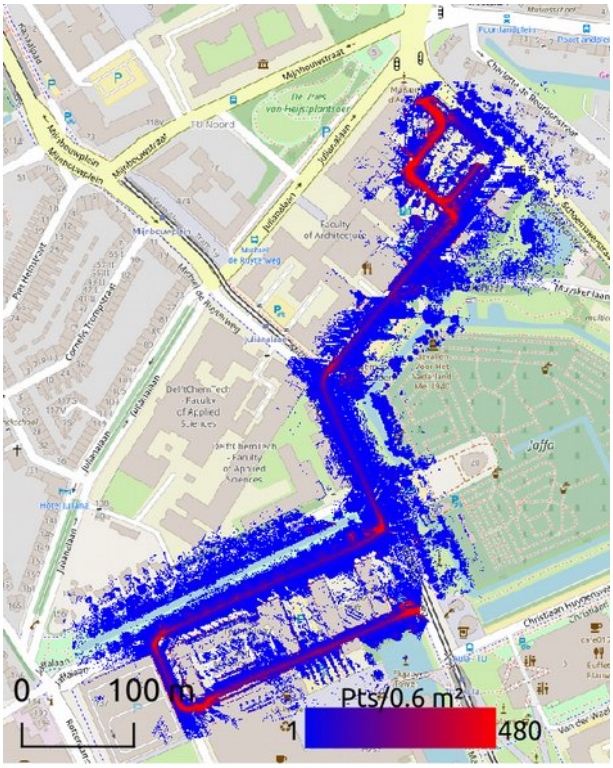


Figure 13: Point density of the 2013 MLS data.

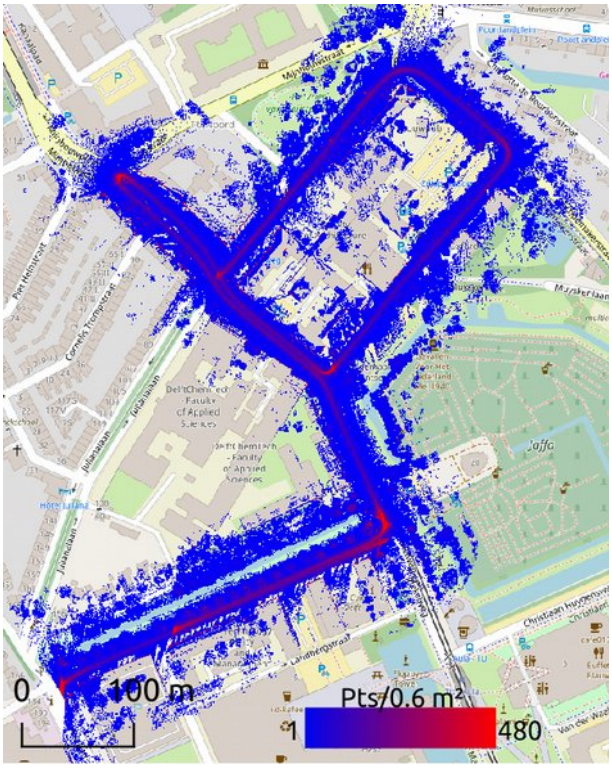


Figure 14: Point density of the 2016 MLS data.

3.D. PDOK Luchtfoto

Nationwide aerial photographs are recorded every year, recently the 'low-resolution' images with a ground resolution of 25×25 cm were made publicly available[30]. Both true colour (RGB) and infrared images were recorded.

Data is provided as a web-service using WMS (on demand, Web Map Service) and WMTS (tiled, Web Map Tile Service). Data is available in various coordinate systems: EPSG:28992 (RD-coordinates), EPSG:4326 (WGS84) as well as EPSG:3857 (Pseudo-Mercator/Web-Mercator).

The aerial photographs may be used to colour the point cloud for easier navigation. Unfortunately AHN and the aerialphotograph do not perfectly align. The aerial photograph is defined to have a position accuracy of 37.5 cm (1σ) at ground level[31] and AHN2/3 has a horizontal accuracy of 50 cm (1σ)[20]. As a consequence some grass can be seen on rooftops and vice versa.

3.E. InSAR (persistent scatterers)

The InSAR timeseries are provided by SkyGeo, a Delft' firm specialising in InSAR deformation monitoring[32]. Provided are the (estimated/expected) position of the scatterer, a time series of

30 "Hogere resolutie luchtfoto als open data by PDOK", PDOK, February 2017, https://www.pdok.nl/nl/actueel/nieuws/artikel/10feb17-nieuw-hogere-resolutie-luchtfoto-als-open-data-bij-pdok. Retrieved 2017-10-30.

31 "Bestekvoorwaarden Lage resolutie luchtopnamen", chapter 6.2.k, Het Waterschapshuis, july 2015, http://www.beeldmateriaal.nl/binaries/content/assets/hwh-bm/downloads/besteksvoorwaarden-2016-lrl-.pdf. Retrieved 2017-10-23.

32 "About SkyGeo", https://skygeo.com/company/. Retrieved 2017-10-30.

measurements and an estimate of the linear trend. All one million points include a two year time series from a single descending TerraSAR-X orbit. An example of such a time series can be seen in Figure 15.

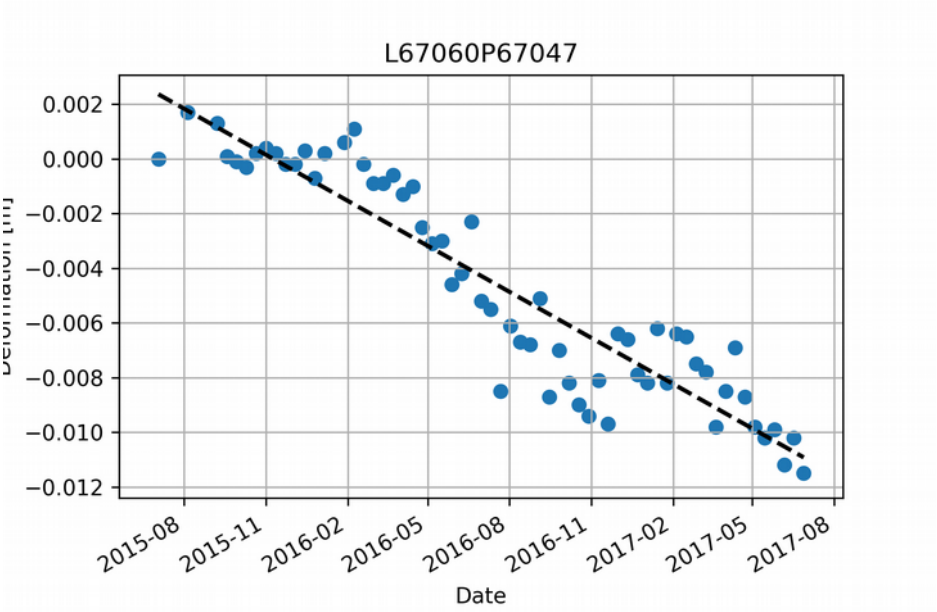


Figure 15: Timeseries of persistent scatterer "L67060P67047", located on or close to a bench at the Mekelpark (university campus). Shown on top (black) is the estimated linear trend.

All data is within an approximately 12½ km diameter circle around Delft. The extends are shown in Figure 16. As the region of interest is small it is assumed that all points share the same satellite viewing geometry.

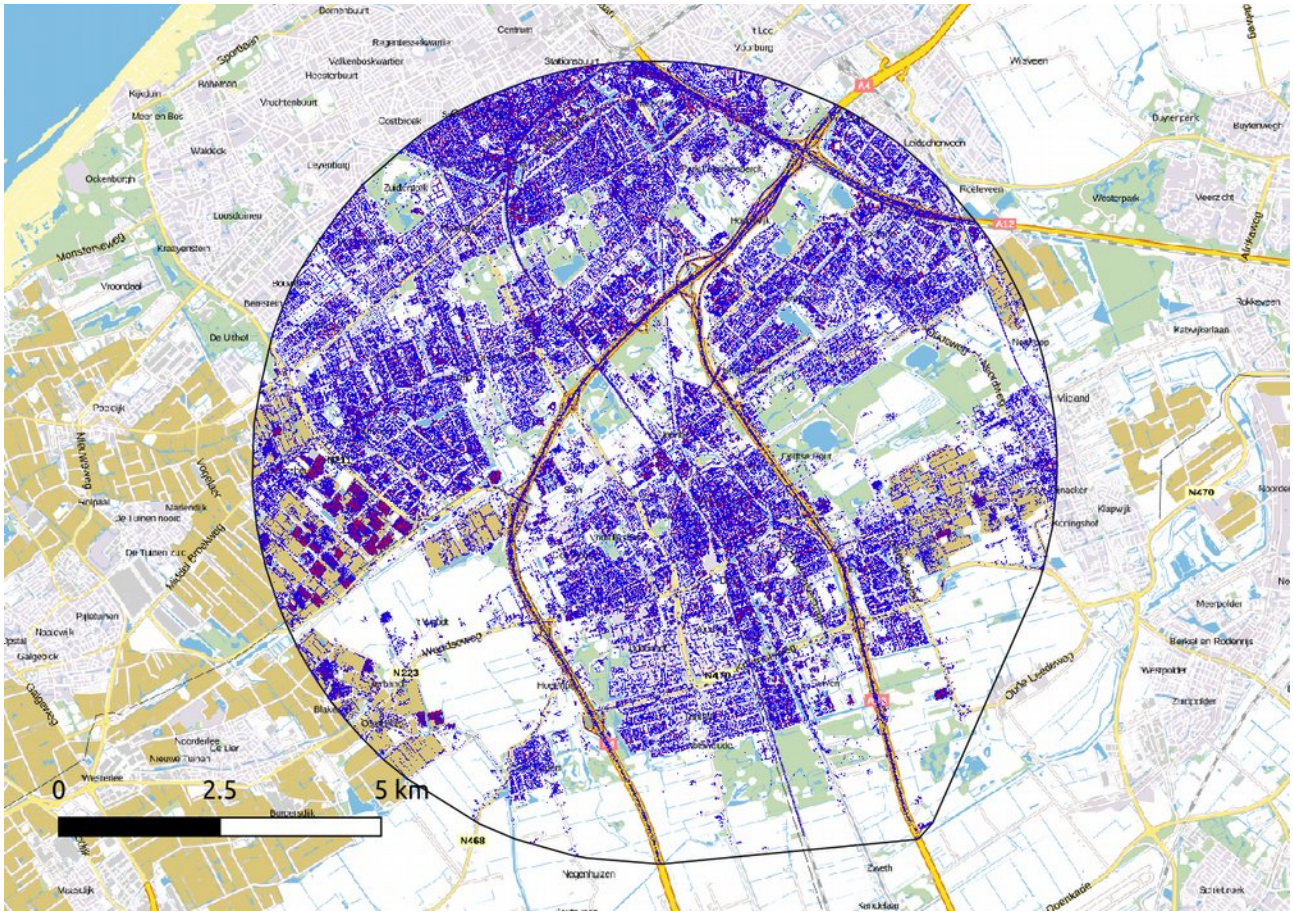


Figure 16: Extends of the InSAR dataset (black). Red patches indicate a high point density (colour scale: blue to red, background: PDOK Achtergrondkaart).

The ratio between the errors, standard deviations, in range, cross-range and azimuth direction was taken from the paper "high-precision positioning or radar scatterers" by P. Dheenathayalan et al.[33] and was estimated to be 1/3/213 in range/azimuth/cross-range. The range error was defined as σ = 0.025 m, ergo: 0.075 m in azimuth and 5 m in range. This results in elongated, flat, ellipsoids. Some examples can be seen in Figure 17.

33 "High-precision positioning or radar scatterers", P. Dheenathayalan et al., Journal of Geodesy, February 2016. (doi: 10.1007/s00190-015-0883-4)

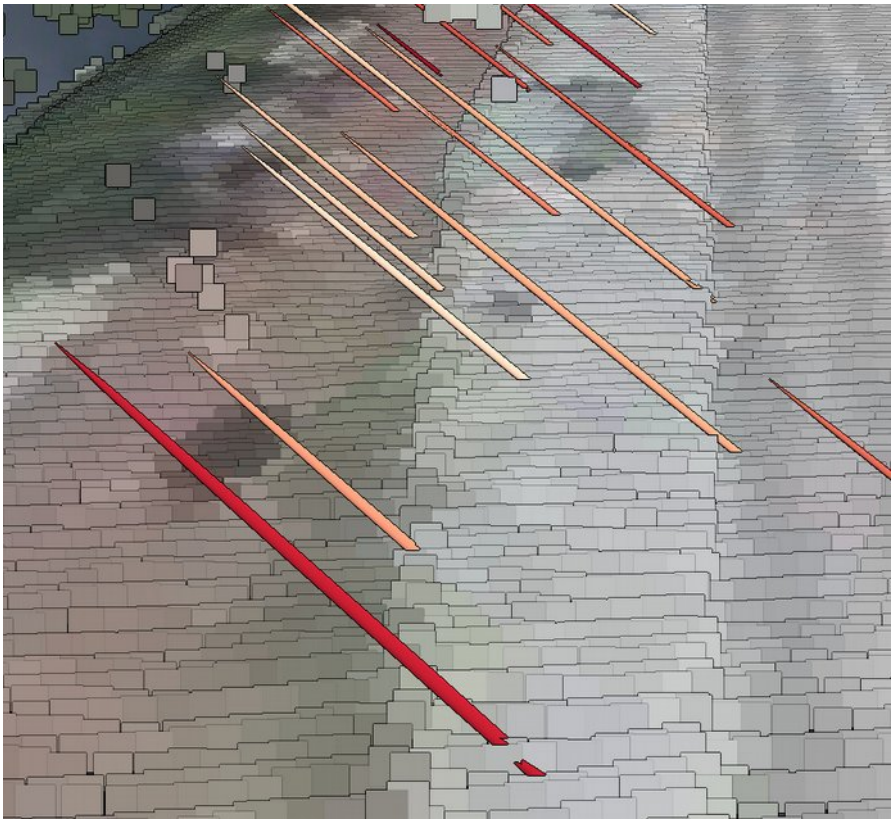


Figure 17: Example error ellipsoids as can be "seen" on the pavement in the demo application. The radar signal is coming from the right (range direction is from top right to bottom left of the image).

3.F. Map sources

In the application the client is offered a map, this map is provided by OpenStreetMap[34]. The map is provided as TMS (Tiled Map Service) in the Web-Mercator projection, compatible with most web-based map applications. OpenStreetMap is based on mapping efforts of volunteers and has worldwide coverage.

As an alternative NL Maps could be used[35]. This service is based on open data released by the Dutch (governmental) institutions and is limited to The Netherlands. And provides standard (vector) maps as well as aerial photos in the TMS format, suitable for web pages.

Labels for towns are based on the TOP10NL product of the Dutch Cadastre and were taken from a dataset published by Imergis[36]. These were not included in the demo application, but an example of those labels can be seen in Figure 25.

34 Implementing OpenStreetMap in OpenLayers: “OpenStreetMap Wiki: OpenLayers”, http://wiki.openstreetmap.org/wiki/OpenLayers. Retrieved 2017-10-24.
35 “NL Maps”, https://nlmaps.nl/. Retrieved 2017-10-24.
36 “Geografische open-data GIS bestanden”, Imergis, http://www.imergis.nl/asp/47.asp. Retrieved 2017-10-25.

4. Data processing

Data processing from input (chapter 3) to output is summarised in this flowchart (Figure 18).

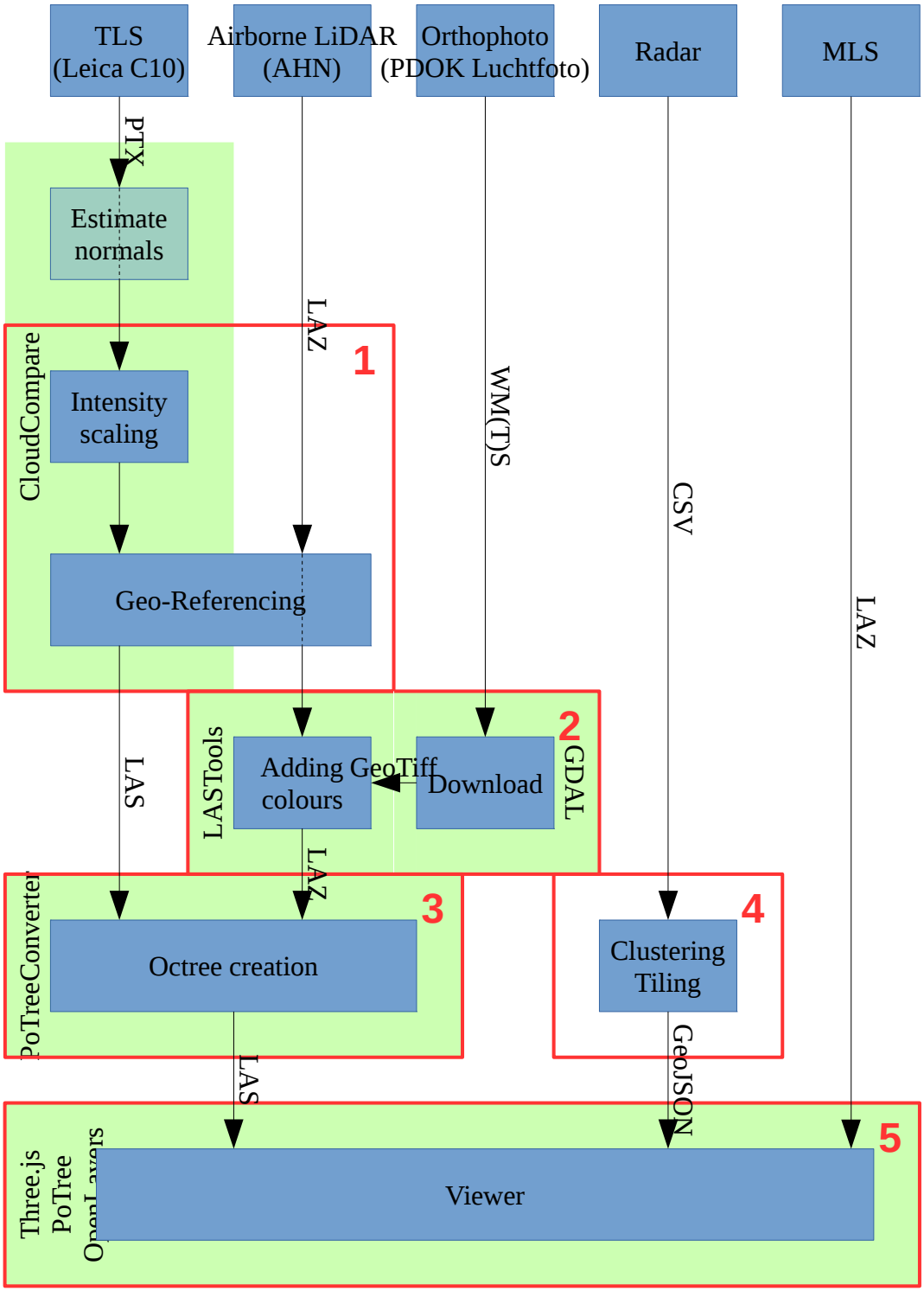


Figure 18: Flowchart data processing. Processes shown in blue, data streams in black and software used is shown as green boxes. The steps as discussed in this chapter are marked in red.

In this chapter the steps in this flowchart (Figure 18) are subdivided in 5 distinct processing steps (red). The purpose of this process is to integrate all available data to an integrated visualisation of a point cloud overlayed with radar data. For each step their input and intended output are discussed (black), followed by the commands involved. Together they will demonstrate in a manual style fashion how to create a similar result.

As an alternative a short summary of how to create a very basic PoTree installation (viewer) with one point cloud and an overlay of radar error ellipsoids is given in Appendix B. The steps in this chapter may be used to expand this basic installation.

4.A. Required software

LASTools[37], CloudCompare[38] (with LASlib integration), PoTreeConverter[4] and GDAL[39] are required for the creation of the (integrated) point cloud. Except for LASTools' lascolor all tools are available as open source software. Furthermore lascolor is the only program limited to Microsoft Windows only, all other software was tested on Linux but should work on Windows and Macintosh too. The processing of InSAR data was implemented in Python[40].

On the users side most modern browsers are supported by PoTree, major exception is Internet Explorer 11. The application was tested and worked with Mozilla Firefox 56 (Linux), Google Chrome 62 (Windows) and Apple Safari 11 (Macintosh).

4.B. Data structures

Taking the limited processing power of the client in mind, data has to be brought to the client efficiently. Chunks of data have to be small enough to be downloaded quickly and the least possible time should be lost on searching the right chunks. To accomplish this a spatial index is used.

For two dimensional searching a Quadtree is used, for 3D structures the similar Octree is used. Both could be considered a tiling schema. Tiling schemes serve a dual purpose: spatially dividing the data and splitting the data in small chunks. The tile will contain only the points within the tile boundaries and is essentially, allowing access to specific points without much redundant (unwanted) data, warranting a fast download (compared to downloading the full dataset and filtering).

The tile structure is such that there is an implicit relation between tiles and coordinates. This allows for downloading the requested tile without doing a search query first, as the tile location (URL) follows from the coordinates.

Quadtree

In a Quadtree data is divided in equal quadrants. This create an easy tile – coordinate relation. Take Figure 19[41] as an example. The top raster (level zero) contains all information of the input dataset

37 “rapidlasso GmbH, LAStools”, https://rapidlasso.com/lastools/. Retrieved 2017-11-08.
38 “CloudCompare”, http://cloudcompare.org/. Retrieved 2017-11-08.
39 “GDAL – Geospatial Data Abstraction Library”, http://www.gdal.org/. Retrieved 2017-11-08.
40 Python 3.5, with GeoPandas and Matplotlib (PyPlot).
41 “Damn Cool Algorithms: Spatial indexing with Quadtrees and Hilbert Curves”, Nick's Blog (Nick Johnson), http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves. Retrieved 2017-11-07.

and can be split into four equal parts (level one). Subsequently these four tiles can be divided in 16 smaller tiles (level two). And so on and so on.

A quadtree may be implemented such that only the highest zoom level is stored. Another implementation would be to include only a (random) subsample of points, adding points in each level up to the zoom level were all points are included.

The first solution allows for rapid access to specific features (their storage location is known based on the coordinates), the second option allows for incremental loading. Incremental loading is praticullary useful for viewing purposes. Immediately showing the a rough preview of the data and refining it (adding points) as soon as more data becomes available.

Finding the tiles based on the coordinates is as easy as (same relation holds for *y*):

$$\text{floor}\left(\frac{x}{x_{max}-x_{min}}2^{z}\right) \tag{1}$$

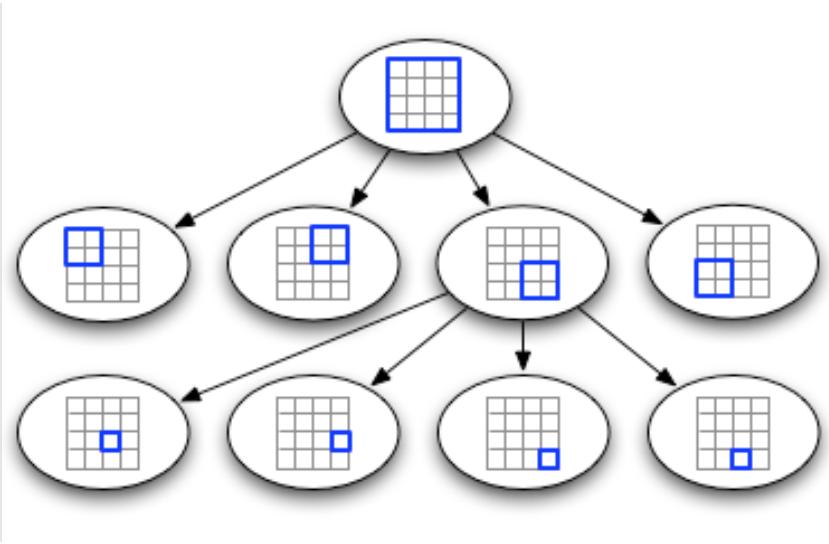


Figure 19: Demonstration of the internal structure of a Quadtree, systematically subdividing a set in equal quadrants. (Nick Johnson)

Octree

An Octree is a 3D implementation of the Quadtree structure. Divding data in equal octants, rather than quadrants. An example can be seen in Figure 20[42]. The same impelmentation methods discussed for the Quadtree hold for the Octree.

42 "GKOctree", Apple Developer Documentation, https://developer.apple.com/documentation/gameplaykit/gkoctree. Retrieved 2017-11-08.

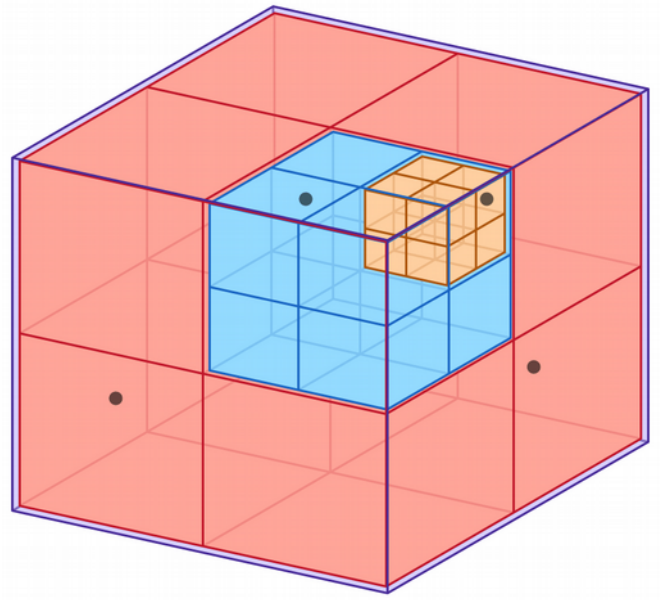


Figure 20: Demonstration of the internal structure of a Octree. (Apple)

Step 1: from TLS to referenced point cloud

For the TLS data (see chapter 3.B) no georeference was available. To combine the datasets it is necessary to align the local TLS coordinates to a coordinate system used by the other datasets. When no georeference is available for the point clouds the point cloud can be aligned based on another or an wide area (eg. airborne) LiDAR dataset.

Based on corresponding points (references) the transformation may be computed. It is important to maintain the scale of the original (input) point cloud. Therefore determining a affine transform using least-squares estimation will not suffice, as it will include shearing and scaling too.

To find the optimal rotation and translation a method described by Besl and McKay (1992)[43] is used. In short the translation is found by first calculating the centroids (average coordinates) of the corresponding points, these form the translation parameters; the rotation is found based on the left and right singular vectors of the singular value decomposition of the covariance matrix of the points. The resulting transformation matrix can then be applied to all points in the input, to transform them from the local coordinate system to the wide area coordinate system.

Furthermore (possibly Leica PTX specific) the intensity value in PTX files is in the range from 0 to 1. By definition LAS intensity values are stored as `unsigned short` (integer, 0 to 65535). Therefore scaling of the intensity values is necessary before further processing.

43 "A method for registration of 3D shapes", P.J. Besl and N.D. McKay, IEEE Transactions, 1992, http://www-evasion.inrialpes.fr/people/Franck.Hetroy/Teaching/ProjetsImage/2007/Bib/besl_mckay-pami1992.pdf. Retrieved 2017-09-11.
A more straightforward explanation is available on:
"Finding optimal rotation and translation between corresponding 3D points", Nghia Ho, May 2013, http://nghiaho.com/?page_id=671. Retrieved 2017-09-11.

The intensity scaling, (now known) transformation and conversion to LAS can be done by CloudCompare. Either by using the graphical user interface or by using the command line interface[44].

First the point clouds were manually aligned to AHN3, by selecting equal points in both datasets using CloudCompare. Based on the coordinates of the corresponding features the coordinate transformation can be calculated. The matching points, their coordinates and the quality of the transformation are shown in Appendix B. The alignment found is of poor quality, with an RMSE of up to 30 cm. This can likely be attributed to the limited amount of corresponding features used in the alignment process, limiting the redundancy of the process for operator error and mismatches between points.

Unfortunately attempts to automatically refine the match/referencing using Iterative Closest Point algorithms failed, likely due to a lack in overlap between the point clouds. Between the different TLS point clouds the overlap is small, as they cover different areas (see Figure 9 - 12). The overlap between AHN3 and the TLS point clouds is small due to the different viewing angle, as the airborne AHN contains mostly roofs and ground while the terrestrial clouds contain mostly facades. But more experiments would be required to come to a concluding answer.

The intensity scaling for PTX files is not automatically done by CloudCompare, but scaling of this variable is easily performed. After estimating the transformation matrix all required steps can be executed at once using the command line interface of CloudCompare[44]. An example of such command can be found below. This command opens the PTX-file, scales the intensity, applies the transformation and saves the resulting point clouds as LAS-file.

```
CloudCompare \
-O '[input].ptx' \
-AUTO_SAVE OFF \
-SF_OP 0 mult 65535 \
-APPLY_TRANS '[transformation matrix].txt' \
-C_EXPORT_FMT LAS \
-SAVE_CLOUDS ALL_AT_ONCE
```

An example script on the calculation of the transformation matrix and generation of the CloudCompare command is available in Appendix C.

Aligned and processed were the TLS point clouds: Mekelpark Tram; Mekelpark DTM; Mekelpark Trees and Mekelpark Gras. Not included were the Faculty of Architecture and the radar reflector in Wassenaar. The Faculty of Architecture is not included for practical reasons (time constraints), the radar reflector in Wassenaar is outside the extents of the “Groot Delft” AHN dataset.

Step 2: colouring AHN

To make the resulting point cloud easy navigable it is desired to add colouring. For this aerial photographs are used. The nationwide 'PDOK Luchtfoto' is available as WMS and WMTS services,

44 “Command line mode”, CloudCompare Wiki, http://www.cloudcompare.org/doc/wiki/index.php?title=Command_line_mode. Retrieved 2017-10-30.

webservices providing data on demand[30]. Using GDAL (`gdal_translate`) this image can be tiled and downloaded as GeoTiff. LASTools' `lascolor` is able to add the data to the points in a LiDAR dataset[46].

Given lack of support for BigTiff, (Geo)Tiff files larger than 4 GiB, in lascolor the images must be tiled to less than 4 GiB each (without compression). In practice this requirement is met when only one AHN tile is processed at a time.

First the GDAL definition file for the PDOK Luchtfoto has to be created[45]. This is done by querying the PDOK WMS or WMTS server for available layers that match the coordinate reference system used by AHN (EPSG:7415, equal to EPSG:28992 for horizontal coordinates). As an example for the WMS service:

```
gdalinfo \
"WMS:https://geodata.nationaalgeoregister.nl/luchtfoto/wms?
request=GetCapabilities&SRS=EPSG:28992&format=image/png"
```

A possible return is, for the 2016 aerial photograph with 25 cm resolution:

```
https://geodata.nationaalgeoregister.nl/luchtfoto/rgb/wms?
SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&LAYERS=2016_ortho25&SRS
=EPSG:28992&BBOX=-
285401.92,22598.08,595401.92,903401.92&FORMAT=image/png
```

This URL can be converted to a GDAL definition file using `gdal_translate`:

```
gdal_translate "https://[previous URL]" wms.xml -of WMS
```

Given this definition file the aerial photograph can be downloaded using `gdal_translate`. It is recommended create a file for each AHN tile, not to hit the size limit of a conventional GeoTiff. A download command for tile `37EN1` will look like this, with `-projwin` defining the bounds of the tile:

```
gdal_translate -tr 0.25 0.25 -of GTiff \
-projwin 80000 450000 85000 443750 wms.xml 37EN1.tiff
```

It is important to define the desired resolution of the output (`-tr 0.25 0.25`, 25 cm), as the GDAL default may be higher than the resolution of the source.

Using `lascolor` (unfortunately only available on Windows) the points can be matched to the colours from the aerial photograph[46]. An example command, that outputs a coloured `C_37EN1_rgb.LAZ` based on `C_37EN1.LAZ` and `37EN1.tiff`, would be:

```
lascolor.exe -i C_37EN1.LAZ -image 37EN1.tiff -odix _rgb -olaz
```

Unfortunately, with the unlicensed version of LASTools, datasets with more than 3 million points will be slightly distorted and their intensity values nulled.

45 “WMS: generation of WMS service description XML file“, GDAL manual, http://www.gdal.org/frmt_wms.html. Retrieved 2017-10-30.

46 “lascolor”, rapidlasso GmbH, https://rapidlasso.com/lastools/lascolor/. Retrieved 2017-10-30.

Step 3: creation of the PoTree datastructure

Once all datasets are available as LAS-file, `PoTreeConverter` can be used to merge and tile the input point clouds to the Octree structure. This structure allows fast loading over the web, downloading only the points to the level of detail necessary for the current camera position and angle.

The coordinate system is a parameter to `PoTreeConverter`, but it is currently only used to show the extent of the dataset on a map[47], ie. the coordinate system is ignored in the conversion[48]. It is therefore important that all datasets are in the same coordinate system before conversion!

Using `PoTreeConverter` the input LAS-file is converted into a series of (open) binary or LAS files. For the coloured version of AHN3 the command will be as follows:

```
PotreeConverter C_37EN1_rgb.laz \
-o ./37EN1_rgb --material RGB -p AHN3 --show-skybox \
--projection "+proj=sterea +lat_0=52.15616055555555
+lon_0=5.38763888888889 +k=0.9999079 +x_0=155000 +y_0=463000
+ellps=bessel +units=m +no_defs" \
--edl-enabled --intensity-range 3 256 -r 256 \
-a CLASSIFICATION RGB
```

Some explanation on the parameters:

- `C_37EN1_rgb.laz`, the input file. Multiple files or a directory may be specified.
- `-o ./37EN1_rgb`, the output directory. The same directory may be used multiple times as long as the title/name (`-p`) of the cloud is different. This will create multiple viewers (in a single directory structure) that can later be merged manually.
- `--material RGB`, the attribute on display. All attributes mentioned under -a are stored, and may be selected for display in the viewer. Supported are: `RGB`, `ELEVATION`, `INTENSITY`, `INTENSITY_GRADIENT`, `RETURN_NUMBER`, `SOURCE` and `LEVEL_OF_DETAIL`. If the material is `RGB`, `INTENSITY` or `CLASSIFICATION` it should be mentioned under `-a` too!
- `-p AHN3`, the filename in the output directory (`-o`), `.html` will be appended automatically.
- `--show-skybox`, show some clouds as background (optional).
- `--projection "..."`, projection (EPSG:28992), ignored by PoTreeConvert but used by PoTree for addding a map.

47 "Projection with PoTreeConverter", https://github.com/potree/potree/issues/344. Retrieved 2017-10-23.
48 “Does PoTreeConverter work with different projection?”, https://github.com/potree/PotreeConverter/issues/54. Retrieved 2017-10-23.

- `--edl-enabled`, enable “Eye-Dome-Lighting” for a more natural looking point cloud.
- `--intensity-range 3 256 -r 256`, although a full intensity range up to 65536 is available, with AHN3 most points have an intensity value between 3 and 256. Without those limits the point cloud will be all black.
- `-a CLASSIFICATION RGB`, attributes to include in the output. Supported are: `RGB`, `INTENSITY` and `CLASSIFICATION`. As `INTENSITY` is truncated by lascolor this attribute may not be included.

This will create a point cloud based on the RGB values, but without intensity (lost due to the unlicensed version of LASTools). A version of the command using the intensity values (of the original AHN3 files) can be found in Appendix B.

Step 4: tiling InSAR data

The InSAR dataset has to be split in tiles that are small enough to be read by the browser. This is achieved by tiling the points in a Quadtree-like tiling schema, where subsequent tiles contain the same amount of points but in a smaller area. The tiles will be used in the map view, while the highest zoom level is used to provide the ellipsoid data for the 3D viewer

Clustering can be applied to group points with equal properties. If the maximum number of points is reached at lower zoom levels these groups can be shown rather than the individual points. The clustering method developed is elaborated on in chapter 5. The number of clusters (k = 75) was chosen as an empirical optimum between a cluttered map and enough clusters to represent the regional behaviour.

The output of this step is a series of tiles of increasing zoom level, with every zoom level the tile is divided into four equal parts. Every tile holds a fixed number of points related to (point) coverage of the tile. If more points are present in the dataset, clusters of points are shown instead of points. At the highest zoom level al points are included. The tiling loop can be written as the following *pseudo-code*:

- If more than 75 points are present and this is not the highest zoom level, apply clustering. Otherwise save all available points for this tile.
- If clustering is necessary:
  - Any point distinctly visible in a previous tile (lower zoom level) will be shown directly and will not be part of the clustering process. For visual consistency points visible at lower zoom levels should be shown at higher zoom levels too.
  - Determine the convex hull of the points. Calculate the ratio of the surface of the convex hull versus the surface of the tile[49]. Full coverage will create 75 clusters. The minimum is 1 cluster.

49 Calculating the convex hull of an irregular shaped group or a combination of two or more disconnected groups of points may result in an overestimation of the surface area. Various other constraints are possible, such as using the envelope instead of the convex hull (reducing computational complexity) or determining the coverage based on density rather than geometric extends. This is briefly elaborated on in chapter 5.

    - Apply K-Means with the previously calculated number of clusters/centroids[50].
    - If any clusters are formed by a single point, copy this point to the output.
    - Copy all centroids (and their member count) to the output.
- Divide the tile in four equal parts and run this process again, until the maximum zoom level is achieved. But only if there is at least one point.

Furthermore the colours indicating the linear deformation velocity are attached to the measurements in this step. As all data is available in this step the extremes can be calculated that form the basis of the colourramp.

As most web mapping applications use the Web-Mercator projection, all points are first transformed to this coordinate system. As the point cloud is in RD-coordinates these coordinates are added to the points. This will allow for using the same tiles for both the map view and the ellipsoids. Transforming the coordinates in the browser is possible, but storing the coordinates as Web-Mercator will allow for using standard tools. Adding the original RD-coordinates will guarantee that the ellipsoids are not distorted by the transformation back and forth (RD-coordinates → Web-Mercator → RD-coordinates).

The tile grid used is equal to the one used by OpenStreetMap, a Quadtree like structure, with the positive *x* tile-coordinate from left to right and the positive *y* tile-coordinate from top to bottom. This is done to ease configuration of the map layer in the final viewer. The dataset is rendered from the lowest zoom level (0) with world coverage, iteratively higher zoom levels are created. As most tiles will be empty (up to zoom level 7) they will not spawn children (of a higher zoom level). Therefore high zoom level coverage is limited to populated parts of the map, limiting the number of tiles generated.

An example on how to use K-Means clustering and colouring is available in Appendix C.

Step 5: PoTree viewer, integrating datasets

All previous steps come together in the viewer. The point clouds will be shown in the same scene as the radar observations (error ellipsoids). As PoTree is built as a plugin to the versatile Three.js 3D library[51], this library can be used to form the error ellipsoids.

As much as possible has been used from the standard PoTree implementation, the application is implemented with the least possible changes to the original PoTree viewer. This allows for quick adaptation to future updates in the PoTree library and minimises the impact of adjustments on browser compatibility. An important (structural) change is that the mapping library, OpenLayers 3, was swapped for the newer OpenLayers 4.

The viewer can be dividend in three connected, but distinct, components:

50 It is possible to re-use the centroids/clusters of the previous zoom level that coincide with the extends of the current tile as basis for the new clusters. In theory this could (visually) stabilise the clustering, as clusters would attain the same or similar position in subsequent zoom levels. In practice this showed very little effect as often new centroids had to be added (at random) as on average only around one quarter of the previous centroids is within the extends of the new tile. Those randomly added clusters disturbed the centroid locations beyond recognition.

51 “three.js”, https://threejs.org/. Retrieved 2017-11-08.

1. 3D view

   Main screen of the application. Drawing and loading of the point cloud and navigating through it. This includes traversing the Octree cells (point cloud) and Quadtree tiles (InSAR) as necessary and drawing them on screen in accordance with the settings. (This is done automatically.)

2. Settings

   A 'passive' component, implemented as a sidebar. Settings influence the viewer directly, but the viewer does not influence the settings (except for camera positioning).

3. Map

   Shows the camera position on the map and shows where information is available.

All three components were edited with the implementation of the radar error ellipsoids. The changes/additions will be listed per component:

1. 3D view

   - Multiple parallel point clouds were added. Although this is a default feature, it is not enabled by default and should thus be mentioned.
   - Ellipsoids are implemented as a Three.JS Level of Detail (LOD) object, with three levels of detail. At lower zoom levels the ellipsoids (represented by a series of vertices) are replaced by boxes, containing less vertices and putting less strain on the browser.

     Using the LOD configuration it is possible to show ellipsoids enlarged at larger distances. This makes it easier to find ellipsoids at large distances. This is currently not implemented.

     Ellipsoids are coloured by the estimated linear deformation speed of the scatterer. Colours are provided by PyPlot in the tiling step and included in the GeoJSON output.
   - An extra loading mechanism was added, loading radar tiles around the focus (target) of the camera. This limits the number of ellipsoids in view, reducing the computational load.
   - A colourbar is added, based on the characteristics of the dataset acquired during loading.

2. Settings

   - Options for the ellipsoid scale were added. Ellipsoids can be scaled to 1/2/3σ.

3. Map

   - OpenLayers 3 was swapped with OpenLayers 4, without further changes.
   - The persistent scatterers and clusters of persistent scatterers were added as a layer to the map.Including matching styling for both clusters and points.

4.C. Optional steps

As an additional step the implicit classification available in AHN1 and AHN2 can be added to the LAS-file as ASPRS compatible code.

Adding classification to AHN1/2

AHN1 and AHN2 are subdivided into two groups: *gefilterd* and *uitgefilterd*. The first one being ground points (ASPRS LAS classification code 2), the second one unclassified (code 0). When added to the LAS-file PoTree will be able to filter points based on those properties.

Using `las2las` this classification code can be added to the LAS-file:

```
las2las -i [input].laz -set_classification 2 -odix _c -olaz
```

The version including the classification (in this case set to 2, ground) will be saved with as LAZ with a c suffix: *input*`_c.laz`.

5. Clustering of scatterers

There are over a million persistent scatterers in the InSAR dataset. Although a million points of the point cloud are shown continuously the information carried by a similar amount of persistent scatterers is much harder to interpret. By clustering similar measurements it is possible to limit the information output to the user (compare Figure 21 to Figure 22). The desired clusters should describe regional behaviour; without masking anomalies. Focussing the users' attention to distinct features within a regional trend.

Figure 21: All measurements in the InSAR dataset overlayed on the university campus. Points overlay each other, complicating analysis.

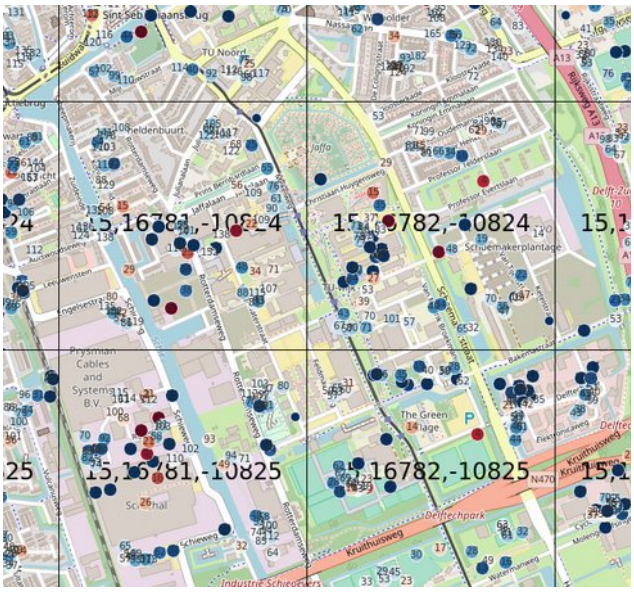


Figure 22: After clustering the dataset (Figure 21) is summarised as clusters. Shown in black are the tile boundaries.

Furthermore it is impossible to show all relatively complex ellipsoids (rendered as a series of faces, approximating the smooth surface of an ellipsoid) in the 3D view without overloading the 3D engine at the client. The test client was limited to around 2000 concurrent appearances before performance issues ensued. To circumvent simultaneous loading of all persistent scatters a tiling technique similar to the Octree used by PoTree is deployed. If more than 75 points are present in a single tile, clustering is applied first.

This clustering is based on K-Means clustering. This technique was chosen for its scalability and the guaranteed reduction of the output to a specified number of clusters. Using K-Means it is possible to estimate a predefined number of centroids of an (almost) infinite set of input points[52]. It has to be noted that a similar procedure may be implemented with a different clustering technique as well.

52 An overview of clustering techniques is available on:
"Clustering", scikit-learn, http://scikit-learn.org/stable/modules/clustering.html. Retrieved 2017-10-24.

The predefined numbers of centroids is placed at random on the map. These centroids are iteratively positioned between (input) points. For each centroid it is determined which points it is closest to, the centroid is then moved to the centre (average coordinates) of these points. This step is repeated until either all centroids remain stable (ie. no points are swapped between centroids) or a predefined number of iterations is reached. Major drawback is that due to this random initialisation of the centroids final results (clusters) may vary each run.

By implementing this procedure in the tiling process it is possible to reduce the number of points shown while maintaining variability in the output. If a random sample or an average would be chosen to represent a number of points all anomalies would be masked by the vast amount of data.

In determining the appropriate number of clusters it is important to consider that not every tiles may be fully 'covered in data' (Figure 23). To circumvent this problem the convex hull of the points is calculated, the ratio between the area covered and the tile size determines how many of the (default) 75 clusters will be shown. Another method would be to calculate the density of the coverage and setting the amount of clusters accordingly.

The first option has the advantage of a evenly distributed map, where all areas of the map are covered in (roughly) the same amount of clusters. The second option honours the variations in density that may be present in the data. In the demo application the first method was implemented.

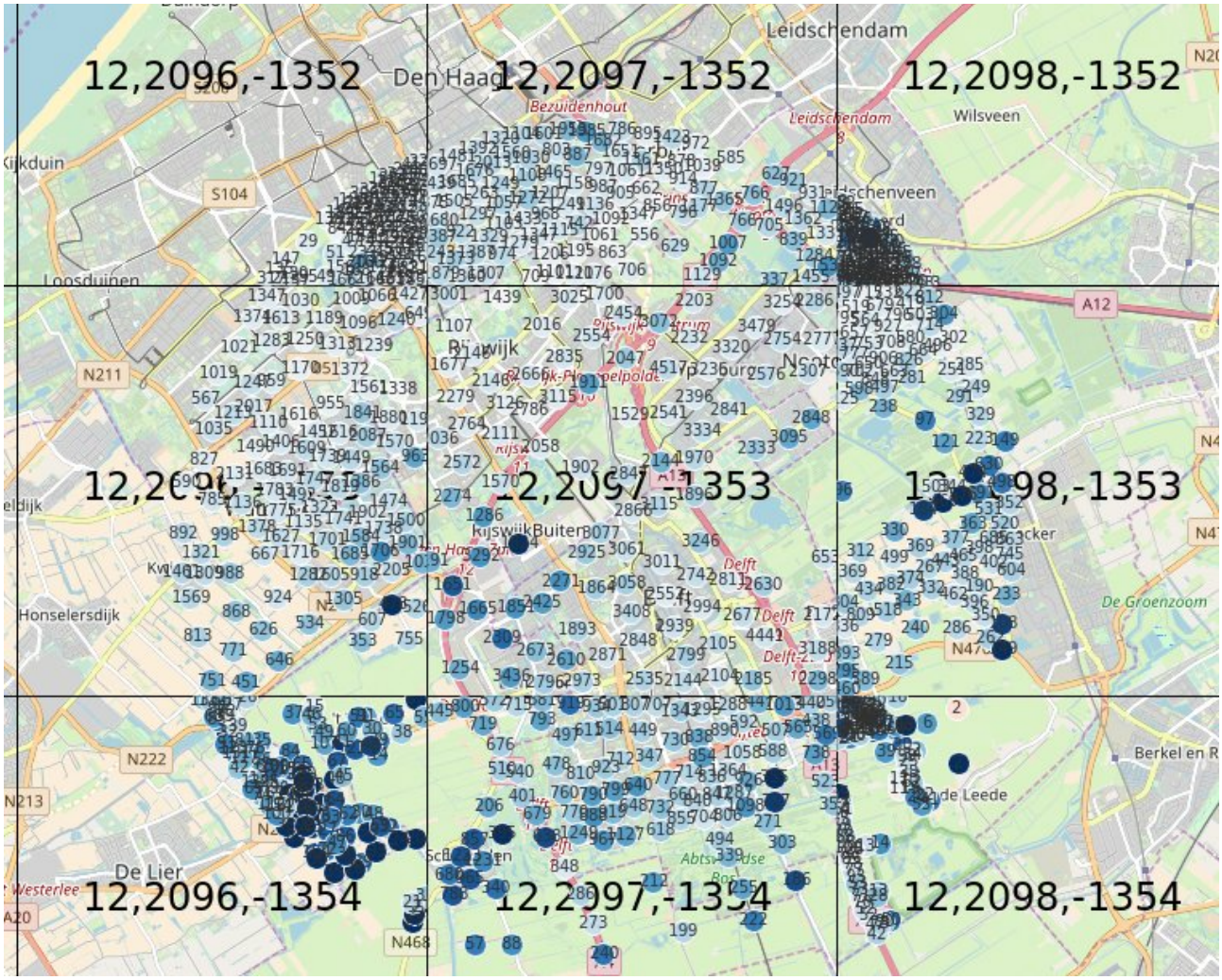


Figure 23: Not all tiles are fully covered in data, while the data in each tile was divided in 100 clusters. As a result some tiles (marked in black) have high (local) densities of points, while others are evenly distributed. (Background: OpenStreetMap)

Including deformation behaviour

Clustering is implemented in four dimensions: position (x, y, z) and the (estimated) linear (deformation) velocity of the persistent scatterer (v). Aim is to not only cluster scatterers that are close together (eg. a house) but include deformation behaviour (eg. subsiding garage, attached to the house) in the clustering process.

Unfortunately the four dimensions (x, y, z, v) are scaled differently. While the positions (x, y, z) of persistent scatterers vary by meters (RD coordinates) the deformation velocity is not larger than ± a few millimetre per year. As the Euclidean distance (in four dimensions) is used to determine which centroid is the closest the (relatively) small difference in deformation velocity is negligible in the process.

By scaling the deformation velocity it's contribution to the final clustering (centroid location) can be enhanced. In Figure 24 the effects of scaling can be seen. When no scaling is applied (left) the clustering is dominated by the geometry of the scene. If the velocity is scaled by 5000 (an empiric number) the points are grouped by a combination of their deformation velocity and their geometric

proximity. An extreme case is shown on the right, by scaling with one million the clustering is governed by the deformation behaviour, connecting points of similar velocity regardless of their distance.

When used for a mapping application it is desirable to find a compromise between the first two options. Geometric integrity is important to show the features in their appropriate place (rather than a meaningless average position) while the deformation behaviour should not vanish in the average of a geometric group.

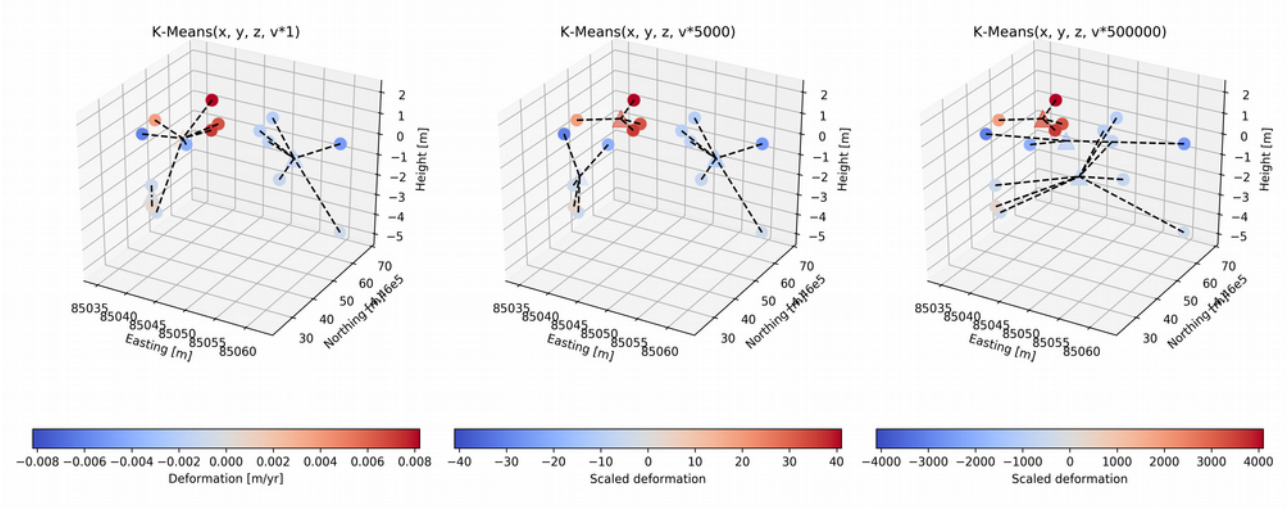


Figure 24: Clusters of data, with increased scaling of the velocity component. Input points are shwon as circles, centroids as pyramids.

Based on this clustering regional trends can be estimated, while retaining (some level of) local behaviour. An example is shown in Figure 25 in 3D. This will guide users in the vast amount of information available in the 3D display.

Figure 25: Clustering of 1 million persistent scatterers in 1500 clusters. Size of the cluster related to the points in the cluster, color indicates (average) deformation speed of the cluster. (Vertical scaling factor: 5000.)

6. Results

In this chapter the main result, the viewer, will be shown and its functions discussed. The effectiveness of the combinations made will be assessed. The other parts of this chapter will focus on practical aspects of the conversion process and storage of the data. A word on the scalability of this process will conclude the chapter.

6.A. The viewer, demo application

The viewer was designed with a broad audience in mind. Accessible to the greater public and built such that it can be built and included in publications by researchers. At the time of writing the demo application was available at: http://dev.fwrite.org/radar/[53].

Accessibility of the viewer is guaranteed by a broad support for (modern) browsers. Furthermore the (native) PoTree controls were intuitive to all users the application was demonstrated to. Building a viewer is not (yet) point an click. But recipe in Appendix A should be doable for all researchers familiar with the commandline interface and some basic programming (or scripting).

As much of the original PoTree installation was retained to ease the creation process, updates and future extensions. By using the original PoTreeConverter structure, static files are used for data storage. With static files no complex webserver is necessary to transfer the files to the client. This allows for adding combinations to publications, where the application should be available years later. Either by attaching it to the publication ('zip-file') or by storing the application in a low-maintenance object storage (see 6.E).

In this chapter the features of the demo application are discussed. The main viewport (Figure 26) can be divided in three parts: the (3D) viewport, settings and map. Each will be discussed separately.

53 Redirecting to https://potree.o.auroraobjects.eu/Groot/Map_vtiles_Groot.html.

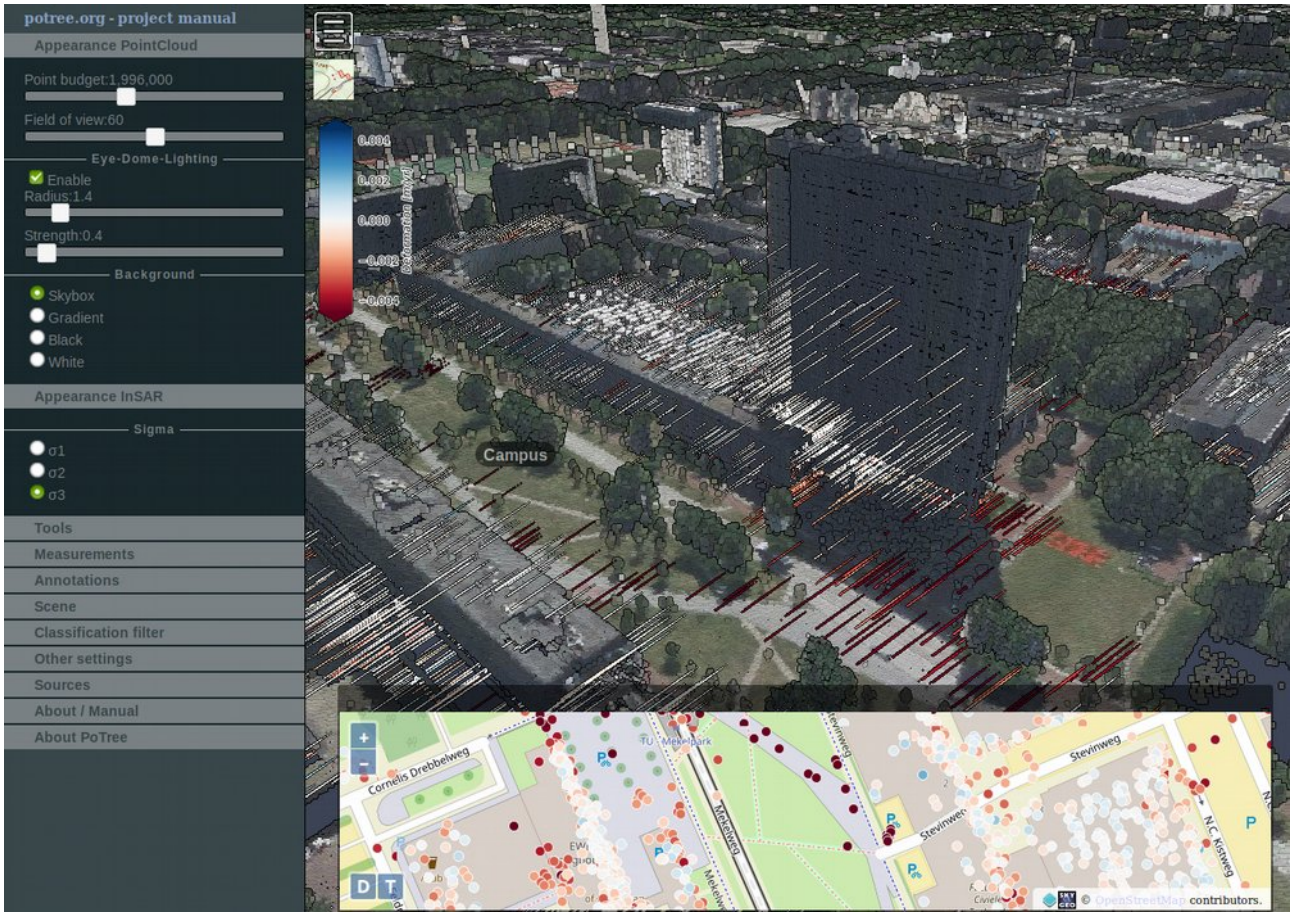


Figure 26: Demo application viewer canvas. The settings menu is on the left, a map view of the scatterers at the bottom. In the map (3D) viewport the colourbar for the linear deformation speed can be seen. (Faculty of EEMCS, Delft University of Technology)

3D viewport

The 3D viewport is controlled using the mouse. A '3 button' mouse, also known as a mouse with a scrollwheel, is recommended for the best viewing experience. The controls are defined as follows: double (right) click to set a target position; left click and drag to rotate around this target position; right click and drag to move around. The scrollwheel is used to zoom.

Ten different point clouds are included in the demo application: AHN1; AHN2 and AHN3 (RGB and intensity) as airborne LiDAR point clouds. From the terrestrial laser scanner Mekelpark Tram, Mekelpark Gras, Mekelpark DTM and Mekelpark Trees are included. From the mobile scanner (Faculty of Architecture and surroundings) a recording from 2013 and one from 2016 are included.

Ellipsoids are dynamically loaded from the same source (tiles) as the map view. A square of 5×5 tiles (of the highest zoom level) around the target is loaded. These ellipsoids are then shown in the 3D view (see Figure 17 for a close-up). If the viewer changes target this process is repeated, loading new tiles (ellipsoids) and purging ellipsoids no longer in view.

Using the same technique it is possible to show regional behaviour (clusters of scatterers) in places where the individual ellipsoids are not currently loaded. An example was shown in Figure 25. This functionality is currently not implemented.

Three buttons can be seen in the left top corner of the viewer: the settings toggle, the map toggle and the colourbar.

Map

Shown on the map (Figure 27) are the clusters (zoomed out) or individual scatters (zoomed in). As a background OpenStreetMap is shown (see chapter 3.F). The colours match the estimated linear deformation rate with the colours corresponding to the colours (and colourbar) shown in the 3D view.

The map is in the Web-Mercator projection (EPSG:3857) instead of the RD-coordinates (EPSG:28992) of the point cloud. The projection difference is unnoticeable in the example application.

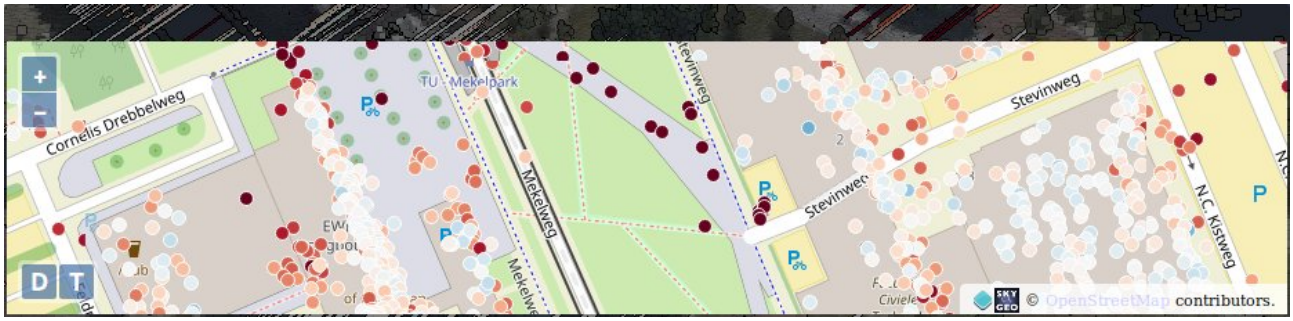


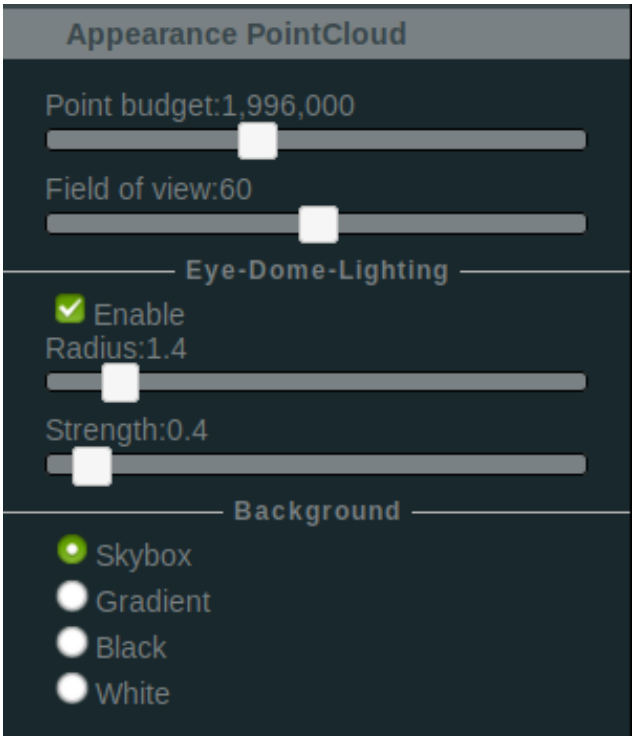
Figure 27: Map, as shown in the viewer. Scatterers are marked with dots and coloured according to their deformation velocity.

The map can be moved independent from the 3D view. As soon as the cursor is on the map the RD-coordinates at the cursor are shown in the top right corner. Double clicking on the map will focus (camera target) the 3D view at this point and move the camera close to the point (camera position).

The map 'window' can be moved by dragging it (click and drag on the top grey bar) and enlarged on the bottom and right side (click and drag).

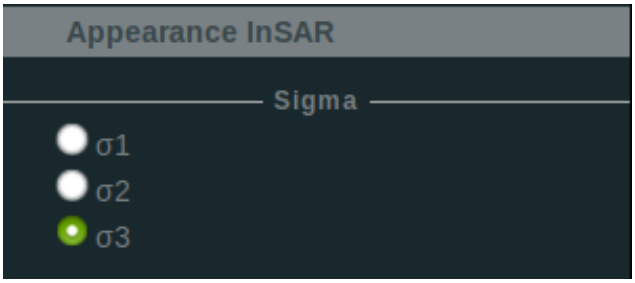
Settings sidebar

Many of those features are standard PoTree features. The InSAR scaling features and the documentation on the sources were added to the sidebar. In this section all functions of the sidebar will be discussed.



Appearance of the point cloud

- Point budget (default: 1 000 000)
  How many points will be in memory (maximum), also known as *high-water mark*. Cells of the Octree will be loaded until this maximum is reached.
- Field of view (default: 60)
  Camera setting, determining the field of view of the virtual camera. Lower values will give a telescopic (binocular) effect. Extreme values may distort the image at the edges as the 3D world is projected on a flat screen.
- Eye-Dome-Lighting (default set during conversion)
  "Is a non-photorealistic, image-based shading technique designed to improve depth perception in scientific visualization images."[54] Shading is added to focus users on points close by (the virtual camera) and accentuate edges[55]. As a side effect the shading added allows for the identification of individual points.
- Background (default set during conversion)
  Backgroundcolour behind the point cloud. Skybox (a cloud like pattern) and gradient will allow for the distinction between up and down.

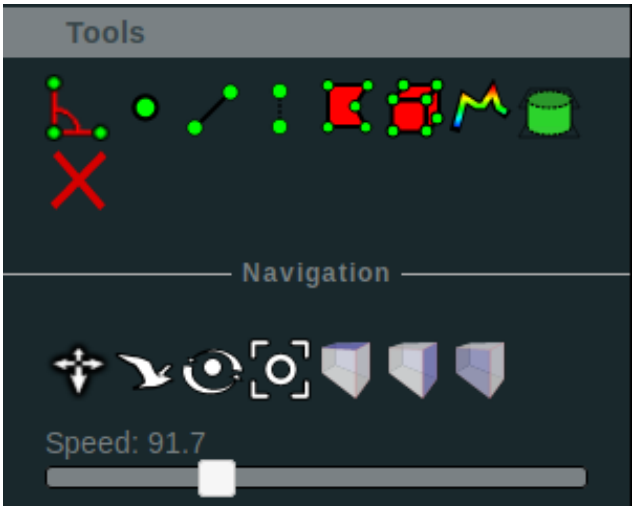


Appearance InSAR

- Sigma (default: σ_1)
  Scaling of the radar error ellipsoids, with 1, 2 or 3 σ (standard deviation).

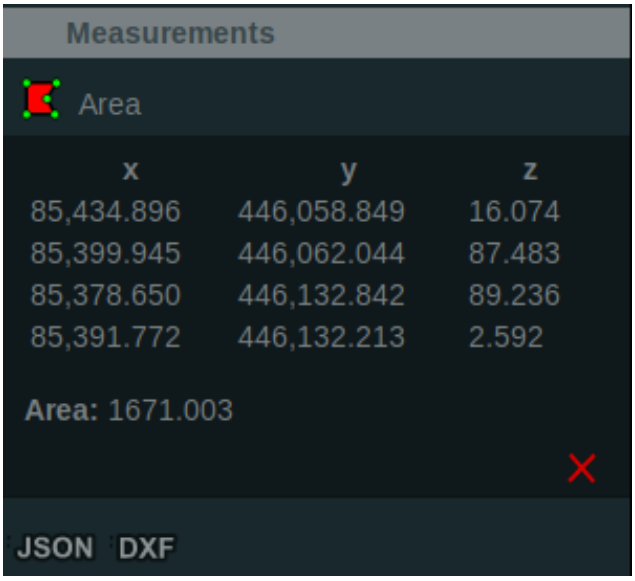
54 "Eye-Dome Lighting: a non-photorealistic shading technique", C. Boucheny & A. Ribes, Kitware Blog, April 2011, https://blog.kitware.com/eye-dome-lighting-a-non-photorealistic-shading-technique/. Retrieved 2017-11-06.

55 "A neighbour pixel will reduce the lighting at p if its depth is lower (i.e. closer to the viewer) than the one of p. This procedure defines a shading amount that depends solely on the depth values of the close neighbours." Thus more shading will be applied to points adjacent to points closer to the camera. Making those points 'darker' will increase contrast between points.

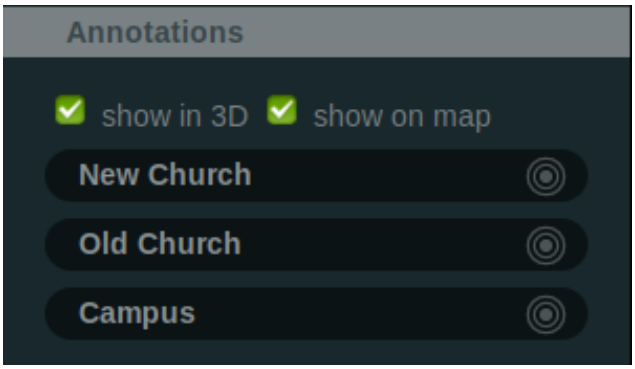


Tools
Default measurement tools available in PoTree.

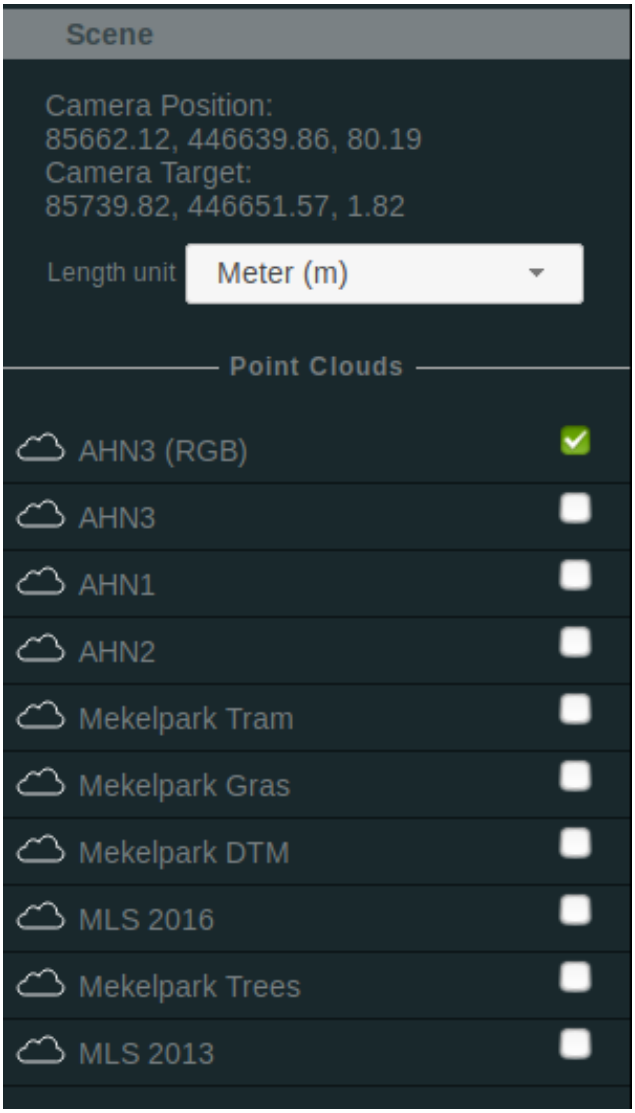
- Tools available
  - : Angles in a triangle;
  - : Point information (height, coordinates, RGB);
  - : Distance between two or more points;
  - : Height difference (between two points);
  - : Surface area;
  - : Volume, show a cube of specified dimensions in the point cloud;
  - : Profile;
  - : Volume, mark points in a specified volume in the point cloud.
- Navigation
  - Various navigation/control modes, and pre-defined camera angles (top view and side view).
  - Speed determines the sensibility of the viewer for cursor movement.



Measurements
Populated after using one of the measurement tools. Shown as an example is an area mesurement. Results can be exported as JSON or DXF file.

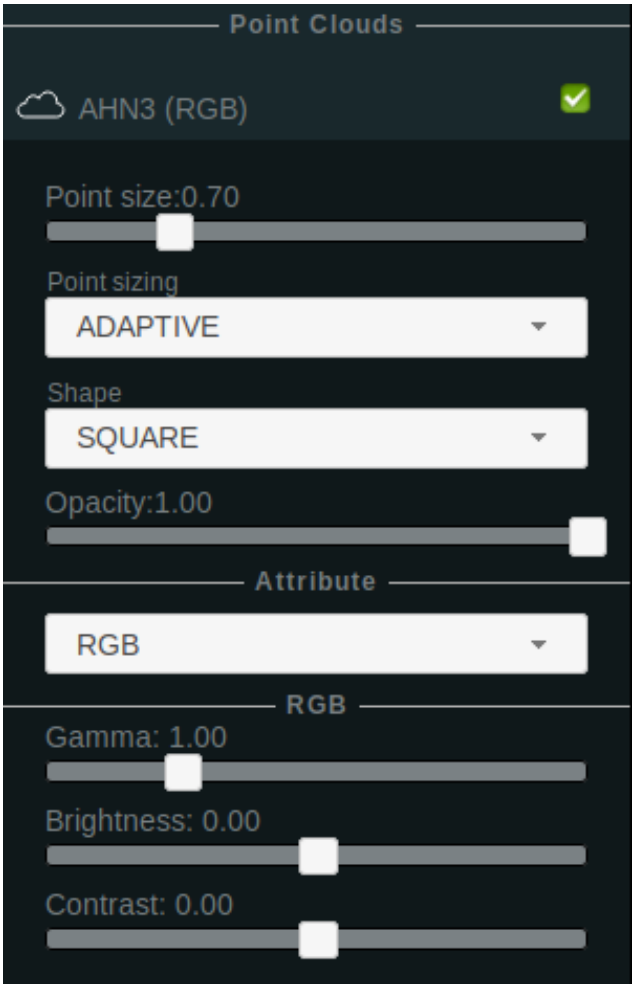


Annotations
List of annotations added to the viewer. Shown both in 3D (as label) and as grey dot on the map. When a target symbol ( ) is present, camera settings are linked to the annotation. Clicking the annotation will then result in 'flying to' the object annotated.



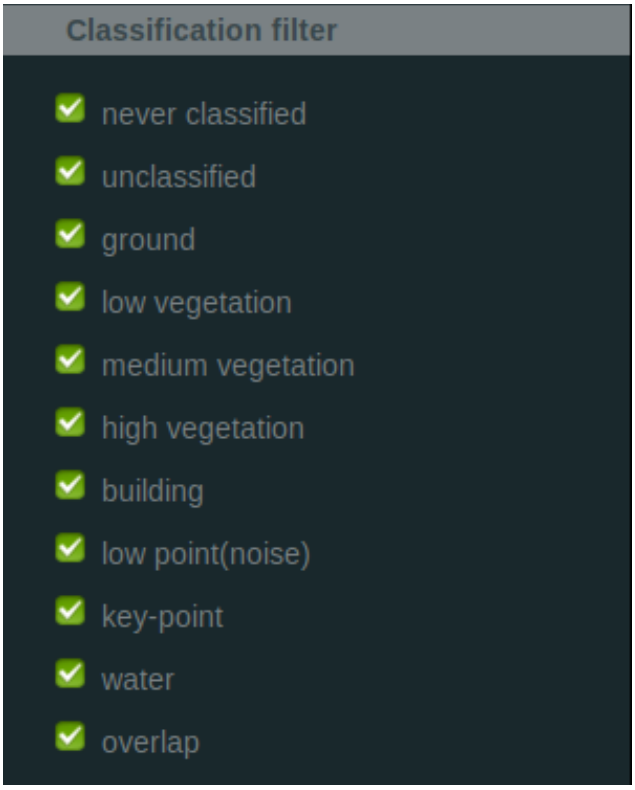
Scene

- Camera position & camera target
  Position and rotation center (target) of the virtual camera. This does not include the direction the camera is pointing in (yaw and pitch of the camera).
  The length unit set is used by the measurement tools for the correct labeling of distances and areas. No conversion is done, this has to be set to the length unit of the unerlying coordinate system.
- Point clouds
  List of point clouds included in this viewer. They may be enabled/disabled using the thickboxes.
  Each pointcloud has detailed settings that can be accessed by clicking on its name or the cloud icon.



Scene, point cloud settings
Settings can be set per point cloud.

- Point size (default depends on point cloud, 0.5 – 1.0)
  Determines the size of each individual point in view. Taken to high (large) points will overlap, taken to low (small) the cohesion between points (eg. points on the same wall) is no longer apparent.
- Point sizing (default: adaptive)
  Adaptive scaling makes points close to the viewer smaller compared to points further away. This prevents points in close proximity to the virtual camera from 'saturating' the view.
- Shape (default: square)
  Circular points are more computational intensive. The paraboloid shape will 'melt' adjacent points to a 3D surface/volume.
- Attribute (default depends on point cloud)
  Select the attribute to be shown, eg. RGB, elevation, level of detail, etc..
- RGB
  Some colouring settings for the RGB viewer.



Classification filter
When ASPRS classification data[16] is present in the point clouds and this information is retained during conversion filters can be applied here. When no classification was provided all data is marked as 'never classified'.

About
Two menu items with information on the sources and the project were added. The tab 'About PoTree' contains information on PoTree and the modules it consists of.

6.B. Effective combinations

To find the dominant scatterer it is necessary to have a high enough level detail in the point cloud at the area of interest. Given the airborne nature of AHN3 this method is less suitable for scatterers on building facades, as limited information will be available on their geometry. This is shown in Figure 28, in this figure window frames and other scattering features of the facade are invisible. While in Figure 29, recorded using a terrestrial laser scanner those features are well defined. See Figure 9/10 in chapter 3.B for a similar comparison between point coverage of various data products.

An effective combination of these sources is a combination where the high density properties of the point cloud are exploited to the fullest. Thus giving centimetre/decimetre level of detail about potential scatterers around the error ellipsoid.

The poor (compared to InSAR) accuracy of the AHN data (horizontal 50 cm, vertical 10 cm, both 1σ, see chapter 3.A) may become troublesome when matching both data sources automatically.

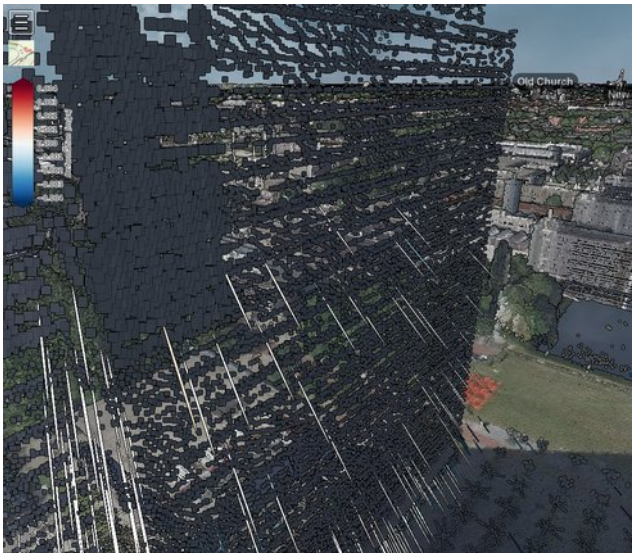


Figure 28: Error ellipsoids on the facade of the faculty of EEMCS (EWI) overlayed on AHN3.

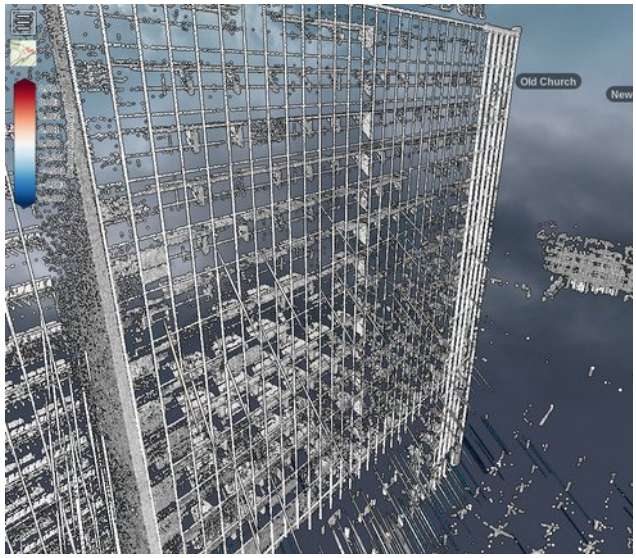


Figure 29: Same ellipsoids and viewing angle as in Figure 28, but overlayed on the TLS cloud "Mekelpark Trees".

6.C. Point density settings

During Octree creation the Octree spacing can be set, setting how far points should be apart to be included in the highest level Octree cell. Higher density settings will create fewer Octree cells, at the cost of including more points per cell.

By default one million =concurrent points are shown in PoTree, allowing for acceptable performance on most clients. This *high-water mark* can be adjusted on the client's side.

Octree cells close to the camera position will be loaded in higher resolutions until the set high watermark is reached. This may include cells that are only partially visible in the current viewport. With increasing density fewer Octree cells can be loaded before the high watermark is reached.

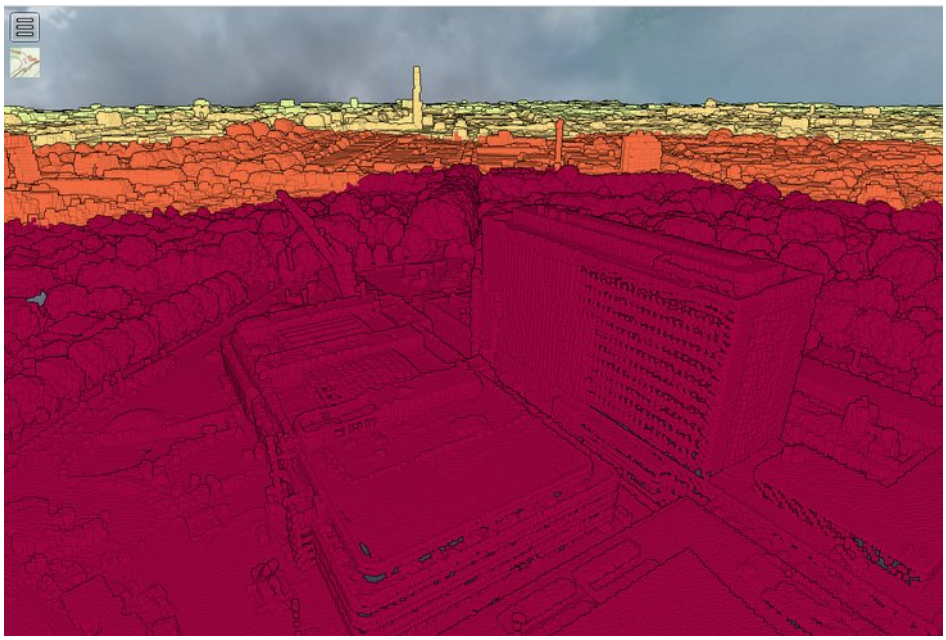
As a result fewer cells are shown in high resolution. This may result in part of the object of interest not being shown in the highest possible resolution. Possible scatterers may be invisible because of this.

At the same time a lower density will require more (small) files to be stored and more requests (from the client) are necessary to load the point cloud. This may result in increased cost at the side of the provider. This will be further discussed in 6.E (Object storage).

As an experiment AHN3 tile `30DZ2` was processed using PoTreeConverter using different density settings (*d*). Shown is the Level of Detail visible in the viewer (high watermark at one million points). Red indicates a high level of detail, orange to green a lower level of detail shown. As an example the facade of the Haga Hospital (Els Borst-Eilersplein, The Hague) was used.

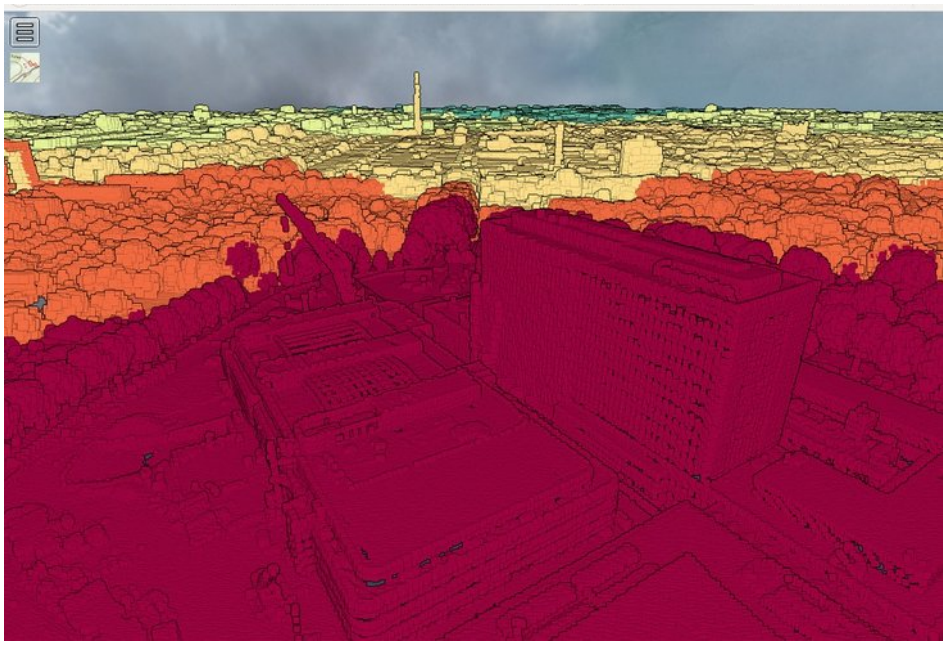
d = 200

115536 files
12 GB
Biggest cell: 1076 kB



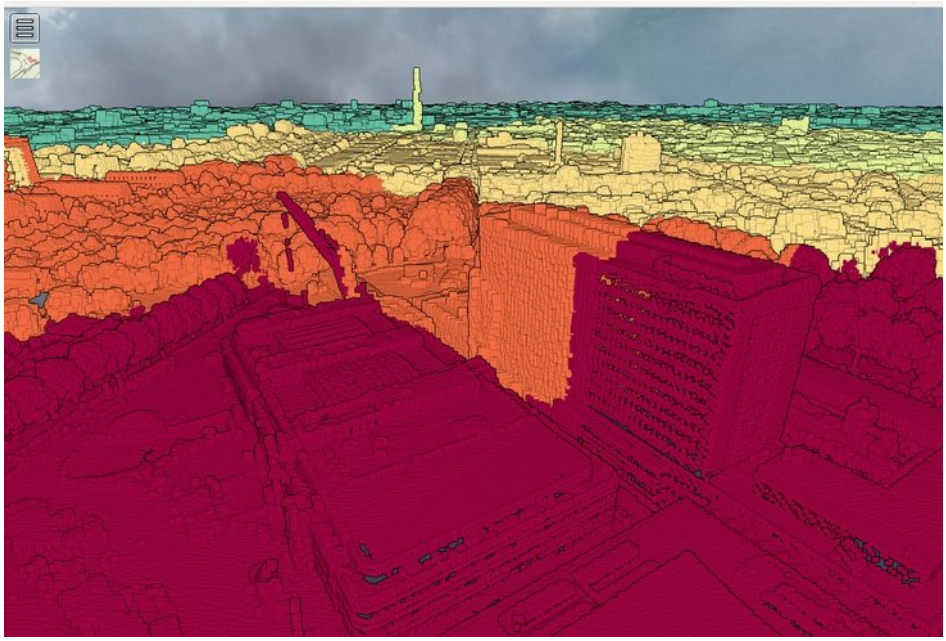
d = 300

88606 files
12 GB
Biggest cell: 2264 kB



d = 400

73275 files
12 GB
Biggest cell: 3960 kB



As a compromise a density of 300 was used, this results in 24% fewer files compared to the default density of 200. Reducing the costs for the provider of the data. The level of detail at short range is acceptable for the application, the whole facade is loaded in the highest available detail.

6.D. Meshes

Meshes could provide an alternative to point clouds, their properties as closed surface could allow analysis is lower density point clouds. As this was not the focus of this work further research is definitely needed before any conclusions can be drawn. It is possible to load meshes in Three.js and add them to the PoTree viewer[56].

56 PoTree examples including various meshes, read from the PLY format, http://potree.org/demo/potree_1.5/examples/meshes.html. Retrieved 2017-10-30.
A OBJ format loader is included in Three.js, http://potree.org/demo/potree_1.5/examples/meshes.html. Retrieved

In Figure 30 to Figure 35 the results of mesh creation with standard CloudCompare tools is shown[57]. In Figure 30 - 32 only the points from AHN3 are used. In Figure 33 - 35 AHN3 is combined with the point clouds Mekelpark DTM and Mekelpark Gras.

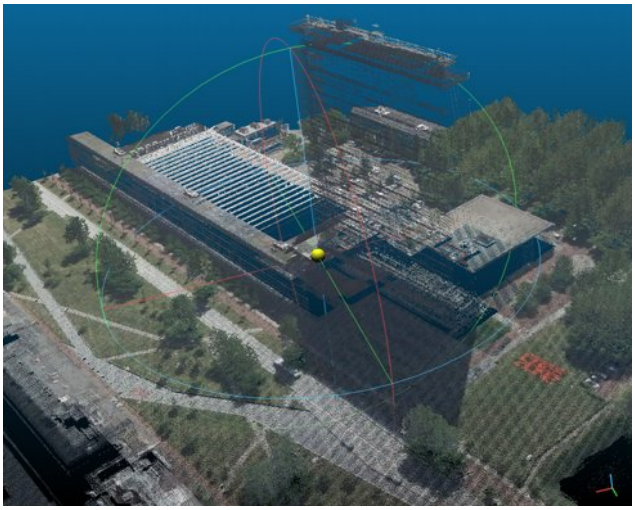


Figure 30: Coloured point cloud from AHN3.

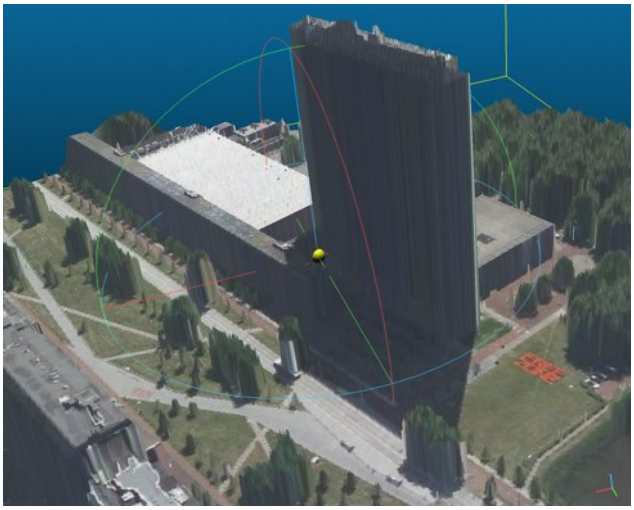


Figure 31: Mesh calculated by 'draping' a cloth over the scene. Note the trees are closed up to ground level.

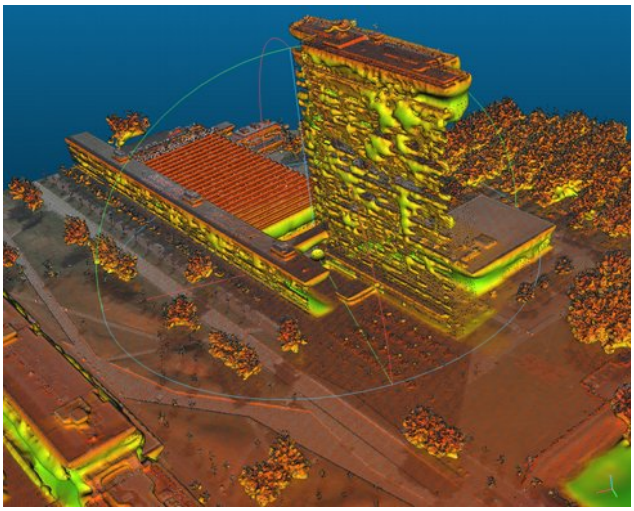


Figure 32: Mesh created using Poisson Surface Reconstruction[57]. Note that in areas of low surface density voids are created.

2017-10-30.

57 "Poisson Surface Reconstruction", CloudCompare Wiki, http://www.cloudcompare.org/doc/wiki/index.php?title=Poisson_Surface_Reconstruction_(plugin). Retrieved 2017-10-30.

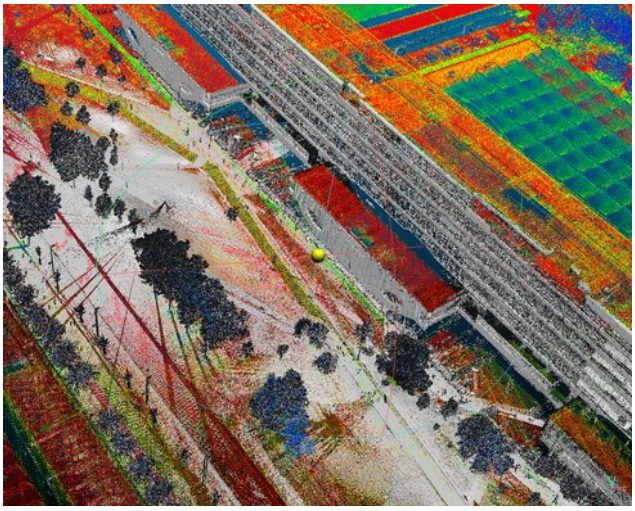


Figure 33: Similar to Figure 9 point clouds from Terrestial Laser Scanners were added to compenstate for the lower point density on the facades.

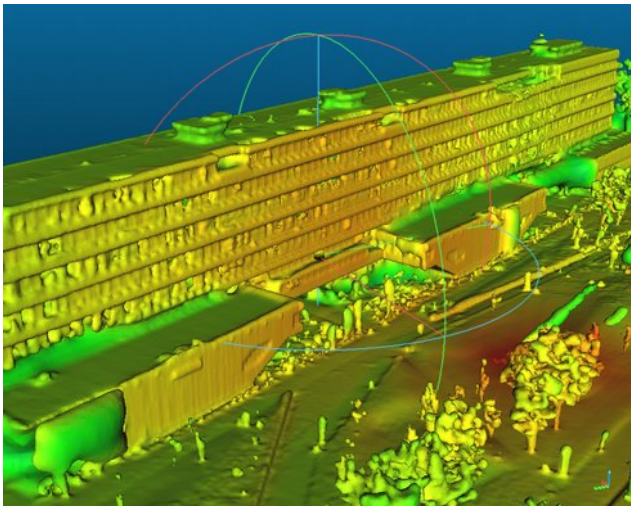


Figure 34: CEG (CiTG) reconstructed using Poisson Surface Reconstruction.

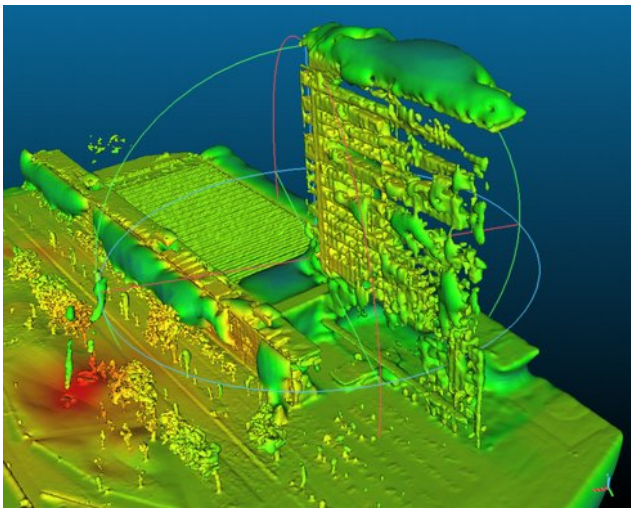


Figure 35: At the faculty of EEMCS (EWI) the result of Poisson Surface Reconstruction[57] is not necessarely better than with only AHN3 as source (compare to Figure 32).

Before those meshes will be web-ready their (file)size has to be reduced significantly or an efficient transport method has to be found. The meshes shown are around 200 MB each. An alternative could be adding the ellipsoids to Google Earth as models. This will allow overlaying the error ellipsoids on the Google mesh.

6.E. Object storage

Given the intended *write once, read many* approach, object storage could be an ideal candidate to store and serve the point cloud data to the viewers. Object storage solutions, Amazon S3 for example, allow for storing an unlimited number of files ('objects') to be stored and served to users on a *pay as you go* basis, without the maintenance responsibilities of a webserver.

Payment is based on objects stored (per gigabyte) and transactions (uploading or retrieving files). Various storage types with different properties exist. Those properties form the balance between transaction (retrieval) and (monthly) storage costs. The S3 Infrequent Access (S3 - IA) product allows for lower storage costs at the expense of higher transaction costs. Ideal for publications with infrequent visitors, but there is a catch.

The Octree consists of many (small) files. With Amazon S3 Infrequent Access the minimum size of an object is 128 KiB, any smaller object will be billed as 128 KiB of storage[58]. Given the Groot Delft dataset (described in 3.A) this results in the file counts as shown in Table 2.

	Size on disk			Billed size on S3 - IA	
	bytes	GiB	Files	bytes	GiB
AHN1	1142258294	1.06	12512	1950132981	1.82
AHN2	62590843891	58.29	946755	142120292416	132.36
AHN3	77164057944	71.86	1058891	174244620482	162.28
AHN3 (RGB)	87319375296	81.32	1064286	183190908124	170.61

Table 2: "Groot Delft" after conversion by PoTreeConverter, with a density setting of 300.

At the time of writing storage of the AHN3 (RGB) cloud on Amazon S3 (Ireland)[58] would cost (without taxes): $ 5.32 to upload the files (transaction fees); and $ 1.87 monthly for storage. With S3 Infrequent Access those numbers would be: $ 10.64 to upload the files and $ 2.13 in monthly storage. Doubling the initial costs and a 14% increase in monthly fees.

This prices do not include retrieval of the data by visitors of the application. Costs incurred by users can not be controlled. For S3 they are $ 0.004 per 10 000 requests, $ 0.090 per GB data. Or $ 0.01 per 10 000 requests excluding a $ 0.01 per GB 'data retrieval fee' and bandwidth for S3 Infrequent Access storage. Although very dependent on visitor behaviour, a minute of scrolling through Delft results in 304 requests and 97 MB of data transferred. With frequent visitors this may incur high costs for data transmission.

6.F. Scalability of point cloud conversion

Processing was done on a Intel i5 with 8 GB of RAM and two SSD's for storage. For two AHN3 tiles processing takes around 15 minutes. Processing of the nine tiles of the “Groot Delft” dataset, consisting of 5 billion points takes more than 2½ hours. Processing was benchmarked for an increasing point load. Iteratively one extra of the following AHN3 tiles (see Figure 6) was added to the processing queue: `30DZ2`, `30GZ1`, `30GZ2`, `37BN2`, `37EN1`, `37EN2`, `37BZ2`, `37EZ1`,

58 “Amazon S3 Pricing”, Amazon, requested 2017-10-17. (https://aws.amazon.com/s3/pricing)

37EZ2. First only 30DZ2 was processed, then 30DZ2 together with 30GZ1 and so on. A conversion of 24 tiles[59] was done to test the behaviour for larger sets of tiles. In Figure 36 the timing results are shown both as function of tiles and point count. A linear trend is plotted on top, a clear linear trend can be seen in the data.

The Netherlands is tiled in 1441 tiles. Given that nine tiles took 159 minutes to convert it would take 424 hours (18 days) to complete all tiles (assuming no restrictions on disk; I/O, memory, etc.). This does not include the time needed for colouring the point cloud with the aerial photograph.

PoTreeConverter is implemented as a single thread, using only one of the available processors at a time. As (in the test setup) the processing power was the limiting factor, this limits the overall processing speed. Masse-PoTreeConverter may provide a solution to this problem, allowing PoTreeConverter to run in parallel. This tool, by the Netherlands eScience Center, was briefly mentioned in chapter 2.A. With this tool the time needed for the conversion of the 597 billion points in AHN2 was reduced from 100 days to only 15 days[60].

PoTreeConverter was likely updated after the paper was published in 2015, accelerating the conversion. The authors report a conversion rate of 250 million points per hour on a much more advanced system (128 GB RAM, 16 cores)[60]. Their estimate of 100 days is based on this number. The consumer processor used in this study was able to process almost 1.8 billion points per hour – about eight times more!

Entwine.io is built specifically for massive point clouds ("terrabytes in scale")[9] and may perform better under those circumstances. Entwine.io was not benchmarked in this study, but publicly available suggests slightly better performance: 2.6 billion points per hour on 30 cores and 60 GB RAM[61]. This includes reprojection to the Web-Mercator projetion.

The conversion between input data and the internal (Octree) structure is not the only time consuming step. Downloading of aerial photographs is a major factor due to their size and their relatively low download speed (due to the nature of the WMS requests needed).

This step may be circumvented by using the live colouring features of the Plas.io viewer or a similar (experimental) feature in PoTree[62]. This will allow having many colouring schemas available at the same time, for example the infrared version of the aerial photograph (3.D) or those available through the Sentinel Playground[63].

In the current implementation scaling of the InSAR processing is not a problem, as pre-processed data is delivered. It has to be noted that InSAR processing is an intensive process and will likely have similar problems in scaling to national scale.

59 The tiles included formed an outer ring around the "Groot Delft" AHN3 selection. (Tiles: 30HZ1, 37EN1, 37FN1, 37DN2, 37BZ1, 30DZ1, 37EZ2, 30GN1, 37BZ2, 37GN1, 37EN2, 30GZ1, 30GN2, 37HN1, 30DN2, 30HN1, 30GZ2, 37EZ1, 30DZ2, 37FZ1, 37BN1, 37GN2, 37BN2)

60 "Taming the beast: free and open-source massive point cloud web visualisation", O. Martinez-Rubi et al., November 2015. (doi: 10.13140/RG.2.1.1731.4326/1)

61 "Performance benchmark or estimate", comment by C. Manning, https://github.com/connormanning/entwine/issues/39#issuecomment-303192453. Retrieved 2017-11-06.

62 "A proof of concept for projecting web maps on point clouds", http://potree.org/demo/experimental/potree_map_projections/examples/viewer_proj.html. Retrieved 2017-10-25.

63 "Sentinel Hub, Sentinel Playground", http://apps.sentinel-hub.com/sentinel-playground/. Retrieved 2017-10-25.

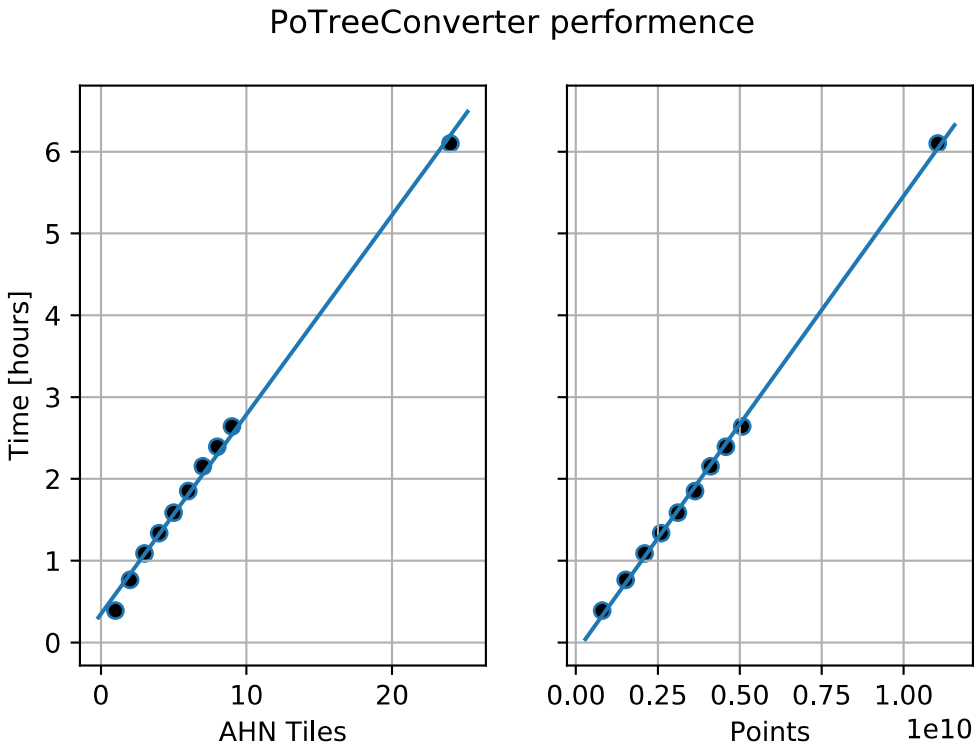


Figure 36: Benchmark results of PoTreeConverter under an increasing dataset size. A lineair trend is shown on top.

7. Conclusion

The advance of WebGL technology enabled the creation of powerfull 3D (point cloud) viewers in the browser. Paving the way for an easy to implement combination of radar and LiDAR point clouds. Although there are no *ready-made* solutions available, point cloud viewers can easily be extended to contain radar data. PoTree was used as the basis for a demo application. Undoubtedly this is a very powerful tool to visualise, analyse and understand the interaction of a radar signal with its surroundings.

To analyse the properties of the radar signal the error ellipsoid of the measurement is shown at the estimated position of the scatterer. Visualising the intersection of the estimation with the dense geometry of the point cloud. This allows for analysis of the (expected) dominant scatterer in the image.

To accomplish this goal it is necessary to have a high density (LiDAR) coverage of the object of interest. For most facades this can be accomplished by either TLS or MLS surverys, while airborne LiDAR will provide for coverage of groud, roofs and roof installations.

Possibilities are mostly limited by the ability of the browser to load (and process) large streams of data. Tiling and Octree structures will allow the client to process only the points or ellipsoids in view. Viewers come with solutions for the creation of the requierd Octree structure. Clustering of points with equal properties reduces the visual clutter and allows the operator to focus on important details elements of the data.

In this study the application was implemented over only a small area (Delft and surroundings). Experiments with PoTreeConverter show that this technique may be used at larger scale with reasonable preparation times. A solution like Massive-PoTreeConverter is defenitly not necessary at city scale projects.

Research questions

As the first three research questions provide a powerfull summary of the work at hand, the results per question will be briefly discussed.

1. What software and applications are already available?

   No ready-made solution for the combination of point clouds with radar information exists as of yet. There are two major point cloud tools available: PoTree and plas.io. Both come with processing software to convert data from various input formats to a structure readable by either the client or a server that transfers data to a client.

   For the preparation of (very) large point clouds the Netherlands eScience Center has developed Massive-PoTreeConverter, a parallel implementation of the traditional PoTreeConverter.

2. What information is required to position SAR data?

1. How are SAR signals and their uncertainties represented?

   SAR signals are presented by an error estimate (standard deviation) in range, cross-range and azimuth direction. Based on the (known) satellite viewing geometry and estimates of the scatterer position error ellipsoids may be placed in a 3D space.

2. Which coordinate systems and (file) formats are involved?

   Horizontal coordinates are provided in RD coordinates. Vertical coordinates are in an unknown height system.

   As file formats ASPRS LAS (point clouds), GeoTiff (aerial photographs) and ASCII (InSAR) were dominant. As output formats GeoJSON (InSAR) and a custom binary format (PoTree) were used.

3. Which combination of data is effective?

   A combination between point clouds from terrestrial (or mobile) laserscanners for facades and an airborne LiDAR survey for the roofs and bigger picture.

   1. How is effectiveness assessed?

      The combination of data is effective if the properties of the (likely) dominant scatterer can be found in the point cloud. This requires high density point clouds of the region of interest. The resolution of AHN may not be high enough in some areas, such as on facades.

   2. How does this combination help in finding the (dominant) scatterer?

      In its current implementation this combination will help a skilled operator estimate the dominant scatterer. Further implementations of this combination may automatically estimate the dominant scatterer based on likelihood as function of the error ellipsoid.

7.A. Recommendations

Geometric/Geological tools

By default PoTree is equiped with angle, distance, height, surface, volume and profile measurement tools. There is no tool for estimating the orientation of a plane (through points) relative to the coordinate axes. This would be a valuable extension for geologic use: estimating strike and dip of the geological features present. (Such application exist for the desktop environment, eg. LIME[64].)

Colourisation

PDAL may provide an alternative to lascolor, adding colours from a aerial photograph without loosing intensity or other information[65]. Another alternative would be implementing a technique to 'live' overlay images on the point cloud (already possible in plas.io).

64 “LIME: Visualisation and Interpretation Software”, http://virtualoutcrop.com/lime. Retrieved 2017-11-08.
65 “PDAL: filters.colorization”, https://www.pdal.io/stages/filters.colorization.html. Retrieved 2017-11-08.

Improvement of alignment

Alignment between datasets will be a problem when expended to (automatic) matching between the radar measurement and most likely scatterer (from the laser point cloud). Small alignment errors could falsly attribute a scatterer to a mis-aligned point.

Currently alignment between the (laser) point clouds is rather poor with the manually defined correspondences. Techniques like Iterative Closest Point (ICP) may improve this match. Initial attempts at doiing so were unsuccesfull.

Currently the TLS scanning procedure in use at the Department of Geoscience and Remote Sensing does not include georeferencing of the scan. Georeferencing while scanning would make manual alignment redundant.

Equal representation of laser point

Currently only radar scatterers are represented by their error ellipsoid. To create a level playing field the points aquired using a laser system should be represented by their error ellipsoid too. In case of AHN, for example, the vertical component will be much larger than the intersection of the ellipsoid in the same direction. Horizontally the accuracy of AHN is much (approximately 10×) better.

Meshesand intersection finding

Meshes could be used to represent the objects in the 3D space. These meshes could then be used to detect the intersection of the expected position of the radar scatterer with the surface of the object – determining the position of the (most likely) dominant scatterer.

Such an approach would use both the redundancy within the point clouds (many points describing the same object) as well as solving the problem of the irregular spacing between points (intersection with the surface may be closer than the closest point).

If it is only about the viewing experience, the radar data could be integrated into an existing solution providing (building) meshes (eg. Google Earth).

Deformation vectors

A deformation vector could be added to the ellipsoid. This will show that deformation is in the range direction and not in the vertical direction.

Appendix A “Quick recipe”

Presented here is a abbreviated version of the manual presented in chapter 4. Focus of this manual is to create a simple PoTree web-interface from a point cloud and add radar measurements to it. This is not the full flowchart (Figure 18) as discussed in chapter 4. In this recipe a simple case of a single point cloud and a small (less than 1500 points) radar file is discussed.

Prerequisites

- A point cloud, AHN3 for example[25]. Please note that AHN1 and AHN2 do not contain intensity information!
- PoTreeConverter[4], available binaries (64-bit) for Windows and as sourcecode for Linux.
- Radar data.

Point cloud

Using PoTreeConverter the input LAS-file is converted into an Octree of binary files. For AHN3 tile `37EN1` the command will be as follows:

```
PotreeConverter C_37EN1.LAZ \
-o ./web --material INTENSITY -p AHN3 --show-skybox \
--projection "+proj=sterea +lat_0=52.15616055555555
+lon_0=5.38763888888889 +k=0.9999079 +x_0=155000 +y_0=463000
+ellps=bessel +units=m +no_defs" \
--edl-enabled --intensity-range 3 256 -r 256 \
-a CLASSIFICATION INTENSITY
```

Some explanation on the parameters:

- `C_37EN1.LAZ`, the input file or files.
- `-o ./web`, the output directory.
- `--material INTENSITY`, the attribute on display. All attributes mentioned under -a are stored, and may be selected for display in the viewer. Supported are: `RGB`, `ELEVATION`, `INTENSITY`, `INTENSITY_GRADIENT`, `RETURN_NUMBER`, `SOURCE` and `LEVEL_OF_DETAIL`. If the material is `RGB`, `INTENSITY` or `CLASSIFICATION` it should be mentioned under `-a` too!
- `-p AHN3`, the filename in the output directory (`-o`).
- `--show-skybox`, show some clouds as background (optional).
- `--projection "..."`, projection (EPSG:28992), ignored by PoTreeConvert but used by PoTree for addding a map.
- `--edl-enabled`, enable “Eye-Dome-Lighting” for a more natural looking point cloud.

- `--intensity-range 3 256 -r 256`, although a full intensity range up to 65536 is available, with AHN3 most points have an intensity value between 3 and 256. Without those limits the point cloud will be all black.
- `-a CLASSIFICATION INTENSITY`, attributes to include in the output. Supported are: `RGB`, `INTENSITY` and `CLASSIFICATION`.

After running PoTreeConverter there should be a file `web/AHN3.html`. When opened in a browser (even locally) this should give a working PoTree installation.

Radar ellipsoids

Next step is adding the radar (error) ellipsoids. Independent of the source of the data JSON is the prefered format to feed the data to the viewer. In this recipe an extra simplified format is used. It may be extended at will.

As of Matlab 2016b `jsonencode()` is available[66]. Other languages will have similar functions. The input does not matter as long as the output is of the same structure.

In this example the structure is a list of series of *x*, *y*, *z* values (as shown below) in the same coordinate system as the point cloud. Whitespace is unimportant. The file should be saved as `web/radar.json`.

```
[{"x":85443.4,"y":446144.6,"z":-1.714},
{"x":85452.0,"y":446150.5,"z":-1.346},
{"x":85450.1,"y":446138.7,"z":-0.73},
{"x":85437.0,"y":446151.9,"z":-0.7},
{"x":85440.7,"y":446157.1,"z":-0.158},
{"x":85434.1,"y":446142.3,"z":-0.569},
{"x":85453.7,"y":446133.0,"z":-1.341},
{"x":85458.1,"y":446135.6,"z":-1.543},
{"x":85433.7,"y":446159.2,"z":-1.077},
{"x":85435.6,"y":446161.7,"z":-0.8}]
```

To add the ellipsoids, add the following lines to `web/AHN3.html`, before `</script>` at around line 72. The current parameters (scaling, rotation) are based on the TerraSAR-X estimates (3.E). See the comments for some guidance.

```
// Read the file
$.getJSON('./radar.json', function(data) {

    // Create a spehere with radius 1
    let sph = new THREE.SphereGeometry(1, 12, 10);

    // Create a material

    let sphm = new THREE.MeshNormalMaterial();
```

66 “jsonencode”, MathWorks/MATLAB documentation, https://nl.mathworks.com/help/matlab/ref/jsonencode.html. Retrieved 2017-10-26.

```
    // Create a rotation matrix
    var eul = new THREE.Euler(0, -1.150172, -1.605703);

    // Loop over all elements
    $.each(data, function(key, val) {

        // Create an element out of the sphere and the material
        let s = new THREE.Mesh(sph, sphm);

        // Scale the ellipsoid
        s.scale.set(0.075, 0.025, 5.00);

        // Rotate the ellipsoid
        s.rotation.copy(eul);

        // Position the ellipsoid
        s.position.set(val.x, val.y, val.z);

        // Calculate the transformation matrix …
        s.updateMatrix();
        s.matrixAutoUpdate = false; // ... only once, to improve
performence

        // Add to the viewer (scene)
        viewer.scene.scene.add(s);

    });
});
```

You should now see ellipsoids in your 3D world!

Appendix B TLS Alignment

In this appendix the alignment between the local TLS coordinate system and the national coordinate system (RD-coordinates, matched on the basis of AHN) is given. Per point cloud the groups of matching points and their coordinates are given, together with the Root Mean Square Error (RMSE) of the resulting fit (based on rotation and translation only).

Mekelpark Tram

RMSE: 0.36 m

AHN			TLS			
N	E	U	N	E	U	Location
85512.500	445878.035	16.263	21.743	36.721	15.248	EWI
85558.945	445966.059	9.439	116.369	7.045	7.191	CiTG
85734.539	445939.430	11.717	120.813	-170.736	10.682	Bld. 36
85533.266	445878.286	-0.631	25.826	16.872	-1.599	Buslane

Mekelpark DTM

RMSE: 0.27 m

AHN			TLS			
N	E	U	N	E	U	Location
85558.945	445966.059	9.439	-5.443	36.236	9.647	CiTG
85545.148	445999.124	9.441	-31.408	11.659	9.721	CiTG
85497.743	446108.008	9.346	-116.300	-71.393	9.365	CiTG
85536.222	445955.041	-0.413	12.680	18.749	-0.201	Buslane
85403.344	446053.721	81.187	-32.728	-140.890	81.274	EWI

Mekelpark Trees

RMSE: 0.24 m

AHN			TLS			
N	E	U	N	E	U	Location
85558.945	445966.059	9.439	43.726	-48.558	8.651	CiTG
85530.845	446028.172	9.338	8.007	9.426	8.587	CiTG
85446.184	446029.268	18.844	-75.813	-0.184	17.972	EWI
85469.303	445972.804	18.85	-45.508	-53.111	18.002	EWI

Mekelpark Gras

RMSE: 0.34 m

AHN			TLS			
N	E	U	N	E	U	Location
85446.184	446029.268	18.844	-51.588	-32.333	18.308	EWI
85469.303	445972.804	18.85	-50.884	-93.162	18.262	EWI
85497.743	446108.008	9.346	25.221	22.014	8.937	CiTG
85403.344	446053.721	81.187	-82.300	7.147	80.516	EWI

Appendix C Scripts

Some of the Python (3) scripts used during the creation of demo application are listed here. Their programming style should not be an example and they should be considered a more elaborate version of pseudo-code.

Alignment of two point clouds

In this example implementation the coordinates are implemented as a hard coded matrix. The first three columns contain the AHN coordinates (EPSG:7415), the following three columns contain the coordinates in the local coordinate system of the TLS. The corresponding points will have to be found manually first, for example in CloudCompare.

The script will output a CloudCompare command that will project the input point cloud with the transformation found, apply the intensity scaling necessary for PTX files and store the results as LAZ-files.

```
import pandas as _pd;
_np = _pd.np;
from shlex import quote as _quote;
from os.path import join as _path_join;

ROOT = '[directory with TLS point cloud data]';

# AHN_X, AHN_Y, AHN_Z, S_X, S_Y, S_Z
Coords = {
   'Mekelpark Tram/Tram_HighestRes_Colour.ptx': _np.matrix(
        [[85512.500, 445878.035, 16.263,  21.743, 36.721,
15.248], # EWI Laagbouw, south-east
        [etc.]
   ]),
   [repeat for other point clouds]
};

for key, coord in Coords.items():

   # Extract coordinates
   coord_AHN = coord[:, :3];
   coord_TLS = coord[:, -3:];

   # Calculate the transformation based on SVD (Besl method)[43]
   centroid_AHN = coord_AHN.mean(axis=0);
   centroid_TLS = coord_TLS.mean(axis=0);

   # Calculate rotation
   H = (coord_TLS -centroid_TLS).T *(coord_AHN -centroid_AHN);

   U, S, Vt = _np.linalg.svd(H);
```

```
   R = Vt.T *U.T;

   if _np.linalg.det(R) < 0:
        R[:, 2] *= -1;
        R = Vt.T *U.T;

   # Calculate translation
   t = -R *centroid_TLS.T +centroid_AHN.T;

   # Add extra row (CloudCompare)
   TransSVD = _np.vstack([_np.column_stack([R, t]),
                   [0, 0, 0, 1]]);

   # Calculate residuals
   ResSVD = _np.dot(_np.column_stack([coord_TLS,
_np.ones((coord_TLS.shape[0], 1))]), TransSVD[:3, :].T)
-coord_AHN;

   # Calcualte the RMSE
   RMSESVD = _np.sqrt(_np.multiply(ResSVD,
ResSVD).sum()/coord_TLS.shape[0]);

   # Save the transformation matrix (SVD) for CloudCompare
   with open(_path_join(ROOT, key) + '.trans', 'wb') as
trans_out:
        _np.savetxt(trans_out, TransSVD);

   # Create the CloudCompare command
   Command = ['CloudCompare',
              '-COMPUTE_NORMALS'];
   Command += ['-O {!s}'.format(_quote(_path_join(ROOT, key)))];
   Command += ['-AUTO_SAVE OFF',
                '-SF_OP 0 mult 65535', # For LAS compatibility
                '-APPLY_TRANS {!
s}'.format(_quote(_path_join(ROOT, key) + '.trans')), #
Transformatie matrix
                '-C_EXPORT_FMT BIN', # LAS
                '-SAVE_CLOUDS ALL_AT_ONCE'];

   # Print command
   print(key, 'RMSE', RMSESVD);
   print(' '.join(Command));
```

Clustering

This script serves as an example on how to apply K-Means clustering in Python3 using the scikit-learn package[67], The output will be a single JSON file, including per cluster colours related to the deformation velocity.

```
import geopandas as _gpd;
_pd = _gpd.pd;
_np = _pd.np;

from shapely.geometry import Point as _point;
from sklearn.cluster import MiniBatchKMeans as _k_means;

import matplotlib as _mpl;
from matplotlib import pyplot as _plt;
from matplotlib import colors as _plt_colors;

# Scaling of the trend
trend_scale = 5000;

# Number of clusters
n_clust = 1500;

Points = _pd.read_csv('[input file]');

k_mtx = Points[['pnt_rdx', 'pnt_rdy', 'pnt_height',
'pnt_linear']].as_matrix();
k_mtx[:, 3] *= trend_scale;

k_means = _k_means(n_clust);

Pnts_gpd = _gpd.GeoDataFrame({'trend':
k_means.cluster_centers_[:, 3]/trend_scale, 'count':
k_means.counts_},
   geometry=list(map(lambda r: _point(*r),
k_means.cluster_centers_[:, :3])),
   crs={'init': 'epsg:28992'});

# Calculate cluster scale/size
Pnts_gpd['scale'] = 3+22*((Pnts_gpd['count']
-Pnts_gpd['count'].min())/Pnts_gpd['count'].ptp());

# Generate a color scale
c_min = Points.pnt_linear.quantile(0.05);
c_max = Points.pnt_linear.quantile(0.95);
c_lim = max(abs(c_min), abs(c_max));
```

67 “scikit-learn: machine learning in Python”, http://scikit-learn.org/stable/. Retrieved 2017-11-08.

```
cmap = _plt.cm.RdBu_r;
norm = _plt_colors.Normalize(vmin=-c_lim,
                             vmax=+c_lim);
sm = _plt.cm.ScalarMappable(cmap=cmap, norm=norm);

Pnts_gpd['color'] = ['#{:02X}{:02X}{:02X}'.format(r[0], r[1],
r[2]).lower() for r in sm.to_rgba(Pnts_gpd.trend, bytes=True)];

with open('[output1.json]', 'wt') as json_out:
   json_out.write(Pnts_gpd.to_json());

# Create an image of the colorbar
fig = _plt.figure(figsize=(2, 4)); # Horizontal: (4, 1)

# Create an axis for the colorbar, and the colorbar itself.
ax = fig.add_axes([0.05, 0.05, 0.3, 0.90]);
cb = _mpl.colorbar.ColorbarBase(ax, cmap=cmap, norm=norm,
extend='both');

# Add a white stroke around the letters, for better readability.
[l.set_path_effects([_plt_pe.withStroke(linewidth=3,
foreground="w")]) for l in cb.ax.yaxis.get_ticklabels()];
cb.ax.tick_params(labelsize=14);

# Add a label (again with white stroke).
#cb.set_label(r'Deformation [$\frac{m}{yr}$]');
cb.set_label(r'Deformation [m/yr]', size=14, style='oblique',
path_effects=[_plt_pe.withStroke(linewidth=3, foreground="w")]);

# Save everything.
_plt.savefig('[output.svg]', transparent=True);
_plt.close();
```