NTNU
Det skapende universitet

LIDAR Datavarehus

Kjartan Bjørset

Master i ingeniørvitenskap og IKT
Innlevert: juni 2013
Hovedveileder: Terje Midtbø, BAT

Norges teknisk-naturvitenskapelige universitet
Institutt for bygg, anlegg og transport

NORGES TEKNISK-
NATURVITENSKAPELIGE UNIVERSITET
INSTITUTT FOR BYGG, ANLEGG OG TRANSPORT

Oppgavens tittel: LiDAR Data Warehouse	Dato: 10.06.2013 Antall sider (inkl. bilag): 121			
	Masteroppgave	x	Prosjektoppgave	
Navn: Kjartan Bjørset				
Faglærer/veileder: Terje Midtbø				
Eventuelle eksterne faglige kontakter/veiledere: Alexander Nossum				

Ekstrakt:

Denne masteroppgaven har hatt som mål å finne ut hvordan et datavarehus bør designes for å kunne sikre effektiv handtering av data produsert av laserscannere (LiDAR – Light Detection and Ranging). LiDAR-scannere er fjermålingssytem som er i stand til å generere nøyaktige og tette punktskyer av omgivelsene. Desse punktskyene kan brukes innenfor en rekke ulike ingeniørdisipliner, men er ofte store og er vanskelige å handtere med dagens programvareløsninger. Fordi LiDAR-scanning er dyrt og fordi punktskyene kan brukes om igjen flere ganger av ulike aktører, kan det ligge langsiktige gevinster i å organisere dataene på en gjennomtenkt måte. Denne masteroppgaven gir en grundig gjennomgang av LiDAR-data og presenterer en rekke programvareteknologier som er nyttige for å bygge en prototype LiDAR-datavarehus. Ny kunnskap om hvordan et datavarehus bør designed ved å utvikle og teste en prototype bestående av en PostGIS database, en Apache Server og en Web-applikasjon. Positive egenskaper ved PostGIS-databasen er at den støtter SQL-spørringer og har en organisert lagringsstruktur, men lagringseffektiviteten er betraktlig lavere enn for LAS og LAZ-filer. Masteroppgaven konkluderer med at LAZ-filer er det beste alternativet med tanke på lagringrseffektivitet. Masteroppgaven ser også nærmere på visualiseringsteknikker for LiDAR-data og konkluderer med at ved å introdusere en Web-basert visualiseringsløsning vil datatilgangen bli enklere og mer oversiktlig.

Stikkord:

1. LiDAR
2. Data Warehouse
3. WebGL
4. Geomatics

Kjartan Bjørset 10.06.2013

Oppgavetittel:

LIDAR data warehouse

Type oppgave:

Masteroppgave

Bakgrunn

- LIDAR data høstes i store mengder av mange ulike objekter – deriblant Nidarosdomen
- Data som fanges er punktskyer, noen med egenskaper utover koordinatfestet punkt
- Dette fører til en stor mengde data med varierende kvaliteter. Det finnes ikke noe system for å samle alle disse data i en felles database og hvor man enkelt kan navigere i en samlet datamengde og laste ned på tvers av datasettene.

Oppgave (Stikkordsform)

- Implementere en proof-of-concept av et datavarehus som kan lagre punktskyer i et felles referansesystem uavhengig av datafangstmetode eller tilhørende attributter.
  - Definere et dataskjema og et grensesnitt for støttet data
- Undersøke skaleringsmuligheter på databasen
  - Velge software, algoritmer/teknikker
  - Designe databasen
- Gjennomføre tekniske eksperimenter på forskjellige aspekter med datavarehuset med reelle datasett
- Implementere et grensesnitt for effektiv uthenting av data
- Implementere en proof-of-concept applikasjon som kan navigere i datamengden lagret i datavarehuset. Fortrinnsvis ved bruk av standard webteknologier.

Veiledere

Eksterne: Nidarosdomen

NTNU: Terje Midtbø / Alexander S. Nossum / Trond Arve Haakonsen

Norwegian University of Science and Technology

Master Thesis

LiDAR Data Warehouse

Author:
Kjartan Bjørset

Supervisor:
Dr. Terje Midtbø

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

Research Group: Road, Transport and Geomatics
Department of Civil and Transport Engineering

June 2013

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Abstract

Faculty of Engineering Science and Technology
Department of Civil and Transport Engineering

Master of Science

LiDAR Data Warehouse

by Kjartan Bjørset

Light Detection and Ranging (LiDAR) scanners generate dense and highly accurate three-dimensional point clouds of their surroundings. These point clouds can be used in a wide range of engineering applications, but are generally large and hard to manage using existing software solutions. Because the point clouds can be used in many different applications, an efficient organization of the data may have positive effects on data utilization and help justify the high acquisition costs. This thesis presents a thorough description of the LiDAR data structure and introduces relevant software solutions for developing a prototype LiDAR data warehouse. By building and testing the prototype, which consists of a PostGIS database, Apache server and a Web application, new knowledge was obtained about how a LiDAR data warehouse should be designed. While the PostGIS database provides a powerful query language and organized management of the data, it has a lower storage efficiency as compared to LAS and LAZ file management solutions. The results show that the LAZ file is the best choice for maximized storage efficiency. The thesis also investigates visualization techniques for LiDAR data and suggests that a web-based point viewer, which has been successfully implemented, could increase the accessibility of point cloud data.

Preface

This thesis summarizes a semester of research and software development on the topic of managing data produced by Light Detection and Ranging (LiDAR) scanners. LiDAR scanners have over the last couple of decades become an efficient and accurate way of mapping the world around us. Through my studies I have had the opportunity of trying a few LiDAR scanners and have been impressed with the accurate 3D point clouds they create. Therefore, when the opportunity of writing a Master thesis about "LiDAR data warehouse" presented itself, the decision process was short. The thesis has resulted in a prototype of a data warehouse that can load, store and extract point clouds, in addition having a Web-based application for 3D visualization. The data that is show-cased in this document, and has been the basis of development and testing of the system, has been lent to me by two companies. I owe Blom Geomatics and Nidaros Domkirke Restaureringsarbeider huge thanks for their cooperation.

Finally, I would like to thank all the people who in some way have helped me get through this thesis. I would like to thank my primary supervisor, Terje Midtbø, for giving me the opportunity to work on this topic. I would also like to thank my co-supervisor Alexander Nossum, who has been an inspiration and of great help throughout the semester. I would also like to thank my second co-supervisor, Trond Arve Haakonsen, who has supported me in my work and helped me acquire data. My fellow students deserves a thank you for the feedback and tips they have given me during the semester. Lastly, but not least, I would like to thank my dear Helen for her feedback and support throughout the thesis.

Contents

List of Figures

List of Tables

Abbreviations

ALH	**A**ir **L**iDAR **H**ydrography
ALS	**A**erial **L**iDAR **S**canning
BIM	**B**uilding **I**nformation **M**odel
BLOB	**B**inary **L**arge **OB**bject
CAD	**C**omputer **A**ided **D**esign
CIM	**C**ity **I**nformation **M**odel
CML	**C**ontinuous **M**apping and **L**ocalization
DEM	**D**igital **E**levation **M**odel
DSM	**D**igital **S**urface **M**odel
DTM	**D**igital **T**errain **M**odel
GNSS	**G**lobal **N**avigational **S**atellite **S**ystem
GPS	**G**lobal **P**ositioning **S**ystem
HDD	**H**ard **D**isk **D**rive
INS	**I**nertial **N**avigational **S**ystem
LiDAR	**Li**ght **D**etection and **R**anging
NCALM	**N**ational **C**enter for **A**irborne **L**aser **M**apping
NIR	**N**ear **I**nfra **R**ed
NSF	**N**ational **S**cience **F**oundation
RADAR	**Ra**dio **D**etection and **R**anging
SDSC	**S**an **D**iego **S**upercomputer **C**entre
SLAM	**S**imultaneous **L**ocalization **a**nd **M**apping
SLR	**S**atellite **L**aser **R**anging
SONAR	**So**und and **N**avigation **R**anging
SSD	**S**olid **S**tate **D**evice

TIN	**T**riangular **I**rregular **N**etwork
USGS	**U**nited **S**tates **G**eological **S**ervice
WKB	**W**ell **K**nown **B**inary
WKT	**W**ell **K**nown **T**ext
WMS	**W**eb **M**ap **S**ervice

Part I

Introduction

Chapter 1

Introduction

In recent years, the advancement of 3D-mapping tools have made high-resolution, accurate "point clouds" available to the engineering industry. In their simplest form, point clouds contain three-dimensional coordinates but, depending on the acquisition technique, additional information, such as colour, intensity or classification values, may be tagged to each point. There are several techniques available for capturing point clouds: SONAR, RADAR, LiDAR and photogrammetry, which are all capable of producing vast amounts of point data. Out of these surveying techniques, LiDAR, an acronym for Light and Detection Ranging, has been deemed the most versatile and accurate. LiDAR systems use laser light to capture millions of points per seconds - resulting in data called "point clouds."

LiDAR data can be used in a wide range of applications, including traditional surveying and mapping, 3D visualization, vegetation mass estimation and archaeology. In recent years, LiDAR has even been used for wind speed measurements and sensory units in autonomous cars. For many of these applications, the point clouds are not the end product, but rather an intermediary step on the way to the 3D representation or derived information that is needed for a specific purpose. Because point clouds usually contain millions or even billions of points, after they have served their purpose, it can be time-consuming and resource demanding to store these in an organized manner. This often leads to data sets ending up in files, on external hard drives or other offline storage solutions.

Taking into account the vast size of point clouds, and that the user wants a simple and easy method to store them, the quick and dirty solution of simply saving the data in files is understandable. However, there might be benefits from having the LiDAR data stored in a more organized manner. Point clouds are versatile and can be re-used many times for different purposes. This becomes more interesting when the acquisition cost is taken into account. Because LiDAR scanning requires expensive equipment, trained personnel and often includes vehicle or aircraft operation, the data sets come with a hefty price tag. By making point clouds more available, it becomes easier for users to harness the power of LiDAR data. This realization has been made by some, such as the governments of Denmark and Finland, who have made all of their raw data available on-line. Another example is the organization OpenTopography, which is being funded by the US government and state institutions to do the same.

However, even when the will to organize LiDAR data is there, the path to selecting a system that can manage the data is complicated. Software solutions for working with and storing LiDAR data are still in their infancy. Because many of the current spatial databases and Geographical Information Systems have poor support for LiDAR data, it is not straight forward to use these as data management solutions. This master thesis aims to investigate how LiDAR data can be stored in an efficient, organized and user-friendly manner. This was done by, firstly, developing a thorough understanding of the challenges related to the structure, acquisition, storage and visualization of LiDAR data. A LiDAR data warehouse prototype was developed, followed by an assessment of what the implementation of such a system would entail.

The report is divided into four parts: Part 1 presents the problem of LiDAR data management. Part 2 presents background knowledge needed to build a LiDAR data warehouse: LiDAR as a remote sensing technique, storage and access of spatial information, and finally, recent advances in Web technology. In part 3, state-of-the-art solutions are explained together with the prototype developed during this master thesis. In part 4, the discussion and conclusions are presented.

Part II

Background

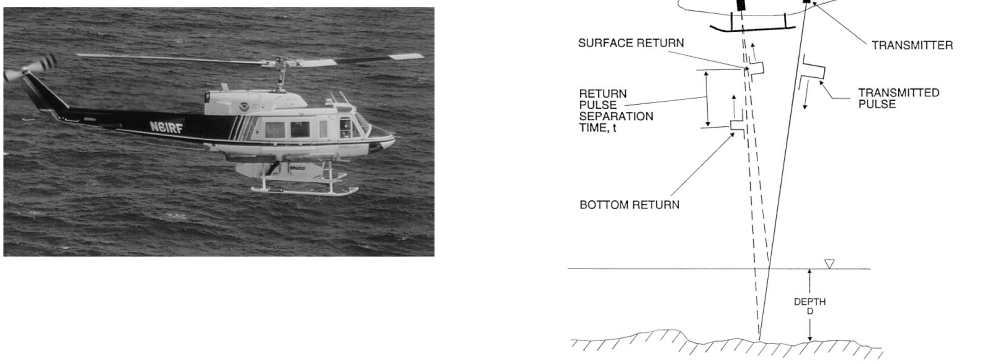
Chapter 2

LiDAR

2.1 Light and Detection Ranging

LiDAR, an acronym for Light and Detection Ranging, is a remote sensing technique based on the principle of measuring the distance to an object by illuminating it and analysing the backscatter. It bears resemblance to Sound and Navigation Ranging (SONAR) and Radio Detection and Ranging (RADAR), but uses light instead of sound or radio waves to measure distances. Since the 1860s, when the speed of light was first accurately measured, LiDAR has been a theoretical possibility, but it was not until the invention of the laser in 1958 that it became an viable solution[1]. Lasers concentrate light beams, which makes it easier to detect and isolate light reflected by objects. One of the first successful uses of LiDAR was measuring the distance to the moon, which was done in 1962 with a few decimetre accuracy[1, 2]. Another NASA-related application of LiDAR is Satellite Laser Ranging (SLR), a crucial part of maintaining accurate satellite altimeter information. Many of the early LiDAR applications were based on measuring single distances between two objects, but it was also realized that by emitting light rays in several directions, LiDAR could be used as a mapping tool. This was first discovered by the US Navy, who came up with the idea of using LiDAR for submarine detection in the 1960s [1, 3]. As the military projects were classified information,

the first research papers on this topic surfaced in the 1970s and operational prototypes were developed in the 1980s [4]. One of the first successful attempts at large-scale Aerial LiDAR Hydrography (ALH) was a joint American-Canadian project called SHOALS, which was completed in 1994.



(a) SHOALS helicopter

(b) SHOALS system overview

FIGURE 2.1: SHOALS Aerial Laser Hydrography System[5]

Over a period of three years the SHOALS system mapped over 2000 square kilometres of coastal area in the two countries[5]. In addition to the LiDAR scanner, SHOALS carried an Inertial Navigational System (INS) and utilized the Global Positioning System (GPS). The GPS and INS helped reduce the number of ground control points needed to georeference the data, and today this is the standard set up for all Aerial LiDAR Scanners (ALS). In 1995, the first commercial aerial LiDAR scans were carried out and in 2001 there were 75 companies operating 60 scanners across the globe[1]. Since then, LiDAR technology has improved in terms of accuracy and information quality and is being used in an increasing number of applications. With the current technology, a single scan can exceed hundreds of gigabytes of data, thus, challenging the existing hardware, software and IT infrastructure solutions.

2.1.1 Technical Description

Current LiDAR scanners are based on one out of two observation techniques for measuring the distance between the scanner and an object: Time-of-flight or phase-shift observation[6]. For time-of-flight observation, the scanner observes the elapsed time for a transmitted laser pulse to return to the scanner. Then the distance is computed by multiplying time with the speed of light, as shown in equation 2.1.

$$d = (t_1 - t_0) * c \tag{2.1}$$

$$t_{flight} = \frac{\phi}{2\Pi * f_{modulation}} \tag{2.2}$$

The second option is to have transmit a laser beam with sinusoidally modulated optical power, which makes it is possible to observe the phase change of the returned signal. From the phase change, the time of flight can then be computed by using equation 2.2. The result is then substituted into equation 2.1. Scanners that use this technique, typically have a far higher data acquisition rate compared to time-of-flight-based scanners, but their range is also significantly shorter: The modulation signal is periodic and introduces ambiguity in the range measurements for longer distances. As an example, Leica's phase scanner, HDS6200 has a maximum range 79 meters, whereas the time-of-flight scanner, ALS60, has maximum flying height of 6 km[6].

2.1.1.1 Laser Light

The laser light used for LiDAR is usually in the lower region of the near-infra-red (NIR) spectrum. Wavelengths between 800 nm to 1550 nm are commonly seen in the industry[6–8]. Using these wavelength results in vegetation reflecting a large portion of the light, whereas other materials such as water and asphalt will absorb

it, all of which can be used for classification purposes. For ALH systems such as SHOALS, a second laser light is used to penetrate the water: Green light with a wavelength of 532 nm is emitted in addition to the NIR light[5, 6]. This means the scanner will observe two different returns over an extended time interval, as illustrated by the idealized return waveform in figure 2.2. This is an important idea in LiDAR scanning which is increasingly being exploited to analyse the data. This concept will be presented in full in section 2.2.1.

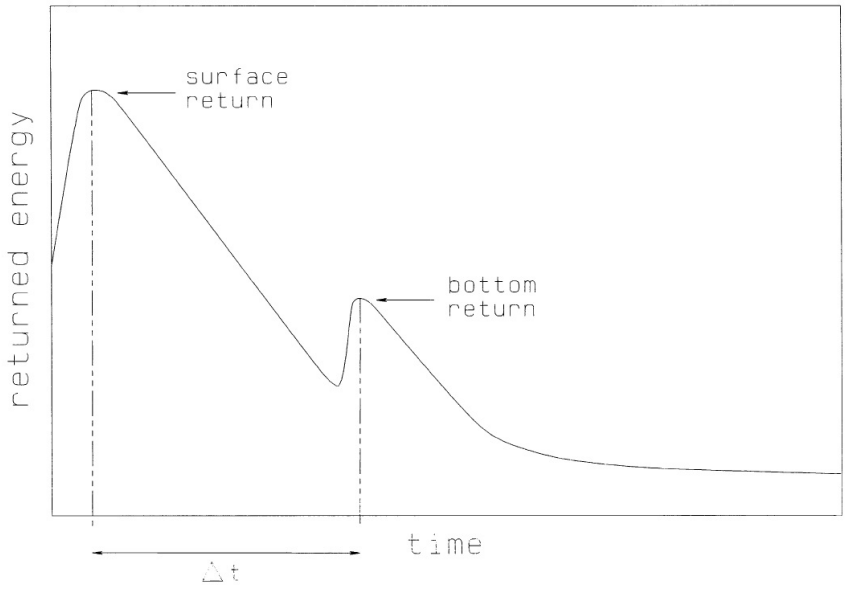


FIGURE 2.2: Laser waveform return[5].

The size of light ray that reaches the illuminated object will vary in size depending on the light source and the scan distance. For ALS this size is usually referred to as footprint: Big footprint laser scanning refers to ground diameter between 10-70 meters, whereas small footprint scanning refers to a ground diameter in the sub-metre area[6, 7]. Typically, the ground-based LiDAR scanning implies small footprint scanners, whereas ALS scanners tend to have a larger footprint. Footprint size is naturally determined by the flight height, which in turn will have implications for the flight speed that can be used to survey an area, as illustrated in figure 2.3.

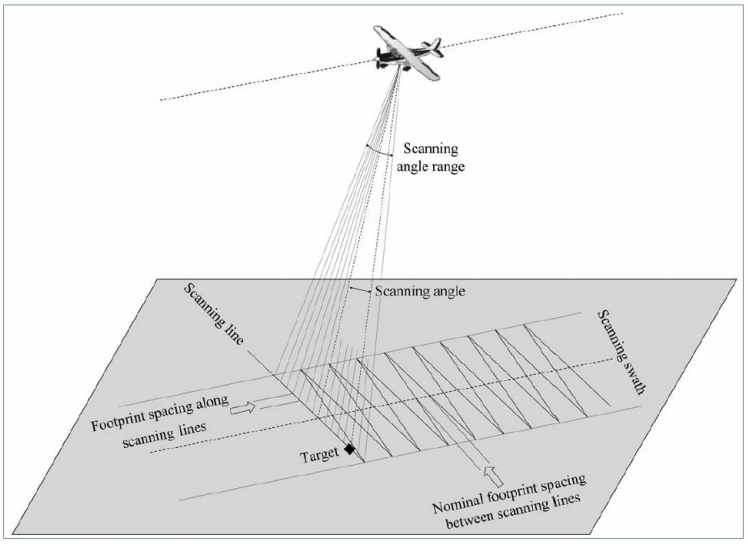


FIGURE 2.3: LiDAR flight overview[7]

Another part of the LiDAR scanner that affects the structure of the gathered information is the mirror motion inside the scanner. There are many different ways of doing this, in figure 2.4 four of them are shown. The see-saw pattern is the most common solution for ALS[7].

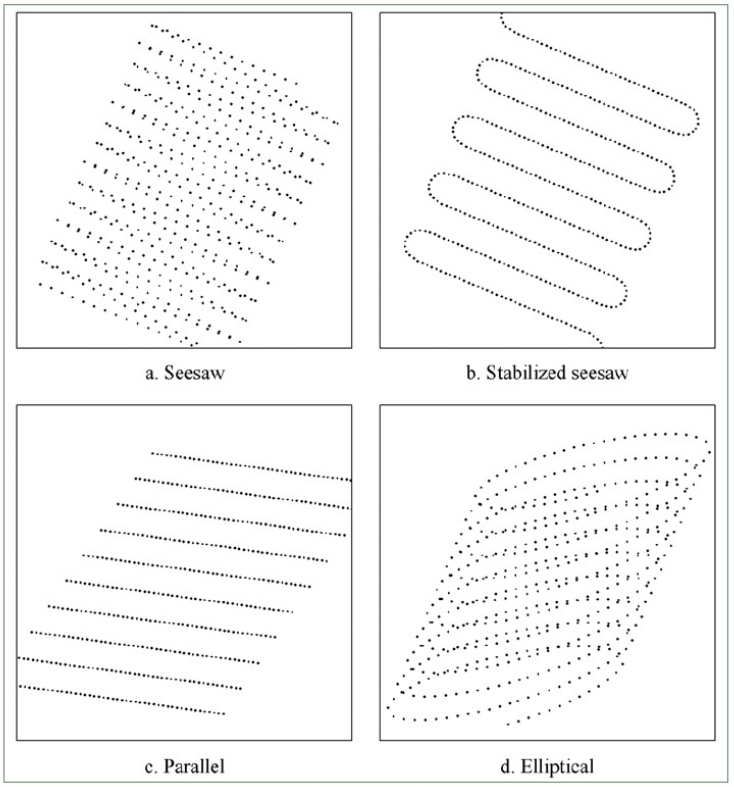


FIGURE 2.4: LiDAR mirror scan motion[7]

2.1.1.2 Analysing the Returned Light

Regardless of which observation method, scan pattern or footprint size is being use, the returning light has to be captured by the optical unit in the scanner. As shown in figure 2.2, the intensity of the returned light has to be analysed in order the identify the position of the illuminated object. While 2.2 shows how this problem can look like for hydrographic LiDAR scans, the principle can also be applied to forested areas: Parts of the light may be returned by leaves or conifer needles, while others will be reflected by the ground. An illustration of this can be seen in figure 2.5, where several returns are returned per light ray.

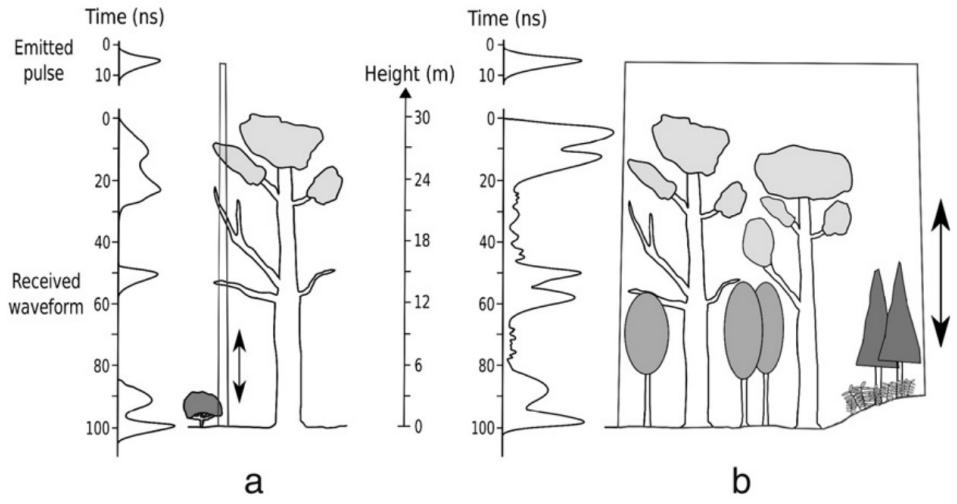


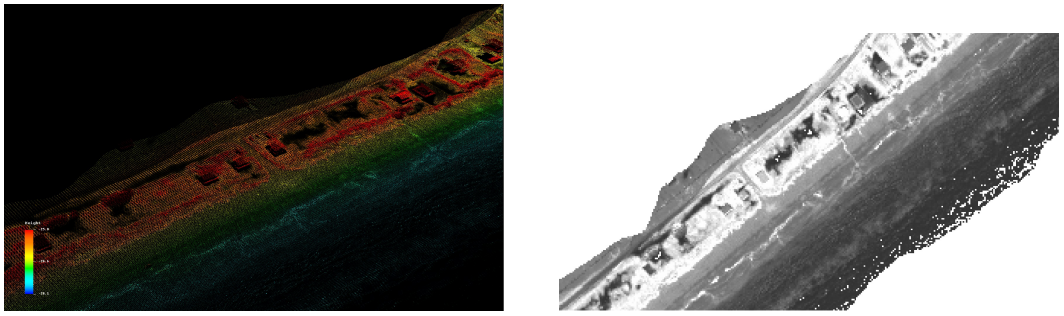
FIGURE 2.5: LiDAR return waveform from trees[6].

This figure also shows how different footprint sizes will affect the returned waveform. Traditionally, LiDAR scanners have collected this information by identifying the significant changes in the return waveform and exported this directly as discrete returns[6]. Typically, two to four returns have been the maximum amount, and there has been no way of seeing the raw waveform data. In 2004, this changed when new scanners capable of storing the entire waveform, were introduced to the market. These scanners, often referred to as "full-waveform LiDAR scanners", were a revolution with regards to data quality. There is however a catch to this technological advance: As reported in a study from 2009, storing full waveform data would lead to data sets five times bigger than the discrete return data sets at the time[6]. Seeing as the digitizing interval could, and probably will due to technological advances, get shorter in the future, these data sets may grow even larger.

2.2 Applications

One of the earliest applications of LiDAR scanning was the creation of Digital Surface Models (DSMs) and raster images. This can be done by running the

point clouds through algorithms that produce 2D or 3D representations. A simple approach would be to snap each point to a regular cell structure and use this as a raster image with elevation or return intensity tagged to each pixel. This is demonstrated in figure 2.6, where the left image shows the raw point cloud coloured by elevation and the right image shows a raster image derived from it.



(a) LiDAR visualization (b) Raster image

FIGURE 2.6: LiDAR data set from Texas, USA

A more sophisticated approach would be to build a Triangular Irregular Network (TIN). If this is done for a LiDAR data set that is unclassified, we get a Digital Surface Model. That is, a surface that follows the highest points in the terrain, including trees, buildings and other structures. As LiDAR data has become more detailed, it has become customary to classify points on whether or not they are ground reflections. Running a triangulation algorithm on these point clouds will result in a "bare-earth" TIN, also referred to as "terrain models" or Digital Elevation Models (DEM). These have a wide range of applications: 3D modelling, geological surveys and identifying archaeological dig sites [9]. The non-ground classified are obviously points off the ground and could be vegetation, buildings and power lines to mention a few. Especially vegetation ha been a prioritized research areas, and lately the full-wavelength data is being used to improve classification algorithms[8].

FIGURE 2.7: A bare-earth DTM based on classified LiDAR data. Created with LasTools and visualized in ArcScene 10.1.

LiDAR is increasingly being used in the civil engineering sector as well: During the planning phase of construction projects, stakeholders are often interested in seeing what the end product will look like. By using 3D models created from LiDAR scans combined with CAD drawings, it becomes easy to visualize a project. In fact, generating entire City Information Models (CIM) has also become a market in itself. Because of the high density and accuracy of phase-based LiDAR scanners, scan data are also being used for as-built-surveys, where the scan has to match the construction plan specifications. A typical use for as-built surveys is scanning tunnels under construction to minimize the needed concrete lining and ensure that height clearance is within acceptable limits [10].

LiDAR has also been shown to be able to detect clouds, water vapour, aerosols and even be used as a measure of wind speed [11]. Finally there is the growing industry of robotics. While current robots basically are production line workers, new and interesting prototypes are being developed that in the future can replace humans for more advanced tasks as well. Since 2010 Google has been developing the Google Self-Driving Car, which incorporates GNSS navigation, INS and LiDAR in order to map the world around it. In the field of robotics this is called Simultaneous Localization and Mapping (SLAM), or Concurrent Mapping and

Localization [12]. These systems are in principle, survey vehicles with an artificial intelligence. Although they are not currently being used for surveying or mapping purposes, the possibility is evidently there. Autonomous drones are also being developed, which is a promising technique for indoor mapping[13].

2.2.1 LiDAR Data

The design of a LiDAR scanner means that each processed pulse at the very least holds information about the distance to the surface that reflected it, a scan angle and an intensity value. Whether or not the full waveform is captured or simply processed and converted into discrete returns, depends on the system design. RGB image information can also be added to the scan data, but that is usually done after the point cloud is processed or 3D models have been created. The customer's order and the application the data is to be used for, dictates the form the point cloud data will be delivered on. A classification scheme for LiDAR data containing six different data processing levels has been suggested by the National Geospatial-Intelligence Agency in the US[14]:

- **Level 0** - Raw data and Metadata
- **Level 1** - Unfiltered 3D Point Cloud
- **Level 2** - Noise-filtered Point Cloud
- **Level 3** - Georegistered 3D Point Cloud
- **Level 4** - Derived Products
- **Level 5** - Intel Products

Level 0 The content of the raw data will mainly differ with regards to two factors. Firstly, whether or not the scan system is moving determines if INS and GNSS data is needed to compile the point cloud. Another factor is whether or not the scanner captures full waveform or simply outputs the discrete returns, as

discussed in section . Meta data, such as scan system, date, time and calibration date are also included.

Level 1 The result of this processing done at this level is an unfiltered point cloud, which means the data's 3D representation is computed by using the raw data. The meta data is also kept for this level.

Level 2 After the 3D point cloud is ready, it is filtered to exclude superfluous points from overlap sections between scans. As in Level 1, all meta data is kept for this stage.

Level 3 The georeferencing step is dependant on whether or not the scanner is stationary. For mobile scans, georeferencing means improving the point cloud accuracy or transforming the data into a different reference system. For static scans that are not dependent on having a global position information to compute the point cloud, this step will give them a global reference.

Level 4 Derived products from laser scans can be DEMs, DSMs and other 3D models created using relatively simple algorithms. For level 4, the meta data is not necessarily included.

Level 5 The last level is full-fledged 3D models or information that has to be created using advanced techniques. City Information Models and Building Information Models, which require a lot of work, either computationally or in terms of man hours, full in under this category. Again, meta data might be missing.

This classification scheme completes this introduction to Light and Detection Ranging. It shows that LiDAR data is not necessarily just a point cloud, but may be represented in different ways, depending on acquisition methods, LiDAR scan systems, and at which processing level the data is at. This makes the task of storing the data a bit trickier, which is what will be covered in the next chapter.

Chapter 3

Data Management

3.1 Storing Information

At it's lowest level, storing digital information consists of writing sequences of bits in a predefined pattern onto a storage medium. For long-term data storage, this is ideally done using Hard Disk Drives (HDD) or Solid State Devices (SSD). These storage devices arrange data bits in blocks and provide functions for manipulating these blocks via a data bus, such as SATA or IDE[15, 16]. In recent years there has been a storage technology revolution, as SSDs has become cheaper and faster. This has had huge implications for performance in data access intensive applications, something which clearly is the case for LiDAR data applications. Because the main goal for this thesis has been to create a software application for LiDAR storage, hardware performance tuning has not been within it's scope. But due to a bit of curiosity, a few tests were run to compare the performance of HDD and SSD. these can be found in appendix C. From these it becomes quite clear that the SSD revolution indeed has a huge impact on system performance.

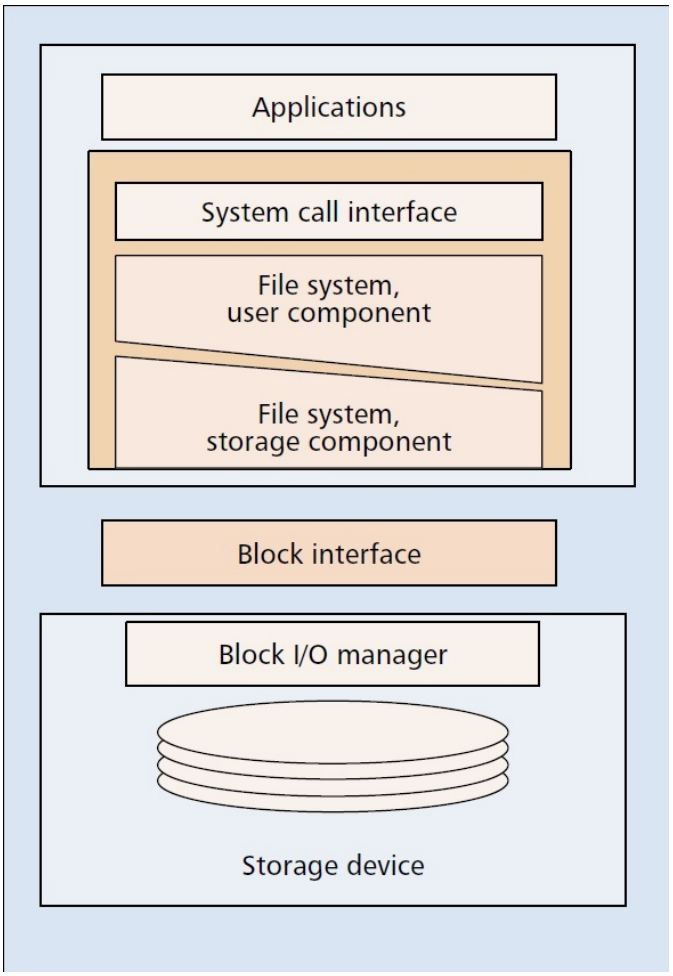


FIGURE 3.1: Traditional computer storage[16].

Software applications connect to storage devices either by accessing the file system of the Operating System (OS), or by directly manipulating the data on a block level, as explained above[16]. The latter approach is mostly used for Database Management Systems (DBMS) that require full control of the data block management[15]. For normal file management, low-level block control is not needed and the OS file system can be used. A model that visualizes the hierarchy of computer storage is shown in figure 3.1. In general, applications that are involved in information management can be split into two groups: Programs that provide direct access to the computer files in a file system, and Database Management Systems that abstracts away the actual data storage and instead provides the ability to manipulate database tables and their content. These two

alternatives have different characteristics and areas of use. In this section we will look at how they relate to LiDAR storage.

3.2 Computer Files

As explained in the previous section, a computer file is simply a sequence of binary values. Computer files are identified by their extension name, which in the case of a image file can be ".png" or ".jpeg". A file extension identifies the specification with which the file was encoded, and is also needed to decode the file. Many computer files have file headers, although it is no prerequisite. Headers are commonly used for identification of file type and version number, as well as potentially useful summary information regarding the file's content. The file version number is essential to decode the file correctly, as file specifications tend to change over time.

Traditionally in the LiDAR industry, each scanner manufacturer has had it's own data file format that has required proprietary software packages to open and process. Historically, this has been a problem whenever file exchange has been necessary. The early solutions were based on exporting data to ASCII files and sharing these, but that naturally had limitations: Without a standardized LiDAR format, there was no way of guarding against inconsistencies between exported information from different programs. Consequences of this could be erroneous data or loss of meta information. This is not a problem today however, as standardized formats have emerged. The next few sections will describe each of the file types that are currently available for LiDAR data storage and exchange.

3.2.1 LAS and LAZ

The file format LAS was published in 2003 by the American Society for Photogrammetry and Remote Sensing and has since it's introduction been updated four times[17]. The current version is LAS 1.4, which had it's last revision in

2012[18]. While it is specifically made for LiDAR data, any three dimensional point cloud can be stored using LAS. The general outline of a LAS file as follows:

PUBLIC HEADER
VARIABLE LENGTH HEADER
POINT DATA

The public header holds summary data about the content, such as bounding box of the coordinates, number of returns, date and year. This header makes it easy to get the size and extent of a LAS file, which can be used to visualize the outline of the file content in a map. The variable length header is for user-specific information, and after that, the point data themselves follow. After launching the first version of LAS, the ASPRS LAS Committee has revised the format several times. Notable additions to the format is listed in table 3.1. The current version of LAS supports up to 15 return values per point, RGB values as added by images from the scan, a range of classification values and full waveform data[18].

Version	Last revision	Major changes
1.1[19]	2003	Classification
1.2[20]	2005	Absolute GPS Time, RGB
1.3[21]	2010	Waveform data
1.4[18]	2012	CRS: WKT instead of GeoTIFF, 64 bit fields

Table 3.1: LAS file revisions

The LAS files have become "de facto" standard for storing LiDAR data in the industry. Most survey companies will include LAS files in the product delivery. LAS files have been purpose-built to handle large amounts of point data, and the number of bytes per point is limited as much as possible. For instance, the coordinates are stored in integers and a scale factor is stored in the header. This reduces a coordinate to 12 bytes and at the same time makes you have to define the accuracy for the data set[22]. Still however, there are improvements that can be made. The company RapidLasso, launched a compressed format of LAS in 2011,

called LAZ. It has been proven to have the capability of losslessy compressing LAS files down to 7-25% of their original size[22]. They also leave the LAS file header intact, which means it can be read without decompressing hte file.

3.2.2 e57

E57 is a file standard created by stakeholders in the 3D imaging industry. It is not exclusively made for LiDAR scanning and has a more general outline than the LAS files. The problem it is meant to solve is that of storing georeferenced imagery and point clouds at the same time. This is useful for creating detailed Building Information Models (BIM), for which a LiDAR point cloud with RGB values may not suffice. The e57 file type therefore has the ability to store both georeferenced images and point data. Similar to the LAS format, e57 also allows for scaling of coordinate values to reduce size[23].

3.2.3 ASCII

ASCII is an text encoding schemes created in the 1960's[24]. It is a standardized scheme for encoding characters that is universal and wide-spread. This makes it easy to rely on for data exchange, which is why it was used as exchange format in the LiDAR industry up until LAS became widespread. ASCII is however not storage efficient, as it uses 7 out of 8 bits for representing characters. Thus, 12.5% of the data capacity is lost for a while[24].

3.2.4 Proprietary formats

While LAS formats are most commonly used for data exchange, the LiDAR vendors still have their proprietary formats. After the introduction of full waveform data, this has perhaps become more important, as more advanced analysis techniques are becoming available. Leica, Riegl and Faro all have their own LiDAR data formats.

3.2.5 Managing the Files

Having gone through the internals of the different files, it is natural to look at how the files themselves are managed. For most operating systems, files are usually organized in the hierarchical file structure that have been used for decades[25]. While organizing files in folders is something any computer user is capable of, it is not necessarily a reliable solution. As anyone who has owned a digital camera for a few years will know, it is bound to get messy unless rigid storage procedures and naming conventions are enforced. And even if that is the case, moving a few files and folders around will cause disruptive changes to the file organization. While it is relatively easy to create an application that reads LAS file headers and displays the information found there in a software application, the underlying file organization is not very robust. Although a potential disaster on the managerial side, the upside to a hierarchical file organization that the data can be accessed by any program that can read the files. As we shall see, there are quite a few.

3.2.6 Software for LiDAR Data

Software that reads, visualizes, transforms and manages LiDAR data are numerous and serve a multitude of purposes. In this subsection, a small selection of these software solutions will be presented. Bear in mind that this is an incomplete list and that there are probably many other solutions out there.

Scanning Software LiDAR scanner vendors usually ship theirs scanners with software for processing the point clouds. The main purpose of these software solutions is to make the point cloud data ready for the end user. Leica, Riegl and Faro are examples of vendors that have their own software for data analysis and processing. These software package are mostly used for data at levels 0-3, as described in section2.2.1. Terrasolid is an example of a stand-alone LiDAR data processing tool. QTModeler and QTReader are other examples of software

dedicated to LiDAR data manipulation and viewing. QTReader was used to visualize the point cloud in figure 2.6.

GIS Software Geographical Information Systems (GIS) have been slow at adopting support for LiDAR data, mainly because it is structured very differently from most other types of GIS data. ESRI, the biggest GIS vendor on the market, added support for loading LAS files in 2006[26]. The spatial database industry, lead on by Oracle, added support for a Point Cloud type in 2008. The PostgreSQL spatial extension, PostGIS, does not include point cloud support or a built-in function for loading LAS files. This is however under development. Among the GIS and spatial databases listed here, full waveform LiDAR data is not supported.

CAD Programs that can be classified as Computer Aided Design (CAD) software, such as Bentley V8, Autodesk Revit or Autodesk Civil 3D, generally support loading LiDAR data both from las, txt and even proprietary formats. A LAS file that has been converted to a Autodesk point cloud format and loaded into Revit, is shown in figure 3.2. These programs don't necessarily have advanced point cloud editing functionality, but plugins are available that will improve the work flow. An example is Imagine's ScanToBim for Autodesk Revit. It makes it easier to create Building Information Models fro point clouds.

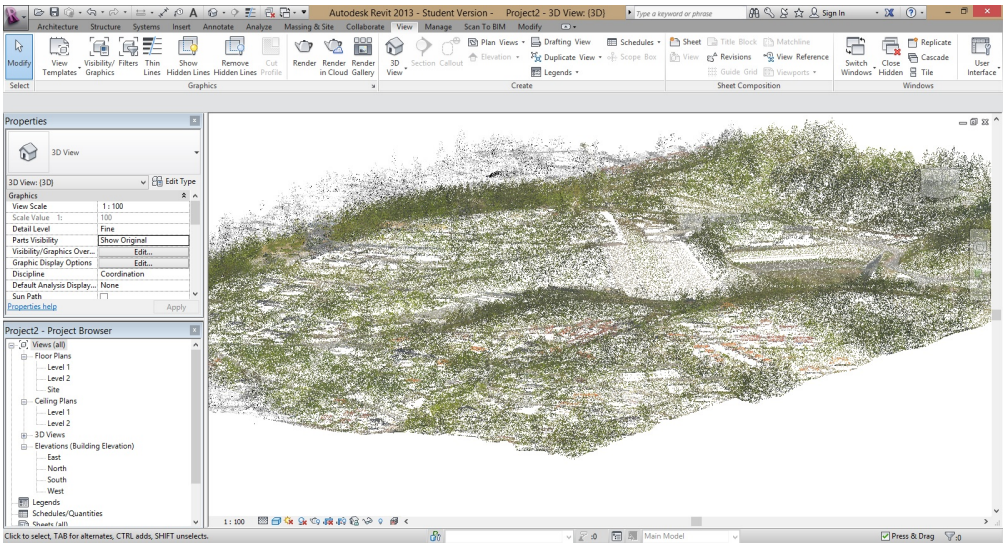


FIGURE 3.2: LiDAR data in Autodesk Revit. Holmekollen data set.

Other Software Solutions Apart from the applications listed so far there are a few more worth mentioning. These software solutions go under the relatively wide description of Transform and Extraction (ETL), or are simply libraries that can be included into bigger programs.

LibLAS, created by the ASPRS, is the official C/C++ LAS conversion library. It is open source and can be used free of charge, which is done by a many companies, among them RapidLasso and Oracle.

FME, developed by Safe Software, is an ETL tool which handles a wide range of file formats. While it's primary concern is to transform and translate data, it has also got GIS functionalities.

LasTools is a collection command programs developed by RapidLasso. It can be used to transform and manipulate LiDAR data. Operations such as creating TINs or transforming point clouds to and from e57 and ASCII files are possible. It also includes a LAS viewer, shown in figure 3.3

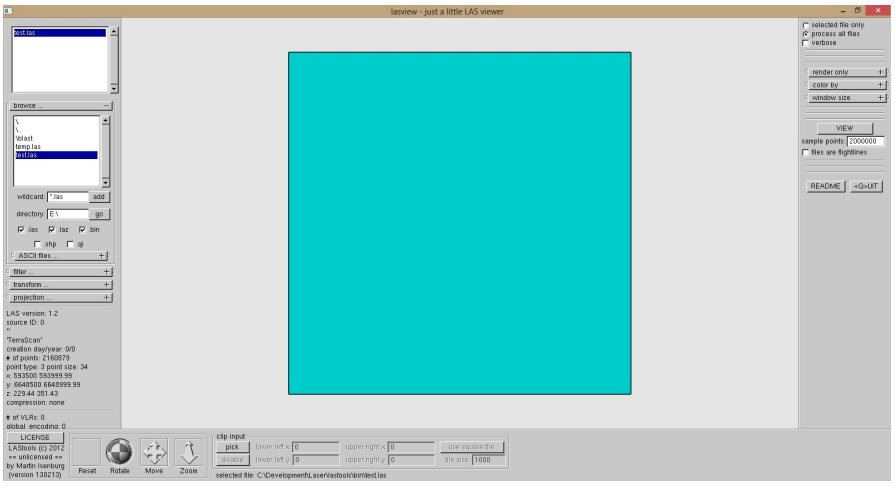


FIGURE 3.3: LasView. Spatial extent and information from a LAS file

3.3 Databases

While storing data in files is an applicable tactic for small projects with few involved parties, it seldom the desired solution for bigger projects with many collaborators and large amounts of data. The bigger the project, the more important it is to keep an organized and universal "truth" of the available information. Luckily, there are systems that can do it for us, called databases.

3.3.1 Relational Databases

During the 1970's, the foundations of modern database industry were laid down, as Boyce published his seminal paper on the concept of a Relational Database Management System (RDBMS) was published in 1970[26]. IBM followed up by developing the database "System R", an attempt to create a relational database that used the concept of transactions and a query language called SEQUEL[15]. SEQUEL was later renamed SQL, short for Structured Query Langauge, which became an ISO standard in 1986. By that time several best practices regarding information modelling has also been established, that for many applications are

valid to this date. The acronym ACID, summarize some of they key principles in traditional database modelling[15].

Atomicity - All or nothing: transaction is fully committed or every change it made is completely rolled back.

Consistency - Transactions takes the database from one consistent state to another consistent state. Examples of violations are NULL values in primary IDs and secondary IDs referring to a deleted row.

Isolation - To ensure data integrity at all times, different concurrency control techniques are implemented to ensure data that is read and updated is valid.

Durability - Committed transactions are permanent, which is ensured by using transactions log and recovery mechanisms.

It should be noted that these principles are far more important for systems that deal with a large number of transactions, than for static data, which usually is the case in the LiDAR industry. Still, these are important principles for general database modelling and taking them into account will help create a more robust and organized system.

3.4 Object-Relational Databases

While RDBMS are excellent for storing simple values such as text and numbers in columns, it is not constructed for storing objects[27]. This became an issue in the late 1980's, as object oriented programming was catching on and object-based storage was needed. This lead to the development of Object-Relational Datbases (ORDBMS), which was included in the SQL:1999 ISO standard. SQL:1999 is generally well-supported by current relational database systems, such as DB/2, Oracle and PostgreSQL. With a ORDBMS it is possible to define new data types and objects, which also implies the possibility of creating spatial database extensions.

This is exactly what creates the foundation for PostGIS and Oracle Spatial Extension. As a result, PostGIS and Oracle have the strict organizational capabilities of relational databases, but also allows for spatial objects to be stored.

3.5 Databases and Geographic Information

This leads us on to the topic of geographic information modelling, which is a huge and advanced topic if need be. For the purpose of this thesis it is however not necessary to delve too far into the definitions of how geographic information should or should not be done. Because a single point is a 0-dimensional entity, and point clouds simply are collections of them, there is little need in worrying about topological questions or other intricacies of the geographic information modelling[26]. What is important in the context of storing point clouds, is to make the point clouds as compact as possible and at the same time retain data quality.

3.5.1 Databases Representing Points

Whereas the file formats that was presented in section 3.2 have mostly been purposely built to store LiDAR data, most current databases do not have their own point cloud types. If you are storing geographic information in a spatial database, chances are that you are doing so using standards maintained by the Open Geospatial Concortium (OGC). For storing geometries, the OGC has provided us with the Well-Known-Text (WKT) and Well-Known-Binary (WKB) for relational databases. These two are identical, but differ in that WKT is ASCII-encoded WKB. A Well-Known-Binary is structured as follows: First the endianness, which is whether or not the least significant bit is put first. Then follows the geoometry code, and after that the number of double precision points that are needed to complete the geometry type[28]. Some geometry types that are interesting in the LiDAR context, are listed in table 3.2.

Type	2D	Z	M	ZM
Geometry	0000	1000	2000	3000
Point	0001	1001	2001	3001
Polygon	0003	1003	2003	3003
MultiPoint	0004	1004	2004	3004
TIN	0016	1016	2016	3016

TABLE 3.2: Well Known Binary Codes, written as integers

In the case of storing a single point geometry, we would end up with the space requirement in table 3.3. This means a single three-dimensional point, if nothing else is stored, would take up 37 bytes. 10 Million points would then amount to 352.86 MB. This can be modeled by equation 3.1.

Field	Endianness	Code	Float	Float	Float	Sum
Bytes	1	4	8	8	8	37

TABLE 3.3: WKB PointZ space requirement

$$S_{PointZ}(n) = 37 * n \tag{3.1}$$

If we look at the case of storing a 10 Million points in one MultiPoint, we will get away with 305.18 MB, saving approximately 8.5 %. MultiPoint size can be modelled by equation 3.2. We are still talking about in the excess of 30 bytes per point.

$$S_{MultiPointZ}(n) = 5 + 32 * n \tag{3.2}$$

3.5.1.1 PostGIS implementation

PostGIS uses WKB to store geometries, but also supports their the Extended-Well-Knwon-Binary. This is an attempt at enforcing spatial reference info for all

geometries, as coordinates have no meaning without a spatial reference. The SRID adds another 4 byte to the total size of a geometry. This leads to PostGIS having a raw byte count as shown in equation 3.3 and 3.3.

$$S_{PointZ}(n) = 41 * n \tag{3.3}$$

$$S_{MultiPointZ}(n) = 5 + 36 * n \tag{3.4}$$

If we compare the raw representation of 10 Million raw coordinates using PostGIS and the LAS implementation we end up with the result in figure 3.4. Taking into account the fact that this is the LAS file, and not the compressed LAZ file, it is clear that PostGIS is not well-suited for LiDAR storage. It is also important to keep in maind that we have not looked at other values, such as number of returns, classification, gps time, or the fact that a database system in itself introduces some overhead[15]. It is evident that the OGC format for storing points is for manholes and lamp posts, not millions of points from LiDAR scans.

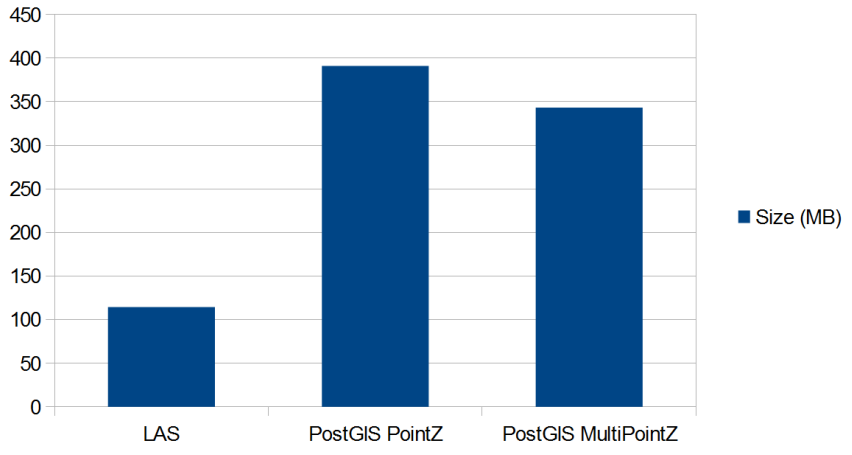


FIGURE 3.4: Coordiante size comparison: PostGIS vs LAS

3.5.1.2 Oracle's Point Cloud

While PostGIS doesn't currently have support for point clouds, Oracle's database 11gR2 with the Spatial Extension has a working solution. Oracle has created a Point Cloud object that is optimized for storing LiDAR data. As can be seen in figure 3.5, Oracle uses a object type called "SDO_PC _BLK" for storing the actual point data, whereas the spatial extent and summaries for each block is stored in a separate "SDO_PC" type. The point cloud blocks are stored as so-called Binary Large Objects (BLOBs), which can be compressed if it is transformed to Oracle's SecureFile type.

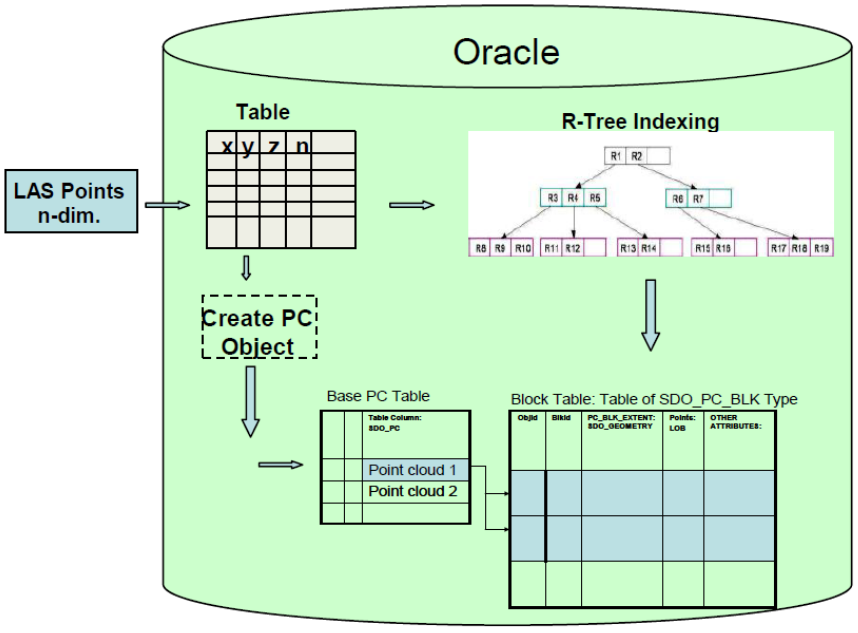


FIGURE 3.5: Oracle Point Cloud[29]

In a presentation given by Oracle in 2009, it was stated that a normal LAS file containing 26 Million points, taking up 552 MB, would be stored in 839.5 MB as a normal BLOB. Furthermore, that as compressed SecureFiles, the LAS file's content would occupy 223.4 MB[30]. This suggests that Oracle Point Clouds can store LAS data with a size reduction of approximately 60%. Apologizes are in order for not

having proper references in this regard, but as always with proprietary software applications, it is hard to come by. Another key feature of the Point Cloud object is the R-tree index on the meta data table, which makes it easier to retrieve data from specific areas. This brings us over on the topics for next chapter, which is indexing and data access.

Chapter 4

Accessing the Data

4.1 Access Methods

Finding a piece of information within a large collection of data is a time-consuming task if no attention is given to the storage structure. In order to speed up data retrieval, a wide range of access methods and storage techniques have been developed for different types of data. The basic principle of these methods is to organize the data with regards to key values, similar to how a librarian organizes books with regards to titles or author names. Which key is chosen, and how it is used, has large implications for performance of a data management system. This section will explain indexing techniques and how they are applied in real world applications.

4.1.1 Hash-based indexing

Apart from storing information in a heap file, i.e. no specific structure, there are mainly two techniques for organizing data: Hashing and tree-based indexing[15]. The former is a way of organizing data by running the data key through a "hash function". This function can be just about anything, as long as it maps input values to a set of output values. The output values are addresses of "buckets"

where the record data is stored. For this technique to work efficiently, the function needs to uniformly distribute data across the different buckets. If one bucket gets all the data, it is no better than storing data in a heap file. When a data record is requested, the record key is run through the has function and the content of the bucket is returned and looked through.

4.1.2 Tree-based indexing

Tree-based indexing is a hierarchical way of organizing key values. Out of all the tree-based indices, the B-tree and it's variations are most frequently used. A B-tree is in a modified version of the Binary Search Tree (BST). The difference between a BST and a B-tree lies in that the B-trees allow several keys and pointers per node, as shown in figure 4.1. This results in a bigger fan-out, more efficient data traversal and reduced index size[15, 31]. Depending on fan-out, the run time of a single key value search will be $O(log_b n)$ in O-notation[15]. Here, the n is data size and b is the blocking factor, which should be adjusted according to the data block size of the underlying storage medium. Different B-trees can differ with regards the number keys or nodes allowed, whether or not keys stored in internal nodes reappear in the leaf nodes, and whether not the leaf nodes have pointers between them.

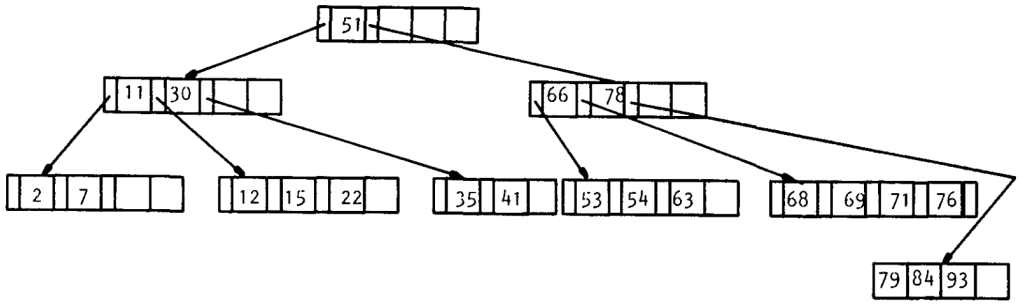


FIGURE 4.1: B-tree[31]

A typical query that will be sped up by a B-tree indexing is Q1. Instead of reading the database table sequentially, also referred to as a scan, the B-tree search will return the record IDs almost instantaneously. Obviously, this is a huge time saver for large database tables. Table 4.1 shows the difference between using a B-tree index to run query Q1, as opposed to having no index at all. The table in the query contains 29 million rows with three integers and a geometry data type.

```
Q1: SELECT * FROM point_table WHERE id>5000000 AND id<5000500;
```

Index type	Average query time
No index	6527.3 ms
B-tree on ID	2.7 ms

TABLE 4.1: Results in milliseconds from running Q1

4.2 Spatial Indexing

Multidimensional indexing requires more advanced techniques compared to indexing one-dimensional keys. Essentially, when two or more dimensions are present, the problem consists of indexing records with regards to two or more columns. Many spatial indices have been proposed throughout the years, but only a few of them have actually been implemented in commercial products. In this section the most important index types will be presented.

4.3 Mapping to One Dimension

It is possible to map multidimensional data to one dimension by using Z-ordering, also known as Morton index, or Hilbert curves. These orderings organize data recursively in a Z- and H-like form, respectively. These orderings are not necessarily used for indexing purposes, but often act either as access patterns or are used during construction of the indices[26, 32].

4.3.1 kD trees and quadtrees

Other examples of tree indices that deal with 2-dimensional data are the 2-D tree and the quadtree. These tree structures subdivide 2-dimensional spaces based on insertions of new points. They differ in that the quadtree will split a square into four new sub-squares, while the 2D-tree use a binary division technique and only ends up with two new rectangles per point. This means the quadtree uses regular spatial division, whereas the kD tree uses a irregular spatial division[26]. Both of these indexing techniques are general and can be implemented for higher dimensions as well. The kD tree would be the 3-D tree for three dimensions, while octtree would be the three-dimensional equivalent of the quadtree.

4.3.2 R-trees

When it was published in 1984, the R-tree was intended for two dimensional structures by using their bounding rectangles as index representation. Similar to the B-tree, the R-tree is height balanced with leaf nodes containing pointers to the data objects[33]. A difference compaired to the R-tree is that there is a possibility of having overlapping nodes. Thus, several sub-trees might have to be visited to find the objects within a bounding box search. The R-tree, as visualized in Guttmann's 1984 publication, is show in figure 4.2[33].

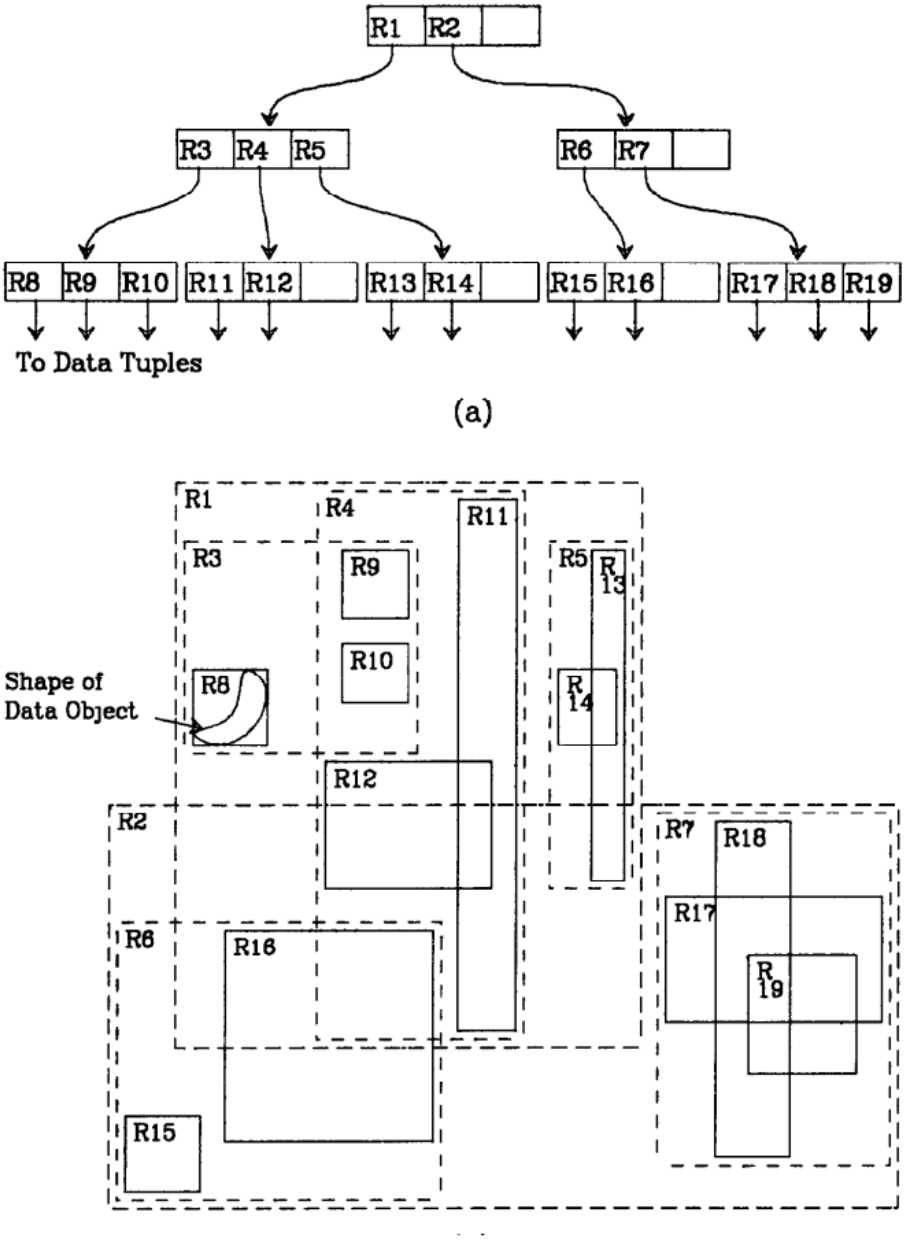


FIGURE 4.2: R-tree[33]

4.3.3 Real World Applications

This section will briefly introduce some real world implementations of spatial indexing. While there are numerous spatial indices in the literature, only a few of them get implemented in commercial products.

4.3.4 PostGIS: GiST and R-tree

PostGIS uses a R-tree that is implemented on top of the Generalized Search Tree Index (GiST) that is provided by the PostgreSQL database. GiST is a general index which can work with both R-trees and B-trees[34].

4.3.5 Oracle Point Cloud: R-tree

In section 3.5.1.2, we read that Oracle uses a R-tree for indexing the point cloud meta data that are derived from the point cloud chunks. The outline of these chunks can be seen in figure 4.3. As can be seen in the figure, the amount of boxes that needs to be indexed is quite small compared to the total size of the survey area.

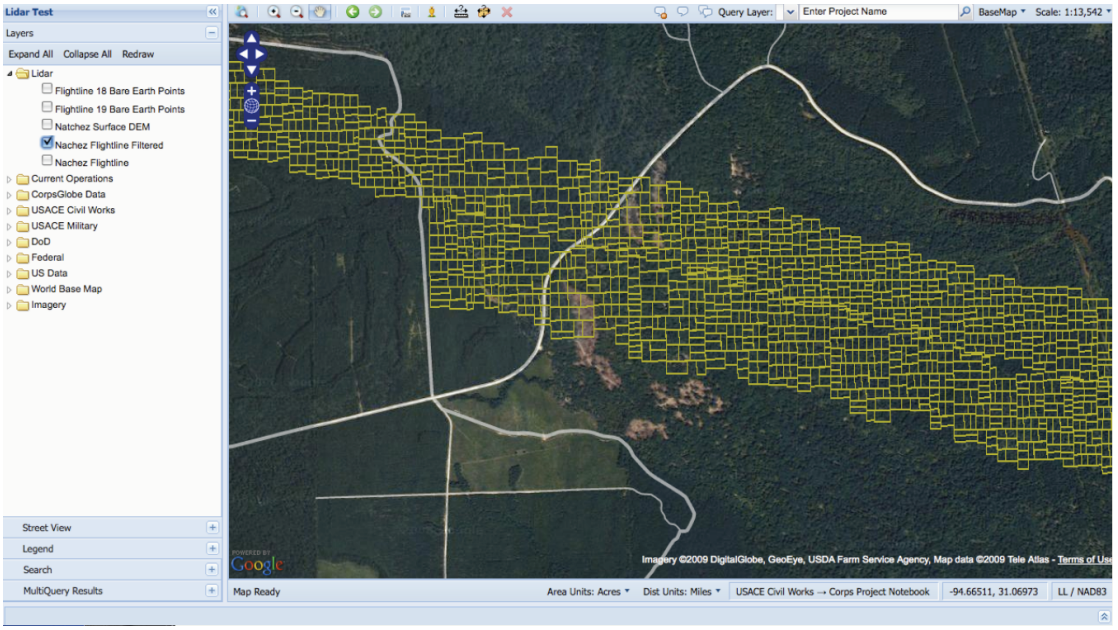


FIGURE 4.3: Oracle R-tree[30]

4.3.6 LAX

Finally, a short note on LAX, which is a file for storing index information, developed by RapidLasso[35]. LAX is a simple file for storing a description for a

quad-tree, which can be used to speed up spatial queries for LAS and LAZ files. According to RapidLasso, this indexing is used by OpenTopography, which will be presented in chapter 6.

Chapter 5

Web Technology

5.1 The Internet and the Web

The Internet combined with the World Wide Web has over the last couple of decades become intrinsic parts of communication and data technology. Many services that were analogue only ten years ago, have been moved to "the Web": Anything from filing tax returns and checking the bank account, to buying concert tickets or simply sharing is just a few clicks away. Entirely new business areas have also emerged: Facebook, Twitter and other social media are examples of companies that couldn't have existed without the Web and globally interconnected computers. The geospatial industry has also seen a considerable market growth over the last decade. After Google Maps was launched in 2006, we have seen an increase in the use of online map services[36]. What the future holds is difficult to say, but it is possible to see some of the potential of future applications by looking at the tools that currently are at our disposal. This section will explain important advances in web technology in the context of web development and 3D content.

5.1.1 HTML5

The Hyper Text Markup Language (HTML) is a markup language created by Tim Berners-Lee in 1989[37]. HTML was initially created as a markup language for researchers to share documents using networks, but quickly caught on in the IT community. Web browsers became popular in the early 1990's, supporting HTML and as well as extension to it for improved functionality. The latter lead to a situation where no real HTML standard was upheld, something which still plagues web browsers used today. As a response to this trend, Berners-Lee created the World Wide Web Consortium (W3C) was created to manage the HTML standard. Despite his effort to standardize HTML, it continues to be a matter of debate where several organizations would like to have their say[38].

HTML5 is the new version of the HTML standard and has a planned completion data in 2022. However, seeing as it is an open standard, many browsers already support some of the implementations that have been finalized. Because a large number of innovative applications are being developed with pieces of HTML5 in them, the term "HTML5" has turned into a bit of a buzz word. This resembles the situation a few years back when the term "Web 2.0" was used to describe the general web trends surrounding user-oriented services[39]. Regardless of whether we are looking at the actual HTML5 standard or the web applications that are being programmed in it's spirit, there are a few ideals that holds true[38]:

- Fewer plug-in programs, such as Flash or Windows Media Player
- Web applications that resemble desktop applications
- Media platform independence

5.1.2 Javascript and the DOM

While the HTML5 standard will make it easier to develop web applications that fulfil the three ideals above, it is already to some extent being done with the aid

of JavaScript libraries in conjunction with the Document Object Model (DOM). The DOM is the form of a HTML page after it has been processed and stored in the web browser, while the JavaScript programs can be used to manipulate these the DOM through the JavaScript API. Because JavaScript is a client-side programming language, applications that are built using it can be more responsive than the ones that require server-side processing.

There are several problems with these two technical solutions. DOM is not consistently defined across different web browsers, which makes development inefficient, seeing as special cases for each web browser must be treated separately. As for the JavaScript, it is tempting to quote Douglas Crockford, who described it as "...a collection of good ideas and few really bad ones[40]." Looking at the amount of JavaScript libraries on the Web, it is however undoubtedly a popular language. New libraries are popping up on almost a weekly basis, covering new and specialized needs for web applications programmers. Examples of popular Javascript libraries are Twitter's Bootstrap library, jQuery and Backbone. For the GIS sector it is worth mentioning that Google Maps JavaScript API, OpenLayers and Leaflet are all based on JavaScript programming. Three of these will be used later on and are presented below.

jQuery is a JavaScript library that makes HTML document traversal and manipulation, event handling and animation easier for the developer. It supports most browsers and create an abstraction level above the different DOM manipulation functions of each browser. As such, this library in part is fixing the mess created by different DOM implementations.

Bootstrap.js is a JavaScript library created by Twitter. It is a framework for creating websites and web applications that work different screen sizes. It makes it far easier to create the user interface and can save developers considerable amounts of grunt work.

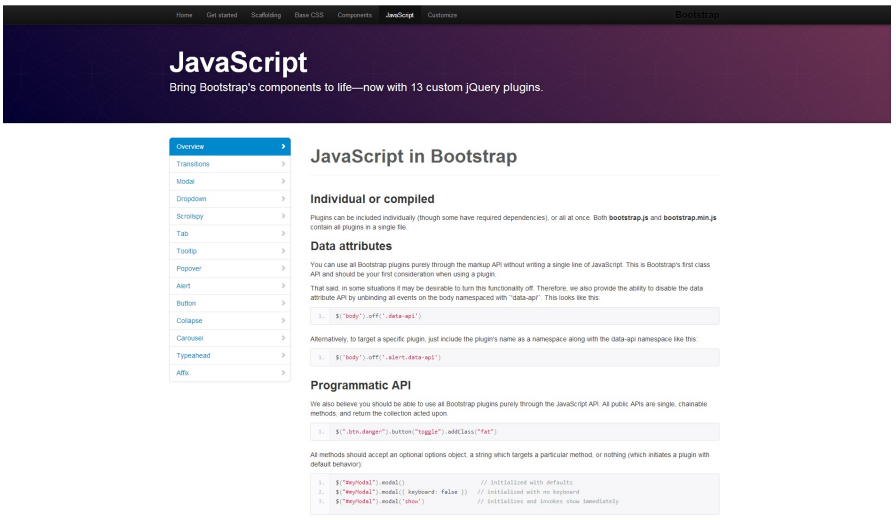


FIGURE 5.1: Bootstrap Example

Leaflet.js is a light-weight JavaScript library for creating web maps. It is in essence a simpler version of OpenLayers (also a JavaScript library), but has fewer and simpler functions. It has become very popular in the last couple of years.

5.1.3 HTML5 in the Third Dimension

Finally we arrive at the most exciting innovations of HTML5, namely the real-time 3D rendering independent of any plug-ins. In 2011, an organization called the Khronos Group launched the first version of the WebGL API, which defines functions to render 3D geometries using the HTML canvas element[41]. WebGL is a truly exciting technology, as it allows the web browser to directly access the Graphics Processing Unit (GPU). Although web browser just recently started supporting WebGL, it is already showing great potential. In figure 5.2, a screenshot from a race car game demo is shown. The car can be driven around just like in a desktop game application, but it is actually running on the web browser.

FIGURE 5.2: WebGL game example

Three.js is a JavaScript library that provides a high-level interface to the WebGL API. Because OpenGL programming is relatively time-consuming, using a high-level JavaScript instead, saves a lot of time and energy. The Three.js library provides relatively easy functions for defining three-dimensional shapes. There are also a multitude of 3D model loaders, so that CAD drawings and SketchUp models can be loaded in as DOM element and then rendered in the browser window. To move the focus back to geographical information, a final example of a 3D globe is shown in figure 5.3. In this example, the earth is rotating and clouds are moving around in real time.

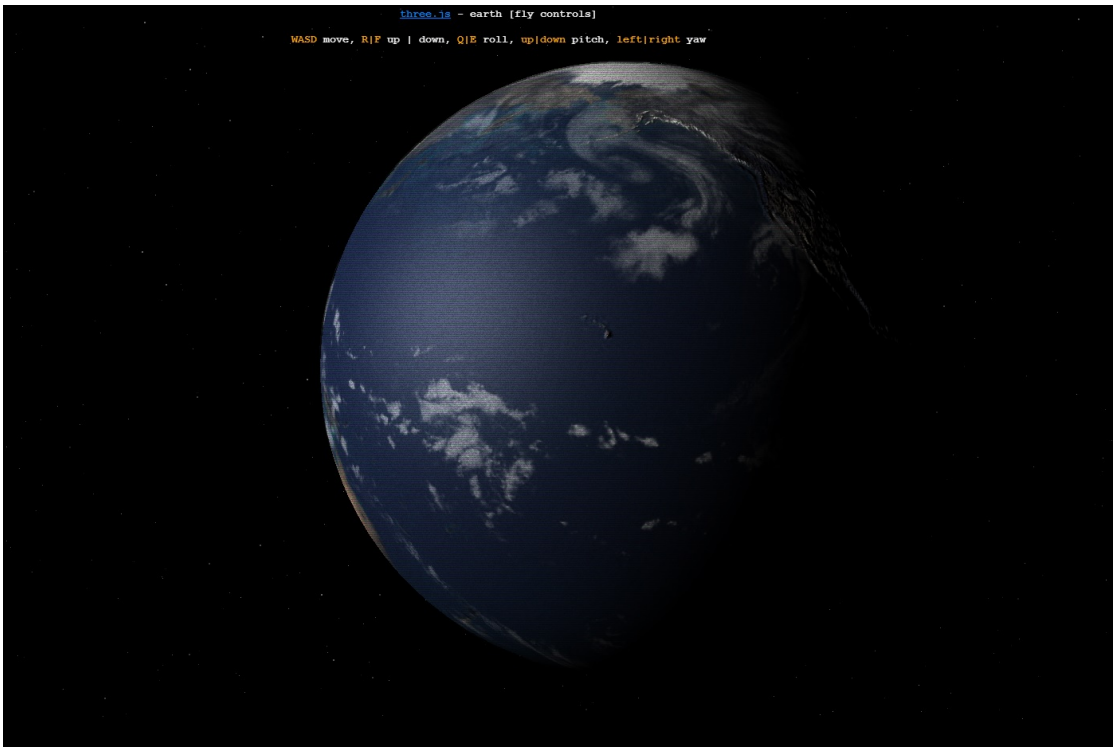


FIGURE 5.3: Three.js globe example

Part III

LiDAR Data Warehouse

Chapter 6

State of The Art

6.1 OpenTopography

OpenTopography is an American organization dedicated to making LiDAR data open and accessible. Hosted by San Diego Supercomputer Centre (SDSC), and sponsored by the National Science Foundation (NSF), OpenTopography has the expertise and financial muscles to deliver stable and efficient LiDAR data services. On their web pages it is possible to find LiDAR scans from all over the United States and order the data on different formats. While OpenTopography hosts most of the data themselves, they also act as a portal to downloading LiDAR data from the United States Geological Service(USGS), the National Center for Airborne Laser Mapping (NCALM) and "Community Contributors", which are smaller data providers. If any of these data sets are chosen, the user is simply redirected to the provider's download solution.

6.1.1 System Architecture

OpenTopography's system was initially based on the architecture from a project from 2010, called GEON. This system used a DB2 database with spatial extension to store the LiDAR data. By loading ASCII-encoded LiDAR data into

the database, and distributing this across several database tables, it allowed the database cluster to serve the data quickly on request. By indexing with B-trees on either longitude or latitude, the data could be partitioned efficiently. The shape of the scanned area would naturally have impact on the efficiency of index-aided retrievals. A drawback by this implementation was that the database storage consumed approximately six times larger than the original scan data [42].

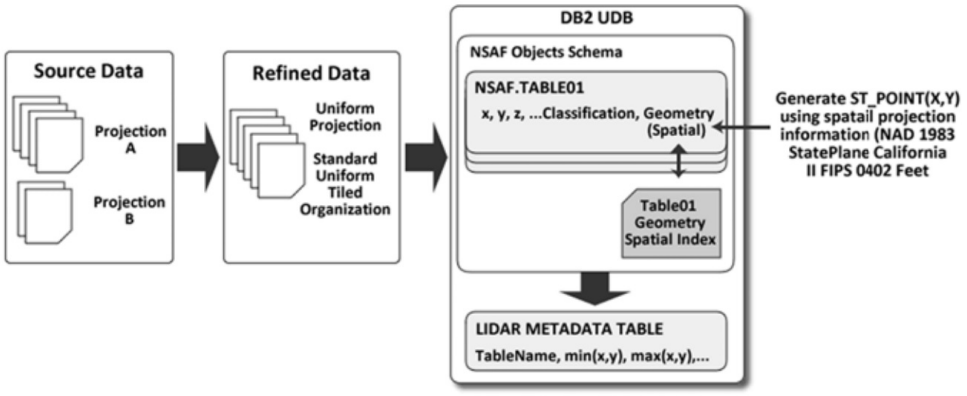


FIGURE 6.1: System architecture of OpenTopography in 2010[42]

In 2011, OpenTopography's system architecture went through changes with regards to storage solution. In addition to having the database from the 2010 configuration, it also supported LAS file storage and processing. By using LibLAS, LAS and LAZ files are easily read, transformed and extracted. However, the LAS files still needs to be organized by a spatial database: Each LAS file's header information is read and stored in a meta data table in the DB/2 database. This database table is used to get file identifiers when a user requests data from a certain area. Any output in addition to the point cloud itself will be generated with 3rd party applications, such as hill shades or KMZ files served by Global Mapper [43]. A system architecture overview from 2011 is shown in figure 6.2.

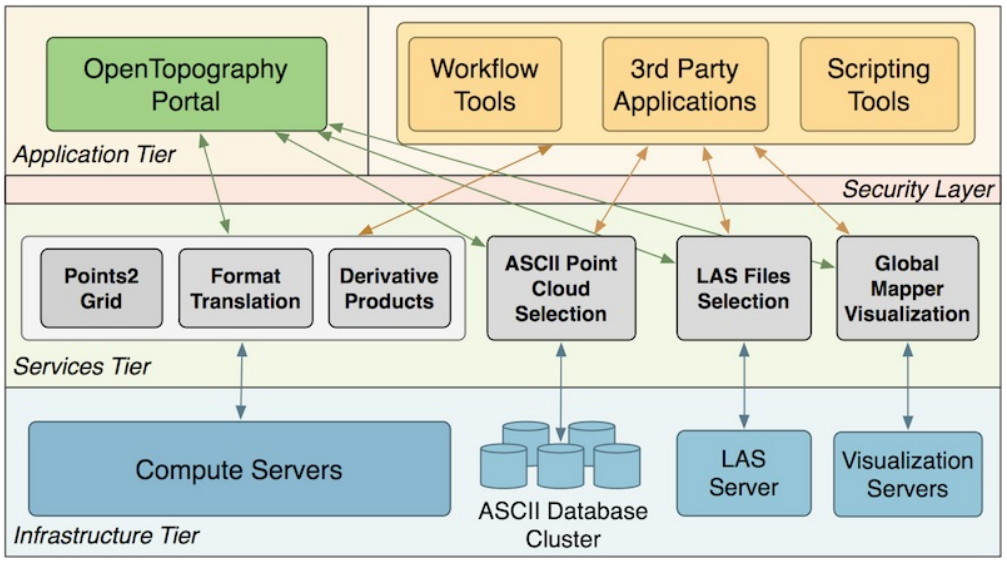


FIGURE 6.2: System architecture of OpenTopography in 2011[43]

6.1.2 Deliverables

The deliverables that are possible to download from OpenTopography can be defined once the "get data" option is chosen. The first choice the user makes is the spatial extent of the data set, which can be done by drawing a bounding box or specifying coordinate bounds. There is a limit of 50 million points per job for anonymous users, and 150 million points for logged in users. The type of returns that should be included can also be chosen; either "all", "ground" or "unclassified". After that is done, the desired products can be chosen and altered with different parameters.

- Raw point cloud: LAS, LAZ or ASCII
- Terrain Model: DEM by local gridding or Delaunay Triangulation (TIN)
- Derived products: Hillshade and slope grids in GeoTiff or Erdas IMG format.
- Visualization: Google Earth KMZ files for visualization

After submitting an order, OpenTopography handles it on the go. The processing time is highly dependent on the size of the data set, as well as chosen products and

preferences. On top of this comes server load at the time of ordering. Table 6.1 shows the same data set being downloaded with different settings. As can be seen, choosing all products will increase the processing time substantially compared to just getting the raw data. It should be noted that selecting ASCII files results in them being zipped, while the same is not the case for downloading the data set as a single LAS or LAZ file. Tests performed on different areas show relatively speaking similar results.

Job	Points	File Format	Products	Time	Point Files Size
ME1	27,158,487	LAS, LAZ, ASCII	All products	325 s	369 MB
ME2	27,158,487	LAS, ASCII	All products	380 s	552 MB
ME3	27,158,487	LAS, LAZ, ASCII	-	151 s	552 MB
ME4	27,158,487	ASCII	-	130 s	309 MB
ME5	27,158,487	LAS	-	40 s	518 MB
ME6	27,158,487	LAZ	-	30 s	59.8 MB

TABLE 6.1: OpenTopography processing time of Mount Edwards in Colorado, USA.

6.2 Denmark: Kortforsyningen

Denmark was one of the first countries in the world to obtain complete LiDAR coverage. The entire data set is available on the Ministry of Environment's web sites, stored as zipped LAS files. While there is no available information on system architecture, it is evident that the LiDAR storage is entirely file-based: As a user you can choose any number of tiles in an online map viewer and proceed to download them. The tiles marked in the map viewer are immediately presented as a list of downloadable zip files, indicating that no processing is being done. This solution is different to OpenTopography, as it does not give you the opportunity to download subsections or combinations of the tiles, nor get derived products from it. Other products, such as a DEMs, can however be downloaded separately by going through the same process of selecting tiles.

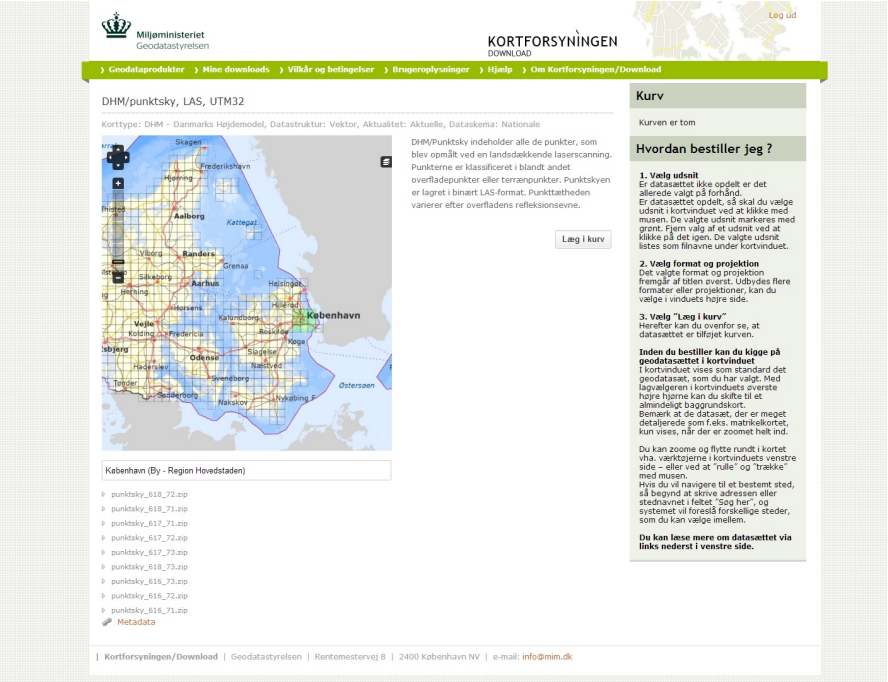


FIGURE 6.3: .

FIGURE 6.4: Denmark LiDAR data set

6.3 National Land Survey of Finland

Just like Denmark, Finland has obtained complete LiDAR coverage and published this online. The web application is quite similar to that of Denmark's, but serves the files on the compressed LAS format, LAZ. As have been previously discussed in section xxx, the LAZ format performs better than zipped LAS with regards to storage efficiency and is far more practical for meta information retrieval and file loading. In practice however, these two solutions are very similar: They both use file servers combined with a map viewer that has references the tiles to file identifiers.

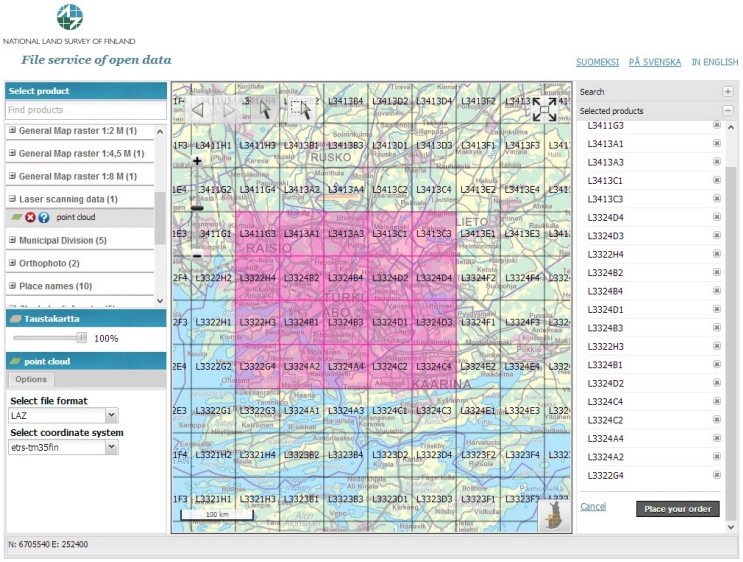


FIGURE 6.5: Finland LiDAR data

6.4 CyArk

While the three projects mentioned so far are led by governmental-funded organizations with a mission of making LiDAR data available to all potential users, CyArk is an ideal organization with a different goal: To digitally preserve cultural heritage sites all around the world. They do this by collecting precise information from heritage sites using LiDAR scanners, total stations and photography. The data is made publicly available through their own web site, www.cyark.org. On this web page they have a map with markers for each scanned heritage site, which links to a site with all related information and 3D models.

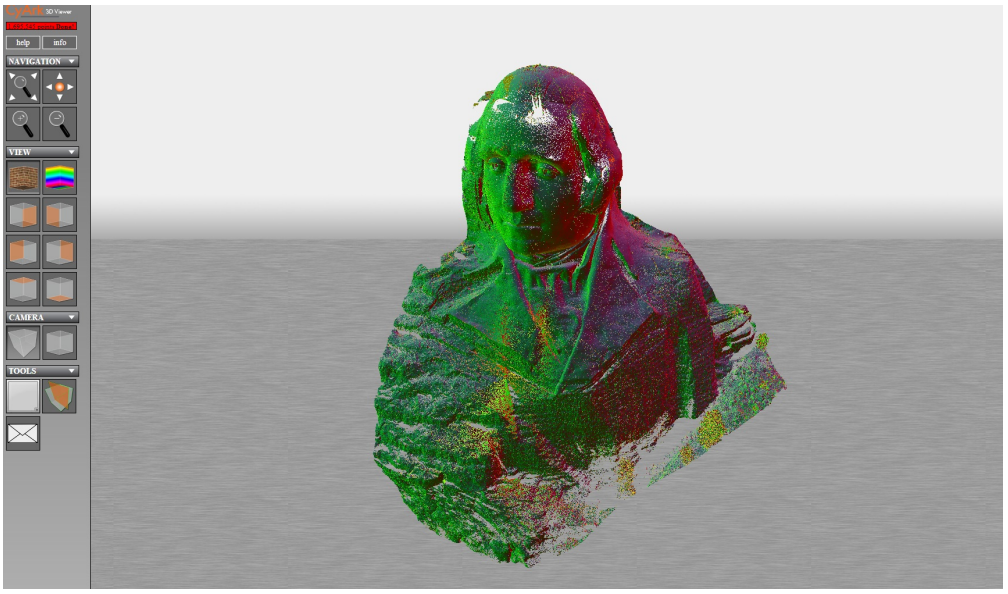


FIGURE 6.6: CyArk Point Cloud Viewer

A WebGL-based 3D viewer for raw point clouds as well as derived models. No raw data extraction is however publicly available, but the points can be seen in a point cloud viewer. This 3D viewer typically loads a point cloud The point cloud loader typically loads the data at a rate of 20 000 points per second. The point cloud visualized in fiugre 6.6 contains 1.7 Million points and loads in approximately 1.5 minutes.

Chapter 7

Prototype

7.1 LiDAR Data Warehouse

In the chapter 2-4, different aspects of LiDAR data acquisition and data management was presented. Chapter 6 finished off part 2 by show-casing the capabilities of modern web technology, before a selection of the state-of-the-art LiDAR online services were presented in the beginning of part three. In this chapter the LiDAR data warehouse prototype built during this master thesis will be presented. This is accomplished by giving a thorough presentation of the system architecture and the rationale behind the design decisions made during the development phase. Finally, tests are performed to assess the system quality.

7.1.1 Prototype Development

The requirements of the LiDAR data warehouse were stated in the project description and given further depth in discussion with the thesis supervisors. The key requirements for the system has been narrowed down to four points:

R1 Support storage of georeferenced LiDAR data

R2 Enable visualization through the use of Web technologies

R4 Implement user interface for efficient extraction of LiDAR data

R3 Use open-source technology where applicable

The prototype was developed using an iterative development process, where a three-step method has been applied for each iteration, as shown in figure 7.1. This approach divides the overall design challenge into manageable problems and makes it possible to create a skeleton system early on[44]. Because of this, what is presented in this chapter not a complete recap of the different development stages the prototype has been through. Detailed descriptions of the changes between the different iterations can be found in appendix A.

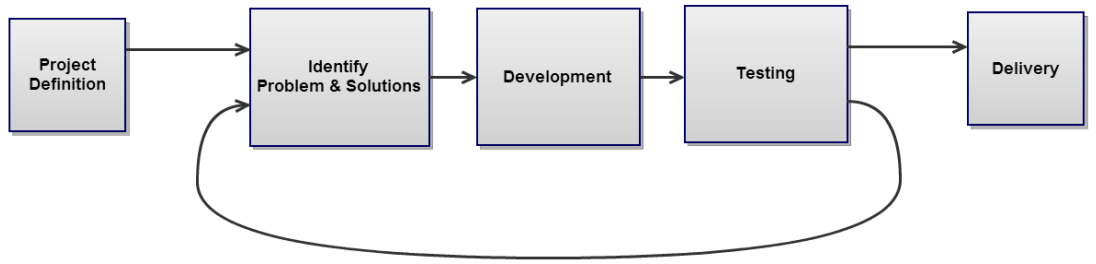


FIGURE 7.1: Development process

7.2 System Components

The LiDAR data warehouse can be split into three different modules or layers: The spatial data storage and processing layer, the server layer and the web client layer. This general outline resembles the structure of the 2011 version of OpenTopography, seen in figure 6.2. By creating well-defined interfaces between the layers, the components within each layer can be changed relatively easily. Out of the three components, the web client and backend storage has received most attention, whereas the server layer is simply handling user data access.

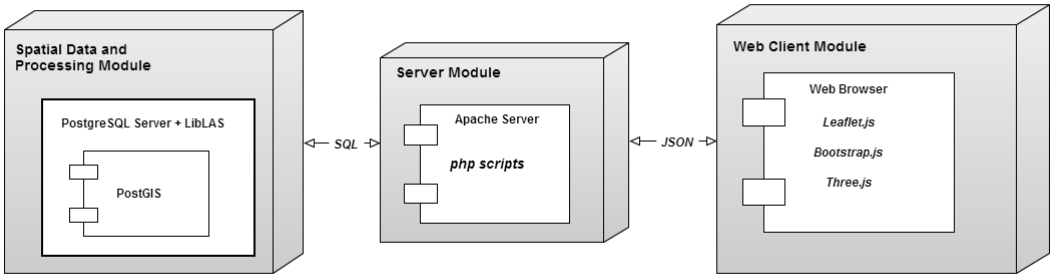


FIGURE 7.2: Cloudy point viewer

7.2.1 Layer 1: Spatial Data Storage and Processing

The first layer comprise of the storage solution chosen for the system, combined with extraction and loading tools. Given the objective of storing georeferenced point clouds, we are effectively looking at level 3 in the NGA's data processing classification list, presented in section 2.2.1. A second option would be to also include full waveform LiDAR, as stored in LAS 1.3 and LAS 1.4[18, 21]. Although this would be interesting and usable for waveform algorithm research, it also increases the storage need approximately by a factor of 5[6]. Because of the increased data size, combined with the fact that full waveform is mostly interesting for algorithm research, it was decided that was not to be included for the prototype.

7.2.1.1 LiDAR Data Storage

The choice between LiDAR storage solutions

As explained earlier, LAS and LAZ files are excellent storage solutions for LiDAR data. Despite this, it was decided to choose the PostGIS for LiDAR storage. The rationale behind this choice is the following: Spatial databases for LiDAR data storage are commercially available. Oracle Spatial Extension is an example of this,

and while the current PostGIS version has not been developed for LiDAR storage, efforts are being made to build a Point Cloud object similar to Oracle's solution[45]. While this means the current PostGIS version is inefficient at storing LiDAR, it still provides us with many of the same spatial functions we would expect from a Oracle database. Because of the modular structure of the prototype, it would be easy to change the database engine, as long as it provides us with the same query capabilities. The mix of a large number of spatial functions and a solid system for transforming between spatial reference systems, results in a versatile and robust system for working with the LiDAR data.

7.2.1.2 Database Structure

The prototype's PostGIS database consists of two types of tables: A meta data table and tables that hold the actual point clouds. In the meta data table, all meta information for a data set is stored, together with aggregated spatial information and a bounding polygon. This means it works in much the same way a LAS file header does, but also includes the exact outline of the scan. Table 7.1 shows an example of what a meta data table could look like. The polygons in the meta data table are indexed using PostGIS GiST and R-tree combination, which speeds up spatial queries on the data.

Metadata table	
id	PRIAMRY ID
Source	VARCHAR
Name	VARCHAR
BoundingPolygon	BBOX3D
SRID	INTEGER
HREF	INTEGER
Number of points	BIGINT
Number of classes	INTEGER
Max number of returns	INTEGER
Scan date	DATETIME
Uploaded date	DATETIME

Table 7.1: Prototype meta data table

Seeing as this database scheme implies storing a summary information and bounding polygon derived from the point raw data tables, the scheme is not compliant with best practices in database modelling. If a set of points is deleted from a point table, this would lead to an inconsistencies between the points stored in the raw data table and the derived information in the meta data table. This is however not a big problem for LiDAR data: Once the point clouds are stored in the database, they are not likely to be updated. In other words, there is in practice a write-once policy for the point data and therefore the ACID violations become less significant. This "lesser form" of database modelling is based on Kimball's data warehouse approach[46]. It was introduced in the 1990's and used in order to improve performance of large data intensive systems, which resembles the current state of LiDAR data.

Point table	
Point	PointZ
Green	INTEGER
Red	INTEGER
Blue	INTEGER
Intensity	INTEGER
Class	INTEGER

Table 7.2: Prototype point data table

The point data in the prototype simply stored as separate tables, as shown in table 7.2. This solution is similar to the architecture chosen for OpenTopography's first implementation[42]. The reason as to why all the points were not stored in one table and then tagged with a "point cloud ID", becomes apparent when the storage overhead is taken into account. A single integer of 4 bytes would add another 38 MB to a point cloud that holds 10 million points. Seeing as table names are unique within a database, they will work as ID's on their own accord. An overhead such as this is however dwarfed by the general storage inefficiency of the PostGIS database, which will be shown in section 7.4.

7.2.1.3 Extracting, Transforming and Loading

What is usually referred to as Extracting, Transforming and Loading (ETL) in the database community, is the process of grabbing data, extracting valuable information from it and then making it compliant with the application's data structure, before finally, it is loaded into the system[46]. For LiDAR data this problem boils down to what type of meta data and raw data that is to be kept. For the purpose of this prototype, all the data was not kept, as this made the development process simpler. Ideally however, no information should be left behind. Seeing as most LiDAR scans are stored on LAS format, the ETL process will usually comprise of gathering all the information from the LAS headers and the point clouds in the raw data. For LAS 1.2 and below, the loading can be done with LibLAS, which like

the LAS format is created by the ASPRS. For this project, the temporary solution of using the command-line programs was chosen for loading LAS files instead of programming a PostgreSQL database loader. This would not be acceptable for a real world application but results in the same data being loaded into the database and saves development time.

7.2.1.4 Data Accessing

In order to enable LiDAR data sharing via Internet, the server has to communicate with the database: The Apache server connects to the PostgreSQL database server using the php extension "pgsql", which enables the two applications to communicate. The Cloudy web client module is served only one type of data: JSON-encoded information. This can either be from the meta data table and raw point cloud data. The former is used for visualization in a web map viewer, whereas the latter is used for point cloud 3D visualization. These data values are easily extracted from the PostgreSQL server using built-in functionality.

While the bounding polygons can be handled quite easily by the web client, the load time and resources required to visualize a large LiDAR data set requires more considerations with regards to data size and processing. If such considerations are not taken, the client browser or the server could crash. The server needs to determine how to scale the data that is requested. Potential pruning of the data set is however also related to the database design.

7.2.1.5 Database Modelling Alternatives

Retrieving a pruned data set quickly from the database can be done using one of two techniques:

- Thinning of data by query
- Thinning of data by storage

The point data could either be selected by a query that leaves out points before they are shipped off the return set. The other option is to have pruned data set stored in the database, working as a quick selection of points.

7.2.2 Layer 2: Server

The second layer consists of the server that grabs data from the backend storage and serves it to the web client. The server was also planned to take care of job scheduling and organize the files ordered by users. Choosing the server was not a overly important architectural choice, seeing as the main challenges as defined in the project description are best solved by the frontend and backend modules. The final decision was to go with an Apache server, because it has a large user base and long development history, both of which is useful when you run into development problems. Currently, the systems serves a total of three different web pages, and each of them are related to a php script. Their php scripts perform the following tasks:

getpoints.php - Returns points based on spatial queries. Determines how much a data set should be pruned **getdata.php** - Returns the meta data information to displayed in the map **export.php** - Starts the export scripts that stores information in text files, which then is transformed to LAS files.

7.2.3 Layer 3: Client Layer

As explaind in section 5.1.2, web technologies have been developed over the last few years that has enabled us to create more responsive and interactive web applications. Because web browsers are becoming increasingly powerful, we can create applications that offload the server with regards to certain tasks. Open-source Javascript libraries, created by eager developers all over the world, makes it easier to develop simple and user-friendly web sites without spending a lot of time on it. For the Cloudy prototype, three libraries have made the development process easier:

- Leaflet.js
- Bootstrap.js
- Three.js

The combination of Leaflet.js and Three.js makes it possible to visualizes the extent of the point clouds in a map viewer, and provides real-time rendering of 3D points. This means that the users does not have to install any software on their computer to get a look at the data: Everything is directly accessible through the web browser. Additionally, Bootstrap.js makes it easy to create a organized and user-friendly web site in a short amount of time. Sequence diagrams for the interaction between the web client and server can be seen in Appendix D. The sequence diagram in figure D.2 shows ow the data is subdivided and sent off to the client's web browser, where each chunk becomes it's own "point cloud object." This object oriented way of organizing the points in the point cloud viewer makes it more scalable and efficient than what would be the case for loading in everything in one chunk. It resembles the "tiling" in a Web Map Services (WMS).

7.3 Presenting Cloudy

The front page is a simple map viewer with the point clouds highlighted as blue polygons. To the left there is an information box for selected point clouds. It contains information about the number of points in the data set, the horizontal coordinate system and vertical reference system. Upon polygon clicks the map viewer will navigate to that polygon. The settings allows the user to change the background map. Currently, OpenStreetMap and the Norwegian Mapping Service maps are available.

FIGURE 7.3: Cloudy front page

7.3.1 Point Cloud viewer

The point cloud viewer is initiated when the user presses the visualization button on the front page. This opens up a new window where the 3D points are loaded into the WebGL window. The loading time is dependent on the extent and density of the point cloud that is being loaded. If it is split up into several point cloud tiles, these will load and get built individually. Throughout the development phase, it was realized that tiling of the point cloud queries is practical for web client performance, this is discussed in the test section. The point viewer can visualize different values of the point cloud, such as RGB, intensity or classification. It is also easy to implement other visualizations that are real-time by the web viewer. Height indication by colour is an example.

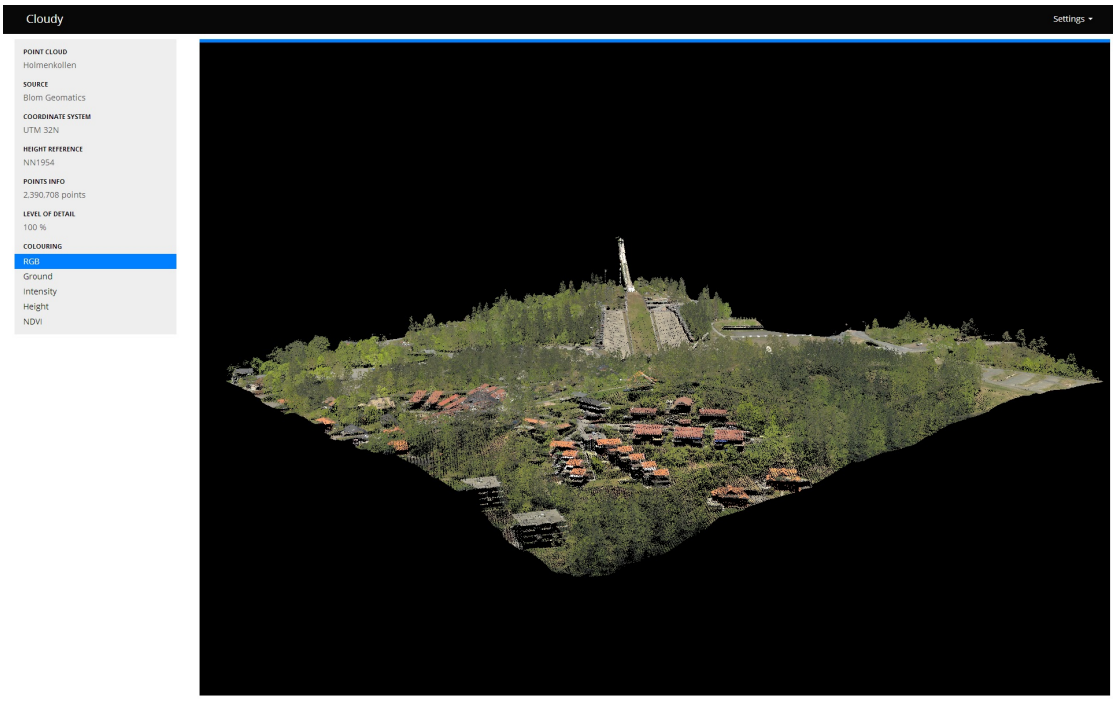


FIGURE 7.4: Cloudy point viewer

The point cloud viewer has functionality for changing colour based on classification values, RGB values, height, intensity value and NDVI. These colour schemes can either be applied via the data loaded from the database, or ratios such as the NDVI and height can be comptued on the go. The NDVI is explained in detail in appendix C.

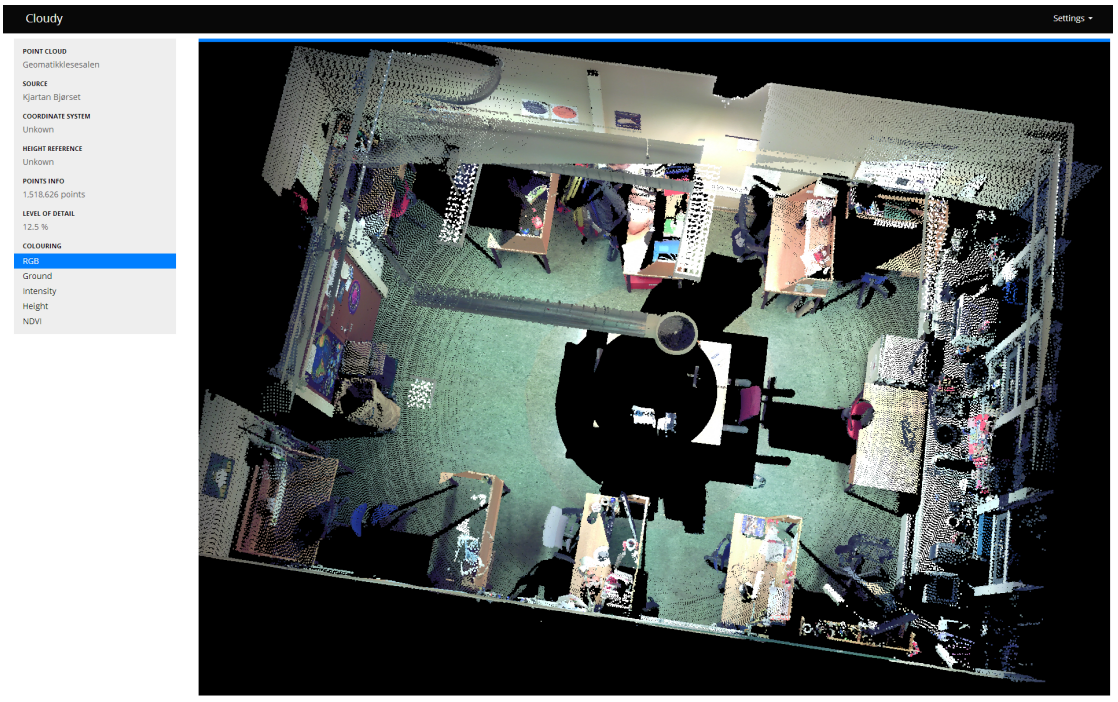


FIGURE 7.5: Cloudy point viewer: Indoor bird's perspective

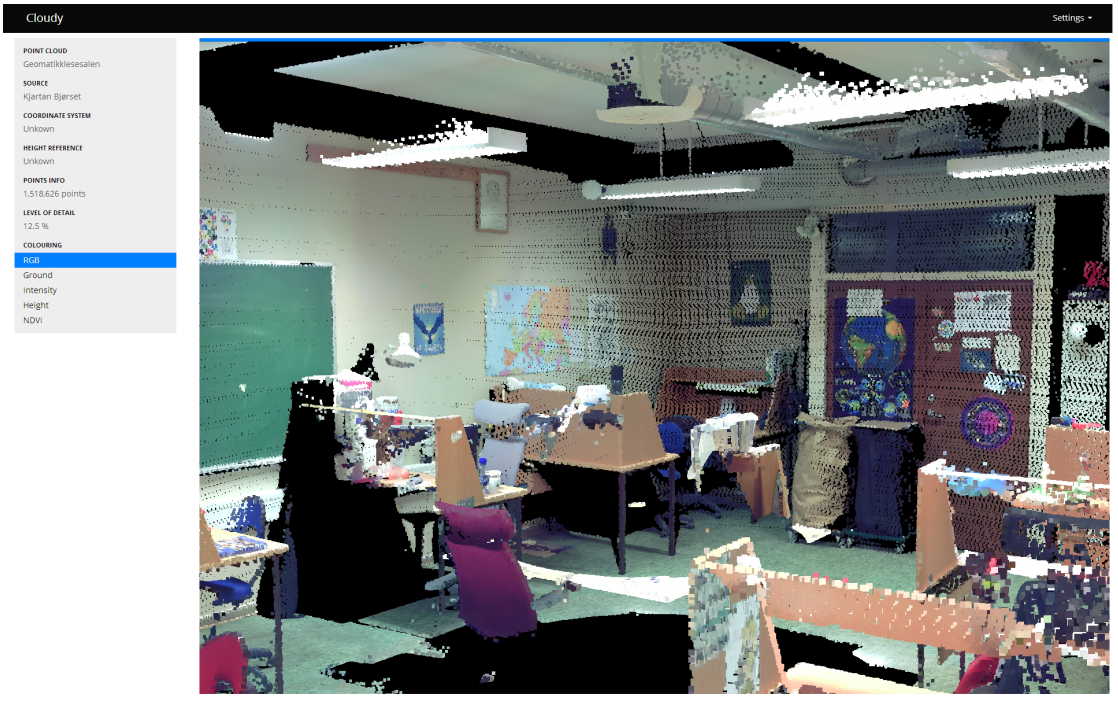


FIGURE 7.6: Cloudy point viewer: Indoor close-up

As can be seen in figure 7.5 and 7.6, the point cloud viewer works well for small, indoor scans as well. The point clouds in these figures have not been georeferenced before they were loaded into Cloudy, and therefore these are tagged as "unknown." Notice also the "Level of Detail" in this picture, which is 12.5 % of the original data set size. The data set in figure 7.4 shows a 100 % level of detail and the 2.4 Million points is on the verge of crashing the Chrome browser running it. If we wanted to load in the entire data sets, as the outline suggests in figure 7.3, the same pruning of the data would be enfored and it would result in the image seen in figure 7.7. Notice also that there is only RGB data for parts of the point cloud.

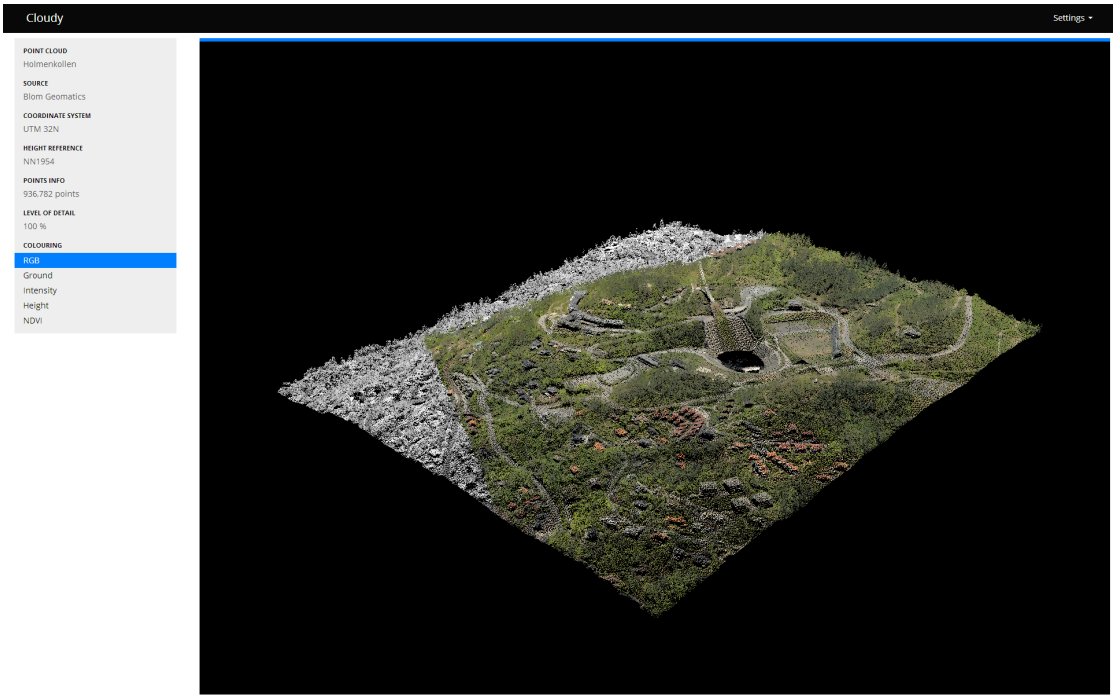


FIGURE 7.7: Cloudy point viewer: Pruned dataset

Some data sets are indeed colour-less and will in that case be coloured as white in the current version of Cloudy. This was the case for LiDAR data acquired for the Nidarosdomen cathedral in Trondheim, shown in figure 7.8.

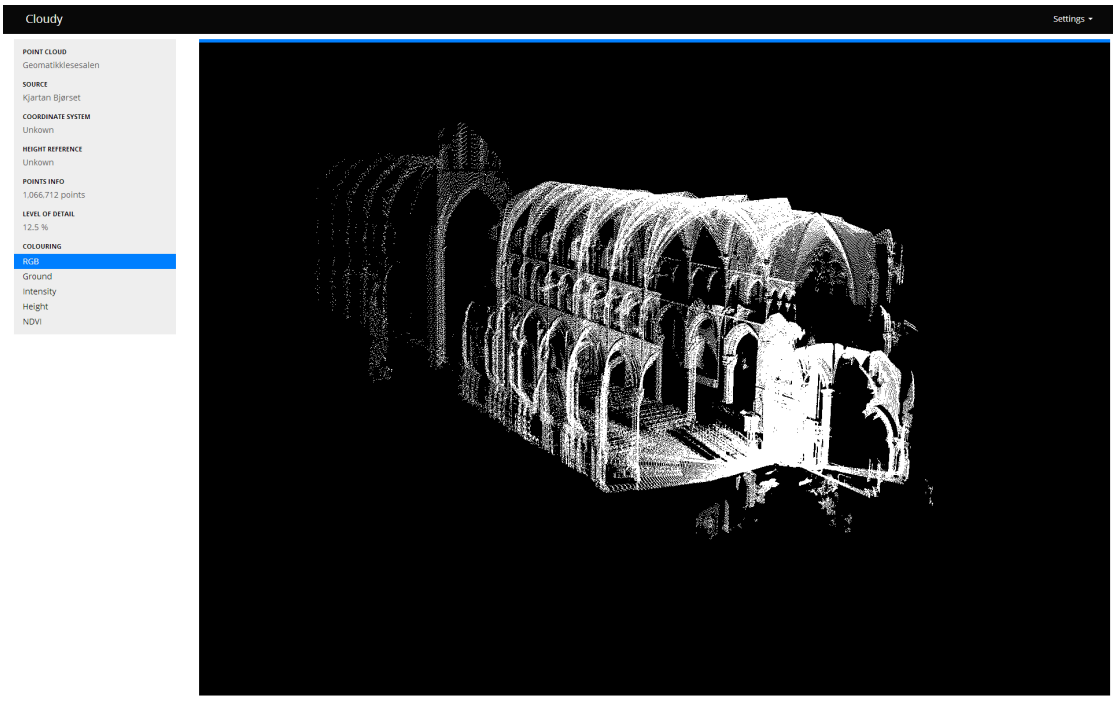


FIGURE 7.8: Cloudy point viewer: No colour

7.3.2 Data Extraction

The data extraction window is launched from the point cloud viewer. In the current version, it is only possible to select each point cloud, but the functionality of selecting subsets is easy to implement with a Leaflet plugin. As with the ETL process, the ideal solution is to utilize LibLAS and create a database script for extracting the data on a LAS format. The implementation of the current version is however limited to exporting the data on a text file which is then manually transformed to the LAS format. The export page is not fully operational in the current version of Cloudy, but the conceptual export page can be seen in figure 7.9.

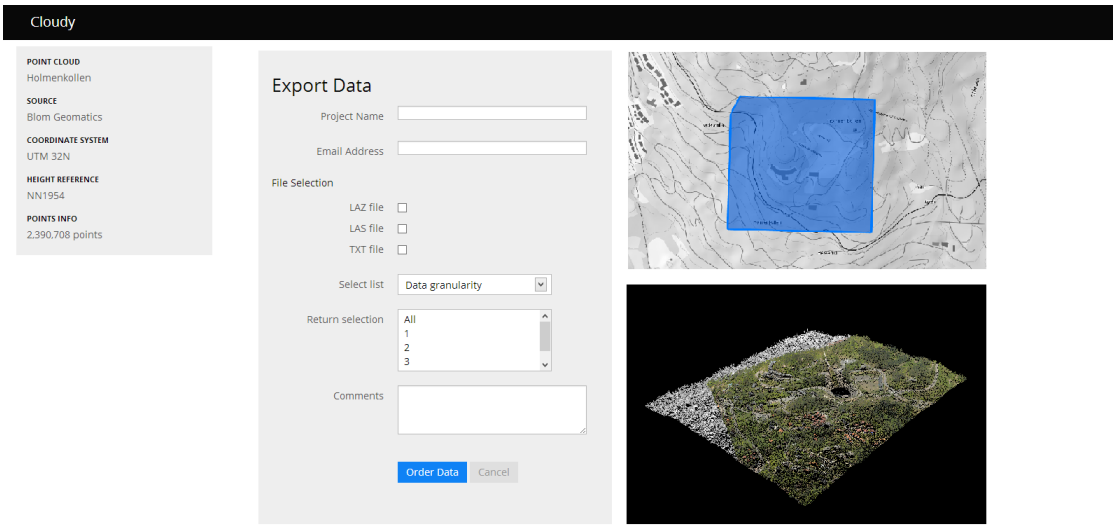


FIGURE 7.9: Cloudy export

The export form page requires the user to fill in email address and select what type of data the database should export. After the form is filled in, it is sent to the server, which creates an export job for the PostgreSQL server based on the data order. This job is run and stored to file. A sequence diagram that show how this interaction can be seen in figure D.3 in appendix D.

7.4 Testing

In order to assess the quality of the system, a number of tests have been devised and run. The tests are divided into two main categories. One test category is focused on measuring the storage efficiency and data access speeds. The other category is concerned with the web viewer's performance. Both test categories have been used throughout the development phase, and helped understand the performance characteristics of different methods.

7.4.1 Storage efficiency

Because one of the biggest challenges with LiDAR data is the data size, the storage efficiency is an important metric in assessing a storage solution's quality. Figure 7.10 shows how the chosen PostGIS implementation performs compared to a LAS 1.2 file, even with less information included. There is far more content in the LAS file and still the PostGIS table is three times bigger. When the GiST index is inlcuded, the result is even worse.

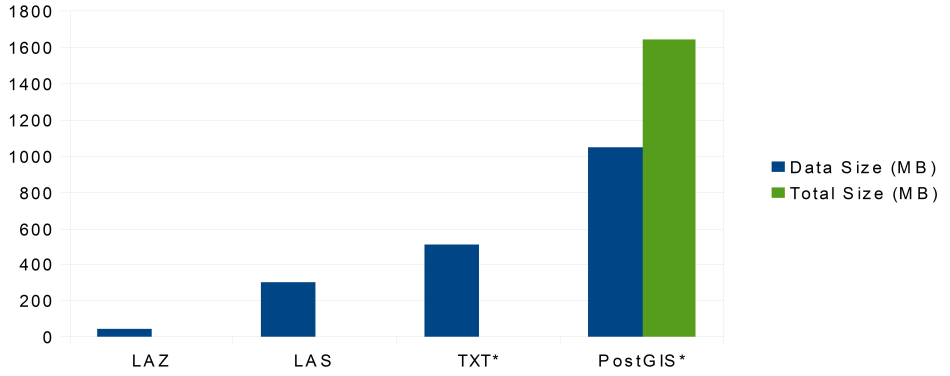


FIGURE 7.10: Relative storage efficiency: 9.4 million points. *The TXT file storage and PostGIS database table contains only XYZ, RGB, Classification and Intensity values. LAS and LAZ files contain orignial scan data.

As was explained in section 3.5.1.1, the main problem with the point geometry is that for each point, a full double precision number is stored per coordinate, in addition to a geometry type and SRID. We can re-run the comparison from section 3.5.1.1 with the database table that is used as reference in 7.10. The difference is mainly the addition of RGB values, intensity and classification. Table 7.11 and 7.12 shows how the values relates to storage implementation in LAS and PostGIS.

Data	WKT PointZ + SRID	R	G	B	intensity	classification	SUM
Byte	28	4	4	4	4	4	52

FIGURE 7.11: PostGIS DB single point storage. All database specific storage use not included

Data	X	Y	Z	R	G	B	intensity	classification	SUM
Byte	4	4	4	2	2	2	2	1	21

FIGURE 7.12: LAS file single point storage. All LAS specification fields not included

By looking at figure 7.10, it is clear that the database implementation takes up more space than what is indicated by figure 7.11. This is most likely due to database implementation, which has not been thoroughly investigated and there it is hard to say anything about it. As can be seen from table 7.3, the table size in PostGIS as reported by the psql relation size function amounts to 1046 MB, which leads to approximately 117 bytes per point. In the table C is short for classification value and I is short for intensity value.

Solution	Points (M)	Content	Size (MB)	Avg. byte/point
LAZ	9.4	Original	42	5
LAS	9.4	Original	304	34
TXT	9.4	XYZ RGB IC	511	57
PostGIS	9.4	XYZ RGB IC	1046	117
PostGIS + Index	9.4	XYZ RGB IC	1640	183

TABLE 7.3: Size Comparison I: Holmenkollen - 9.4 Million points

The biggest data set to be loaded into the database, was a collection of approximately 60 individual scans from the Nidarosdomen cathedrag. All together, the data set contained over 1.5 billion points. This data set was too big for the SSD hard drive without making a serious effort at deleting data. When loaded into PostGIS, which took 15 long hours on a Seagate Momentus XT hard drive, the point cloud containing three coordinates in addition to 7 integer values took up over 100 GB. Comparatively, a LAZ file takes up 2.24 GigaBytes, which is 7.3 % of the LAS file. The PostGIS table is almost 4 times bigger compared to the LAS file. The e57 file and the LAS file are almost the same size.

Solution	Points (B)	Content	Size (GB)	Avg. byte/point
LAZ	1.556	Original	2.24	1.7
LAS	1.556	Original	30.44	20.0
e57	1.556	Original	10.27	20.5
TXT	1.556	XYZ RGB IC RN	56.79	38.3
PostGIS	1.556	XYZ RGB IC RN	119.0	71.7

Table 7.4: Size Comparison II: Nidarosdomen - 1.556 Billion points.

7.4.1.1 Loading data

Loading data was done using the COPY command in PostgreSQL. This can either be done from the psql command line, from PgAdmin3 or by running the function from a psql script. The first solution that was used for loading data from a text file, consisted of the following three steps:

- Copy data from file
- AddGeometryColumn
- Create Index

Tests were run using these loading steps on the full Holmenkollen data set, which contains 29 Million points. The results are shown in figure 7.13, where the total load time is 37 minutes. Indexing is the most time-consuming part of the operation.

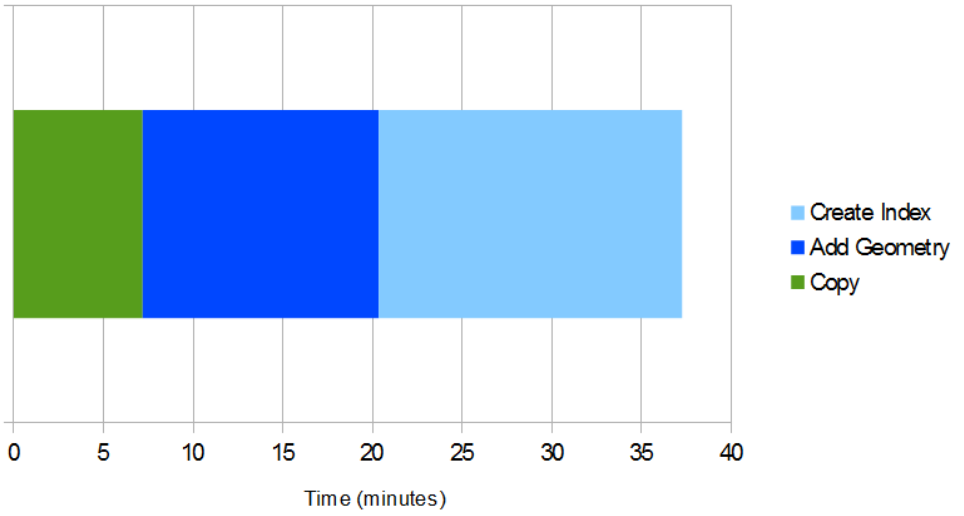


FIGURE 7.13: Holmenkollen loading time consumption: 29 Million points

A more efficient approach would be to create the geometries on the go instead of loading in x, y and z to a table and then generating the Point geometry. This saves us the middle "Add Geometry" part in figure 7.13. This idea was implemented and defined as it's own function on the PostgreSQL server. The load function is called "lidarload" and creates a point data table directly from the text file input:

- Copy data from file and create geometry on the go
- Create index

Unfortunately, no timed tests were not run for this function, but it is clearly a better solution because it gets rid of an extra step.

7.4.1.2 Index Size and Efficiency

The index size of the point tables were tested as well. Indexing was not done for the Nidarosdomen data set simply because it would take too much time. Typically, indexing would add a 50% space requirement to the point data tables. The efficiency of the PostGIS R-tree and GiST combination can be seen in figure 7.14.

For small queries, there is a signficant speedup, but as the query size approaches 1/3 of the data set, the index has little effect.

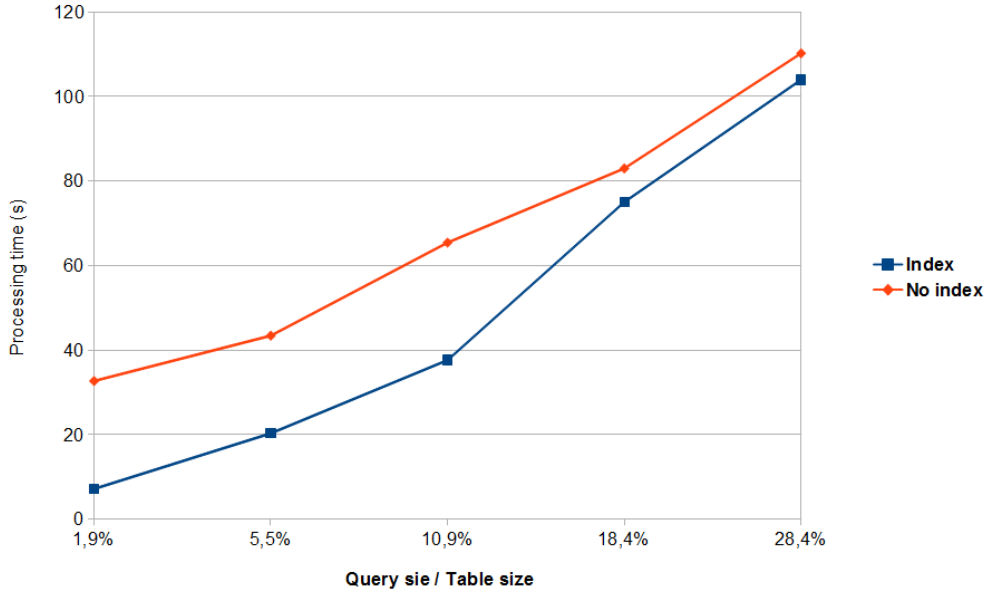


FIGURE 7.14: PostGIS Query efficiency: 29 Million points

As is explained in section, each point will be represented as one geometry in the combined GiST and R-tree index, which makes the indexing scheme extremely inefficient. Because of this, attempts were made to look at indexing schemes more similar to those implemented by RapidLasso and Oracle. One approach was to store data points as MutliPoint geometries instead of points. A PostgreSQL script was created for this purpose. A short, pseudocode representation is shown in listing 7.1.

```
for x = minx; x < xmax; x =+ xstep {

    for y = miny; y < ymax; y =+ ystep {

        collection = Select points in bbox(x, y, x + xstep, y + ystep);

        Insert into newtable (collection);

    }
}
```

LISTING 7.1: Grid function queries

By adjusting the xstep and ystep, it is possible to create different types of bounding boxes. When the GiST command is used on MultiPointZ instead of PointZ, the result is fewer geometries to index. This naturally results in smaller index size. This was done for different step size, as shown in table 7.5.

Table	Points (M)	Content	Table rows	Index size (KB)
H-Standard	9.4	PointZ RGB IC	9 367 818	443 392
H-Grid 5m	9.4	MultiPointZ	39 752	1992
H-Grid 10m	9.4	MultiPointZ	10 000	520
H-Grid 20m	9.4	MultiPointZ	2 500	192
H-Grid 40m	9.4	MultiPointZ	625	32

TABLE 7.5: Index size comparison

From this table we can see that indexing on a few thousand multipoints will improve the index storage efficiency considerably. From these numbers we can also compute number the number of bytes each row adds to an index: Each point amounts to approximately 50 bytes. To see what effect these inedxing schemes would have on the performance on queries, a few tests based on the code in listings 7.2 were run. The four bounding boxes used together constitute an area that contains all 9.4 Million points in the Holmenkollen set used in table 7.4 and visualized in figure 7.7.

```
For i=0; i<4 i++ {

        SELECT geometry FROM H_Table
        WHERE ST_Intersects(multipoint,ST_MakeEnvelope(BBOX i));

}

For i=0; i<4 i++ {

        SELECT id FROM H_Table
        WHERE ST_Intersects(multipoint,ST_MakeEnvelope(BBOX i));

}
```

LISTING 7.2: Grid function queries

The sesults show that some gridding has positive effect on select queries, but only up to a certain point. The 40x40m grid has not got a positive effect on the query processing. It should also be noted that the MultiPointZ data have not gotten any RGB, intensity or classification values. This might have unforeseen effects on the data. Other tests were performed as well, where single points were being picked out. In these tests, the MultiPointZ indexings performed considerably worse, compared to the PointZ index.

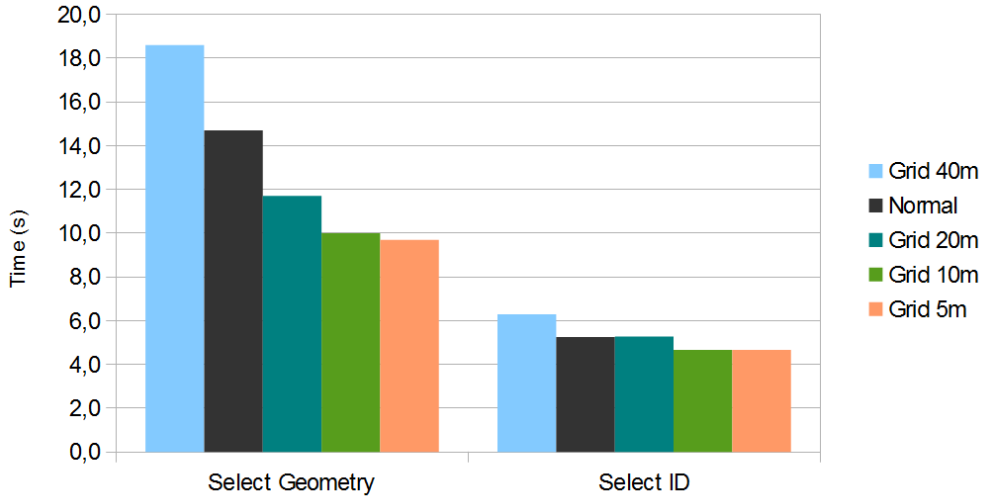


FIGURE 7.15: Index test 1: Full dataset

Table 7.6 shows the result of selecting points within a 10x10m grid in the Holmenkollen data set used in 7.5. The index tests that have been run have shown how the size and speed relates to each other. It is however important to keep in mind that the MultiPointZ tables are smaller because the data tagged to each point has not been included.

Table	Time (ms)
H-Standard	2
H-Grid 5m	6
H-Grid 10m	8
H-Grid 20m	33
H-Grid 40m	112

Table 7.6: Index test 2: 10x10m bounding box

7.4.2 Web browser testing

Because the point cloud viewer runs on the web browser, these test are highly dependent on the efficiency of the web browser. In fact, some browser don't support WebGL properly and cannot run the point viewer at all. The effect of different browser performance can be seen in figure 7.16, where a total number of 2.4 Million points is being loaded into the point viewer. The points are loaded into the browser in four chunks containing approximately 600 000 points each. Interestingly, Chrome starts struggling at 1.8-2 Million points, whereas the FireFox browser keeps on going after that point. The results of the browser tests indicated that memory management could be the issue: FireFox crashed time and time again when approaching 2.4 GB RAM, whereas Chrome did the same around 1.8 GB RAM usage. This result was consistent for different computers. The point record for the prototype was 3.2 Million points, running on FireFox.

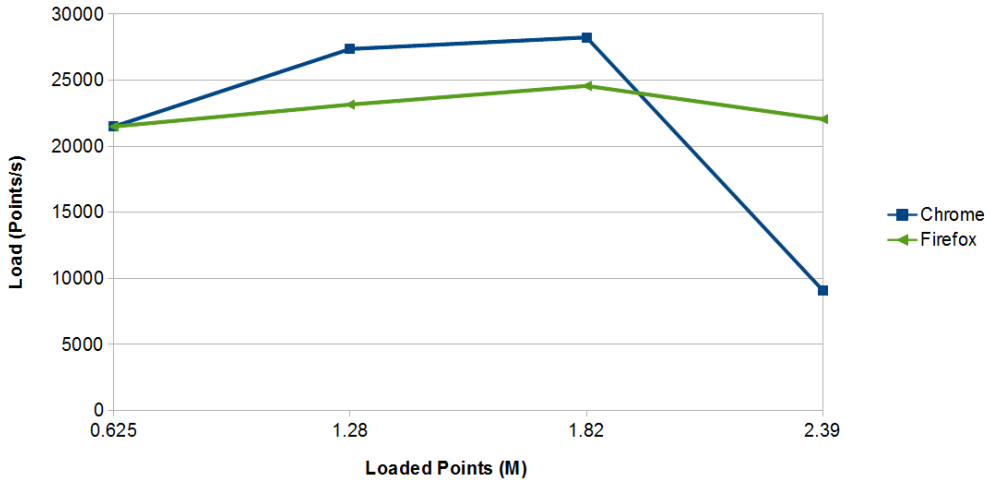


FIGURE 7.16: Browser comparison: Loading 2.4 Million points

This chunking has, through a wide range of testing during the development phase shown to be more efficient than loading everything at once: The PostgreSQL database server uses a certain amount of time for each request, and the bigger or more advanced a request is, the longer the processing time will be. In figure 7.17, a test of how 1.7 Million points are loaded into the browser is shown. The bottom x-axis shows the transfer time from query was sent till the time it had all the points. Each coloured box refers to one data chunk being loaded into the browser. The number of points per chunks are found in the graph explanation. This was tested for a 100 Mbit network and then the 1.5 Mbit transfer time was calculated. "Inifinite bandwidth" is the query time.

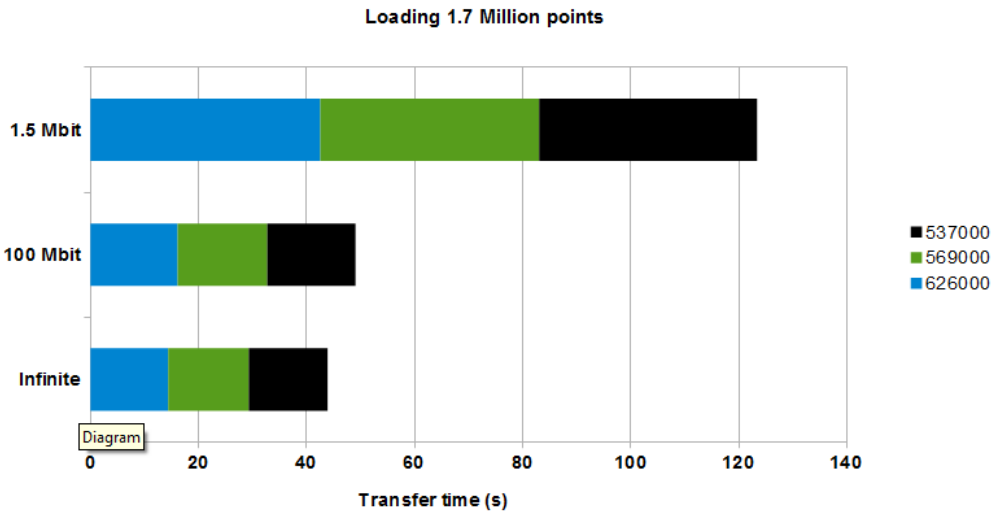


FIGURE 7.17: Bandwidth effects. Loading 1.7 Million points

No have been run on that tests the 3D acceleration via metric such as frames per second. The reason is simply that there has been no need for this. From running the application on different, the general observation has been made that the number of points and browser's memory usage limitations are restrictive long before the actual 3D rendering becomes an issue.

Part IV

Conclusions

Chapter 8

Discussion and Future Work

8.1 Discussion

In Part III, state-of-the-art LiDAR online services were presented together with the thesis prototype. Because any serious attempt at developing a prototype should result in a system that surpasses current solutions or that has some innovative aspects to it, the following is a discussion about whether the proposed LiDAR data warehouse meets these criteria. This will be done by looking at how well the prototype fulfils the system requirements and by comparing it to the state-of-the-art solutions. The tests from section 7 will be used to assess the prototype and will, together with knowledge from part II, help unravel potential shortcomings in the design. Finally, the lessons learned from this thesis will be presented together with suggestions for future work.

8.2 Fulfilling the Requirements

The requirements for the system were stated in the beginning of section 7. These have acted as guidelines during the development of the system and will, in the following paragraphs, be used to assess the end product.

R1: Support storage of georeferenced LiDAR data

Data storage in the prototype is handled by PostgreSQL's spatial extension, PostGIS. While PostGIS is excellent at storing relatively small geographic objects, it is ill-suited for large amounts of point data. The test section is a testament to this fact, seeing as the prototype's storage efficiency is extremely poor compared to existing file-based solutions. A LiDAR data set stored in a PostGIS database table took up 3.5 times the space of a LAS file, and staggering 42 times more than a LAZ file. The inefficiency lies in the data structures used in PostGIS, which have been thoroughly described in section 3.5. However, compared to these early notes, the tests show that the PostGIS database is even less storage efficient than anticipated. Unsuccessful attempts have been made at figuring out how the database storage works on a low level. Index size has also been an issue, although some promising work was done on this area. A reduction of index size by a factor of 200 was attainable using 5x5m grids of MultiPointZ geometries instead of the normal PointZ geometry. Furthermore, there are also positive effects of using PostGIS for LiDAR storage: Queries can easily be run using standardized SQL queries, which also includes a wide range of spatial operators. It is also important to note that there are database solutions that are optimized for LiDAR data and, therefore, storing LiDAR data in a spatial database is not inherently a bad idea.

R2: Enable visualization through the use of Web technologies

The prototype has two types of LiDAR data visualizations. One is based on a traditional web map approach, where the outline of the scans are visualized as polygons. The second, and most interesting, is a 3D point viewer built using a WebGL-based JavaScript library. The point viewer has proved to be capable of loading in 3.2 million points if run on a Mozilla FireFox browser. This number is high enough for the point viewer to visualize large aerial scans or detailed terrestrial scans. For the data sets that exceed this maximum point number, data pruning techniques have been employed to load them in. An advantage of the web-based application is that it runs directly in the browser and requires no extra installation.

The point viewer is also capable of visualizing different properties of the LiDAR point cloud, such as classification values and RGB colours. Using a 100 Mbit line it was shown that the point viewer would build approximately 35 000 points per second. This is transfer rate exceeds the observed performance of the CyArk viewer. Shortcomings of the point viewer is that it does not stream the data, but loads it in piecewise. The infinite case of the the chunk loading described in section 7.4.2 would be a constant stream. This is however more challenging to implement, especially seeing as the Three library doesn't support dynamically changing "ParticleSystem" objects.

R3: Implement user interface for efficient extraction of LiDAR data

The implementation of the data export user interface is described in section 7.3.2. It has not received the same amount of attention as the rest of the system. None of the export data tests were included for the thesis report, as they would have added little insight. Exporting the data itself is simply done by running an SQL query with the export specification and then saving this to a file. Thus, the processing time would be very similar to what has been thoroughly tested in other parts of the systems. The user interface is quite similar to the solutions presented in section 6, but has the extra feature of showing a pruned version of the data set as well.

R4: Use open-source technology where it is applicable

One of the motivations behind the use of open source technology was that it would make the system easy to replicate while simultaneously lowering the costs of development. Finding the open source software packages that would make it possible to develop the system was not a difficult task. All of the JavaScript libraries used in this project are freely available to the public. The same is true for the Apache server and the PostGIS database. The latter, however, has had a detrimental effect on storage performance. From the discussion above and in

section 3.5.1.2, it is evident that the Oracle database would have been a preferable storage solution.

8.3 Lessons learned

While the prototype did not turn out to be the perfect LiDAR data warehouse, it has offered insight into the issues and challenges related to LiDAR data management. This section will present they key observations which has been made during the thesis.

Simplification of Data: Given a globally referenced 3D box, the internal points of that box can be stored with smaller data type such as an integer, if it is accompanied with a scaling factor. This simplification step is an important part of the LAS and e57 file specification, as explained in section 3.2. The result of this trick is that the coordinates of a point can be stored in 12 bytes instead of 24, which would be the case if the coordinates are stored using double precision float data types. The same idea applies to other data types: Limit the data types as much as possible.

Compression is Key: The storage efficiency tests that were carried out in section 7.4, show how important compression is for efficient LiDAR data storage. The LAZ files are 7-25% of the original LAS files, which amounts to substantial amounts of storage space for large data sets. While LAZ files show the best performance, the idea of compression has also been implemented by Oracle. The take-home message is no LiDAR data management system without compression.

The Potential of Web-based GIS: Whereas the backend database of the prototype in no way is competing with current solutions for LiDAR storage, the front end web client is an example of an application that introduces something new: It is capable of retrieving point data, TINs, CAD models and more. It makes the whole concept of georeferenced data a lot more exciting, as it comes to live in the

3D viewer. That this can all be done from a browser opens up a new world of opportunities for GIS.

8.4 Future Work

The objective of developing a LiDAR data warehouse is not an easy one. Because knowledge about LiDAR and geographic information storage is required, in addition to being capable of using several different programming techniques, the system naturally will have some shortcomings. This section gives topics related to the LiDAR data warehouse that requires further work in order to create a better solution. Finally, a slightly visionary and interesting research topic is presented.

Use a LiDAR-optimized database: Because the storage solution of the prototype is inefficient compared to file-based solutions, it becomes difficult to argue for choosing the database option. However, by using a spatially enabled Oracle database or an equivalent solution, the benefits of database management won't come at a high price.

Dynamic visualization: The Web-based point cloud visualization implemented for the prototype uses a relatively rigid point selection algorithm. The point viewer itself has however been built in an object-oriented manner, where a chunk of points of any size can be loaded in an visualized according to it's coordinates. This opens up for a more dynamic way of getting points. As an example, querying for point clouds using a road section presented as a polygon would make it possible to get a denser point cloud surround the road and sparser coverage in the round lying areas.

Full waveform LiDAR: A topic which was barely touched upon in section 2, was full waveform LiDAR. Getting easy access to large amounts of full waveform data through something like a spatial database would be interesting, as different algorithms for automatic detection or improved precision through better waveform analysis could be the result. These data are stored in LAS 1.3 and 1.4 as byte

strings that represent the actual waveform. These have a starting point and length, meaning there is a three dimensional spatial extent when the scan angle and scan system position is taken into account. This makes it possible to store as a spatial object, and how that should be done in a database context would be interesting to look closer at.

Virtual reality: An exciting idea that should be investigated regarding LiDAR data, is the visualization of LiDAR data on smart phones or other portable devices. In Appendix A, a screen shot from a cellphone running the point cloud viewer is shown in figure A.5. This was an early version of the point cloud viewer, but still rendered over half a Million points on a 2 year old smart phone. By creating an application that tracks the camera and orientation change, it would be possible to create a virtual reality viewer of the point cloud. With recent product launches such as Google Glass, this might be an interesting idea for the future.

Chapter 9

Conclusion

9.1 Conclusion

This thesis has investigated how a data warehouse could be designed in order to facilitate efficient management of data produced by Light Detection and Ranging (LiDAR) scanners. This has been done by gaining an understanding of the LiDAR data structures and how these can be stored and managed. Furthermore, a LiDAR data warehouse has been developed, tested and compared to existing solutions. The results of this study indicate that the chosen approach of using a PostGIS database for point cloud storage is suboptimal due to poor storage efficiency. The data organization that causes this to be the case has been explained in section 3.5, thoroughly tested in 7.4 and finally analysed in chapter 8. The conclusion is that the PostGIS table implementation, which has a smaller amount of information than the LAS file, takes up 3.5 more storage space. Moving on to LAZ files, the worst case scenario is that the PostGIS table takes up 42 times more storage space. Therefore, the finding of this thesis indicates that for maximized storage efficiency, LAZ files should be used for storing LiDAR files.

Accessibility of the LiDAR data is another issue that has been thoroughly investigated during the prototype development. Current online LiDAR solutions mainly use web maps to visualize the extent of the data sets. This was easily implemented

with the PostGIS database in combination with a Leaflet-based web site, but the prototype also offers three dimensional visualization of the point clouds. The idea of creating a Web application to increase LiDAR data accessibility has been influenced by CyArk's web viewer, which was developed to visualize laser scans of heritage sites. A similar web viewer has been developed for this prototype and has shown promising test results. Up to 3.2 Million points have successfully been visualized in the Web application and, with no required plug-ins, looking at LiDAR data now requires the same amount of effort as reading the online news.

Since the first commercial LiDAR surveys were carried out less than 20 years ago, LiDAR technology has grown to become an industry of it's own. While aerial scans and construction site surveys are typical uses of LiDAR scanners today, this might change in the future. Exciting technology such as autonomous cars and drones might cause an increase in the number of sensory units, which would lead to even more data being gathered. Managing these data and making them easily accessible should be a primary concern of LiDAR data users. Countries such as Sweden, Denmark and the US have recognized the value of making these point clouds available online, for anyone to download. Hopefully this trend will continue, in which case the lessons learned from this thesis could provide insight into making an optimized warehouse for LiDAR data storage.

Part V

Appendices

Appendix A

Prototype Development

A.1 Developing Cloudy

Developing the LiDAR data warehouse started off after the initial planning phase was finished. Using an iterative development process, it has been possible to feed the knowledge from each iteration step into the next one. Thus, the system has improved drastically over the course of this master. The largest revisions of the prototype will be presented in this appendix chapter.

Iteration 1: Getting started

The initial version of Cloudy was finished in late February. For the initial version of the system, the Holmenkollen data set was used. At this point the system included the following software packages:

- Leaflet.js
- Three.js
- Simple HTML/CSS
- PostgreSQL + PostGIS

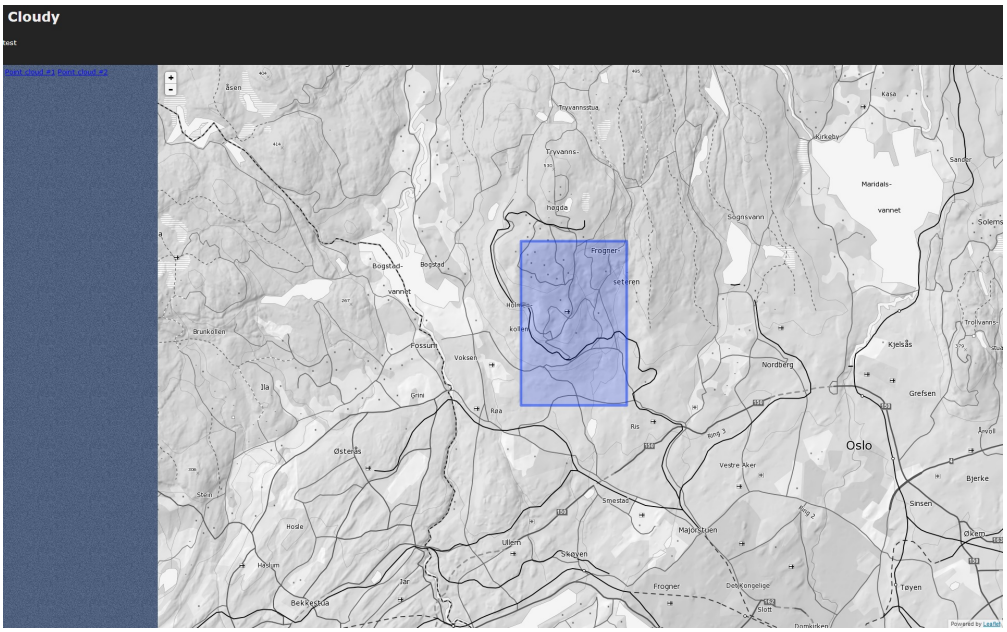


FIGURE A.1: LiDAR version 1

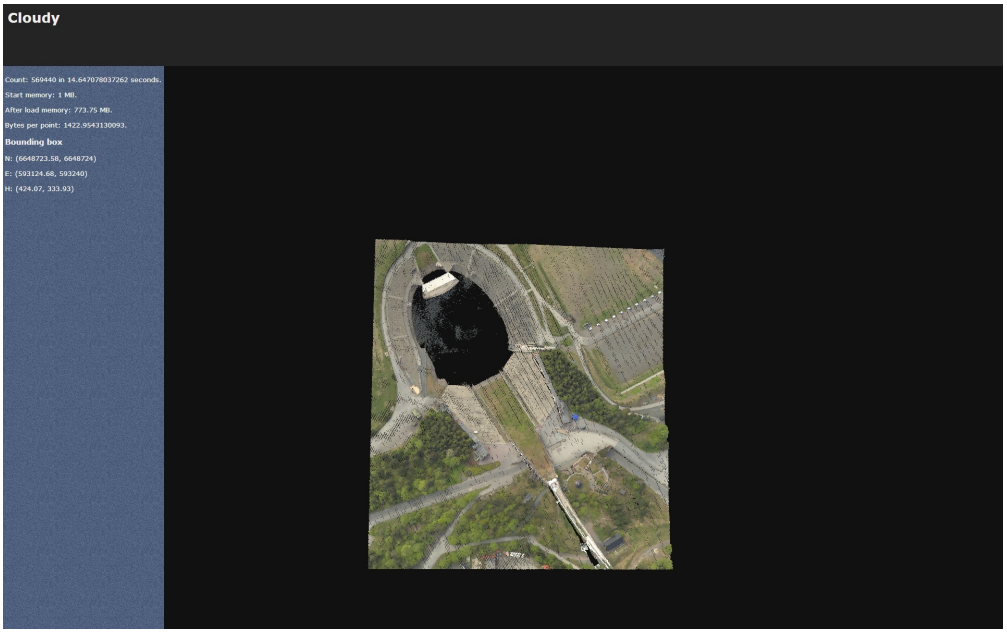


FIGURE A.2: LiDAR point viewer version 1

Iteration 2: Additional Scans and Bootstrap

The second revision is shown in figure A.3. It included the data set from Nidaros Domkirkes Restaureringsarbeider (NDR), which was an indoor scan of the Nidaros cathedral in Trondheim. Sadly however, colour information was not present for this scan. Another indoor scan, captured with a Faro 3D scanner, was also added to the database. A notable change in the Cloudy system is the use of bootstrap.js, which makes it easer to build a neat looking and User Interface (UI). It was however not implemented for the point viewer at this point. Out of curiosity, a mobile test was run as well, on a Samsung Galaxy S2 with the Google Chrome browser set to development mode. It actually rendered a point cloud of 568 000 points. This can be seen in figure A.5 Summarized, the changes were:

- Bootstrap.js
- New data sets: Indoor scans

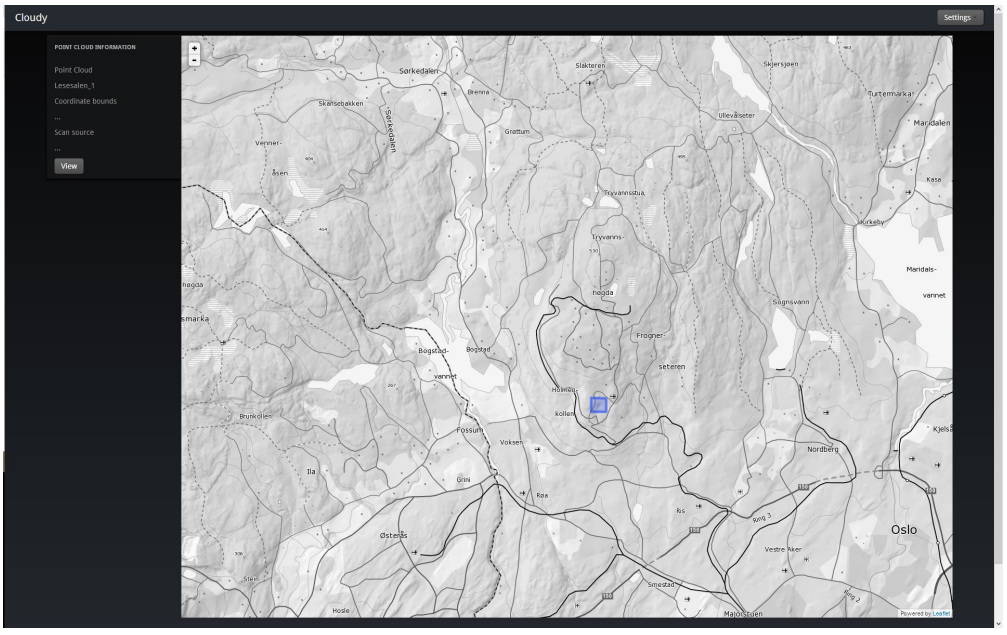


FIGURE A.3: LiDAR version 2

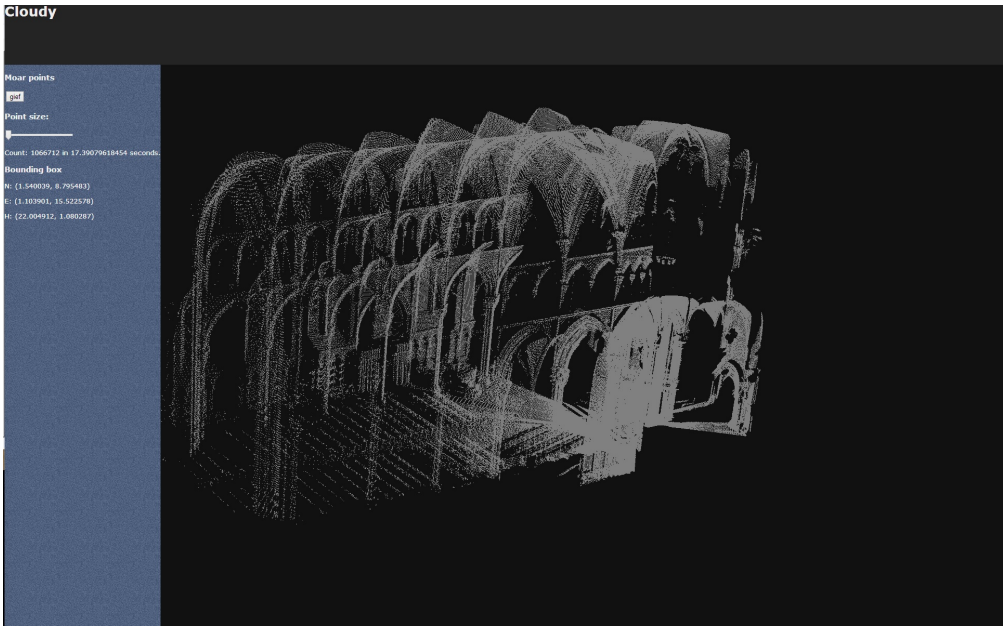


FIGURE A.4: LiDAR point viewer version 2

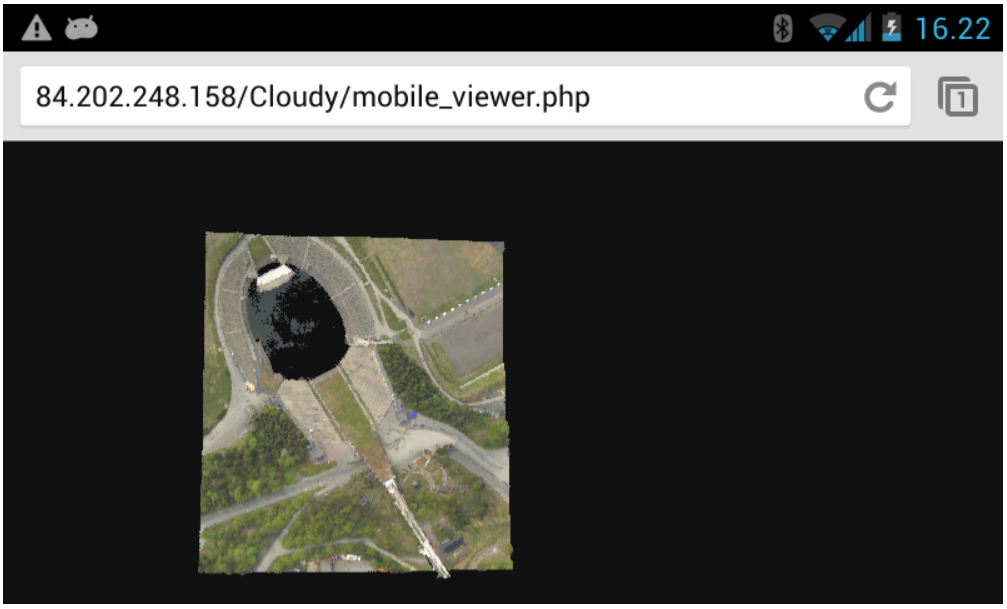


FIGURE A.5: Cloudy version 2: Cellphone

Iteration 3: New Look and Improved Controls

For the third iteration, the look was changed to create a better user experience. The controls for the point cloud viewer were also significantly improved, as they up until this point had simply been changing camera position based on mouse position. Controls were now implemented using a system working in the following manner:

Notable additions:

- New Look
- Improved user interface
- Export functionality

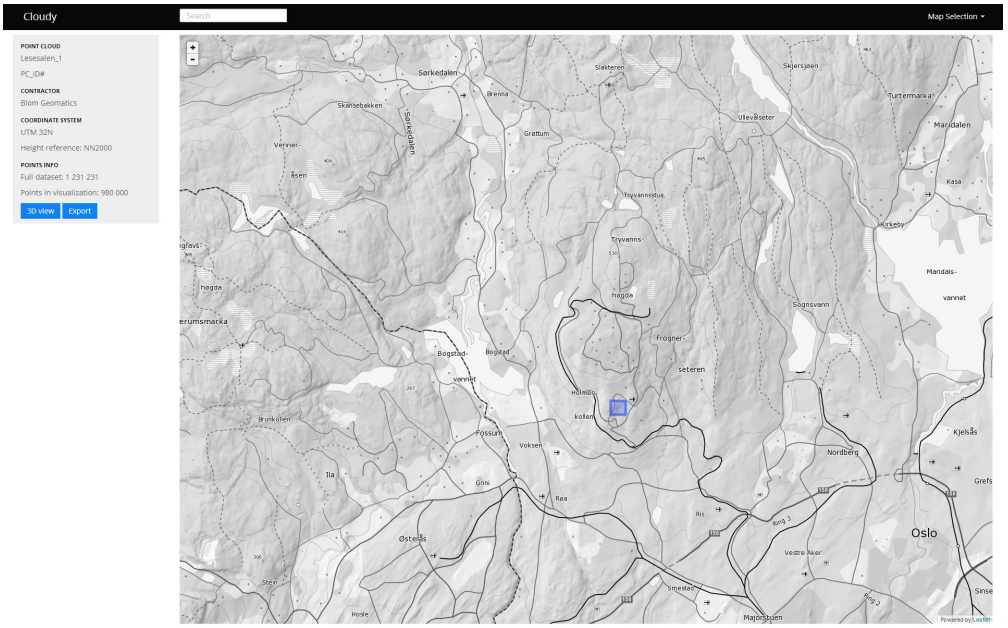


FIGURE A.6: LiDAR version 3

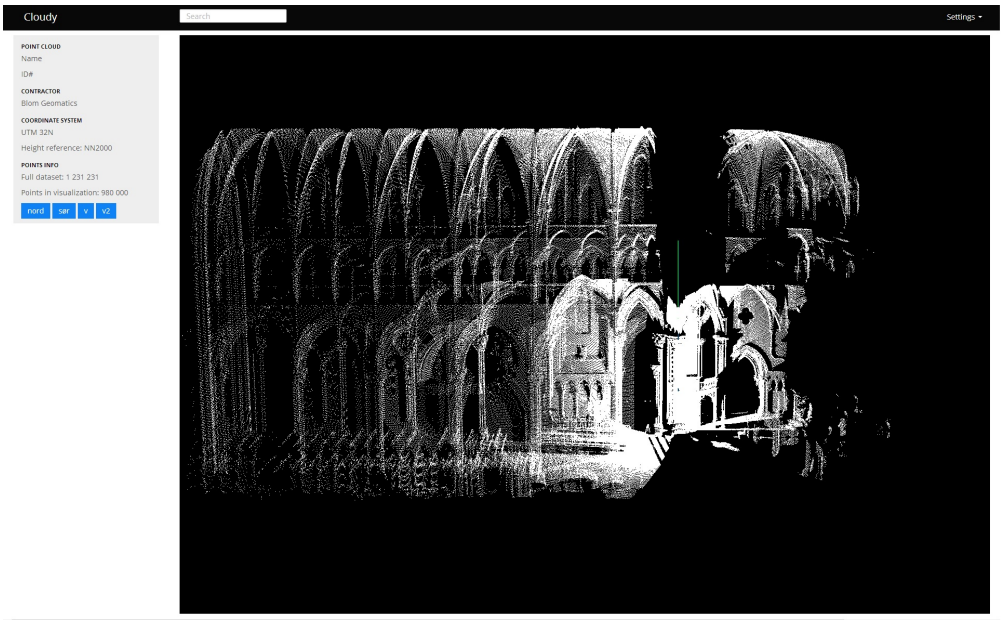


FIGURE A.7: LiDAR point viewer version 3

Iteration 4: Final Delivery

Cleaning up the code and adding final touches, such as a loading bar which indicates when the system is working. For the final versions, the point cloud chunks have been finalized with call functions for colouring based on different variables. Figures A.9 and A.8 shows what height and intensity colouring looks like.

- Progress bar
- Object-oriented Particle Systems

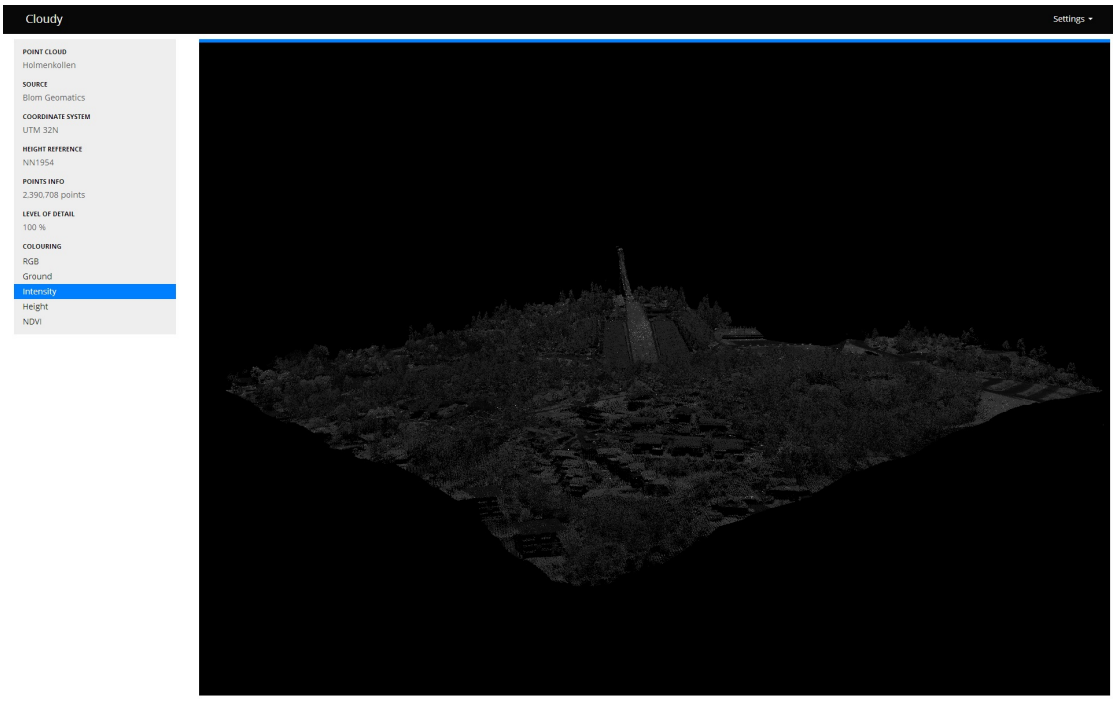


FIGURE A.8: LiDAR point viewer version 4

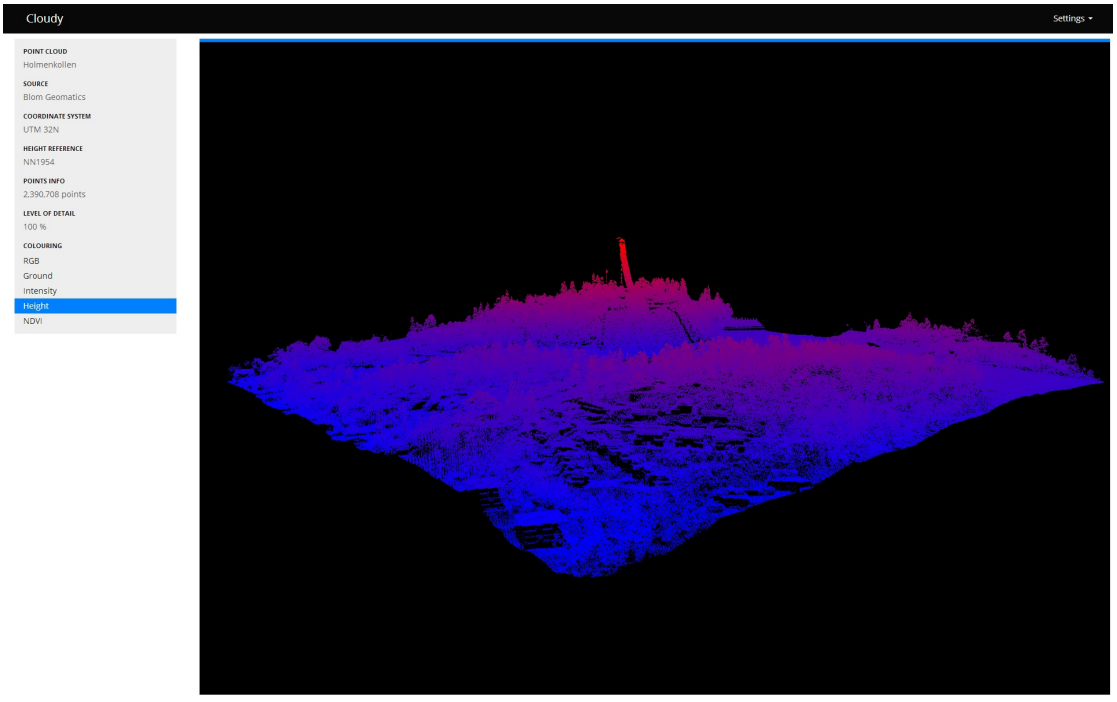


FIGURE A.9: LiDAR point viewer version 4

Appendix B

Prototype Experimentation

B.1 Extra Functionality

Creating a LiDAR data warehouse prototype lead to many hours of research and development, something which also spawned creative processes that lead to solutions that not necessarily are within the project's development goals. These solutions are still interesting however, as they show how potent the combination of the spatial database and web GIS is. Two such spin-off projects will be presented here: One section will present derived 3D model visualization and analysis. The other section will explain how Normalized Differentiated Vegetation Index (NDVI) can be implemented and run as a service on the Web Client itself.

B.2 3D Models From Point Clouds

RapidLasso's command line tools, LasTools, includes functionality for creating Triangular Irregular Networks from las files. Using this program, combined with the selection of ground-classified points from the Holmenkollen data set, a bare-earth DEM was created. The output format of the program is OBJ, which is simply a list of points and triangle definitions. This data set was previously used as an example of a bare-earth mode in figure 2.7. In that figure, ArcScene was

used for visualization. However, it turned out that it was easy to get hold of a OBJ loader to the three.js library, which made it easy to import and visualize this data set in the point cloud viewer. A result of this is seen in figure B.1, where a Collada model of the Empire State building also was added by using a 3D model loader library.

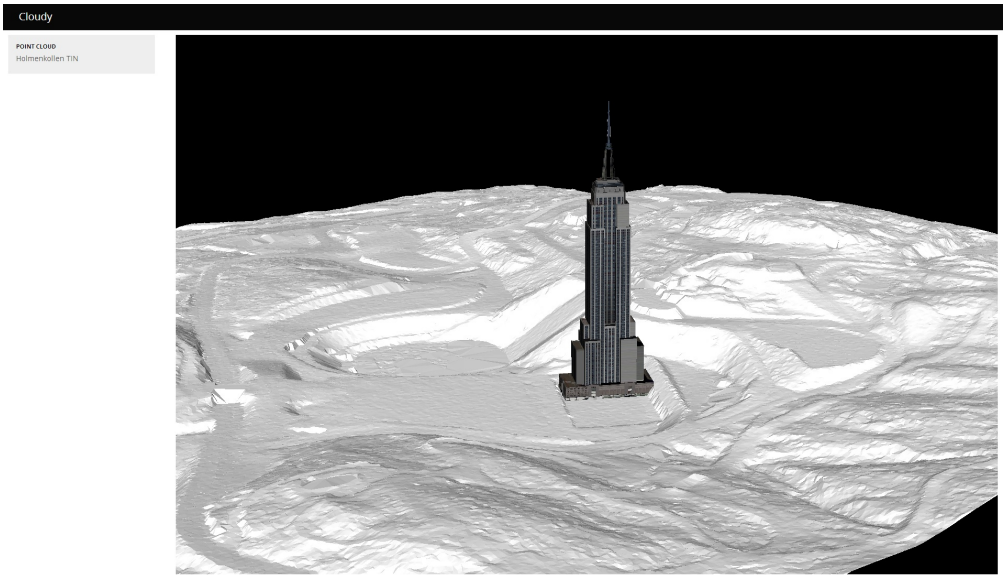


FIGURE B.1: Cloudy TIN and CAD: DTM from Holmenkollen and a CAD file loaded into the Cloduy point viewer

B.2.1 3D Analysis

As soon as we are working with surface models, it is easy to extend the functionality to do 3D analysis. Simple operations such as doing volumetric measurements could be thought of. Another type of analysis is line of sight and shadow analysis for planned buildings. The latter was easily implemented by constructing a box, adding it to the scene and letting it cast shadow onto the DTM. The beauty of a web application such as Cloudy is that as long as it has unambiguous coordinates, everything can easily be added to the scene. The true potential of georeferenced data is by no means fully utilized by current applications.

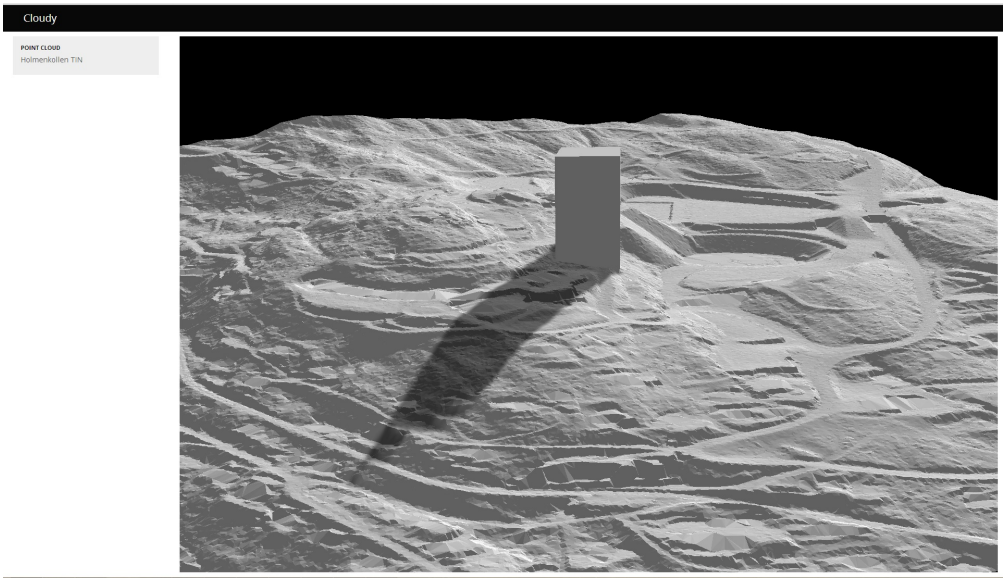


FIGURE B.2: DTM Shadow Analysis using a box model to represent a tall building

B.3 Real-Time Processing: NDVI

The Normalized Differentiated Vegetation Index (NDVI) is a technique for detecting chlorophyll-based organizations among other objects. It is an integral part of mapping vegetation coverage from satellite and aerial photography. NDVI provides an easy way to see growth and decline in a bio mass, and will in fact detect a decline in the productivity of a plant's photosynthesis before it can be detected in leaves' colour. This is because the NDVI is based on the fact that leaves and other chlorophyll-based biomass absorbs a lot of red light and reflects a lot of the infra-red light[47]. This is a property almost no other objects exhibit, and means that it can be used for identification purposes if both red colour channel and Near-Infra-Red light (NIR) are available. The NDVI index is defined in equation B.1.

$$NDVI = \frac{VIS - NIR}{VIS + NIR} \tag{B.1}$$

Interestingly, as explained in section 2, most LiDAR scanners are based on NIR laser. This means that one of the prerequisites for creating a NDVI index is naturally present in LiDAR data sets. If the RGB values have been matched to the point cloud as well, everything is in place for a NDVI analysis. There are few publications on the topic of LiDAR NDVI, and it is not commonly used in the industry, but still a feasible option for vegetation detection[48]. The Holmenkollen data set is shot with a 1064nm laser and has RGB attached to it, which made it applicable for a NDVI test. The result is shown in figure B.3. Bear in mind that this is not the standard NDVI index colouring scheme, but a simple red-blue gradient.

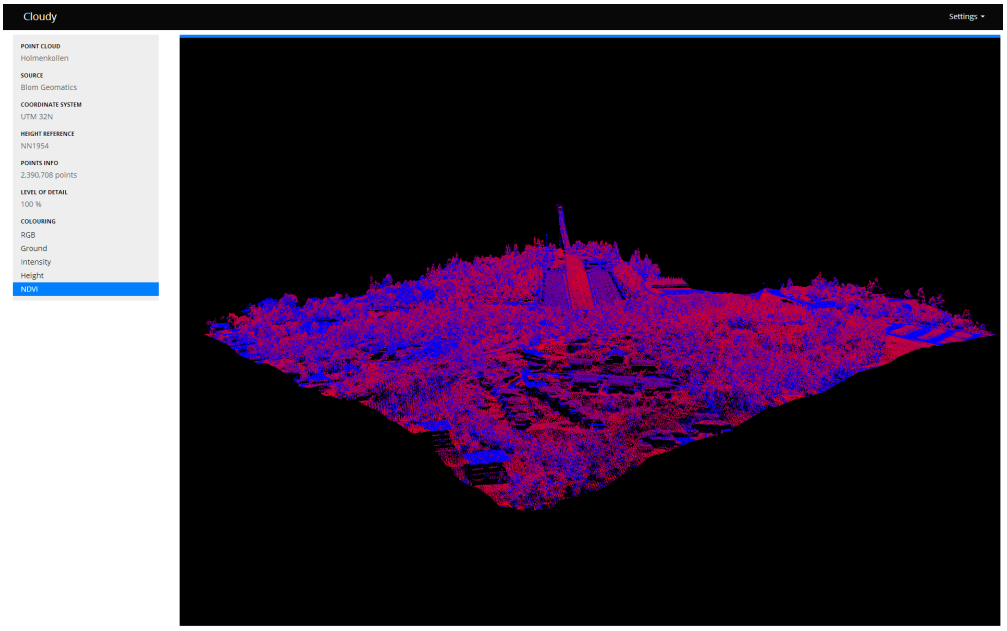


FIGURE B.3: Holmenkollen NDVI

Appendix C

Hardware Comparison

C.1 Motivation for testing

Almost all the of the development of the prototype has been done using a Windows 8 system installed on an Intel 520 240 GB storage device. The Intel 520 is a Solid-State Device, which means that it utilizes NAND flash cells to store information, as oppposed to a Hard Disk Drive (HDD), which reads and writes bits to magnetized metal disks[15]. Traditionally, SSDs have been expensive and high-performing storage mediums. The random data access has been particularly impressive on these devices, and was the main selling point on earlier devices citexxx. In fact, five years back the SSD and HDD would have comparable sequential read performance, and the random read performance alone could often not justify the high price. Today, the situation is completely different, as relatively cheap SSDs with impressive performance and high capacity can are available. Whereas many companies have adopted the SSD, it is still expensive in terms of GB per dollar. Seeing as the LiDAR industry requires a lot of GigaBytes, they are likely to do this change at a later stage. This section simply shows how big the performance difference is between a HDD and a SSD.

Test machine	SSD	HDD
Hardware		
RAM CPU Storage	8 GB DDR3 Intel i7-720QM Intel 520 240 GB	8 GB DDR3 Intel i7-720QM Seagate Momentus XT 500GB 7200RPM
Software		
OS Database Spatial Extension	Windows 8 PostgreSQL 9.2.2 PostGIS 2.0	Ubuntu 12.04 PostgreSQL 9.2.2 PostGIS 2.0

Table C.1: Storage Test Machines

C.2 Tests

In order to see the performance difference in using a SSD compared to a traditional magnetic disks for spatial queries on a PostGIS database, four tests were constructed and run on two systems with the specifications listed in table C.1. As can be guessed from this table, the only difference in the hardware was the storage solution and OS. The database tables were identical on both systems and both had been clustered on a GiST index prior to testing. While every measure was taken to ensure equal test set up, no data has been gathered on performance difference for PostGIS running on Ubuntu compared to running on Windows. The results are therefore more of an indication than a correct result.

C.2.1 Test 1

Test 1 is the control query, which simply performs a data table scan for 200 000 rows. This means no spatial indices are being used, the data records are simply retrieved from disk.

```
SELECT * FROM holmenkollen LIMIT 200000;
```

Listing C.1: Test 1 query

C.2.2 Test 2

Test 2 performs four queries, as shown in listings C.2. The result of this query is the data set shown in figure C.1. Summed up the query returns roughly 2,39 Million rows.

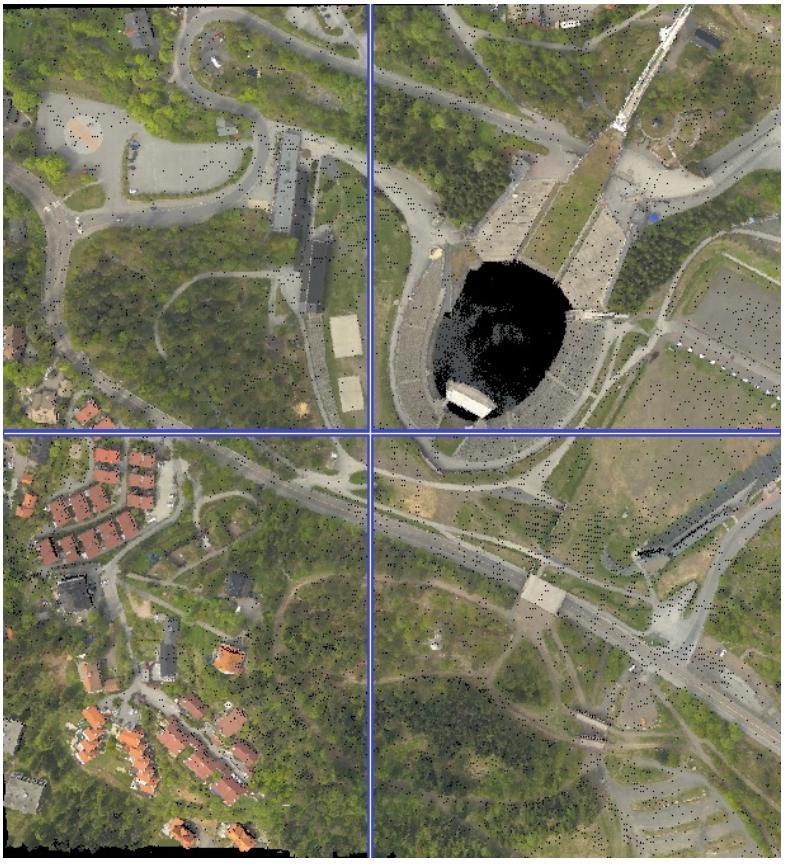


FIGURE C.1: Test 2 query result

```
SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(592960, 6648724, 593240, 6648474, 32632));

SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(593240, 6648724, 593520, 6648474, 32632));

SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(592960, 6648224, 593240, 6648474, 32632));

SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(593240, 6648224, 593520, 6648474, 32632));
```

LISTING C.2: Test 2 queries

C.2.3 Test 3

Test 3 queries consists of four separate queries that access four separate, squares of 30x30 meters. Queries can be seen below in listing C.3.

```
SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(592735, 6648735, 592765, 6648765, 32632));

SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(593235, 6648735, 593265, 6648765, 32632));

SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(592735, 6648265, 592765, 6648235, 32632));

SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(593235, 6648265, 593265, 6648235, 32632));
```

Listing C.3: Test 3 queries

C.2.4 Test 4

Test 4 queries, shown in listings C.3, consists of two stripes that are orthogonal to eachother. They are have the dimensions 30x970 meters and 30x1000m.

```
SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(593235, 6648265, 592265, 6648235, 32632));

SELECT id, ST_x(point), ST_y(point), ST_z(point), r, g, b, i, c FROM holmenkollen
WHERE ST_Intersects(point,ST_MakeEnvelope(593235, 6648000, 593265, 6649000, 32632));
```

Listing C.4: Test 4 queries

C.3 Results

The test results shows that the Cloudy prototype is well served with being developed using a SSD-based system. The throughput for all queries are approximately an order of magnitude faster on the SSD-based than on the HDD-based system.

Surprisingly, there is not much difference in the relative performance gain with respect to different access patterns. In fact, there is a slight tendency towards the SSD being even faster in the case of a database scan query.

Test	HDD	SSD
1	32.28	1.71
2	285.35	23.57
3	4.60	0.36
4	54.14	4.27

TABLE C.2: Test: Time spend (seconds)

In figure C.2, the throughput in rows per second is shown for the SSD in green and HDD in blue. The SSD returns 100-120 000 rows per second, while the HDD delivers a 6000-8000 rows per second. Had a HDD-based system been used to run the Cloudy prototype, the system would have been considerably slower.

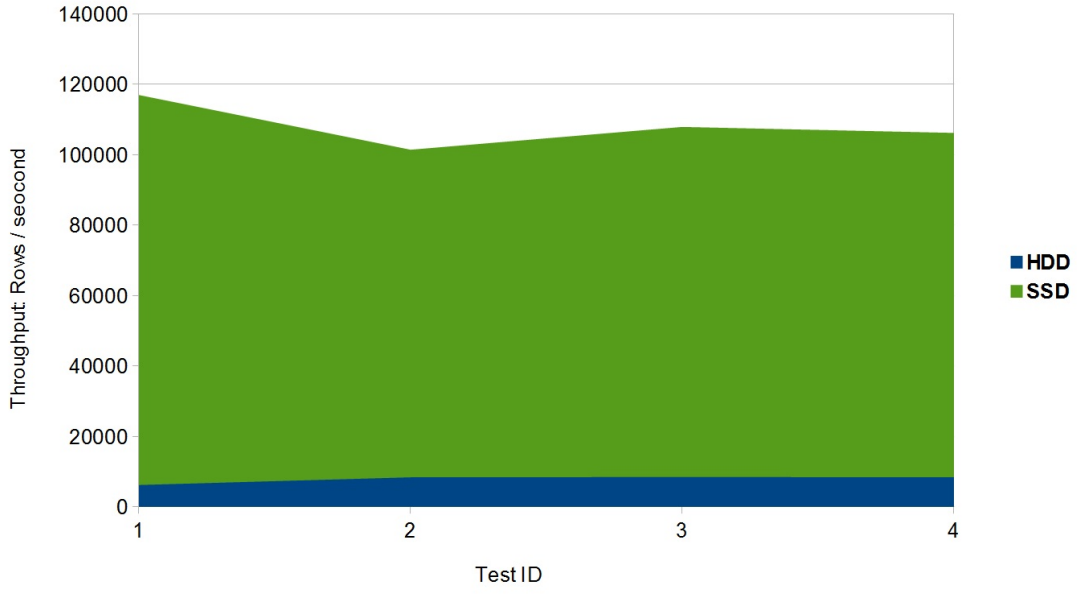


FIGURE C.2: Test: Throughput

Appendix D

Sequence Diagrams

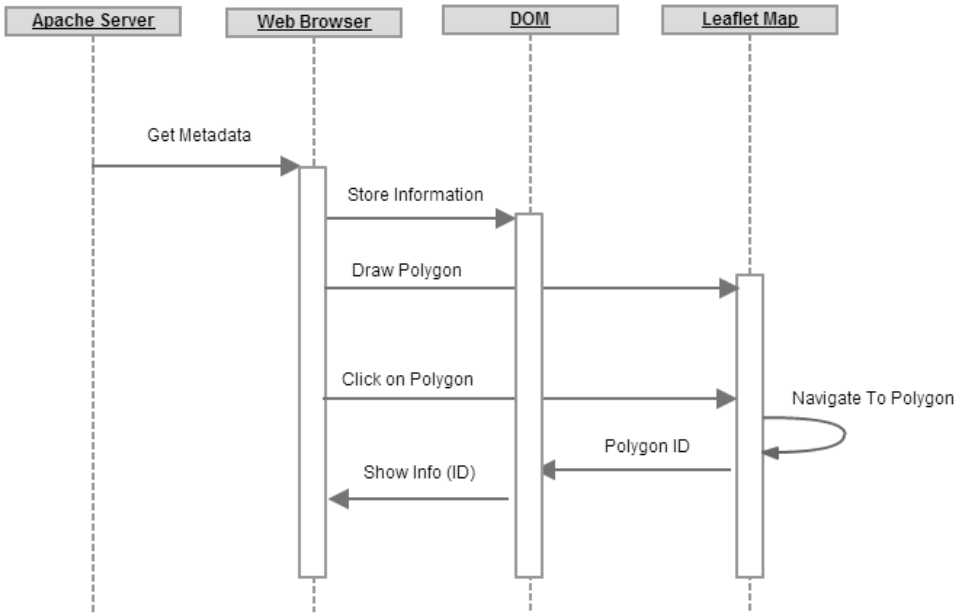


FIGURE D.1: Web Map Sequence Diagram

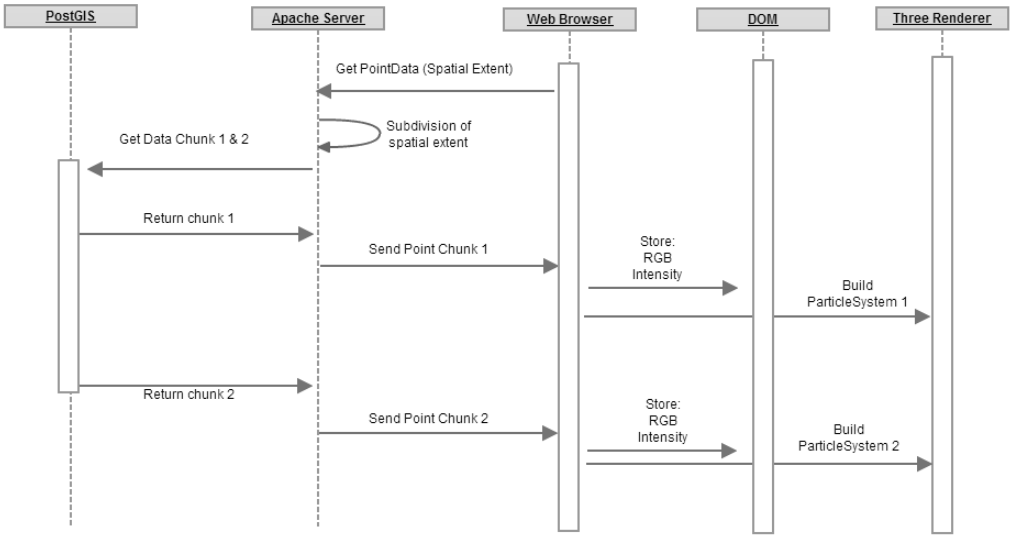


FIGURE D.2: Browser Point Cloud Creation

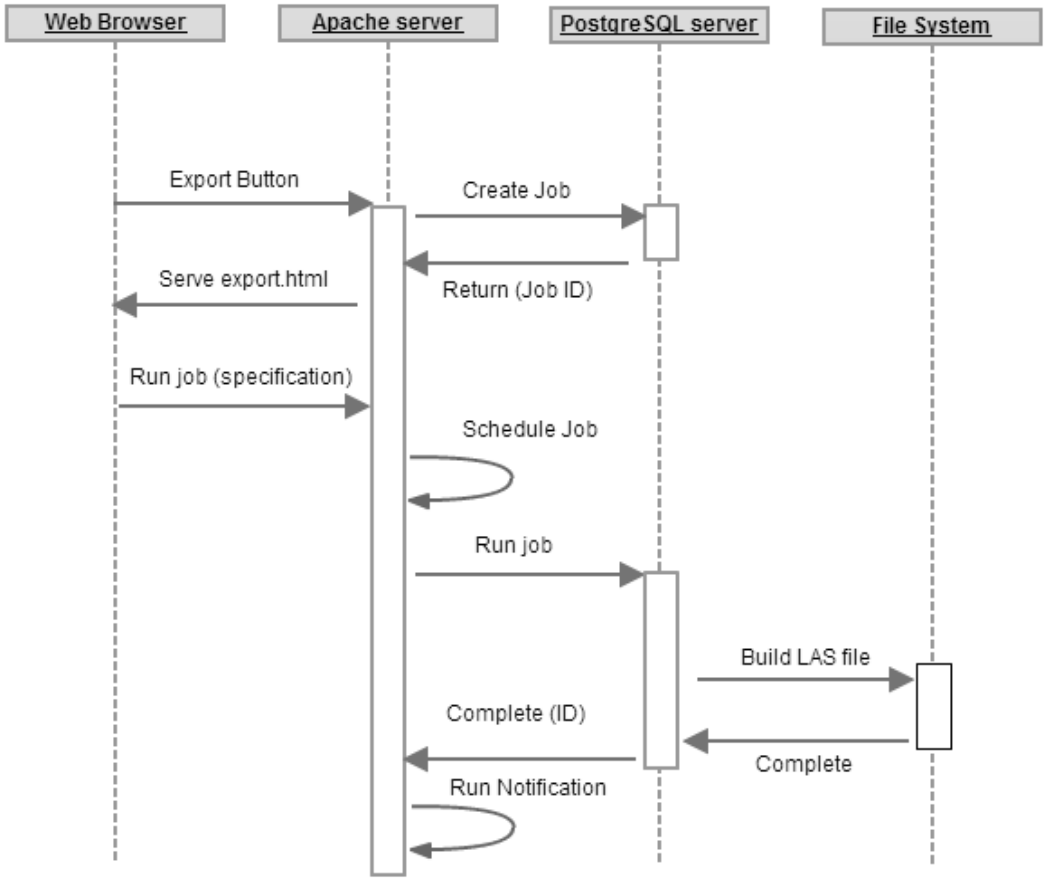


FIGURE D.3: Export Sequence Diagram

Bibliography

[1] M. Flood. Lidar activities and research priorities in the commercial sector. *International Archives of Photogrammetry and Remote Sensing*, 2001.

[2] P. A. Forrester and K. F. Hulme. Review: Laser rangefinders. *Optical and Quantum Electronics*, 1981.

[3] G.C. Guenther. Airborne laser hydrography. Technical report, US Department of Commerce, Washington, DC, 1985.

[4] Banic J. Sizgoric, S. and G. C. Guenther. 1970-1990: Airborne lidar hydrography status. *EARSeL Advances in Remote Sensing*, 1992.

[5] Irish J.L. and White T.E. Coastal engineering applications of high-resolution lidar bathymetry. *Coastal Engineering*, 1998.

[6] Mallet C. and Bretar F. Full-waveform topographic lidar: State-of-the-art. *Journal of Photogrammetry and Remote Sensing*, 64(1):1–16, 2009.

[7] U.S. Government. *A Guide to Lidar Data Acquisition and Processing for the Forests of the Pacific Northwest*. General Books, 2011.

[8] F. Bretar A, A. Chauve A, C. Mallet A, and B. Jutzi B. Managing full waveform lidar data: A challenging task for the forthcoming years. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII. Part B1. Beijing 2008*.

[9] Arlen F. Chase, Diane Z. Chase, John F. Weishampel, Jason B. Drake, Ramesh L. Shrestha, K. Clint Slatton, Jaime J. Awe, and William E. Carter.

Airborne lidar, archaeology, and the ancient maya landscape at caracol, belize. *Journal of Archaeological Science*, 38(2):387 – 398, 2011.

[10] Marko Pejić. Design and optimisation of laser scanning for tunnels geometry inspection. *Tunnelling and Underground Space Technology*, 37(0):199 – 206, 2013.

[11] Y. B. et. al. Kumar. Development of lidar technologies under project lidar. *Ineternational Journal of Advanced Engineering Sciences and Technologies*, 2011.

[12] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. In *Proceedings of the 2008 conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 363 – 371, 2008.

[13] A. Bachrach, A. de Winter, Ruijie He, G. Hemann, S. Prentice, and N. Roy. Range - robust autonomous navigation in gps-denied environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1096–1097, 2010.

[14] National Geospatial-Intelligence Agency. Nga standardization document light detection and ranging (lidar) sensor model supporting precise geopositioning. Technical report, 2011.

[15] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. 3 edition, 2003.

[16] Mike Mesnier, Gregory R Ganger, and Erik Riedel. Object-based storage. *Communications Magazine, IEEE*, 41(8):84–90, 2003.

[17] The American Society for Photogrammetry and Remote Sensing. Asprs lidar data exchange format standard version 1.0, 2003. URL http://asprs.org/a/society/committees/standards/asprs_las_format_v10.pdf.

[18] The American Society for Photogrammetry and Remote Sensing. Las specification version 1.4 - r12, 2012. URL http://asprs.org/a/society/committees/standards/LAS_1_4_r12.pdf.

[19] The American Society for Photogrammetry and Remote Sensing. Las specification version 1.1, 2005. URL http://www.asprs.org/a/society/committees/standards/asprs_las_format_v11.pdf.

[20] The American Society for Photogrammetry and Remote Sensing. Las specification version 1.2, 2008. URL http://asprs.org/a/society/committees/standards/asprs_las_format_v10.pdf.

[21] The American Society for Photogrammetry and Remote Sensing. Las specification version 1.3 – r11, 2010. URL http://asprs.org/a/society/committees/standards/LAS_1_3_r11.pdf.

[22] M. Isenburg. Laszip: lossless compression of lidar data, 2012.

[23] D. Huber. The astm e57 file format for 3d imaging data exchange. pages 78640A–78640A–9, 2011.

[24] Robert W. Bemer. A proposal for character code compatibility. *Communications of the ACM*, 3(2):71–72, 1960.

[25] Gary Marsden and David E. Cairns. Improving the usability of the hierarchical file system. In *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, SAICSIT '03, pages 122–129, 2003.

[26] *GIS: A Computing Perspective, 2nd Edition*. CRC Press, Inc., 2004.

[27] *Readings in database systems*. The MIT Press, 2005.

[28] Open GIS Consortium. Opengis implementation standard for geographic information - simple feature access - part 2: Sql option, 2010.

[29] Horhammer M. Ravada S. and Kazar B. M. Point cloud: Storage, loading, and visualization.

[30] M. D. Smith and D. C. Finnegan. Visualization, analysis and management of 3d lidar topography in oracle spatial 11g, 2009. URL download.oracle.com/otndocs/products/spatial/pdf/osuc2009_presentations/osuc2009_usace_smith.pdf.

[31] Douglas Comer. Ubiquitous b-tree. *ACM ACM Computing Surveys*, 11(2): 121–137, 1979.

[32] Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 500–509, 1994.

[33] A. Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, 1984.

[34] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In *IN PROC. 21 ST INTERNATIONAL CONFERENCE ON VLDB*, pages 562–573, 1995.

[35] M. Isenburg. Lasindex - simple spatial indexing of lidar data, 2012. URL http://www.youtube.com/watch?v=FMcBywhPgdg.

[36] T Edwin Chow. The potential of maps apis for internet gis applications. *Transactions in GIS*, 12(2):179–191, 2008.

[37] Matthew B. Hoy. Html5: A new standard for the web. *Medical Reference Services Quarterly*, 30(1):50–55, 2011.

[38] *Pro HTML5 Programming (Powerful APIs for Richer Internet Application Development)*. Apress, 2010.

[39] Johan van Wamelen and Dennis de Kool. Web 2.0: a basis for the second society? In *Proceedings of the 2nd international conference on Theory and practice of electronic governance*, pages 349–354, 2008.

[40] *JavaScript: The Good Parts*. O'Reilly, 2008.

[41] Khronos Group. Webgl specification version 1.0.2, 2013. URL https://www.khronos.org/registry/webgl/specs/1.0/.

[42] Viswanath Nandigam, Chaitan Baru, and Christopher Crosby. Database design for high-resolution lidar topography data. In *Scientific and Statistical Database Management*.

[43] Sriram et. al. Krishnan. Opentopography: A services oriented architecture for community access to lidar topography. In *Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications*, COM.Geo '11, pages 7:1–7:8, 2011.

[44] C. Larman and V.R. Basili. Iterative and incremental developments. a brief history. *Computer*, 36(6):47–56, 2003.

[45] Paul Ramsey. Postgis point clouds source code, 2013. URL https://github.com/pramsey/pointcloud.

[46] Ralph Kimball et al. The data warehouse. *Toolkit. John Wiley*, 1996.

[47] R. S. Defries and J. R. G. Townshend. Ndvi-derived land cover classifications at a global scale. *International Journal of Remote Sensing*, 15(17):3567–3586, 1994.

[48] A. M. Griffin. Using lidar and normalized difference vegetation index to remotely determine lai and percent canopy cover at varying scales.