Msc. Thesis, submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Visualization on a Budget for Massive LiDAR Point Clouds

Berend Wouda
kirk@kirkwarez.com
1100726

August 1, 2011

Computer Graphics and CAD/CAM Group, Department of Mediamatics, Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology

Supervisors: Dr.Ir. Gerwin de Haan, Ir. Frits Post

TU Delft
Abstract

The recent emergence of LiDAR scanning technology has resulted in the availability of very large three dimensional point cloud data sets. An example of such a data set is the AHN2 data set, which contains a high-density point cloud for the Netherlands. The direct visualization of these point cloud data sets is important because it provides the first step in understanding them. However, the size of these data sets prevents them from simply being displayed, let alone from being explored interactively. As such, they present a challenge in visualization.

In this thesis, we describe the methods used by earlier works to beat this challenge. These methods involve the use of space driven spatial data structures to provide efficient out-of-core access to the point cloud data set, and the alleviation of one or more of the bottlenecks in the point cloud visualization pipeline to improve interactivity.

Choosing to use a different approach, we present a method to interactively visualize these point cloud data sets using the concept of visualization on a budget. We reduce the complexity of a point cloud visualization by using continuous level-of-detail, and we minimize its impact on the effectiveness of the visualization by using a point analogue of visual importance, for which we present a metric. The complexity of the visualization is automatically adapted to maintain interactivity. As a result, the visualization always consists of those points that, for a given metric, best represent the point cloud data set for the required interactivity.

We present a prototype toolchain that includes the implementation of our method and demonstrate its performance on a variety of point cloud data sets. We describe its performance characteristics and present a preliminary user study for the method. We evaluate initial results and conclude that our method works, and propose the development of a better metric that does not require user intervention. We end by showing that the prototype toolchain can be used to continue research on this promising method.
Preface

This document is the master's thesis of a MSc. thesis at the section of Computer Graphics, at the faculty of Electrical Engineering, Mathematics, and Computer Science, at the Delft University of Technology, in Delft, the Netherlands.

The Visualization section of the Computer Graphics group, headed by Frits Post, has had an interest in point cloud visualization for quite some time now. Upon my approach to Gerwin de Haan in search for a Master's thesis, he suggested the topic to me as well as two others. Soon thereafter, I selected point cloud visualization as the topic for my Master’s thesis.

I chose point cloud visualization because it had the right elements for me: it was in-depth, technical, and touched on the core aspects of 3D computer graphics. It was right within my interest zone, and a great opportunity to learn. Indeed, I am thankful for finding this project.

But I did not have to go through it all by myself! I would like to thank Gerwin and Frits, my supervisors, for guiding me and putting up with all my quirks. I would like to thank Ruud and Bart, for providing their expert technical support. I would like to thank Martin and Cathinka, my parents, for their unrelenting and financial support. Finally, I would like to thank Jeannette, my best friend, for pulling me through the rough parts.

Berend Wouda
Delft, the Netherlands
August 1, 2011
Contents

Prelace v

Contents vii

1 Introduction 1
  1.1 Point cloud visualization 1
  1.2 Point cloud data sets 7
  1.3 Problem description 9
  1.4 Overview 10

2 Point clouds 11
  2.1 Description of point clouds 11
  2.2 Acquisition of point clouds 14
  2.3 Representation of point clouds 16
  2.4 Visualization of point clouds 18

3 Visualization on a budget 29
  3.1 Continuous level-of-detail 29
  3.2 Importance 36
  3.3 Automatic level-of-detail adjustment 38

4 Prototype toolchain 41
  4.1 Original toolchain 41
  4.2 LAS support 45
  4.3 Data structure improvement 47
  4.4 Importance experiment 53
  4.5 Continuous level-of-detail experiment 64
  4.6 Visualization on a budget experiment 68

5 Results and evaluation 73
  5.1 Performance analysis 73
  5.2 Preliminary user study 82

6 Conclusions and future work 89
  6.1 Conclusions 90
  6.2 Future work 91

List of Figures 95
1 Introduction

Since the turn of the millennium, there has been a renewed interest in the visualization of three dimensional point data [20, 28, 15, 36]. Part of the reason for this is that data sets have become available that natively contain only points, mostly due to a surface scanning technology called Light Detection And Ranging (LiDAR). The constant increase in availability of these point-only data sets, so called point clouds, mandated the research and development of methods for dealing with their specific aspects and applications.

1.1 Point cloud visualization

These point cloud data sets started out as LiDAR scans of objects of various sizes, from industrial parts (seen in figure 1.1) to historical statues (seen in figure 1.2). The application area has been broadened to include domains such as meteorology (seen in figure 1.3), architecture (seen in figure 1.4), archeology (seen in figure 1.5), and geography (seen in figure 1.6).

LiDAR is a relatively new technology however, and while responsible for the recent surge in interest for the field of point cloud visualization, there had already been done quite a bit of research in the related field of point-based rendering before the technology and the data sets it produces ever existed.

Point-based rendering

In fact, the origin of this field can be traced all the way back to 1984, when Robert Cook suggested using a texture map to displace a tabular array of points into a geometric surface in order to deal with the problem of not being able to render procedurally generated

Figure 1.1: Point cloud visualizations of various industrial parts. Reproduced from [35] and [20].
CHAPTER 1. INTRODUCTION

Figure 1.2: Point cloud visualizations of Lucy, a statue of an angel (left), and David, the well-known statue by Michelangelo (right). Reproduced from [24].

Figure 1.3: Point cloud visualization of the aerosol pollution over China, India, and Bhutan. Reproduced from [23].
1.1. POINT CLOUD VISUALIZATION

Figure 1.4: Point cloud visualization of the Stephansdom church in Vienna, Austria. This scan was recorded from the inside. This type of scan is known as an interior scan. Reproduced from [30].

Figure 1.5: Point cloud visualization of the Domitilla Catacomb in Rome. It is a vast network of tunnels and catacombs from the Roman times, that extends far under the ground. Reproduced from [32].
CHAPTER 1. INTRODUCTION

Figure 1.6: Point cloud visualization of a part of a city. Not only does LiDAR record the surface of the earth, but it also records the manmade structures upon it. Reproduced from [27].

data in image-order [15, 36]. The foundations for the field of point-based rendering were laid when Levoy and Whitted published their seminal work on using points as display primitives [21]. This work is regarded as the first work in its field.

Points as display primitives

When they published that work, their interest in points was rooted in an entirely different aspect. They used points as display primitives and showed that points can be used to represent surfaces.

Up until then, visualization was almost entirely polygon based. Points have been used a few times before for particle simulation and volumetric effects such as modeling smoke, fire, or trees [21, 15, 36], but regular three dimensional models were all expressed in meshes of polygons. Rendering algorithms for these models could use either image order rendering or object order rendering, but object order rendering required a brute force method such as the Z-buffer, which was simply too expensive at the time. While image order rendering was adequate for most cases, certain cases could not be rendered using image order, and as such had to be rendered using object order [15, 36, 21]. Points however possessed a few qualities that appeared to be the answer.
Object order rendering

Points are both dimensionless and volumeless. They do have a position, and possibly other attached information, but they do not occupy space. As such, they cannot overlap, occlude each other, or be partially in view. On top of that, they are individual entities without any coherence, they are not connected to each other. This means they can be rendered in object order.

These simplifications make handling them very different when compared to handling polygons. Polygon based rendering approaches utilize a pipeline that executes steps such as polygon splitting, rasterization, and shading interpolation. Because points are unconnected however, point based renderers require a pipeline that is far less complex. On the other side of the pipeline this is the case as well, where polygon based renderers need to maintain and process a mesh of polygons but point based renderers only need to maintain one point at a time.

For Levoy and Whitted, it meant that points could be used to solve the object order problem. Since points do not occupy space, they cannot occlude each other, and they do not have to be sorted. Of course, a visualized point occupies space (in terms of displayed pixels), but they solved this problem by using the A-buffer algorithm which for single points occupying at most one pixel is much less expensive than for polygons. They had effectively decoupled geometry from rendering.

The universal display primitive

Using this notion they even went a little further, and described how one could use points as a universal display primitive. If rendering points is very simple, and surface primitives (such as polygons) can be converted to points, then points could essentially replace the current surface rendering approaches and change the landscape of visualization. It could even replace bump mapping. Of course, for this to work, surface primitives would have to be convertible to points, and the conversion and point rendering operations combined would have to be less expensive than just rendering the surface primitives directly.

Another trend they had picked up on was that model complexity kept rising to create better and better approximations. They noted that the coherence in these models was becoming less and less useful as screen estate was increasing at a much slower rate. For naturally complex models (such as trees), coherence already didn't even make a difference anymore. At a certain point, polygons would cover less than one pixel, and if that were to happen, points would completely supersede them as coherence would no longer add anything and each surface could just be represented by one point.

All in all it seemed that points could deal with the then current problem of object order rendering without having to resort to a slow, expensive brute force approach, while inadvertently introducing a whole new visualization paradigm that at the time seemed the answer to everything.
CHAPTER 1. INTRODUCTION

Unfortunately, the lack of dimension and coherence of points also signifies their largest deficit. They do not cover space. When visualized directly, this causes holes (as can clearly be seen in figure 1.7). While solutions to fill these holes have been devised [26, 23, 24], they diminish the simplicity and speed advantages over polygon based renderers. Additionally, the resulting visualization quality is often less than the quality produced by polygon based renderers due to a lack of surface information.

Levoy and Whitted opted for a simpler method. If there is a point behind every pixel, there will be no holes in the visualization, and the surface will be represented perfectly. This method was also used by others in later approaches. Alexa et al. even resampled their data sets on the fly so that there was no more than one point behind every pixel [2]. However, this meant that the density of the data set would have to be locked to the resolution of the display, to a certain degree. Aside from this, the method didn’t work entirely as expected, as it turned out that with the method originally devised for nonregular geometry (the aforementioned trees and such), it didn’t apply to regular geometry all that well.

The lack of coherence also makes operations that depend on coherence information impossible or very expensive [36]. For example, neighbourhood searches devolve to at worst a linear search of the entire data set. Phong shading cannot be simply reimplemented as the normal interpolation operation requires the normals of the adjacent surfaces, which

Figure 1.7: A point cloud of a child’s face. The lack of coherence can clearly be seen. Reproduced from [6].
do not exist in the data set. Ultimately, points didn’t make it as the universal display primitive \[15, 36\]. The technological advances that, against all expectations, allowed the Z-buffer algorithm to solve the object order issue made sure that polygons remained the display primitive of choice. The hole issue remained a problem as well. After a system called REYES \[7\] went back to polygons (albeit micropolygons, with point-like properties) to visualize holeless surfaces, Levoy and Whitted abandoned their research. The interest in points as display primitives further decreased afterwards, and eventually died. During the last decade of the millennium, there was little going on at all in the field of point-based rendering.

**Emergence of LiDAR and point cloud data sets**

But then a certain scanning technology by the name of LiDAR started its rise to fame. Combined with technological advances in computer hardware, the availability of point-only data sets spurred a renewed interest in points. This time not so much as a display primitive (much less the universal display primitive), but as a data unit, giving rise to the field of point cloud visualization. Building on the seminal work by Levoy and Whitted as well as the following works in the field of point-based rendering, new methods were devised that made use of current technologies as well. This time, the advantages of points could be used to their full extent, while the disadvantages could be treated as an aspect of the data set. For example, if you have a point cloud data set, then holes are an “acceptable” side effect during visualization. But even for that problem advances are made. All in all, points are back in the minds of researchers all around the globe, and both fields are again in full swing.

**A lesson learned**

Looking back on it all, Levoy remarked that he should never have tried for points to be the universal display primitive \[15, 36\]. They definitely have their uses, but those uses are specific to certain fields and applications. Levoy (proclaiming himself as the author of the rise and fall of points as display primitives) leaves us with the wisdom to clearly distinguish between “those things points can do, but so can other representations, those things points can do that other representations cannot, and those things points cannot do or will never do well”.

**1.2 Point cloud data sets**

The data sets involved in geography are commonly known as aerial LiDAR point cloud data sets. Geographical Information Systems (GIS) have always used some form of surface approximation, most notably two dimensional maps, to support a multitude of applications. Aerial LiDAR point cloud data sets provide a new form of this approximation, but with more detail, an extra dimension, and other additional information not found on the classic map.
CHAPTER 1. INTRODUCTION

Figure 1.8: Point cloud visualization of a small part of the AHN2 aerial LiDAR point cloud data set. It shows a sliver of Dutch glory. Reproduced from [9].

AHN2 data set

The Visualization Group at the Delft University of Technology has gained early access to such an aerial LiDAR point cloud data set. This data set is called Actueel Hoogtebestand Nederland (AHN) and is an up-to-date height data base of the Netherlands\(^1\). There are two versions currently available, AHN1 (recorded from 1997 to 2003, with a density of about 1 point per square meter) and AHN2 (recorded from 2008 to 2013\(^2\), with a density of about 10 points per square meter). An example of the AHN2 data set can be seen in figure 1.8. Figure 1.9 shows a section of the AHN1 data set, and the same section as it is found in the AHN2 data set. The difference in density between the older AHN1 and the newer AHN2 is clearly visible.

Exponential growth

In fact, this difference in density between the old and the new is an ongoing trend among aerial (but also terrestrial) LiDAR data sets. Advances in LiDAR technology are constantly being made, and as a result its resolution and range keep increasing. However, its adoption is also continuing to grow. More and more parties are interested in the data sets it can produce as there are a multitude of applications for them, as is the trend with geographic data in general, but also since the data is three dimensional. As more surface is covered, at higher resolutions, the size of the produced data sets is increasing exponentially.

Currently, the amount of points in the resulting aerial LiDAR point clouds can range from millions to billions. They are therefore often massive, with data set sizes ranging between tens and hundreds of gigabytes. The AHN2 data set goes beyond even these numbers, with the final version being predicted to contain a trillion points, with a size of around 8 terabytes. In the mean time, the exponential growth in aerial LiDAR point

\(^1\)We would like to thank Rijkswaterstaat for providing the AHN2 data set. More information about the AHN1 and AHN2 data sets can be found on [http://www.ahn.nl](http://www.ahn.nl).  
\(^2\)Estimated end date, at the time of writing the recording of AHN2 was still in progress.
1.3 Problem description

The size of these data sets has come to a point\(^3\) where simply displaying them doesn't work anymore, let alone exploring them interactively. As such, the issue here is how to efficiently visualize and allow interaction with this massive amount of data. The visualization of a point cloud data set has to be done in a timeframe that allows interactivity so the user can navigate in real-time. It also needs to maintain a visual quality that is high enough for the user to understand the data set. Furthermore, it requires handling the enormous amount of data behind the scenes. In short, we want to be able to interactively visualize massive LiDAR point clouds.

Research goals

Given that we want to interactively visualize massive LiDAR point clouds, we set ourselves the following goals.

1. Design and implement a method that provides interactive visualization of massive point cloud data sets.

2. Take advantage of the limits of the human visual system to reduce the complexity of a point cloud visualization while minimizing the impact on the effectiveness of that visualization.

3. Design and implement a prototype toolchain that supports current and future research on this subject.

\(^3\)No pun intended.
1.4 Overview

In this thesis, we describe a method for interactive visualization of massive point cloud data sets.

In order to discuss this method, we need to know what point clouds are and how they have been visualized in the past. In chapter 2, we provide a detailed description of the concept of point clouds, and give an overview of the earlier work.

We develop our method by noting that research into point cloud visualization has not yet focused on taking advantage of the limits of the human visual system. In chapter 3, we provide a detailed description of the method we developed.

To implement our method, we developed a prototype toolchain for point cloud visualization and evaluation. In chapter 4, we provide a detailed description of the prototype toolchain, and our implementation of our method.

To evaluate our method, we have performed objective and subjective studies. In chapter 5, we provide a detailed description of our evaluation methodology, the full results, and the evaluation of our method.

From our evaluations we draw our conclusions, and from our conclusions we propose future work. In chapter 6, we provide a detailed description of our conclusions, and describe further research possibilities.
2 Point clouds

We have seen in chapter 1 that point cloud data sets, also known as point clouds, have mandated the research and development of methods for dealing with their specific aspects and applications. Section 1.3 in particular describes the issues with point cloud data sets when it comes to their visualization. In this chapter, we explore what point clouds are to better understand them, and summarize the earlier works to gain insight in the issues with their visualization.

A detailed description of point clouds is given in section 2.1. They are often recorded using LiDAR technology, which is described in section 2.2. Afterwards, they are stored in record-like data sets, described in section 2.3. Finally, they are visualized using point cloud visualization pipelines, as is described in section 2.4.

2.1 Description of point clouds

Data sets containing only point samples (and data related to point samples) are colloquially known as point clouds, as they essentially represent clouds of points. Somewhat similar to water particles in the fluffy kind of cloud, points in a point cloud do not have any coherence [21]. Indeed, when visualized, this causes the collection of points to often look like one or more cloud-like structures (for example, see figure 2.1).

Lack of coherence

Because points in point clouds are not interconnected, they are essentially individual entities. Groups of points may still represent surfaces or volumes, but this information is not explicitly stored. With points in a point cloud being individual, the point cloud can in this way be simply thought of as a database containing a set of records, which are the points. Each record in turn simply contains a set of fields, which are the point’s attributes. This analog shows that point clouds are actually very simple data sets [29].

As such, point clouds may contain any number of points, and any number of attributes per point. The more points a point cloud has, and the more attributes a point has, the larger the data set becomes.

Attributes of points

In all cases, points in point clouds have an attribute for each dimension of the space the point cloud resides in, describing their position within the point cloud [21, 40, 27]. These are normally known as the x, y, and z attributes (depending on the number of dimensions). In most cases, points in point clouds also have shade attributes [21, 26].
Shade attributes represent the lighting function at the point. They generally only include the diffuse color components, normally known as the $r$, $g$, and $b$ attribute triplet (depending on the color space), but they can include other lighting properties as well (such as the specular and emissive color components, or the shininess). In some cases, points in point clouds have normal attributes. Normal attributes indicate the gradient of a surface at the position of the point. They normally are the $u$, $v$, and $w$ attribute triplet (depending on the number of dimensions).

Points in point clouds can additionally have any other set of attributes as well, which makes them very extensible. One can use this feature to tag points or store processed information per point (see section 2.3).

Definition of point clouds

Given this description of point clouds, we can give the following definition.

**Definition** A point cloud is an collection of point samples, where every point sample is an $n$-tuple with position attributes. This $n$-tuple may also contain other attributes such as shade or normal attributes.

For example, a common point cloud data set is one that is a list of $(x, y, z, r, g, b)$ tuples.
2.1. DESCRIPTION OF POINT CLOUDS

Figure 2.2: Left: A ground-based LiDAR scanner. Right: An aerial LiDAR scanner.

Figure 2.3: Part of the Stephansdom ground-based point cloud data set. The diamond-shaped holes that can be seen in the floor mark the consecutive locations of the LiDAR scanner. This is because the scanner is unable to scan below itself. Reproduced from [30].
2.2 Acquisition of point clouds

Section 1.2 states that the primary reason for the renewed interest in point clouds is because they natively result from certain forms of data set acquisition. This is also the reason for the definition in section 2.1 as the data sets are acquired from a stream of data that needs to be stored quickly, and as such the data is stored in a way that is simple and linear.

Acquisition of point clouds with LiDAR

The primary form of point cloud data set acquisition is Light Detection And Ranging (LiDAR). It is a laser scanning technique analogous to RADAR, sometimes known as LADAR \[44\]. The LiDAR scanner (seen in figure 2.2) produces a laser pulse in a direction, which is reflected off the nearest surface and registered by the scanner. The time dilation between transmission and reception gives, together with the known speed of light, the distance to the object. If a surface is semi-transparent, a LiDAR pulse can even produce multiple reflections as the laser penetrates the surface and reflects off a surface behind it. The light model of the surface defines how much light is reflected back towards the camera. Surfaces that are dark or highly specular will reflect less intense laser pulses. The intensity of the reflected pulse is recorded, and can be stored as an attribute of the point sample (see section 2.1). Besides simply the intensity, the color of the surface can
2.2. ACQUISITION OF POINT CLOUDS

Figure 2.5: The Domitilla Catacomb ground-based point cloud data set. The catacomb consists of a vast network of tunnels below the ground, and as such required many scan positions. Reproduced from [32].

Figure 2.6: Part of the AHN2 aerial LiDAR point cloud data set. In this image, the points have been pseudo-colored to represent their respective heights. Reproduced from [9].
also be recorded and stored as an attribute \[40\]. This is done by a camera built into the scanner that can photograph along the line of sight of the scanner.

The LiDAR scanner does this continuously while panning and tilting over the field of view to be registered, and in this way records a data set of three dimensional points that make up the field of view. In order to align these points to their frame of reference, the position of the scanner needs to be properly calibrated. With a fully rotating LiDAR scanner placed on the ground at different places, a complete model of an environment can be made up by merging the resulting data sets into one large data set that contains the entire environment. For example, this was done during the Stephansdom campaign \[30\] (seen in figure 2.3) and the Domitilla Catacomb exploration \[32, 31\] (seen in figure 2.5).

One can also mount the LiDAR scanner on an aircraft, and fly it over a surface (shown in figure 2.4). In this way, a data set that contains a three dimensional map of a part of the surface of the earth can be recorded (known as a aerial LiDAR point cloud data set). The scanner, which is looking downward, only pans in this case, but the motion of the aircraft ensures that all of the surface is scanned in sweeps following the aircraft’s path. For example, this is done for the AHN2 data set \[9\], as can be seen in figure 2.6. Color is instantly photographed by a line camera along the sweep of the scanner, from which the colors are sampled.

Acquisition of point clouds with other methods

Due to the simplicity of point cloud data sets, any process that produces a collection of points can produce a point cloud data set. One of these methods is image matching, where points are inferred from images (photos or videos) taken of real-world objects from multiple viewpoints, using image-based computer graphics methods. Another way of acquiring point clouds is by generating them using simulations. As was mentioned in section 1.1, this was the original reason for their invention because procedurally generated data could not be rendered in image order \[21, 36\]. Producing a point cloud from a simulation is somewhat similar to the LiDAR scanning process, as points are also sampled from surfaces and stored in a point cloud. However, these surfaces are virtual, represented by mesh or curve-based computer models \[26\]. And because the entire surface is actually known in these models, the point samples can actually have many more attributes (see section 2.1) assigned to them. For example, because the surface around a point sample can be interpolated from, a point sample can store a perfect surface normal and advanced lighting attributes. Such attributes allow full visualizations of the original model (for example figure 2.7), while maintaining the advantages of point clouds described in section 2.1.

2.3 Representation of point clouds

As described in section 2.1, points in point cloud data sets can be seen as records with fields within a database. Storage-wise, this means that the standard point cloud data set is a simple, flat, text or binary file without any order or hierarchy (or a group of them
2.3. REPRESENTATION OF POINT CLOUDS

Figure 2.7: Point cloud visualizations using surface elements (surfels). Surfels are point samples with additional attributes that represent the surface at the point sample. Reproduced from [26].

due to file system limitations). Such a file may have a header describing some general properties of the point cloud or the file. Not all files may have the same format, however, as different scanner software may store data in different ways. Software such as libLAS [38] allows one to convert between these formats.

Structure of point cloud data sets

But while the data set produced by the LiDAR scanning process may be simple, it often does have somewhat of an underlying structure. In ground-based scans, points are linearly recorded in sweeps that depend on the rotating motion of the scanner, and have therefore a certain order within their data set (even though points in point clouds are technically unordered due to their unconnected nature). In aerial scans something similar occurs, the path of the aircraft and the range of the scanner cause the points to be ordered in strips (also visible in figure 2.4). When data sets from multiple scans are combined, the data is generally not reordered and as such points from each original data set form groups within the new data set. All this causes points that are close together in the scanned space to very often also be close together within the data set. This is useful for stream or block operations that need to evaluate groups of nearby points.

A data set can be said to have a density. Density indicates how close together the points in the point cloud are when compared to the space they are in, although there may be local variances. The density is not always known beforehand, and as such must often be calculated from the data set. Data sets produced by a ground-based LiDAR scanning process tend to have a highly nonuniform density because of the varying distance of the scanned surfaces to the camera. If data sets from multiple scan positions are combined, the density becomes even less uniform. On the other end of the spectrum, the data sets produced by the aerial LiDAR scanning process often have near uniform density (and the data sets are as such even somewhat gridlike), because of the regular sweeping motion of the scanner, the “constant” speed and altitude of the aircraft, and the relative flatness of the surface. This can also be seen from figure 2.4. However, there may be local variances if data sets from multiple scans overlap and are combined (see section 4.4). If a data set does not have a uniform density, it can be resampled so that it does [20].

17
CHAPTER 2. POINT CLOUDS

Processing of point cloud data sets

Calculating the density of a data set and resampling a data set are both examples of processing a data set. As they are expensive and long operations, they are often done before regularly accessing the data set. This method of saving time is called preprocessing because it prepares a data set for access. It often requires a tradeoff between space and time, because the result of the operation has to be stored statically, essentially creating a new (and possibly larger) data set. On the other hand, it is also possible to remove data from a data set during preprocessing to purposefully make it smaller. The end result of preprocessing is always a data set that has the properties required for its intended use.

As such, LiDAR point cloud data sets are often preprocessed. For example, instead of measuring the color using the camera on the scanner, it is also possible to interpolate the color attributes of a point from an image (of much lower density) during preprocessing. Alternatively, one may decide to instead remove all color attributes and perform the interpolation in real-time. Another example are surface normals, which cannot be recorded during the scanning process because this is not possible with current LiDAR technology. Instead, they are calculated using nearby points in the data set as these form a surface approximation [40]. Because calculating all the surface normals for a point cloud data set is a very expensive operation, it is almost always done during preprocessing. The addition of surface normals in a preprocessed data set causes the data set to be larger than its original, however.

Additional attributes in point cloud data sets

As mentioned in section 2.1, an additional attribute that is sometimes seen in point cloud data sets are tags or meta data [5]. This information can be added automatically during preprocessing, or can be added manually. It can even be produced by the scanner. It is generally not essential to the point cloud itself, describing aspects of points or groups of points that serve a different purpose. For example, features that result from the points in the point cloud may be tagged as such, or the scanner may store an acquisition time with every point.

2.4 Visualization of point clouds

Visualization is the presentation of information to a user in such a way that the user understands the visualized information. For point cloud data sets, this means that the user has to be able to perceive the shape of the data sets.

However, another very important aspect is interactivity, which means a frames-per-second (FPS) rating of at least 30. 60 FPS is preferred, as this is the update frequency of most display devices, and one wishes to avoid visual tearing[1].

1Visual tearing occurs when the frame buffer the GPU is drawing into is only partially filled with a new image when the buffers are flipped. The resulting image contains a part of the current frame and a part of the previous frame.
2.4. VISUALIZATION OF POINT CLOUDS

![Diagram of point cloud visualization pipeline](image)

Figure 2.8: A simple point cloud visualization pipeline. The upper arrows indicate the flow of data, with the lower arrow representing the user interaction loop.

These two requirements are often orthogonal to each other, as a better shape representation usually requires more points, and an interactive visualization generally requires less points. Finding an acceptable compromise between these two requirements is the main challenge in point cloud visualization.

**Simple point cloud visualization pipeline**

The most simple point cloud visualization is actually rather straightforward. Points are displayed as splats with some size, and the human visual system is able to infer the shape that the points represent. A splat is a two dimensional drawing, for example a quad or a disk, or even just a single pixel, at the location where the point should be displayed [9].

Such a simple point cloud visualization is produced by a simple point cloud visualization pipeline. Figure 2.8 shows such a pipeline, as well as the flow of point data, from its acquisition to its perception, to the interaction the user has with it. Points are acquired, usually by a LiDAR scanner, after which they are stored in a simple way on some form of high-volume transportable media, often an external hard disk. From this storage they are preprocessed, and stored in a data structure on some form of external memory, usually a hard disk inside a computer. This could be a hard disk in a local client, or a hard disk in a remote server. From this external memory the points are transferred over the network or disk bus into internal memory. The CPU retrieves the points from internal memory through its memory bus, and pushes them through the video bus into video memory. In turn, the GPU then retrieves the points through its memory bus, and pushes them through its rendering pipeline. This eventually displays them on the display device, which generally is some sort of monitor or projector, and light, the physical medium, allows the user to see the visualization. The user then interacts with the visualization, changing which points should be visualized, and a new run through the point cloud visualization pipeline is required.

The point cloud visualization pipeline consists of a number of steps, but they are not required to be in sequence. An often used paradigm is where the point cloud visualization system performs both the visualization and the data access at the same time, retrieving point data from the point cloud data set asynchronously, providing it to the visualization when it becomes available [9, 5].
The steps don’t even have to be on the same system. The point cloud data set may be stored on remote external memory, and accessed through a network connection. Figure 2.9 shows such an asynchronous client-server framework.

Spatial data structures

To be able to more efficiently access point cloud data sets, one can index them. This is done using specially designed spatial data structures. Many of the earlier works attempt to solve the problem described in section 1.3 at least partly by using efficient spatial data structures. There exist two main types of spatial data structures, space driven (image hierarchy) and data driven (object hierarchy) [29].

Space driven spatial data structures

Space driven (image hierarchy) spatial data structures divide the space of the data set in partitions based solely on the dimensions of the space. This means that the data structure constructed for a data set is not dependent on the data set except for its depth. These space partitions are often regular as that produces the most balanced trees with uniformly spaced data sets, but they are not required to be regular.

Perhaps the most commonly known space driven spatial data structure is the quadtree [29, 11]. Quadtrees recursively divide space along all dimensions of the data (see figure 2.10). While a quadtree is by design a two dimensional data structure, it can be used to spatially index three dimensional data by simply ignoring one of the dimensions of the data set. This may be useful when spatially indexing data that has a very limited range in one dimension, when compared with the range of its other two dimensions, such as point cloud data sets obtained with the aerial LiDAR scanning process where the range of the $z$ axis is very small compared to the range of the $x$ and $y$ axes [9].

A three dimensional version of the quadtree is the octree [29]. Instead of dividing two dimensional space into four quadrants, it divides three dimensional space into eight octants (see figure 2.11). Octrees have been successfully used in the visualization of many point cloud data sets [20, 40, 27].
2.4. VISUALIZATION OF POINT CLOUDS

Figure 2.10: Quadtree space decomposition of the space of a node. Each quadrant is assigned to a child node, and the decomposition is repeated per child node until no more decomposition is necessary.

Figure 2.11: Octree space decomposition of the space of a node. Each octant is assigned to a child node, and the decomposition is repeated per child node until no more decomposition is necessary.
CHAPTER 2. POINT CLOUDS

Figure 2.12: A two dimensional kd-tree constructed on a data set of 6 points. It has 3 levels (right) with dimensional switches that occur per level (left). Reproduced from [43].

Data driven spatial data structures

Data driven (object hierarchy) spatial data structures divide the space of the data set in partitions based on the data in the data set. The shape of such a data structure therefore depends on the data set, and is not always known beforehand. There are two main paradigms in data driven spatial data structures, the separation of space on data points, and the bounding of space on data points [39, 29].

A simple example of the first paradigm is the point quadtree. Point quadtrees are quadtrees that locate the origin of the splitting planes on a data point [29]. Point octrees are again the three dimensional variant of point quadtrees.

The most common example of the first paradigm however is the kd-tree, shown in figure 2.12. Like point quadtrees, kd-trees locate the origin of the splitting plane on a data point, but they have only one splitting plane per level, and its orientation is perpendicular to the dimension associated with that level [11, 29, 43]. This dimension alternates per level, between the k dimensions. There exist various versions of the kd-tree, such as adaptive kd-trees [12, 29, 11].

The prime example of the second paradigm is the R-tree (also referred to as an object driven spatial data structure), shown in figures 2.13 and 2.14. R-trees are spatial data structures that group objects in bounding boxes recursively [16, 1, 39]. This effectively makes them two dimensional B-trees. There exist various versions of the R-tree, such as the R+-tree [33, 15, 14], R*-tree [3, 46], and the Hilbert R-tree [10, 17, 18, 42].
2.4. VISUALIZATION OF POINT CLOUDS

Figure 2.13: A two dimensional R-tree constructed on a point data set. Reproduced from [46].

Figure 2.14: A three dimensional R-tree constructed on a point data set. Reproduced from [47].
**CHAPTER 2. POINT CLOUDS**

**Space driven versus data driven spatial data structures**

Earlier works use space driven spatial data structures, mostly notably the octree, to efficiently index their data. The reason they all use space driven spatial data structures and not data driven spatial data structures is most likely because the visualized massive point cloud data sets have a density that is high and regular enough for space division to be an efficient and versatile enough index for the data.

The problem here is that in data driven spatial data structures, the data becomes part of the structure. Because there is so much data that has to be evaluated, it makes building the data structure an inefficient process. They are also prone to unbalance if not treated very carefully, while with the aforementioned dense regular point cloud data sets, space driven spatial data structures are generally rather balanced.

**Combinations of spatial data structures**

One can combine spatial data structures to use all of their advantages, or to denounce some of their disadvantages. This is normally done by storing instances of spatial data structures in the nodes of another [39, 30].

One paradigm is to store small subtrees in a larger tree to present smaller, local versions of the problem to be solved in order to save on space and time [39, 40].

Another paradigm is to store data driven subtrees in a space driven subtree to allow for efficient local data access once the space driven index has efficiently located the space of the node [22].

**Bottleneck alleviation**

As the point cloud visualization pipeline from figure 2.8 processes larger and larger data sets, its bottlenecks start affecting interactiveness, shown in figure 2.15. As more and more data needs to be stored and transferred, the maximum memory and throughput capacities of bottlenecks are exceeded, and the entire pipeline starts to slow down. This is essentially the cause of the problem described in section 1.3.

Earlier works often augment their choice of spatial data structure by alleviating one or more of these bottlenecks. To reduce use of all the bottlenecks, the size of the data set can simply be reduced. To reduce the use of internal memory, the out-of-core paradigm can be used. To reduce the use of the CPU, work can be moved to the GPU. To reduce the use of the GPU, the amount of visualized points can be reduced.

**Reduction of data set size**

As described in section 1.3 point cloud data sets are often very large, and processing and visualizing them presents a challenge. To increase performance one can simply start by reducing the size of the data set.

One way to do that is to simplify the data set [25, 34, 9]. Simplification sacrifices detail for size, but simplification methods often try to simplify the data set in such a way that it still retains its function.
2.4. VISUALIZATION OF POINT CLOUDS

Figure 2.15: The bottlenecks in a simple point cloud visualization pipeline when dealing with massive data sets, indicated by the upper arrows. The middle arrows indicate the flow of data, with the lower arrow representing the user interaction loop.

Another way to reduce the size of a data set is to compress it [5, 30]. This can be used for the transfer of data as well [13]. As this method sacrifices time for space, it only increases performance when used during transfer.

Lastly, to reduce the size of a data set one can quantize its contents [28, 13, 5, 30]. Quantization is the process of allocating less space to the representation of the data. This method sacrifices accuracy for size.

Out-of-core paradigm

If a point cloud data set is too large to be stored in internal memory, the out-of-core paradigm can be used. A data set that is stored out-of-core is stored in external memory while the parts of it that are needed for the visualization are maintained in internal memory. These parts are fetched on demand, and are managed in such a way that the available internal memory is not exceeded.

For example, the Least Recently Used (LRU) paradigm maintains a list of parts in internal memory, sorted on their age. When additional internal memory is required, the oldest part is discarded [40, 31]. The out-of-core paradigm is used in most earlier works [40, 27, 32].

Transfer of work to the GPU

As can be seen from figure 2.8 a point cloud visualization is eventually visualized by a Graphics Processing Unit (GPU). At the basis, the GPU is a parallel linear algebra computer, allowing many linear algebra operations in a short time. Recently it has also come to support shaders, which are small programs that can be run on the GPU. Furthermore, the GPU has access to low latency video memory.

All of this makes the GPU a faster processor than the CPU for certain operations, and this means one can move parts of the point cloud visualization algorithm to the GPU.
CHAPTER 2. POINT CLOUDS

Figure 2.16: Three levels-of-detail. The level-of-detail increases from left to right, as can be seen by the increase in density. The bounding boxes show the spatial subdivisions used to make the multiresolution representations. Reproduced from [40].

Figure 2.17: Level-of-detail providing multiple resolutions in a point cloud visualization. The color of each point indicates the depth of the level it is in, ranging from red (lowest) to purple (highest). Reproduced from [9].
2.4. VISUALIZATION OF POINT CLOUDS

for increased performance [40, 27, 20, 30]. However, video memory is generally limited in size when compared to internal memory, and if there is too much data then the use of video memory will have to be managed in some way [27, 30].

Reduction of the amount of visualized points

The performance of the GPU can be increased greatly by reducing the amount of points that it has to visualize. Points that lie outside of the field of view of the camera can be entirely skipped, this is known as view frustum culling. As points have no volume, they can easily be culled because they are either completely inside or completely outside the field of view of the camera. However, as the amount of points in a point cloud increases, culling groups of points instead of single points may provide better performance.

Another way to reduce the amount of points visualized is to use a multiresolution representation. With this method, a point cloud data set is maintained in multiple resolutions, and given a maximum of points that can be visualized, the highest resolution that provides a visualization still below that maximum is used during the visualization. A common multiresolution approach is level-of-detail, shown in figure 2.16. In discrete level-of-detail, multiple resolutions are maintained, and a metric such as distance to the camera determines which resolution is used. In continuous level-of-detail, the resolution is directly calculated from the metric. Resolutions can be maintained for an entire point cloud or for subselections of a point cloud [29, 31], as seen in figure 2.17. Multiresolution representations of a point cloud data set are normally generated from that data set by using simplification. This process can be applied recursively to generate all required resolutions. If the simplification method is fast enough, it can even be used for continuous level-of-detail.

In this chapter, we have seen that point cloud data sets are collections of points samples, which in turn are collections of point attributes. These attributes can be anything, but always include position attributes, and often include shade and normal attributes. Furthermore, points are unconnected, single entities, and they have no volume.

We have seen that point cloud data sets are often recorded by LiDAR scanning technology, but that they may also result from other sources such as image matching or simulation. While they are technically unordered, due to the scanning technology, they are generally stored in ways that enable a decent measure of locality. Furthermore, we have seen that a point cloud data set may or may not have a uniform density, but can be preprocessed to do so, as well as make available any other aspects that are required.

We have also seen that point cloud visualization is about the user interactively understanding the shape of a point cloud data set. This presents a problem as shape and interactiveness are adversary. We have discussed the point cloud visualization pipeline, and the issues that arise when it deals with massive point cloud data sets.

Finally, we have reviewed the spatial data structures and bottleneck alleviation used by earlier works for point cloud visualization. As it turns out, earlier works attempt to solve the problem by using efficient spatial data structures to store and access the data set, while alleviating the usage of one or more bottlenecks within the point cloud.
visualization pipeline. All earlier works use a similar out-of-core, multiresolution, space based spatial data structure, indicating that this may be a solution that works well when dealing with massive amounts of points. Their choices of bottlenecks to alleviate vary, although they all improve the interactivity of the visualization to some degree.
3 Visualization on a budget

We have seen in chapter 2 that many of the earlier works that visualize massive LiDAR point clouds use efficient spatial data structures combined with various bottleneck alleviations. They achieve varying degrees of efficiency, especially when it comes to scalability. Scalability is key to performance as the size of these point cloud data sets will only continue to increase (as described in section 1.3). However, if instead of alleviating the various bottlenecks shown in figure 2.15 we interpret them as a budget, we can tackle the problem from the other end. If we treat the bottlenecks as the budget we have available for a point cloud visualization, we can reduce the complexity of the visualization according to that budget so that it remains interactive. We call this paradigm visualization on a budget.

Points lend themselves very well to this concept. Their lack of coherence (described in section 2.1) forces the user to infer the visualization from only the points themselves, causing the complexity of a point cloud visualization to depend directly on which points are visualized. Therefore, to reduce the complexity of a point cloud visualization, one can simply leave points out of the visualization, and this is easy also due to their lack of coherence.

We achieve this by using continuous level-of-detail. This decreases the density of points by leaving them out of the visualization as they are further away from the user. Their decreased density provides a lower level-of-detail, but since they are further away from the user, their perceived density should still be still the same. This is described in more detail in section 3.1.

However, we want the visualization to remain effective as well as interactive. This means we have to leave points out of the visualization while minimizing the impact on the effectiveness of the visualization. For this we introduce the concept of point importance. The point importance of a point represents its importance to the human visual system, and points are left out of the visualization according to this importance. Point importance is described in more detail in section 3.2.

Finally, in order to maintain interactivity as the user interacts with a point cloud visualization, we have to continuously change the complexity of a point cloud visualization. This is achieved by automatically adjusting the level-of-detail of the visualization to provide the interactivity that is required. This is described in detail in section 3.3.

3.1 Continuous level-of-detail

In order to provide an interactive visualization of massive LiDAR point cloud data sets, we can reduce the complexity of the visualization by leaving out points. For this we use the concept of level-of-detail.
Level-of-detail

In section 2.4 we have seen that level-of-detail has been used by earlier works as an optimization technique. It increases interactivity by reducing the amount of points processed by the visualization pipeline.

The basic principle is that points that are further away from the viewer can be left out if that doesn’t reduce the user’s perceived density. In other words, as the distance from the user increases, the density of the visualized points at that distance can be decreased. This is because of the perspective transformation that occurs in point cloud visualization, points are seen at a higher density if they are further away. But because we only need to see a certain density, any points beyond that density can be left out.

An instance of a density of points is known as a level-of-detail, because it, as the name implies, represents a certain level of detail. There are two main paradigms in using level-of-detail, discrete level-of-detail and continuous level-of-detail.

Discrete level-of-detail

In the discrete level-of-detail paradigm, a point cloud data set has a finite amount of levels-of-detail. Each level-of-detail has a density associated with it, that is a certain factor smaller than the density of the next level-of-detail. Each level of detail also has a range associated with it, which is compared to the distance from the user to decide whether the level-of-detail is part of the point cloud visualization (shown in figure 3.1). This range is a certain factor larger than the range of the next level-of-detail. For example, in a two dimensional data set a level-of-detail has a density that is a quarter of the density of the next level-of-detail, and a range that is twice the range of the next level-of-detail. This means that if the distance from the user decreases from the range of a level-of-detail to the range of the next level-of-detail, the perceived density will remain the same. Between these two ranges however, the perceived density will decrease. This gives a discrete transition where points appear to “pop” into the visualization, hence the name.

Usually, a single discrete level-of-detail is visualized, which is selected by comparing the ranges of the levels-of-detail to the distance from the user. This generally involves some redundancy. However, since points lack coherence we can forego on this redundancy, by instead showing the selected level-of-detail and all the previous levels-of-detail simultaneously.

Hierarchical level-of-detail

We use hierarchical level-of-detail to reduce the complexity of a point cloud visualization. It is a form of discrete level-of-detail where each spatial subdivision of a space driven hierarchical spatial data structure (described in section 2.4) contains a subset of the points within its space in a density according to the level-of-detail for the depth of that subdivision.

In our case, the hierarchical spatial data structure is either a quadtree or an octree, depending on the (dominant) dimensions of the data set. As such, each subdivision
3.1. CONTINUOUS LEVEL-OF-DETAIL

occupies a quarter or an eighth of its parent’s space, the density of its associated level-of-detail is four or eight times that of its parent’s, and its associated range is half of that of its parent’s. Usually, for aerial LiDAR point cloud data sets that are predominantly flat a quadtree is sufficient, as they are almost two dimensional. If such a data set contains very high points, an octree generally provides better performance.

The hierarchical spatial data structure provides two optimizations (shown in figure 3.2). First off, the spatial subdivisions present in the data structure allow for efficient, hierarchical culling of subdivisions that are outside the field of view of the user (this is essentially view frustum culling for points, see also section 2.4). This is required because point cloud data sets are very large (as described in section 1.3), and this leaves out many of the points that the user cannot see. Furthermore, the hierarchy of the data structure allows for efficient level-of-detail selection for the spatial subdivisions that are inside the field of view of the user. This leaves out the points the user cannot perceive anymore. In other words, the hierarchical data structure allows efficient selection of the points that should be visualized.

Continuous level-of-detail

Unfortunately, hierarchical level-of-detail by nature has a finite amount of levels-of-detail (one for each level in the hierarchy) and as explained this causes discrete transitions between the levels-of-detail (the effect of which can be seen in figure 3.3). Since this has an impact on the effectiveness of the visualization, we want to use the continuous level-of-detail paradigm instead.

In this paradigm a point cloud data set has an infinite amount of levels-of-detail, which are constructed on demand for a required density of points. The required density is derived from the distance from the user. As opposed to discrete level-of-detail, this paradigm gives a continuous transition where points are “streamed” into the visualization.

While we still use hierarchical level-of-detail for efficient culling and level-of-detail preselection underneath the continuous level-of-detail, the visualization has no discrete transitions.

This is achieved by visualizing only a subset of the points in a level-of-detail if the
CHAPTER 3. VISUALIZATION ON A BUDGET

Figure 3.2: Hierarchical level-of-detail. The included depth of the data structure is higher at close distances, allowing visualization of points of higher levels-of-detail. As distance from the user increases, the included depth becomes lower and as such points of lower levels-of-detail are visualized. Also visible is the efficient hierarchical culling that the data structure provides of points that are out of the field of view of the user.
Figure 3.3: Discrete boundaries in hierarchical levels-of-detail. The range of a level-of-detail is compared with the distance from the user, but from the center of each spatial subdivision. Ergo, some spatial subdivisions from a level-of-detail may not be visualized if they are too far away from the user.

distance from the user is smaller than the range of the level-of-detail but larger than the range of the next level-of-detail (see also figure 3.4). The subset of visualized points is proportional in size to the distance from the user as it is positioned between the two ranges (essentially a linear extrapolation). As such, the density of the level-of-detail is proportional to this as well, and a linear interpolation of density between the two levels-of-detail is achieved. For example, if the distance from the user is halfway between the range of a level-of-detail and the range of the next the level-of-detail, half of the points in the level-of-detail will be visualized, and the density will be exactly between the densities of the two levels-of-detail. The previous level-of-detail will have all its points visualized, and the next level-of-detail will have none of its points visualized.

As the distance from the user changes, the amount of visualized points changes proportionally, and so does the density. This gives a completely continuous transition, and as such allows continuous level-of-detail.
CHAPTER 3. VISUALIZATION ON A BUDGET

Distribution over levels-of-detail

Continuous level-of-detail on top of hierarchical level-of-detail allows us to reduce the complexity of a point cloud visualization by leaving out points efficiently while maintaining perceived density. However, this does require that the points are distributed over the levels-of-detail in such a way that the density of each level-of-detail is correct. The change in density between two levels-of-detail should be inversely proportional to the change in range between them, and should account for the (dominant) dimensions of the data set. For example, if the range of a level-of-detail is half of the range of the previous level-of-detail, and the data set is predominantly two dimensional (for example a aerial LiDAR point cloud data set), the density of the level-of-detail should be four times as much as the density of the previous level of detail. This ensures that the perceived density is always the same.

In a hierarchical spatial data structure, each level-of-detail is also split into spatial subdivisions to accommodate for efficient culling, with the amount and size depending on their depth. This means that the points also have to be correctly distributed over these subdivisions. The change in density between a subdivision and its parent should be inversely proportional the change in space between them. For example, if the space of a spatial subdivision is half of the space of its parent, and the data set is again two dimensional, then the density should be twice as much as the density of its parent.

With a regular subdivision (as used in quadtrees and octrees) this means that every subdivision needs to contain roughly the same amount of points. This is only possible if the data set has a relatively uniform density (see also section 2.3). As described in section 2.1 aerial LiDAR point cloud data sets generally have a relatively uniform density.

It is not surprising that a uniform density emerges as a property required for proper hierarchical level-of-detail in point cloud data sets. A point cloud visualization needs to have a relatively uniform density for the user to be able to perceive shape, and as described in section 2.4 perception of shape is required for an effective point cloud visualization.
3.1. CONTINUOUS LEVEL-OF-DETAIL

Figure 3.5: Distribution of points over levels-of-detail. Points are assigned in their order, first to the lowest level-of-detail until it contains enough points to meet its density requirement, then to the next, and so on. In this one dimensional simplification, each level-of-detail needs (roughly) twice as much points as the previous one to meet its density requirement.

Figure 3.6: The random shuffle is a way of randomizing the order of a set of points. First, a uniformly sampled random number is assigned to each point. Then, they are sorted on these values. The resulting random distribution is guaranteed to be uniform.

Ordering on points

In order to distribute the points over the levels-of-detail correctly, we define an ordering on the points and assign them to each level-of-detail according to their density requirements, starting with the lowest level-of-detail and working our way down along the levels-of-detail (see figure 3.5). During this process, we also assign the points to the correct spatial subdivision of the hierarchical spatial data structure for their level-of-detail. The result is a balanced hierarchical level-of-detail data structure with the points correctly distributed over its spatial subdivisions.

For this to work the order of the points needs to guarantee that each level-of-detail will have a relatively uniform density. Fortunately, if one starts out with a data set that has a relatively uniform density (for example an aerial LiDAR point cloud data set), then a uniform random subselection will produce a subset with a relatively uniform density. So the ordering on the points simply needs to be random.

If a point cloud data set is not randomly ordered, one can “shuffle” it to make it randomly ordered [48]. A simple shuffling algorithm is to assign a uniform random value to each point, and then order the points on these random values (as shown in figure 3.6). This results in a randomly ordered point cloud data set. If the original data set has a relatively uniform density, then the randomly ordered data set will also have a relatively uniform density.
CHAPTER 3. VISUALIZATION ON A BUDGET

3.2 Importance

Now that we can reduce the complexity of a point cloud visualization by using continuous level-of-detail as described in section 3.1, we would like to reduce the complexity of the visualization while minimizing the impact on its effectiveness. To do this, we can take advantage of the limits of the human visual system.

The most important property of the human visual system is perception, which determines a user's understanding of a visualization, or in other words, whether the visualization is effective. Due to the way the human visual system works, perception depends a great deal on visual importance. The human eye is naturally drawn towards the more important parts of a visualization, and the human mind uses the important parts to infer the less important parts of the visualization.

As a result, the less important parts of a visualization contribute less to the perception of the user. This means that the complexity of these parts of the visualization can be reduced with minimal effect on the perception of the user, and as such this minimizes the impact on the effectiveness of the visualization.

A point analogue of visual importance

These less important parts of a point cloud visualization consist of the points that have low visual importance. This means that if we can determine which points have low visual importance, then we can determine which points can be left out of the visualization.

To be able to do this, we introduce point importance. The point importance of a point represents its importance to the human visual system, in visualizations of its point cloud data set. As such, point importance is a point analogue of visual importance, and low point importance indicates points that can be left out.

As points are individual entities, each point individually has importance to the human visual system. Because of this, the point importance of a point can be seen as an attribute of the point (see also section 2.1). As such, it can be defined as follows.

Definition In a point cloud, a point sample has a point importance attribute. This attribute is a number between 0 and 1, where a higher number indicates a higher point importance for that point sample.

Point importance metrics

The point importance is calculated by a point importance metric, an algorithm that measures the importance of a point somehow. Since visual importance is a more global notion and points are unconnected individual entities (as described in section 2.1), such a metric is generally a heuristic that attempts to approximate the visual importance by using information about the point and its surroundings. For example, this information

\footnote{For more information on this subject, one should look into the concept of visual saliency.}
could be the height of a point, the density around a point, or whether a point is part of a feature.

For example, we make use of a point importance metric called the sum-of-squared distances metric, which is a simple density based heuristic that applies well to aerial LiDAR point cloud data sets. Because in such data sets edges and vertical surfaces provide more visual information, this metric deems them more important. The sum-of-squared distances metric is described in detail in section 4.4.

Importance ordering on points

Section 3.1 describes how a uniform random distribution of points over a hierarchical spatial data structure allows the user to continue to perceive shape, while reducing the complexity of the point cloud visualization so that it remains interactive. However, we are looking to take advantage of the limits of the human visual system to reduce the complexity of a visualization while actually minimizing the impact on its effectiveness, by leaving out the points that have low point importance.

As described, the point importance of a point represents its importance to the human visual system. If we order points on this importance, and use this ordering to distribute them over the levels-of-detail as described before (see also figure 3.5 again), then as the level-of-detail increases points become less important to the human visual system. In the resulting point cloud visualization, the less important points will always be left out of the visualization before the more important points. This way, the visualization prefers to visualize the points that are more important to the human visual system, and as such the impact on the effectiveness of the visualization is minimized (see section 3.2).

Weighted random ordering on points

Taking advantage of the upper limits of the human visual system by ordering points on importance entails a distribution of the points over the levels-of-detail that allows a point cloud visualization to prefer to visualize points that are more important. However, it turns out that this can actually exceed the lower limits of the human visual system. As described in section 2.4, a point cloud visualization can only be effective if the user is able to perceive shape, and as described earlier in this section this requires the levels-of-detail to have a relatively uniform density. As importance is not required to be uniformly distributed over points, ordering points on importance no longer guarantees that each level-of-detail will have a uniform density, and as such the resulting point cloud visualization will be very ineffective at having the user perceive shape from them.

In order to address this issue, instead of ordering points on their importance directly, one can use the importance to produce a weighted random ordering on the points. Essentially, points are still ordered on their importance, but they have a chance of moving up or down the order. This allows more important points to sometimes be ordered down and less important points to sometimes be ordered up.

This is achieved by using a normal distribution with the importance of a point as its mean instead of a uniform random distribution, as shown in figure 3.7. If the normal
distribution is parametrized with a properly selected standard deviation, this shuffle can produce an ordering on importance that guarantees that each level-of-detail will have a relatively uniform density. However, it should be noted that this is a parameter that has to be tuned manually per data set.

When using this ordering to distribute the points over the levels-of-detail, the resulting point cloud visualization will leave out less important points before it will leave out more important points, but not to the extent that its relatively uniform density is lost. In fact, given a level-of-detail, the ordering should allow the continuous level-of-detail to provide the points that best represent the point cloud data set for that level-of-detail (given the used metric). So when this ordering is used in combination with continuous level-of-detail on top of hierarchical level-of-detail, we are able to take advantage of the limits of the human visual system to reduce the complexity of a massive LiDAR point cloud visualization so that it remains interactive, while minimizing the impact on the effectiveness of that visualization. However, one should note that the level-of-detail has to be set manually to provide the correct amount of interactiveness during the visualization, and as such is a parameter that has to be tuned manually.

3.3 Automatic level-of-detail adjustment

As described in section 3.2, continuous level-of-detail can be used to change the complexity of a point cloud visualization and as such change the interactiveness of that visualization. This requires a level-of-detail parameter. However, this level-of-detail

Figure 3.7: Like the random shuffle, the weighted random shuffle assigns a random number to each point, and then sorts them on these values. The random number is however sampled from a normal distribution with its mean set to the point’s importance. Its standard deviation provides control over the strength of the effect.
parameter has no direct relation to the actual interactivity of a visualization. As a user interacts with a point cloud visualization, different levels-of-detail are required to maintain interactivity.

**PID controller**

In order to enable this final step in visualization on a budget, we use the concept of a PID controller to control the interactivity of a visualization [49].

A PID controller is a feedback loop that can be used to steer processes to a desired outcome. It uses the output of a process to change the input of a process, by using the proportional, integral, and differential of the difference between the current output and the desired output. The proportional is the difference between the current and desired output, and represents the current error. The integral is the sum of the previous differences, and represents the past error. The differential is the difference between the current and previous differences, and represents the future error. A PID controller uses carefully tuned factors of these errors to adjust the input of a process, steering the output of a process towards the desired output.

Our controller is given a target interactivity. The error of the current interactivity with the target interactivity indicates how much the interactivity needs to change. It uses the magnitude of this value to adjust the level-of-detail. Because decreasing the level-of-detail decreases complexity and as such increases interactivity, the level-of-detail is decreased if the error is positive and the other way around. This process moves the current interactivity towards the target interactivity in steps, where the magnitude of the error decreases per step and as such the change in level-of-detail also decreases per step. This decrease in change of interactivity allows the current interactivity to ease in and settle on the target interactivity.

**Dealing with changes**

As the user interacts with a point cloud visualization, the interactivity constantly changes as the current level-of-detail is no longer correct for the new visualization. The controller is capable of dealing with this as it always depends on the current error. This also means that it is capable of dealing with changes in the budget in general, because changing the level-of-detail will change the load on the entire point cloud visualization pipeline. For example, if another process requires CPU time, this can lower interactivity if the visualization is using a lot of CPU time. If this happens, the controller will change the level-of-detail to reduce the load on the CPU, so that interactivity is again maintained.

In this chapter, we have described how to provide an interactive visualization of massive LiDAR point clouds by using visualization on a budget. We have also described how to reduce the complexity of a point cloud visualization, by using continuous level-of-detail. We have suggested that by taking advantage of the limits of the human visual system through point importance, which is calculated by a heuristic metric, we can
order the points in such a way that continuous level-of-detail always provides the best visualization, given that metric. Finally, we have described how interactivity can be maintained, by using a PID controller that adjusts the level-of-detail in steps until the required interactivity is achieved.
Chapter 3 describes visualization on a budget as our method to interactively visualize massive LiDAR point clouds, and point importance as our method to reduce the complexity of a point cloud visualization while minimizing the impact on its effectiveness. To implement these methods we developed a prototype toolchain for point cloud visualization.

The prototype toolchain (shown in figure 4.1) is a collection of steps that enable us to implement and evaluate interactive visualization of massive LiDAR point clouds. Each step of the prototype toolchain is designed to be modular so that it can be implemented, optimized, replaced, or evaluated separately. The steps consist of a preprocessing toolchain, a point cloud visualization pipeline, and a set of evaluation tools. The preprocessing toolchain is capable of reading a massive LiDAR point cloud data set, calculating the importances for its points, and generating the data structure for it. The point cloud visualization pipeline is capable of interactively visualizing it. The evaluation tools are described in chapter 5.

The prototype toolchain is a natural extension of the workflow of our research (shown in figure 4.2), as we traversed all the steps of the prototype toolchain largely in order until an initial cycle was complete and a working prototype was developed. This initial cycle included adapting the original toolchain (described in section 4.1) into our new toolchain (described in section 4.2), the development of an improved data structure (described in section 4.3), an importance metric experiment (described in section 4.4), a continuous level-of-detail experiment (described in section 4.5), and a visualization on a budget experiment (described in section 4.6). The evaluation is described in chapter 5.

The main reasons we focused on developing an actual toolchain however, were to get acquainted with the software, data structures, and algorithms involved, and to allow future researchers to optimize, replace, or evaluate its steps now that the initial cycle has been completed (see section 6.2). It provides them with a platform on which they can perform more valuable comparisons between methods.

4.1 Original toolchain

When this research was started, there was already a toolchain available for the visualization of massive point cloud data sets [9]. This toolchain was developed by Gerwin de Haan at the Delft University of Technology during earlier research into the same subject. It has served as a basis for the new toolchain.

The original toolchain consists of two tools, a preprocessing tool to generate a data structure from the point cloud data set, and a visualization tool to display the point cloud
Figure 4.1: The prototype toolchain. At the right of the dashed line, the three steps are visible. At the left of the dashed line, the tools for these three steps are visible, as separated by the horizontal lines.
4.1. ORIGINAL TOOLCHAIN

Figure 4.2: The workflow of our research. Each step of the prototype toolchain warranted an implementation, and we developed these largely in order.

data set using that data structure. They consist of a number of components, which are described here.

**OpenSceneGraph**

The tools make use of an open source library and toolset called OpenSceneGraph\(^1\). OpenSceneGraph is an implementation of a *scene graph*, a design pattern used for the display of three dimensional data that is based on objects living in a scene. The objects are implemented as nodes in this graph, and as such the graph can allow for efficient traversal of all the objects in the scene, which is required for all basic operations in the visualization pipeline. OpenSceneGraph supports many types of objects, but the tools only need the more basic ones for the point cloud data structure due to the point data being very simple. OpenSceneGraph uses OpenGL\(^2\) to generate its graphics.

**osgswig**

The tools are both implemented in Python\(^3\), a high level programming language. This allows for rapid prototyping, a software design method where features can be added in

---

\(^1\) [http://www.openscenegraph.org](http://www.openscenegraph.org)

\(^2\) [http://www.opengl.org](http://www.opengl.org)

\(^3\) [http://www.python.org](http://www.python.org)
quick succession without having to design all the software beforehand. This is very useful for research. OpenSceneGraph however is implemented in C++, a lower level programming language. It doesn’t lend itself for rapid prototyping very well, but its performance is very good. The tools are connected to OpenSceneGraph through a software layer called osgswig, an open source Python binding for OpenSceneGraph. This way, one can enjoy the rapid prototyping of Python, but the performance of C++.

Preprocessing tool

The preprocessing tool supports point cloud data sets that are stored in a text file, a so-called XYZ file. This file consists of a number of lines, with each line consisting of a set of numbers separated by whitespace or interpunction. A line represents a point, and the numbers represent its attributes (see also section 2.1). The preprocessing tool supports a limited number of separator characters, and it supports only position, intensity, and color attributes. It is capable of reading a variety of orders and combinations of these attributes, but not all of them.

The preprocessing tool performs two passes over such a file. During the first pass, it gathers some statistics of the point cloud, such as the amount of points in the point cloud. During the second pass, it uses these statistics to create and set up an initial data structure, after which it evaluates and stores the point data in the data structure. Finally, it stores the data structure as a set of files. These files are generated by OpenSceneGraph, and use a format native to it.

Data structure

The data structure produced by the preprocessing tool and its implementation in OpenSceneGraph are shown in figure 4.3. It consists of a hierarchical tree of PagedLOD nodes, with a Group node at the top of the tree. The group and each paged level-of-detail contains a Geode node, a node designed to store geometry. As OpenSceneGraph uses OpenGL for graphics, a geode contains a Geometry object, which contains a PrimitiveSet object and a number of VertexArray objects, with the primitive set containing the point primitives and the vertex arrays containing the actual point data.

The hierarchical structure of the tree implements a quadtree or octree space division (described in section 2.4), which allows use of OpenSceneGraph’s view frustum culling optimization. The group and each paged level-of-detail implements the part of a level-of-detail for its spatial subdivision (described in section 3.1), which allows use of OpenSceneGraph’s level-of-detail optimization. Finally, each paged level-of-detail also provides paging, which allows use of OpenSceneGraph’s out-of-core memory management.

Visualization tool

The visualization tool loads the data structure from the files as needed and visualizes it using OpenSceneGraph. It implements support for easy navigation using a space mouse (shown in figure 4.4), and it is built upon another Python/C++ library called VRmeer,
4.2 LAS support

The very first thing that we did was implement support for LAS files in the original preprocessing tool using libLAS\footnote{http://www.liblas.org}, an open source library for reading and writing LAS files. Fortunately, libLAS has a Python binding available, which allowed us to use it in the preprocessing tool so we could continue to use rapid prototyping.

The support for LAS files was implemented for a number of reasons. First, LAS is a de facto standard file format for the exchange of point cloud data recorded by LiDAR technology. It supports various attributes such as position, intensity, color, and time, but also technical ones such as scanner angle or number of returns. Support for LAS files prepares us to preprocess any LAS files we may obtain in the future.

Furthermore, the original preprocessing tool was not capable of handling all combinations and separators that may occur in the XYZ files described in section 4.1. Replacing the XYZ reader with a LAS reader allows us to keep our preprocessing tool simple and compact, because we don’t have to allow for all possible cases.

LAS files also store a header with information about the point cloud data set such as its amount of points and its extent, so the preprocessing tool no longer has to perform two passes over a data set. This again keeps our preprocessing tool simple and compact, and improves performance.

Figure 4.3: The data structure of the original toolchain. Its quadtrees or octrees space division supports view frustum culling, and its paged levels-of-detail support multiresolution representations and out-of-core access.

which allows for easy stereoscopic visualization and additional rapid prototyping \cite{8}. An example visualization using this tool is shown in figure 4.5.
CHAPTER 4. PROTOTYPE TOOLCHAIN

Figure 4.4: The space mouse is a device for navigating in three dimensional space. Its central controller can be twisted for rotation, and nudged for translation, along all three major axes.

Figure 4.5: A frame from a flythrough visualization done by the original visualization tool. It shows a part of the campus of the Delft University of Technology.
libLAS toolchain

Finally, libLAS comes with a set of versatile open source preprocessing tools that enable us to perform more preprocessing actions than we were able to before, and with relative ease. They all work from the command-line and have a similar interface, and as such can also be scripted.

For example, XYZ files can be converted to LAS files using the `txt2las` tool, which supports all XYZ file formats by allowing one to specify the format on the command-line.

The `las2las` tool allows various filtering options, such as noise removal, but most importantly the coloring of points using aerial photography. This is required to preprocess the AHN2 data set. The original preprocessing tool also already did this, but having it in an external tool again allows us to keep our preprocessing tool simple and compact. It also allows us to color data sets once and then store them as LAS files for future use, something that was not possible with the original preprocessing tool.

The `lasmerge` tool allows us to easily merge multiple data sets. Finally, the `lasinfo` tool allows one to inspect a point cloud data set from the command-line.

Issues with LAS support

Unfortunately, there turned out to be some downsides as well. LAS files contain a lot of different attributes per point, whether they are empty or not. This results in LAS files being only somewhat smaller than the source XYZ files, even though it is a binary format.

The most unfortunate problem however is the performance of actually reading from a LAS file. Since our preprocessing tool is written in Python, the reading is done through the libLAS Python binding, and this turns out to be very slow. However, as this is a preprocessing tool, the performance is not critical to the point cloud visualization, and we simply adopted the versatility and partially addressed the issue by performing the preprocessing on a powerful server.

`las2osg`

Because the libLAS toolchain has a somewhat unified interface, we adapted our preprocessing tool to have a similar command-line interface, and we dubbed it `las2osg`. This was the first step towards the development of the prototype toolchain.

4.3 Data structure improvement

Now that we were able preprocess data sets we were unable to preprocess before, we encountered a data set that provides a very poor performing visualization when preprocessed into the original data structure (described in section 4.1). This is the Kop van Zuid data set. It measures 1.5km by 1.5km by 200m, and as such has a sizable third

---

5We would like to thank Gemeentewerken Rotterdam for providing the Kop van Zuid data set.
Figure 4.6: The empty cells in an octree data structure when used on a predominantly two dimensional data set.

Figure 4.7: The bounding sphere as it applies to a cell near the bottom of a quadtree. As the cell is narrow, but very high, the bounding sphere will contain a lot of space that is not part of the quadtree cell.
dimension when compared to its other two dimensions.

Normally, such a data set is a prime candidate for an octree, to deal with the size of the third dimension. However, as this size is caused by a few high buildings, such an octree will contain a lot of empty cells (as shown in figure 4.6). The original data structure includes and stores these empty cells, which during a visualization are constantly retrieved even though they add nothing to the visualization. This massively increases data retrieval as there are a very large amount of these empty cells, and as such a poorly performing visualization is provided.

If such a data set is preprocessed into a data structure based on a quadtree instead of an octree, then the data structure will not contain such empty cells. However, each cell of the quadtree will be very high as it is not spatially divided along the third dimension. Normally, this would not be a problem, but OpenSceneGraph uses bounding spheres instead of the more traditional bounding boxes to enable efficient hierarchical culling, as they are more efficient than bounding boxes when checking for intersections with the view frustum. Such a bounding sphere, when fitted to a quadtree cell, is forced to contain a lot of space that is not part of the quadtree cell (as can be seen from figure 4.7). This is a natural side effect of using bounding spheres that is normally offset by the gain in hierarchical culling efficiency. However, for such a quadtree cell it causes intersection with the view frustum when it is not visible to occur much more often than usual, and this means OpenSceneGraph will retrieve the data contained by the quadtree cell and then immediately cull it a lot. Again, this massively increases data retrieval and as a result provides a poorly performing visualization.

Additionally, as can be seen from figure 4.7 the original data structure fits the bounding spheres to the extents of a cell, and not to the data in that cell. This causes the bounding sphere to encompass even more space that is not part of the cell.

We were able to research and understand these issues by adding a bounding sphere visualization option to the preprocessing tool. It produces an identical data structure that contains spherical shapes instead of the point data, where the spheres are sized according to the bounding sphere of the cells.

We addressed the issues by fitting the bounding sphere for a cell to the data of that cell, and by removing all the empty cells from the data structure. This massively decreases the amount of data retrieval. It allowed us to use an octree when preprocessing the Kop van Zuid data set. As this data set has an irregular shape, this actually even removed empty cells found along the other two dimensions, as can be seen from figure 4.8. This optimization allowed us to interactively visualize the Kop van Zuid data set to an acceptable degree.

Index

The original data structure stores each cell of the quadtree or octree separately, as well as the data in the cell, to enable out-of-core internal memory usage. This means that in order to be able to cull a cell, OpenSceneGraph first has to retrieve that cell. After that, if a cell is deemed in view, the data in the cell is retrieved. This too increases data retrieval and as a result this too adds to the poorly performing visualization. Additionally, as a
cell itself contains little data, the file for a cell is very small, offsetting its use versus its retrieval time.

In order to address this issue we converted the data structure to consist of an “index” with only the actual data being available out-of-core (shown in figure 4.9). The index is the quadtree (or octree) of efficiently cullable nodes, and is retained in memory at all times. It is retrieved only once, instead of being constantly managed in an out-of-core manner. This is possible because its memory usage is actually quite low.

We also addressed an oversight that occurred at the bottom cells in the tree that caused the spatial subdivisions for the highest level-of-detail to behave like its parent, where either all four or none of the subdivisions were included in the visualization. This was because a shortcut meant as an optimization caused them to measure their distance from the user from the same location, essentially making them behave as a single spatial subdivision of the same size as their parent. The index provides a massive performance boost in hierarchical culling as no data retrieval is necessary anymore to determine which cells are in view.

**Scalability through tiling**

While the index provides better performance, it should be noted that it is not scalable. A single index for a very large data set could potentially have memory usage issues. To address this we simply make use of scalability through tiling. Since the memory usage of the preprocessing tool has always been linearly proportional to the size of the data set, we use las2las to create tiles from data sets and preprocess these tiles individually. Some data sets, such as the AHN2 data set, are already available in tiles. This is not the ideal preprocessing solution, but for now it is sufficient.

Using this notion, we have developed an out-of-core superstructure that is essentially a list that retrieves separate index tiles as needed. This means that they may be retrieved
4.3. DATA STRUCTURE IMPROVEMENT

Figure 4.9: Example of an index in a data structure. Green circles represent the cells, yellow circles represent the data in the cells. The solid lines indicate the index, the dashed lines indicate the out-of-core data.

more than once, but only if they have gone completely out of view. However, for this to be truly scalable, this needs to be a hierarchical superstructure, that itself should also be an out-of-core index (for example, as shown in Figure 4.10). As long as each substructure stays within internal memory limits, the exact shape of the superstructure is open for development. For example, for the AHN2 data set, we envision a superstructure based on its innate TOP10 hierarchical map assignment system, as its tiles are already allocated using that system [37].

Issues with the index

However, the index was not without its problems. The first issue we encountered was that data was retrieved as it came in view, but not disposed of when it was no longer in view. As such the internal memory usage quickly exceeded its limits. OpenSceneGraph provides a native out-of-core implementation, which is handled by a part of it known as the data base pager. This data base pager makes sure paged levels-of-detail dispose of their children when they need to, but for this to work they have to be registered with the data base pager first so it can find them. In the original visualization tool, this registration was automatically performed by OpenSceneGraph as paged levels-of-detail were retrieved. However, when using the index they are no longer retrieved, and as such they have to be registered with the data base pager manually by the visualization tool. This required a small change to the visualization tool.

The second issue was that while the index can remain in internal memory completely, optionally as part of an out-of-core superstructure, it turns out that the interactivity decreases as the index becomes larger. As described the data base pager gives each paged level-of-detail registered with it a chance to dispose of its children. As this is done every frame, this requires an amount of time linearly proportional to the amount of paged levels-of-detail registered with the data base pager. As the data base pager registers not just the paged level-of-detail given to it, but all the children of that paged level-of-detail that are also paged levels-of-detail, we can only register either the entire index with it, or no index at all. As we need to address our first issue, we are forced to register the
CHAPTER 4. PROTOTYPE TOOLCHAIN

Figure 4.10: Example of an out-of-core hierarchical superstructure. Solid lines indicate cells that are in the same index, dashed lines indicate cells that are in separate indices.

Figure 4.11: The data structure of the new toolchain. Its quadtree or octree space division supports view frustum culling, and its paged levels-of-detail support multiresolution representations and out-of-core access. Compared to the original however, they are arranged in an index and are kept in memory simultaneously, while the bottom of the data structure now has a proper space division. The top group is also replaced with a paged level-of-detail to allow for superstructures to work out-of-core.
entire index with it. This means that for a larger data set, more time is needed every frame, and for very large data sets this once again provides a very poorly performing visualization.

In order to address this issue, we had to make changes to OpenSceneGraph itself. As described, the data base pager registers paged levels-of-detail recursively, and this causes the entire index to be registered. However, the point of the hierarchical data structure is to provide efficient hierarchical culling so that the data structure does not have to be completely traversed, allowing for scalability. With this knowledge, we were able to change the data base pager so that it only registers those paged levels-of-detail in our data structure that are part of the visualization. We know that only these paged levels-of-detail retrieve data, and as such only these paged levels-of-detail need to be given the chance to dispose of their children. Once they stop being a part of the visualization, they are deregistered. This change to OpenSceneGraph made it able to handle our index while retaining scalability.

**Improved data structure**

The improved data structure and its implementation in OpenSceneGraph are shown in figure 4.11. Its index consists of a hierarchical tree of PagedLOD nodes that are not paged. Each paged level-of-detail contains a Geode node that is paged. As before, each geode contains a Geometry object, which contains a PrimitiveSet object and a number of VertexArray objects, with the primitive set containing the point primitives and the vertex arrays containing the actual point data.

The improved data structure is implemented in our las2osg tool. It produces an index based out-of-core data structure that minimizes data retrieval, and it has the option of producing a bounding sphere visualization as well. As described, our visualization tool and OpenSceneGraph also required a number of changes to be able to correctly handle the index. The improved data structure signified the second step in the development of the prototype toolchain.

**4.4 Importance experiment**

With las2osg and the improved data structure completed, we were able to start our importance experiment. The goal of the experiment was to see if we could determine some form of point importance for the points in our data sets by using nearest neighbour retrieval.

**Sum-of-squared-distances metric**

Therefore, we developed and implemented a very basic, initial point importance metric that uses nearest neighbour retrieval (shown in figure 4.12). We intended to replace this metric once the initial cycle was complete, but it turned out to work better than expected. As such, we decided that this metric was good enough not to put any more effort into developing a better metric as part of this research.
Figure 4.12: The sum of squared distances metric. First, the neighbouring points within a particular range are retrieved (1). Then, the squares of the distances to those neighbours are summed (2). These values are gathered for all points in the data set. The minimum and maximum of the values are determined, and each value is then normalized. Since each value actually gives an indication of the unimportance of a point (3), its complement is taken so that it represents the importance of the point (4).
We call this metric the sum-of-squared-distances metric (SSD metric), as it works by first retrieving the neighbouring points of a point within a range, and then summing the squares of the distances of those neighbours to the point in question. It then determines the minimum and maximum of the resulting values, and normalizes them to fall between 0.0 and 1.0. Since the resulting values are actually a measure of the unimportance of a point, it takes the complement of the value as that is a measure of the importance of the point.

**Nearest neighbour retrieval**

Retrieving the nearest neighbours of a point in a massive LiDAR point cloud data set is not an easy operation. A naive implementation that evaluates all the points in such a data set is very inefficient. In order to optimize this operation, kd-trees (also seen in section 2.4) can be used [3]. As such, we use a kd-tree to efficiently retrieve the nearest neighbours within range.

We use an open source library called Approximate Nearest Neighbours (ANN) for this. ANN has a Python wrapper available called scikits.ann which allows us to use it in our rapidly prototyped tools. ANN constructs a kd-tree of a collection of points given to it, and allows efficient retrieval of the points from that collection that are nearest to a location in space. It supports both approximate and exact nearest neighbours, with the former method providing a performance advantage. We use the exact nearest neighbours method for accuracy purposes. The library requires one to specify the location in space and the amount of points to return (and optionally the error that is allowed), and returns a list of the specified size, containing the nearest points and their squared distances to the specified location in space. Since we actually want to retrieve all the points within a certain range, as opposed to a certain number of them, we devised a method where the library is repeatedly called with increasing list sizes until points that are out of range are found. If a point that is out of range is found, then all nearer points have already been found, and we can be sure that we have all points within range.

Unfortunately, as described in section 2.4, the construction of data driven spatial data structures such as the kd-tree is not very efficient for point cloud data sets. Indeed, it turns out ANN is not able to cope with our data sets within the limits of available internal memory and time. In order to address this issue, we developed an optimization that reduces the size of the data set for individual groups of points, shown in figure 4.13. It works by bucketing the points from a point cloud data set into a grid. Even though the points are three dimensional, we chose the grid to be two dimensional as a compromise between memory usage and kd-tree construction time. Each cell of the grid is traversed individually, and during the traversal of a cell a kd-tree is constructed from the points in that cell and its directly neighbouring cells. For each point in the cell, this kd-tree is then used to retrieve the points in range. The cells have a dimension that is slightly larger than the range required during the retrieval, which guarantees that the range for a point in a cell always falls within the boundary of the directly neighbouring cells. This


Figure 4.13: The grid optimization used for the SSD metric. Points from a point cloud data set are bucketed into a grid according to their position in space. Each grid cell is then handled separately. When a grid cell is handled, a kd-tree is constructed from the points in that grid cell and its surrounding grid cells, to enable nearest neighbour retrieval for the points in that grid cell. The grid cells are sized in such a way that the range of the retrieval is always guaranteed to fall within the boundaries of the grid cells used to construct the kd-tree.

way, the kd-tree is guaranteed to provide all the neighbouring points within range. This method provides enough efficiency to calculate the SSD metric within acceptable times, although the majority of the time is still used to construct the kd-trees.

**Application to aerial LiDAR point cloud data sets**

As mentioned, we developed the SSD metric to provide an initial implementation of the importance calculation step of the prototype toolchain. The main reason was to find out if we could determine nearest neighbours efficiently, as we deemed that the most promising importance heuristic. The primary reason it sums the squared distances to its nearest neighbours is because that is a simple operation that provides a heuristic of the density around a point, especially since ANN already provides us with the squared distances.

However, as it turns out the SSD metric actually has some merit, and we were able to use it during the remainder of our research. This is mostly because the metric has some unexpected features when it is used for aerial LiDAR point cloud data sets (shown in figure 4.15).

Usually when using a density heuristic, high density deems points less important as they provide less visual information per point. Low density deems points more important...
Figure 4.14: The effect of the very low point importances assigned by the SSD metric to points in swath overlay areas. The weighted shuffling process produces a distribution with a nonuniform density and this is visible in the visualization. Also visible is the effect of the high importances assigned by the SSD metric to points on the boundaries of tiles.

Figure 4.15: The application of the SSD metric to a typical section of an aerial LiDAR point cloud data set. The horizontal surface has a high uniform density, and the vertical surface has a very low density. The point near the edge is assigned a higher importance because the further away neighbours are missing, and the vertical surface doesn’t interfere because of its low density.
because they provide more visual information per point. The SSD metric indeed favours points that have few neighbours, due to the summing of the distances, and favours points less as more neighbours are within range. However, it also favours points less as neighbours are further away, due to the squared distance being used in the sum. This is counterintuitive, but appears to work well for aerial LiDAR point cloud data sets.

Aerial LiDAR point cloud data sets are predominantly flat, and they are recorded from the air using a sweeping LiDAR scanner that points down towards the surface of the earth (see also section 2.2). This causes these data sets to have horizontal surfaces with high uniform densities, and vertical surfaces with very low densities.

The metric deeming a point nonlinearly less important if neighbours are further away has no effect if the data has a uniform density, as is the case with the horizontal surfaces in these data sets. However, the metric deems a point nonlinearly more important if these further away neighbours are missing. This is the case around the edges of the horizontal surfaces because any adjoining vertical surfaces have very low densities, essentially causing the metric to be able to detect edges in these data sets and deem the points near those edges to be more important. This is a very useful result, as edges provide much information to the human visual system and as such have high visual importance.

Unfortunately, as the preprocessing toolchain uses scalability through tiling, this also causes the SDD metric to erroneously deem the boundaries of the tile as more important as there is no adjacent point cloud data available (the effect of which is visible in figure 4.14). This could addressed by simply including the directly neighbouring tiles during the importance calculation for a tile, much like in the grid optimization.

Finally, the vertical surfaces are deemed very important due to their very low density, and this is also useful as in aerial LiDAR point cloud data sets vertical surfaces provide more information to the human visual system than horizontal surfaces due to the data set already being predominantly flat.

**Range parameter**

As mentioned, the SSD metric retrieves neighbouring points within a certain range. This range is a parameter of the metric, and has to be set manually.

A range parameter that is set too high causes importance to change very gradually near edges, and hardly ever over surfaces with uniform densities. A range parameter that is set too low causes importance to change very sharply near edges, and even over surfaces with relatively uniform densities.

As the range parameter increases, the changes in importance become more gradual. With a larger range, more neighbouring points are part of the calculation done by the SSD metric, and as such local differences in density matter less. Near edges, a larger range causes more inward points to enjoy the higher importance given by the presence of the edge. Figure 4.16 shows the effect of the range parameter on a part of the AHN2 data set.

The correct setting of the range parameter depends on the density of the data set. As can be seen from figure 4.16, a range parameter of 3 meters seem to work well for this data set. The horizontal surfaces no longer show any irregularities but the edges are still
Figure 4.16: Importance as assigned by the SSD metric, with increasing range parameter. The color scale used in the importance visualizations runs from red to purple, and then from blue to yellow. Red indicates very low importance, purple and blue indicate low importance, and yellow indicates high importance. Note the low importance of the horizontal surfaces, the high importance of the vertical surfaces, and the high importances of the edges. As the range increases, the pronunciation of the importance starts to fade.
Swath overlays

The biggest issue we ran into when using the SSD metric for aerial LiDAR point cloud data sets was that the data sets contain areas of hyperincreased point densities (shown in figure 4.17). Aerial LiDAR technology records point cloud data sets in strips known as “swaths” (shown in figure 2.4) and when merging these swaths the overlapping areas (that we have dubbed “swath overlays”) have a density that is essentially two times as large. The SDD metric relies on absolute density to determine importance, so in its previously described form it incorrectly assigns lower importance to the points in those areas (shown in figure 4.18). Normally, the weighted shuffling process described in section 3.1 produces a distribution over the levels-of-detail that gives them relatively uniform densities, but in such sizable areas where many of the points have a very low importance this process fails and the resulting distribution gives them a extremely nonuniform densities (the effect of which can be seen in figures 4.14 and 4.19).

Two-pass sum-of-squared-distances metric

From figure 4.18 we can deduce that the swath overlays constitute only a small amount of the points, but are responsible for a very skewed importance distribution over the
Figure 4.18: Importance as assigned by the SSD metric, with a range parameter of 3 meters. Visualizations of the importance are at the top, histograms of the importance as it appears in the data set are at the bottom. The histograms show the color scale used to indicate the importance in the visualizations. The color scale runs from red to purple, and then from blue to yellow. The red to purple areas signify the regions of very low importance, caused by the swath overlays. From the histograms it can be seen that these regions constitute a relatively small part of the data set.
Figure 4.19: Comparison of the importance distribution resulting from the one-pass SSD metric and the importance distribution from the two-pass SSD metric. The left column shows the distribution and its effects after the first pass. The right histogram shows the distribution and its effects after the second pass. Note that the two-pass histogram has almost 5 million points with an importance of 0.0, a side effect of the second pass.
points in the point cloud data set. Less than five million points, only one fifth of the
data set, take up the importance range from 0.0 to 0.625. We can see that all these
points lie within the swath overlays. Therefore, we decided to experiment with a second
pass for the SSD metric. This pass sets the offending points to have an importance of
0.0, and rescales the importance of the other points accordingly. This frees up the entire
importance range of 0.0 to 1.0, and produces a more balanced importance distribution
(as can be seen in figure 4.19). As a result, the weighted shuffling process is able to
produce orderings that guarantee that each level-of-detail will have a relatively uniform
density (also shown in figure 4.19). However, some information is lost as the offending
points are all deemed equally important. On top of that, the second pass also requires a
parameter, the importance below which to set the importances to 0.0, and this parameter
also has to be set manually.

**Issues with the parameters**

The range parameter, the weighted shuffle parameter, and the second pass parameter all
have to be set manually. This behaviour is undesired as it requires user intervention as
part of the prototype toolchain. As a visualization already has to be available to a user for
that user to be able determine the parameters, data sets will have to be preprocessed with
one or more essentially guessed parameters before the proper parameter with which they
have to be preprocessed can be determined. This introduces unnecessary backtracking
in the prototype toolchain.

To address these issues one could attempt to automatically determine the parameters
from the point cloud data set. However, the sum-of-squared-distances metric was never
intended as a “final” metric, and given its problems with varying density, we recom-
mend replacing it altogether. This is further explained in section 6.2. In future metrics
automatic parameter estimation may still be required, however.

**las2las2**

The importance calculation is implemented in a Python tool we have dubbed *las2las2.*
As a part of the preprocessing toolchain, it has a similar interface to las2osg and the
libLAS toolchain.

It is capable of reading a point cloud data set from a LAS file, calculating the impor-
tance for each point in the data set using one or more point importance metrics, and
writing the resulting “importanced” point cloud data set to another LAS file. It can also
read importance values previously stored in a LAS file, and incorporate them in a new
calculation. This allows for the second pass of the SSD metric, for example.

As LAS files have no innate support for point importance, it stores the importance
of a point in the time field for that point. The time field is not otherwise used for its
original purpose by our toolchain. It is nicely available as a 64 bit floating point value
in Python, which is required to properly store the importance for a point.

To be able to evaluate the importances calculated by las2las2, we adapted las2osg to
store the importance of a point in the alpha channel of its color. Using a shader, the
point importances can be visualized as a color (as visible in for example figure 4.16). For this we adapted the visualization tool to support a number of such shaders.

Like las2osg, las2las2 suffers from the slow reading of LAS files through the libLAS Python binding. However, as it needs to perform multiple passes over LAS files because the point cloud data sets in them are too large to be kept in internal memory, it suffers even more. Additionally, the construction of the $kd$-trees also requires a lot of time.

Again, as the preprocessing performance is not critical to a point cloud visualization, we only partially circumvented these issues by using a powerful preprocessing server. However, the time required by las2las2 still remains somewhat of an issue.

4.5 Continuous level-of-detail experiment

With the calculation of importance in place, we continued with a continuous level-of-detail experiment. During it, we developed the continuous level-of-detail described in section 3.1, using point importance to improve the effectiveness of the visualization.

Simulation of continuous level-of-detail using importance

Given that las2osg is able to store the importance for a point in the alpha channel of the color of the point, we started out with a “fake” visualization to get an idea of what continuous level-of-detail using point importance might look like in a point cloud visualization. We developed a shader that gradually showed points over time in the order of their importance, and included it in the visualization tool. This approach is a simulation because the actual amount of points visualized is not reduced until very late in the point cloud visualization pipeline when using a shader, and practically no performance is gained.

The resulting visualization had a very “converging” feel to it, where the less important points (which are visualized after more important points) mattered less to the change in effectiveness. This was the kind of effect we were looking for, as it confirmed our suspicion that less important points can be left out of a visualization while minimizing the impact on its effectiveness (as described in section 3.2).

Discrete level-of-detail using importance ordering

In order to be able to implement continuous level-of-detail however, we first had to adapt las2osg so it could use a point importance ordering to distribute points over levels-of-detail, as described in section 3.1. The resulting data structure already provides a discrete level-of-detail visualization where each level-of-detail contains the points that are more important than the points that are in the next level-of-detail. As the more important points are more prominently featured, this already improves the effectiveness of the visualization when compared to the random ordering used by the original preprocessing tool.

To enable distribution of points over levels-of-detail using point importance ordering, we adapted las2osg to include a first pass where it reads the importance of the points from
4.5. CONTINUOUS LEVEL-OF-DETAIL EXPERIMENT

their time fields in the LAS file, and uses the weighted shuffle described in section 3.1 to randomize them while maintaining a relatively uniform point density. The resulting order is then used to distribute the points over the levels-of-detail as described in section 3.1. This does mean that las2osg needs to keep the entire order in internal memory during the preprocessing process. As we use scalability through tiling, this is not much of an issue, but a streaming approach would be better.

We also adapted las2osg to be able to store the order of a point in the alpha channel of its color instead of the importance. This allows us to visualize the order as a color, again using a shader developed as part of the visualization tool.

Simulation of continuous level-of-detail using order

It also allowed us to perform the “fake” visualization of continuous level-of-detail using point importance with the actual order instead of just the importance. This gave us a better impression what continuous level-of-detail might look like. As one might expect, it looked more uniform as the result of the weighted shuffle was now included in the visualization.

Visual continuous level-of-detail using importance ordering

Given that importance is now included in the levels-of-detail, we were able to implement an initial, visual variant of continuous level-of-detail that does reduce the amount of points visualized by the point cloud visualization pipeline. The variant gradually adapts the amount of points shown per spatial subdivision of the data structure that is in view, until they are all visualized. It is, however, a local approach and works separately per spatial subdivision. This also means that it will eventually visualize all points, providing no complexity reduction and still showing its discrete boundaries. It did however give a good impression of visual continuous level-of-detail, and we discovered that a duration of 2 seconds for showing a spatial subdivision works very well for visual continuous level-of-detail.

We implemented this variant to traverse the data structure, and for each Geode node traversed, to gradually increase the primitive count set in the PrimitiveSet object until it reaches the amount of points available in the VertexArray object (see section 4.3 for an overview of the data structure). However, since we implemented this in Python, the data structure traversal was very slow.

To support this variant, we had to adapt las2osg as well, so it would order the points by their importance ordering within the Geode node’s VertexArray object. This is needed because increasing the primitive count includes more points in the visualization in the order they are stored in the VertexArray object, and if they are stored in order of their importance ordering, the resulting visual continuous level-of-detail will automatically use point importance ordering.

We also had to adapt las2osg to store 1 as the primitive count in the PrimitiveSet object instead of the amount of points available in the VertexArray object, so that the
visual continuous level-of-detail would start with a single point. We would prefer to start with no points, but unfortunately this causes OpenSceneGraph to crash.

Finally, we had to replace the use of display lists with vertex buffer objects, as display lists are static by nature and cannot have their points changed. Vertex buffer objects are slightly less efficient, but are dynamic and allow changing which of their points are part of the visualization. Fortunately, OpenSceneGraph allows us to simply specify which one to use, and we implemented this in las2osg.

**Continuous level-of-detail using importance ordering**

Although it looked promising, the visual continuous level-of-detail was not actual continuous level-of-detail, and its Python implementation was very slow. As such, we decided to implement continuous level-of-detail by changing OpenSceneGraph, much like we did with the database pager to support the index improvement of the data structure (see section 4.3). In this case, we had to change the paged level-of-detail node.

Proper continuous level-of-detail implemented on top of discrete, hierarchical level-of-detail is described in section 3.1. As such, we adapted the PagedLOD node to compare the distance from the user to the its own range and the range of its children (see section 4.3 for an overview of the data structure). This provides a linearly extrapolated factor that can be used to set the amount of points in its Geode node that are visualized by the point cloud visualization pipeline.

The result of setting the amount of visualized points this way is continuous level-of-detail (shown in figure 4.20). It provides instantaneous, seamless transitions between levels-of-detail, and is a global approach. However, while decreasing the amount of visualized points is always seamless, increasing them isn’t necessarily seamless since there may be a delay in the retrieval of the data. As a result, a discrete transition may still occur.

To combat this, we use the previously described visual continuous level-of-detail as well when the amount of visualized points is increased. Contrary to its previous implementation, it is implemented in the PagedLOD node and gradually increases the amount of points towards the amount of points dictated by the continuous level-of-detail. From its previous implementation, we know that two seconds is good transition time for the entire set of points, so for the amount of points dictated by the continuous level-of-detail we use a proportional duration.

Note that this implementation of continuous level-of-detail requires all the changes made to las2osg for visual continuous level-of-detail, as well as the changes to OpenSceneGraph.

**Continuous level-of-detail using random ordering**

As can be seen from section 3.1, continuous level-of-detail doesn’t actually need point importance to function. As such, we also implemented a variant that uses simple random ordering instead of point importance ordering.
4.5. CONTINUOUS LEVEL-OF-DETAIL EXPERIMENT

Figure 4.20: Continuous level-of-detail using random ordering (left) and importance ordering (right). The metric used to calculate the importance is the SSD metric from section 4.4. This metric assigns higher importance to edges and vertical surfaces, improving the effectiveness of the visualization. The more pronounced edges and the higher density of the vertical surfaces is visible in the comparison.

Interestingly, this variant of continuous level-of-detail also provides rather effective visualizations (see figure 4.20). However, as the ordering is uniform and random, each point is simply treated equally. On the other hand, this visualization doesn’t suffer from importance related problems such as swath overlays.

Issues with continuous level-of-detail

There are still a few issues with this implementation of continuous level-of-detail. The data structure underlying the continuous level-of-detail is still a discrete, hierarchical data structure. As such, its out-of-core data is still retrieved in blocks. This has its advantages, as retrieving blocks of data is generally efficient when dealing with hard disks.

However, this is not the case when the data is retrieved over the network. They also need to be retrieved all the way before they can be used, and this introduces a delay (which we currently address with visual continuous level-of-detail). Many of such requests simultaneously can stress the data retrieval bottleneck. Finally, it also means that all the points of a spatial subdivision are retrieved even if not all of them are visualized.

In order to address these issues, a streaming implementation could be used, that constantly retrieves points one by one. This doesn’t cause any sizable delays, reduces the use of the bottleneck, and retrieves only the points that are visualized.

Finally, the continuous level-of-detail currently requires a custom version of OpenSceneGraph specifically tailored to continuous level-of-detail visualizations. As such, this OpenSceneGraph can no longer be used for discrete level-of-detail visualizations.
4.6 Visualization on a budget experiment

With working continuous level-of-detail, we were able to move on and implement the final part of visualization on a budget, the automatic level-of-detail adjustment. As explained in section 3.3, the level-of-detail required for an interactive visualization is not always the same. Therefore, we need to be able to change it during the visualization.

Level-of-detail scale

While the ranges of the levels-of-detail are not actually fixed, updating them all requires a lot of traversal. Instead, OpenSceneGraph includes a level-of-detail scale, a number that is multiplied with the distance from the user to calculate the actual distance used when performing the comparison with the ranges and deciding which discrete levels-of-detail are visible.

One can adjust this level-of-detail scale to control which discrete levels-of-detail are visible in a visualization-wide manner. With our continuous level-of-detail implementation, one can even use this to control the exact level-of-detail visible. This means one can precisely control the amount of points that are visible, and as a result precisely control the interactivity of the visualization.

PID controller

This can be done during a point cloud visualization, and we have used it before to manually tune the interactivity of a visualization. However, this level-of-detail scale can also be adjusted automatically.

We adapted our visualization tool to include the PID controller described in section 3.3. It works from the frame rate statistic reported by OpenSceneGraph. It moves the current frame rate to the target frame rate by changing the level-of-detail scale (and thus the level-of-detail) up or down in a gradual manner. If the current frame rate is too high, the level-of-detail scale will be increased, and if the current frame rate is too low, it will be decreased. To prevent it from running off, the level-of-detail scale is kept between 0.001 and 10.0.

As this works using only the difference between the current and target frame rate, the controller is able to handle any target frame rate. Because of this it is also able to handle any changes in the budget available for the point cloud visualization pipeline. For example, if too many points are displayed due to navigation causing the current level-of-detail to be too high, the level-of-detail scale will be decreased. If another process, either inside the visualization tool (such as the performance statistics recording mentioned in chapter 5.1) or outside the visualization tool (such as paging), requires CPU time, the level-of-detail scale will be decreased to keep the frame rate at the target.

Tuning the PID controller

PID controllers are however notoriously hard to tune, and require specific tunings for specific environments. Fortunately, we were able to tune our PID controller to work
quick and accurately in nearly all cases, with nearly no overshooting.

As described in section 3.3, our PID controller uses a factor proportional to the difference between the current frame rate and the target frame rate, that we call the rate. Early attempts of tuning the PID controller using only the rate ended up in steady oscillation. The level-of-detail scale would eventually increase to include new discrete levels-of-detail, and during their retrieval the frame rate would temporarily decrease somewhat. This caused the level-of-detail to be decreased again, which immediately caused the discrete levels-of-detail to be disposed of again. This process would then repeat itself.

In order to address this issue we introduced a moving average that takes the average of the previous 5 frame rates to smooth out this out-of-core specific behaviour. Values higher than 5 tended to cause a new kind of oscillation, a slower version that somehow exhibits a delayed reaction to the level-of-detail increase. Values lower than 5 didn’t smooth out the original oscillation enough. We also noted that even values don’t seem to help at all. We suspect this is because the retrieval of the next level-of-detail takes exactly one frame, and as such the moving average always includes that one frame rate that throws the PID controller.

However, even with the moving average the frame rate would still jitter around the target frame rate instead of converging on it. This was especially noticeable around discrete level-of-detail transitions. In order to combat this we included a range of acceptable target frame rates, implemented by a deviation value. The PID controller now stops changing the level-of-detail scale if the current frame rate is less than one deviation under or over the target frame rate. The deviation required for correct convergence appears to be roughly 15% to 20% of the target frame rate.

Our PID controller is technically a P controller, as only a factor proportional to the difference between the current frame rate and the target frame rate is used to adjust the level-of-detail scale. We did try to use a PD controller, which also includes a factor that is proportional to the difference in the current and previous differences. However, this seemed to make little difference in the performance of the PID controller, if any at all.

The end result of all this tuning, and with it this research, is visible in figure 4.21.

**Issues with the PID controller**

The PID controller has some issues as well. The largest issue is that the PID controller appears to be sensitive to the base performance of the visualization pipeline. For example, if the performance of the visualization pipeline is changed by running it on a different machine or turning full screen anti-aliasing on or off, then the PID controller needs to be retuned to be able to cope with the changed base performance.

The origin of this issue is probably that the PID controller is frame based. It changes the level-of-detail scale gradually, performing one step of the change every frame. However, this not only means that its performance is dependent on the base performance of the visualization pipeline, it also means that its performance is dependent on itself! If it increases the level-of-detail scale because the current frame rate is too high, it will slow itself down. If it decreases the level-of-detail scale because the current frame rate is too low, it will speed itself up. This issue could possibly be addressed by including the frame
time in the calculation of the gradual change in the level-of-detail scale.

Finally, we experience incorrect behaviour with the current tuning of the PID controller when the user is very far away from the levels-of-detail. It appears to enter some sort of oscillation, and the frame rate decreases significantly. We are currently unaware as to what causes this, although it may not even be a tuning related issue.

Visualization tool

The PID controller is currently implemented in our visualization tool, in Python. It could potentially be implemented in OpenSceneGraph instead, to go with our continuous level-of-detail implementation. This would allow visualization on a budget for point cloud data sets without having to use the Python viewer, as OpenSceneGraph comes with a general purpose viewer application. However, the performance gain would be small, as the PID controller is not very performance intensive.

In this chapter, we have described the prototype toolchain, which consists of a preprocessing toolchain, a point cloud visualization pipeline, and a set of evaluation tools. We showed that it corresponds to our research methodology, and that an initial cycle has been completed. We explained that we focused on developing this toolchain primarily to enable future researchers to more easily implement improvements or replacements, to more easily conduct more experiments, or to perform additional evaluations.

We have described the original toolchain that was available at the start of this research. It uses OpenSceneGraph, osgswig, and OpenGL to provide two rapid prototyped tools implemented in Python and C++, which are a preprocessing tool and a visualization tool. The preprocessing tool generates a data structure from a point cloud data set, and the visualization tool displays this data structure and allows easy navigation using a space mouse. The data structure makes use of OpenSceneGraph’s view frustum culling, level-of-detail, and out-of-core optimizations to allow interactive visualization of massive point cloud data sets.

We have described our addition of LAS support to the toolchain. It simplifies our preprocessing tool, enables it to use the de facto standard for point cloud data sets, and introduces a number of useful tools to our toolchain. We discovered it decreases the performance of our toolchain, but noted that the performance of the preprocessing tool is not critical to a visualization.

We also described our improvements to the data structure. To improve performance, we adapted our preprocessing tool to remove empty cells and better fit the bounding spheres used for hierarchical culling. We also developed an index optimization that decreases data retrieval, and ran into a scalability issue where each cell of the index was granted a chance to perform its out-of-core activities. We addressed this by changing OpenSceneGraph so that cells only get to do this if their data is in memory. Finally, we described scalability through tiling, and how a superstructure can be used to implement this.

We then described our importance experiment, consisting of the development and evaluation of an initial importance metric called the sum-of-squared-distances metric.
4.6. VISUALIZATION ON A BUDGET EXPERIMENT

Figure 4.21: A point cloud data set visualized at increasing frames-per-second, using automatic level-of-detail adjustment, continuous level-of-detail, and point importance calculated by the SSD metric. In other words, visualization on a budget. At lower frames-per-second, the differences are very small. At higher frames-per-second, the shape of the data set is still visible.
CHAPTER 4. PROTOTYPE TOOLCHAIN

We noted that $kd$-trees used for the nearest neighbour retrieval cannot be constructed efficiently for an entire data set, and addressed this issue by bucketing the data set into a grid and constructing the $kd$-trees locally. We saw that the sum-of-squared distances metric applies well to the characteristics of aerial LiDAR point cloud data sets, but that it requires a second pass to deal with large variations in density. We also saw that this metric requires manually set parameters, which we deemed undesirable.

We followed this with a description of our continuous level-of-detail experiment, in which we aimed to remove the discrete transitions still visible in the visualizations. We first produced a simulation and noted that the point importance provided by the metric already produces a converging visualization. We then implemented order, and noted that it makes discrete level-of-detail look better already and the simulation look more uniform. We then implemented visual and regular continuous level-of-detail, and while it requires changes to OpenSceneGraph, we saw that it allows us to visualize points without discrete transitions. As this implementation relies on an underlying discrete hierarchical level-of-detail implementation, the data retrieval still occurs in blocks. We noted that this can cause issues with the data retrieval bottleneck, and that a streaming solution may improve on this. Finally, we saw that continuous level-of-detail even works without point importance, although this produces a less converging visualization.

Finally, we described our visualization on a budget experiment, in which we developed a method to maintain the frame rate at a preset target. We described how the level-of-detail scale provided by OpenSceneGraph can be used to adjust the continuous level-of-detail. We showed that a PID controller can be used to maintain the frame rate at its target by adjusting the level-of-detail scale automatically, and that a moving average and a deviation are needed to prevent oscillation. The current frame rate is moved to the target frame rate in steps, with one step per frame. We showed that this means that the PID controller is based on the performance of the visualization pipeline, and as such that the PID controller has to be retuned if the visualization pipeline changes. We noted that this issue can likely be addressed by basing the PID controller on time instead.

\[ \text{72} \]
5 Results and evaluation

In order to assess the quality of the implementation of our method, we performed two studies, an objective study and a subjective study. Both studies make use of the same integral part, a set of flythrough visualizations of three different data sets (listed in table 5.1) for four different configurations of the method (listed in table 5.2). In order to perform the evaluations, a set of tools was developed as part the prototype toolchain described in chapter 4.

The objective study consisted of recording performance statistics during each flythrough, comparing the results between the different configurations per data set, and explaining the differences. It also includes an overview of the performance characteristics of the preprocessing toolchain. To enable this study we developed tools that record performance statistics during a flythrough and that generate graphs from the recorded performance statistics. This study is described in more detail in section 5.1.

The subjective study consisted of having a group of human subjects rate the flythroughs on visual quality by using “split-screen” videos of the flythroughs that show a configuration on the left side and a configuration on the right side. Subjects indicated which side they preferred. To enable this study we developed tools that record, replay, and synchronize flythroughs, and we created a set of templates and guidelines for producing both the “single-screen” and “split-screen” videos. This study is described in more detail in section 5.2.

5.1 Performance analysis

As the prototype toolchain essentially consists of two evaluable parts, the preprocessing toolchain and the point cloud visualization pipeline, so does the objective evaluation. It was performed on the systems listed in table 5.3. The server system (Hera) was used for preprocessing, while the client system (Vaduz) was used for visualization.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data set</th>
<th>SSD metric</th>
<th>Data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kop van Zuid</td>
<td>3DPilot, Kop van Zuid 2010</td>
<td>Two-pass</td>
<td>Octree</td>
</tr>
<tr>
<td>AHN2 Urban</td>
<td>AHN2, section 30g2z1</td>
<td>One-pass</td>
<td>Quadtree</td>
</tr>
<tr>
<td>AHN2 Rural</td>
<td>AHN2, section 65cm2</td>
<td>One-pass</td>
<td>Quadtree</td>
</tr>
</tbody>
</table>

Table 5.1: The three data sets used for the evaluations.
CHAPTER 5. RESULTS AND EVALUATION

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Level-of-detail</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration 1</td>
<td>Discrete</td>
<td>Random</td>
</tr>
<tr>
<td>Configuration 2</td>
<td>Discrete</td>
<td>Importance</td>
</tr>
<tr>
<td>Configuration 3</td>
<td>Continuous</td>
<td>Random</td>
</tr>
<tr>
<td>Configuration 4</td>
<td>Continuous</td>
<td>Importance</td>
</tr>
</tbody>
</table>

Table 5.2: The four configurations of the method that are evaluated. Level-of-detail and order are described in detail in chapter 3.

<table>
<thead>
<tr>
<th>Technical specification</th>
<th>Hera</th>
<th>Vaduz</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU clock</td>
<td>2.80 GHz</td>
<td>2.40 GHz</td>
</tr>
<tr>
<td>CPU architecture</td>
<td>64 bit</td>
<td>64 bit</td>
</tr>
<tr>
<td>Internal memory type</td>
<td>DDR3</td>
<td>DDR2</td>
</tr>
<tr>
<td>Internal memory size</td>
<td>6 GB</td>
<td>2 GB</td>
</tr>
<tr>
<td>External memory type</td>
<td>SATA III</td>
<td>SATA</td>
</tr>
<tr>
<td>External memory RAID</td>
<td>RAID 0 (4 disks)</td>
<td>N/A</td>
</tr>
<tr>
<td>Graphics card type</td>
<td>N/A</td>
<td>GeForce 8800 GTS</td>
</tr>
<tr>
<td>Network connection</td>
<td>3 GB/s</td>
<td>1 GB/s</td>
</tr>
<tr>
<td>Operating system</td>
<td>Ubuntu Lucid</td>
<td>Ubuntu Lucid</td>
</tr>
</tbody>
</table>

Table 5.3: Technical specifications of the preprocessor server (Hera) and the visualization client (Vaduz).

<table>
<thead>
<tr>
<th>Property</th>
<th>Kop van Zuid</th>
<th>AHN2 Urban</th>
<th>AHN2 Rural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>1.1 GB</td>
<td>16.7 GB</td>
<td>15.6 GB</td>
</tr>
<tr>
<td>Points</td>
<td>40 million</td>
<td>502 million</td>
<td>468 million</td>
</tr>
<tr>
<td>Dimensions</td>
<td>1500x1500x195m</td>
<td>5000x6250x139m</td>
<td>5000x6250x44m</td>
</tr>
<tr>
<td>Data structure</td>
<td>1.1 GB</td>
<td>13.9 GB</td>
<td>13.0 GB</td>
</tr>
<tr>
<td>Files</td>
<td>3784</td>
<td>68072</td>
<td>68007</td>
</tr>
</tbody>
</table>

Table 5.4: Properties of the data sets.

<table>
<thead>
<tr>
<th>Data set</th>
<th>las2las2</th>
<th>las2osg</th>
<th>las2osg (importance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kop van Zuid</td>
<td>5.5 hours</td>
<td>1.5 hours</td>
<td>2 hours</td>
</tr>
<tr>
<td>AHN2 (single tile)</td>
<td>3 to 4 hours</td>
<td>30 minutes</td>
<td>45 minutes</td>
</tr>
</tbody>
</table>

Table 5.5: Time taken by the tools in the preprocessing toolchain for our data sets.

<table>
<thead>
<tr>
<th>Data set</th>
<th>las2las2</th>
<th>las2osg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kop van Zuid</td>
<td>2 GB</td>
<td>1 GB</td>
</tr>
<tr>
<td>AHN2 (single tile)</td>
<td>1 GB</td>
<td>600 MB</td>
</tr>
</tbody>
</table>

Table 5.6: Memory used by the tools in the preprocessing toolchain for our data sets.
5.1. PERFORMANCE ANALYSIS

Preprocessing toolchain

To evaluate the preprocessing toolchain, we provide an overview of its performance characteristics. These characteristics were recorded by querying the data sets listed in table 5.1 for their static properties (listed in table 5.4) using the libLAS toolchain, timing their preprocessing (listed in table 5.5), and recording their memory usage during preprocessing (listed in table 5.6) using built-in operating system tools. Note that a single AHN2 tile is roughly one-fiftieth of the AHN2 Urban or AHN2 Rural data set, on average about 10 million points and 300 MB.

From these characteristics we can see that the preprocessing scales roughly linearly in time, which keeps performance acceptable. Note however that the importance calculation adds considerable overhead. We can also see that scalability through tiling (as described in section 4.3) is required to keep memory usage within acceptable internal memory limits. This is primarily because the tools involved currently maintain some linearly proportional form of the data set in internal memory.

As the actual performance of the preprocessing tools is not essential to the visualization, we can also see from the characteristics that the preprocessing tools are capable of handling massive LiDAR point cloud data sets within an acceptable timeframe. However, the performance could definitely benefit from being improved. For more information on how this could be done, see section 6.2.

Point cloud visualization pipeline

In order to be able to perform an objective evaluation of the point cloud visualization pipeline we recorded a flythrough that is typical for interactive exploration for each of the data sets listed in table 5.1. Excerpts of these flythrough are shown in figures 5.1, 5.2, and 5.3. We then replayed these flythroughs for each of the configurations of the method listed in table 5.2 and recorded the resulting visualizations as videos. During these flythrough replays we also recorded performance statistics retrieved from the point cloud visualization pipeline. The end result is a video and a set of performance statistics over time, for each configuration of the method, for each data set.

We should note that the generation of performance statistics during the flythrough replays has a negative effect on the performance of the visualization. The impact is small but noticeable. This is unfortunately a necessary side effect when recording a
CHAPTER 5. RESULTS AND EVALUATION

Figure 5.2: An excerpt of the flythrough for the AHN2 Urban data set.

Figure 5.3: An excerpt of the flythrough for the AHN2 Rural data set.

visualization. If performance statistics are collected that require traversal of the data structure (such as total amount of points in memory), then the impact is quite large. For this reason, we chose not to collect these performance statistics. Fortunately, they are not immediately required to evaluate the point cloud visualization pipeline.

Creation of single-screen videos

The flythrough is recorded using one of the evaluation tools that is part of the prototype toolchain. It is a Python viewer with recording and replaying capabilities, as well as performance statistics recording capabilities. The flythrough is written to a file from which it can later be read and replayed, while then recording the performance statistics to a file. Afterwards, the performance statistics are read by another tool that is part of the prototype toolchain, called osg2png, which creates a set of graphs from them and outputs those graphs as images.

The video is recorded during this replay as well, using the setup shown in figure 5.4. This setup is used over the more traditional approach of capturing the visualization frame for frame using software, because such a capture process adversely affects the performance of the visualization. This also has the side effect of often causing a perfect (but time-lapsed) visualization, as most of the data can be retrieved and visualized within the time it takes to capture the previous frame of the visualization. The setup using an external video camera records exactly what a human subject would see, but allows storage of the visualization for later use. Unfortunately, this setup also produces visualization noise due to the physical medium and the limits of the video camera. Lighting conditions,
Figure 5.4: The recording setup for the videos. The high-definition video camera records the image on the display device exactly as a human would see it.
Figure 5.5: The behaviour over time of all performance statistics, for each configuration of the method individually, for the objective studies on the Kop van Zuid data set. From left to right: importance and random. From top to bottom, continuous and discrete. Note that the requests axis has been normalized per graph.

video resolution, and video capture rate all have an influence on the recorded video. To address this, one could use a device that does not use the physical medium, such as a frame recorder with sufficient frame rate.

The video is processed using templates for an open source video editing (and modeling) tool called Blender\(^1\) that were developed as part of the prototype toolchain. This provides cropping of the areas around the display device that were also recorded. Its output is an image file for every frame of the resulting video, which are stitched together using a custom built version of ffmpeg\(^2\), an open source video conversion tool, and converted to a video using the H264 codec. The result is a relatively small but high quality video.

**Evaluation of interactivity**

The videos can then be examined alongside the graphs for their flythrough visualizations. For our objective study, the graphs that display the behaviour over time of all the performance statistics for each configuration of the method individually are given in figures 5.5, 5.6, and 5.7 for the Kop van Zuid, AHN2 Urban, and AHN2 Rural data sets respectively. From these figures we will show that the interactivity is directly related to the amount of points in view.

The graphs that display the behaviour over time of each statistic individually for all

\(^{1}\)http://www.blender.org
\(^{2}\)http://www.ffmpeg.org
5.1. PERFORMANCE ANALYSIS

Figure 5.6: The behaviour over time of all performance statistics, for each configuration of the method individually, for the objective studies on the AHN2 Urban data set. From left to right: importance and random. From top to bottom, continuous and discrete. Note that the requests axis has been normalized per graph.

Figure 5.7: The behaviour over time of all performance statistics, for each configuration of the method individually, for the objective studies on the AHN2 Rural data set. From left to right: importance and random. From top to bottom, continuous and discrete. Note that the requests axis has been normalized per graph.
Figure 5.8: The behaviour over time of each statistic individually, for all four configurations of the method, for the objective studies on the Kop van Zuid data set. From left to right: frame rates, points in view, and requests.

Figure 5.9: The behaviour over time of each statistic individually, for all four configurations of the method, for the objective studies on the AHN2 Urban data set. From left to right: frame rates, points in view, and requests.

Figure 5.10: The behaviour over time of each statistic individually, for all four configurations of the method, for the objective studies on the AHN2 Rural data set. From left to right: frame rates, points in view, and requests.
four configurations of the method are given in figures 5.8, 5.9, and 5.10 for the Kop van Zuid, AHN2 Urban, and AHN2 Rural data sets respectively. From these figures we will show that the interactivity of the visualization is maintained by our visualization on a budget both when there would otherwise be too many points and when there would otherwise be too few points.

For example, between the 10 and 35 second marks in the Kop van Zuid flythrough videos, a zoom-out occurs, and as a result the discrete configurations experience reduced interactivity because the amount of points in view becomes very large (visible in figure 5.5). However, the continuous configurations maintain interactivity at the requested level (visible in figure 5.8). This shows that in those configurations of the method, interactivity is indeed maintained by decreasing the amount of points in view if the interactivity would otherwise become too low.

On the other end, between the 20 and 30 second marks in the AHN2 Urban flythrough videos, a zoom-out occurs, and the discrete configurations experience increased interactivity because the amount of points in view becomes very small (visible in figure 5.6). Again, the continuous configurations maintain interactivity at the requested level (visible in figure 5.9). This shows that on those configurations of the method, interactivity is also maintained by increasing the amount of points in view if the interactivity would otherwise become too high.

In general, all three data sets show that the interactivity is maintained at the requested level in the continuous configurations, as opposed to the discrete configurations. Another thing to note with regard to interactivity in the discrete configurations, is the atypical behaviour of the performance statistics for the discrete importance configuration of the AHN2 Rural flythrough (visible in figure 5.10). This is caused by swath overlays (as described in section 3.2), which are present in the AHN2 Rural data set. The importance calculated by the one-pass SSD metric is nonuniformly distributed over the points in the data set very because of the greatly varying differences in point density, and as a result the flythrough encounters areas with greatly varying amounts of points in them.

Evaluation of requests

The relation of the amount of requests to the other performance statistics or the different configurations is less clear. Note first that the amount of requests doesn’t represent the actual amount of files that will be loaded, just the amount of files requested to be loaded. Requests can be cancelled if the loading is no longer necessary.

The early rise that occurs in the flythroughs for the continuous configurations (visible in for example figure 5.8) is caused by the visualization starting out with no points. The PID controller (described in section 3.3) increases the level-of-detail quickly to counter the initially extremely high interactivity, and as a result many requests are placed. Most requests are then cancelled as the interactivity quickly lowers while more and more points are being visualized.

Another interesting thing to note is that the amount of requests is often lower in the continuous configurations of the method when compared to the discrete configurations, if ignoring the early rise (visible in for example figure 5.10). This is likely caused by
the gradual level-of-detail adjustment causing less unnecessary transitions between the
discrete hierarchical levels-of-detail (described in section 3.1), and as such less requests.
This is an unexpected positive side effect of the method, that potentially reduces the
amount of data retrieved.

**Evaluation of order**

Finally, most performance statistics are similar between the importance and random
configurations of the method (visible in for example figure 5.8). This is because the
distribution is largely still uniform for either configuration, and while different points
are in view, the amount of points in view is as such still largely the same. However,
differences may be seen during the subjective study in section 5.2.

**Evaluation of memory usage**

To close, a statistic that wasn’t recorded over time was the internal memory usage during
the visualizations. This is because the internal memory usage is and always has been low,
and experiences little change. For example, for the AHN2 Urban data set, the internal
memory usage is nearly always around 600 MB. This is due to the out-of-core algorithm
used (as described in section 4.3).

Note that if the level-of-detail is manually increased to a very high setting, internal
memory usage can become an issue as a lot of data is consequently visualized. This should
be avoided when tuning the level-of-detail parameter in discrete and continuous level-of-
detail, and is automatically avoided in visualization on a budget unless the interactiveness
target is set very low.

### 5.2 Preliminary user study

In order to be able to perform a subjective evaluation that allows us to rank the vari-
ous configurations of our method, we came up with a somewhat novel approach. As a
sequential visualization requires a subject to remember the preceding visualization, and
a side-by-side visualization requires a subject to focus on two things at the same time,
we combine these two traditional approaches and show the subject a “split-screen” visual-
ization. The user is shown one flythrough, but with the left side and the right side of
the flythrough consisting of different configurations. Figure 5.11 shows an example.

This approach is a first attempt at such a subjective evaluation, and as such has not yet
entirely been thought through. As with all steps in the prototype toolchain, we provide
an initial implementation that can be improved upon, given what we will learn from our
first attempt.

**Creation of split-screen videos**

Such visualizations are made by using the videos recorded for the objective evaluation
described in section 5.1. Again, Blender templates that were developed as part of the
5.2. PRELIMINARY USER STUDY

Figure 5.11: An example of the “split-screen” visualization used for the subjective study. The left and right sides show different configurations of the method, but together still make up one flythrough.

prototype toolchain are used to combine the videos, placing one half on each side, with a divider between them. Some synchronization is required. The flythrough recorder ensures that the flythroughs for each configuration of the method can be synchronized. Again, image files are produced by Blender, and ffmpeg is used to convert them into a small but high quality video using the H264 codec.

Subject backgrounds

Before showing the videos to each subject, we asked about their experiences with the topics listed in table 5.7 to gain an understanding of their background. As can be seen from the table, the subjects largely have the same background. This background indicates good experience with 3D computer graphics.

Rating of split-screen videos

These visualizations allow the subjects to focus on one flythrough but instinctively compare the two configurations of the method. The rest of the study then consists of showing these videos to the subjects in any order, and asking them to indicate which side they prefer. Figure 5.12 shows the information sheet used to communicate this to the subjects. Subjects are asked to rate the videos on a scale of 1 to 7, where 1 equals “I prefer left a
Figure 5.12: The information sheet used to explain the subjective study process.
5.2. PRELIMINARY USER STUDY

<table>
<thead>
<tr>
<th>Experience</th>
<th>Subject answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computers</td>
<td>Yes Yes Yes Yes</td>
</tr>
<tr>
<td>Consoles</td>
<td>Yes Yes No Yes</td>
</tr>
<tr>
<td>3D games</td>
<td>Yes Yes Yes Yes</td>
</tr>
<tr>
<td>Handhelds</td>
<td>Yes Yes Yes Yes</td>
</tr>
<tr>
<td>Google Maps</td>
<td>Yes Yes Yes Yes</td>
</tr>
<tr>
<td>Google Earth</td>
<td>Yes Yes Yes Yes</td>
</tr>
<tr>
<td>Stereo 3D</td>
<td>Yes Yes Yes Yes</td>
</tr>
</tbody>
</table>

Table 5.7: The experiences of the subjects in the subjective study. As can be seen, all subjects have largely the same experiences.

<table>
<thead>
<tr>
<th>Code</th>
<th>Data set</th>
<th>Left LOD</th>
<th>Left order</th>
<th>Right LOD</th>
<th>Right order</th>
</tr>
</thead>
<tbody>
<tr>
<td>KvZ CI-CI</td>
<td>Kop van Zuid</td>
<td>Continuous</td>
<td>Importance</td>
<td>Continuous</td>
<td>Importance</td>
</tr>
<tr>
<td>KvZ CI-DR</td>
<td>Kop van Zuid</td>
<td>Continuous</td>
<td>Importance</td>
<td>Discrete</td>
<td>Random</td>
</tr>
<tr>
<td>KvZ DR-CI</td>
<td>Kop van Zuid</td>
<td>Discrete</td>
<td>Random</td>
<td>Continuous</td>
<td>Importance</td>
</tr>
<tr>
<td>KvZ DR-DI</td>
<td>Kop van Zuid</td>
<td>Discrete</td>
<td>Random</td>
<td>Discrete</td>
<td>Random</td>
</tr>
<tr>
<td>Urban CI-CR</td>
<td>AHN2 Urban</td>
<td>Continuous</td>
<td>Importance</td>
<td>Continuous</td>
<td>Random</td>
</tr>
<tr>
<td>Urban DR-CR</td>
<td>AHN2 Urban</td>
<td>Discrete</td>
<td>Random</td>
<td>Continuous</td>
<td>Random</td>
</tr>
<tr>
<td>Rural DI-DR</td>
<td>AHN2 Rural</td>
<td>Discrete</td>
<td>Importance</td>
<td>Discrete</td>
<td>Random</td>
</tr>
<tr>
<td>Rural CI-CR</td>
<td>AHN2 Rural</td>
<td>Continuous</td>
<td>Importance</td>
<td>Continuous</td>
<td>Random</td>
</tr>
<tr>
<td>Rural DR-CR</td>
<td>AHN2 Rural</td>
<td>Discrete</td>
<td>Random</td>
<td>Continuous</td>
<td>Random</td>
</tr>
</tbody>
</table>

Table 5.8: The videos used in subjective study.

<table>
<thead>
<tr>
<th>Code</th>
<th>Subject ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>KvZ CI-CI</td>
<td>4 4.5 4 4 4 4 4</td>
</tr>
<tr>
<td>KvZ CI-DR</td>
<td>7 3 6 5 4 6.5 6 5.5</td>
</tr>
<tr>
<td>KvZ DR-CI</td>
<td>6 6 6 4 5 4 5 4</td>
</tr>
<tr>
<td>KvZ DR-DI</td>
<td>7 5 4 5 5 5 5 7</td>
</tr>
<tr>
<td>KvZ DI-CI</td>
<td>2 5 5 4 5 5 5 1</td>
</tr>
<tr>
<td>Urban CI-CR</td>
<td>7 4 5 5 5 6 5 5.5</td>
</tr>
<tr>
<td>Urban DR-CR</td>
<td>5 6 3 6 6 6 6 3</td>
</tr>
<tr>
<td>Rural DI-DR</td>
<td>5 5 6 6 5 6 5 4</td>
</tr>
<tr>
<td>Rural CI-CR</td>
<td>7 6 6 6 6.5 6 6 7</td>
</tr>
<tr>
<td>Rural DR-CR</td>
<td>3 2 5 2 3 3 5 2</td>
</tr>
</tbody>
</table>

Table 5.9: The ratings of the subjects in the subjective study.
Figure 5.13: The results of the subjective study as a box plot. The horizontal axis lists the code names of the videos (see table 5.8). The vertical axis represents whether the left or the right side of the video was preferred, and to what degree.

lot, 4 equals “I prefer neither”, and 7 equals “I prefer right a lot”. They are allowed to pause a video or watch a video multiple times.

The videos we used for our subjective study are listed in table 5.8. The ratings given to these videos by our subjects are listed in table 5.9. A box plot generated from these ratings is shown in figure 5.13.

From the boxplot we can see that subjects preferred the right sides except for the videos KvZ CI-CI and Rural DR-CR. That the right side of most videos is preferred is a coincidence, although perhaps it is an oversight on our part at properly randomizing the videos.

As can be seen from table 5.8, the two sides of video KvZ CI-CI consist of the same configuration. This served as a calibration video, and as can be seen from the results almost all subjects indicated both sides looked the same.
5.2. Preliminary User Study

Evaluation of level-of-detail

Video H was used as the practice video, where subjects were allowed to get a feeling for the process. During the first playback of the video, we simply asked whether they saw a difference and which side they preferred. During a second playback, we pointed out the discrete transitions in the right side of the video, and asked if they noticed it. If they did, we asked them if they were bothered by it. During the third playback, we pointed out the drop in frame rate in the right side of the video, and again asked them if they noticed it and were bothered by it.

As it turns out, most subjects either did not see or were not bothered by the discrete transitions in video KvZ CI-DR. Almost all of them also indicated to prefer the higher amount of detail over the decrease in frame rate. This was contrary to what we expected, but as can be seen from the rest of the results, ironically enough most subjects preferred the more smoother visualization for all other videos that compare a discrete configuration to a continuous configuration. This includes video KvZ DR-CI which was simply the inverse of video KvZ CI-DR. The drop in frame rate was generally undesired, as was the discrete transition in most cases. This leads us to believe that video KvZ CI-DR is some sort of outlier, and perhaps a poor choice for a practice video.

Loss of smoothness

Subjects that preferred the discrete configuration generally indicated that they preferred detail over smoothness. This is an interesting result, but not necessarily a negative one for our research. During the recording of the videos, we were unaware that the effective frame rate of the visualization would drop from 60 frames-per-second to only 25 frames-per-second. As a result, a lot of the smoothness of the visualizations was lost. This means that subjects were rating a substandard visualization, as they should have either seen an even smoother visualization at 60 frames-per-second, or a more detailed visualization at 25 frames-per-second. Either visualization may have tempted them to prefer it instead. In future subjective evaluation, this will have to be taken into account (see also section 6.2).

Smoothness crosstalk

We also noticed that a smooth visualization on one side of the video tends to cause a subject to be more comfortable with a less smooth visualization on the other side of the video, which is rather contrary to what one would expect. It would appear that the smooth side somehow “drags” the less smooth side along, for the perception of a subject.

Evaluation of order

For the other important configuration comparison, importance versus random order, the results are pretty conclusive. Almost all subjects prefer the visualization that uses importance, regardless of whether discrete or continuous level-of-detail is used.
CHAPTER 5. RESULTS AND EVALUATION

Notable exceptions are videos Urban CI-CR and Rural CI-CR. Video Rural CI-CR is of the AHN2 Rural data set, which suffers greatly from the swath overlay problem. As such a random order provides a much more uniform looking visualization. Video Urban CI-CR is of the AHN2 Urban data set, and the random order provided a more uniform look when zoomed out. When zoomed in the importance order was preferred. The AHN2 Urban data set likely just requires a better tuned weighted shuffle parameter for continuous level-of-detail. Naturally, both data sets would benefit from a tuned second pass of the SSD metric to combat their nonuniform look.

In this chapter, we have seen that in order to evaluate the method we devised, we constructed tools as part of the prototype toolchain for objective and subjective studies. Using these tools, we created “single-screen” and “split-screen” videos, the former for objective evaluation, and the latter for subjective evaluation.

We performed an objective evaluation by comparing performance statistics recorded over time between the flythrough visualizations of the four configurations of the method. We saw that interactivity is maintained by visualization on a budget, both when the amount of points in view would otherwise be too high and when the amount of points in view would otherwise be too low. We saw that the performance statistics are similar between the two orders, because the amount of points in view is still largely the same. Finally, we saw that the amount of requests is somewhat lower when using visualization on a budget.

We performed a preliminary subjective evaluation by reviewing the ratings given by human subjects to the “split-screen” videos. We saw that the practice video appeared to be some sort of outlier, but that most subjects preferred smoothness over detail. We noticed that we accidentally provided substandard visualizations due to a loss of smoothness while recording the videos, and that a proper visualization may possibly sway the subjects that preferred detail over smoothness as more smoothness or more detail can be visualized. Finally, we saw that most subjects preferred the importance order over the random order, except when the visualization suffered from the swath overlay problem or poor tuning.
6 Conclusions and future work

In this thesis, we have described a method for interactive visualization of massive LiDAR point cloud data sets.

We introduced the point as a three dimensional unit of data found in point cloud data sets. We focused on aerial LiDAR point cloud data sets, most notably the AHN2 data set, which are geographical data sets recorded by aerial LiDAR technology. Their availability and usefulness has caused demand for them to grow. With the technology also advancing, point cloud data sets have been growing exponentially. The resulting massive size of these data sets presents an issue in visualization.

We explained that point cloud data sets are collections of point samples. A point sample is a collection of point attributes. We described how point cloud data sets are recorded by a LiDAR scanner, and the resulting points are stored in simple record-like data structures. We explained that points lack coherence and volume, that point cloud data sets have a density, and that point cloud data sets are often preprocessed. We then showed that interactive visualization of point cloud data sets is a subject of much study, due to the large data sets involved. To improve interactivity, earlier work in the field of point cloud visualization focuses on alleviation of various bottlenecks in the point cloud visualization pipeline, as well as providing efficient access to the data using spatial data structures.

We suggested that research into point cloud visualization had not yet focused on taking advantage of the limits of the human visual system. We explained how to do this by using visualization on a budget, a method we devised that increases interactivity by leaving out points that are less important to the human visual system. This is done automatically, always providing an interactive visualization, using the points that best represent the data set. In order to do this, continuous level-of-detail is used, with the points in an order derived from their importance. This importance is assigned by a metric. The level-of-detail is automatically adjusted according to the interactivity of the visualization.

To implement visualization on a budget, we developed a prototype toolchain for point cloud visualization and evaluation. This toolchain consists of a preprocessing toolchain, a visualization pipeline, and a set of evaluation tools. We based this toolchain on an original toolchain that was already available. This toolchain provides an initial, working implementation of visualization on a budget, and allows future researchers to more easily improve on various steps of the toolchain, conduct more experiments, and perform more evaluations (see also section 6.2).

Finally, we evaluated the method we developed. Initial results show that the interactivity of the visualization is maintained. They also show that the visualization “looks better” than the visualization provided by the original toolchain. The former is based on
CHAPTER 6. CONCLUSIONS AND FUTURE WORK

an objective study that we performed, where performance statistics were recorded and compared for four configurations of the method, for three data sets, during a flythrough visualization. The latter is based on a subjective study that we performed, where human subjects viewed side-by-side comparison videos for a number of permutations of four configurations of the method, for three data sets, of a flythrough visualization.

All that rests us now is to draw our final conclusions, which is done in section 6.1 and propose work for future researchers, which is done in section 6.2.

6.1 Conclusions

Given the results of our evaluation in chapter 5, we can conclude that our method provides interactive visualization of massive point cloud data sets. This conclusion means that we have reached our first goal.

The objective studies show that the interactiveness of the visualization is maintained at an acceptable level across all data sets. They also show that the method increases interactiveness by reducing the amount of points if the visualization is not interactive enough, and that it reduces interactiveness by increasing the amount of points if the visualization has interactiveness to spare. This enables the visualization to show the amount of points required for a given interactiveness at all times. In other words, the implementation of visualization on a budget is a success.

The subjective studies show that, when compared with our initial version, our method already increases the effectiveness of the visualization. This is due to the replacement of the discrete visualization with a continuous variant that is hard for the human visual system to actually see at work. We can conclude that our method is able to adjust the interactiveness of a visualization without directly affecting the effectiveness of the visualization. In other words, the implementation of continuous level-of-detail is a success.

From the same results, we can conclude that we can reduce the complexity of a point cloud visualization while minimizing the impact on its effectiveness by taking advantage of the limits of the human visual system, through employment of heuristic point importance metrics. This conclusion means that we have (tentatively) reached our second goal.

The subjective studies show that the variant using the two-pass metric provides a more efficient visualization than the variant that does not. However, the subjective studies also show that the variant using the one-pass metric provides a less efficient visualization than the variant that does not. From this we can conclude that the metric largely defines the effectiveness of the visualization. We can also conclude that the one-pass metric is not acceptable for all data sets, and that the two-pass metric is acceptable for at least one data set. However, both versions of the metric require at least one manually tuned parameter per data set. We conclude that this is undesirable in the long run, and as such we can conclude that the metric in its current form is not what we are looking for. In other words, the concept of importance was a success, but the implemented metric was not.

Finally, we can conclude that we successfully implemented a prototype toolchain that
6.2. FUTURE WORK

has supported us in our research and implementation, and that also provides a solid, easy
to use basis for future research on the subject. This conclusion means we have reached
our third goal as well.

As a final note, we point out that the performed subjective studies were only initial
studies, and to definitively reach our second goal, a more intensive study should be
performed. Section 6.2 provides a detailed explanation.

6.2 Future work

We can see from our conclusion in section 6.1 that some steps of the prototype toolchain
described in chapter 4 have produced less impressive results. Fortunately, the toolchain
allows us and future researchers to continue improving our method of interactive point
cloud visualization by improving the steps separately.

Importance metric

A step that is of note is the point importance metric described in section 3.2. As the
conclusion in section 6.1 explains, it is not fit for data sets with varying density, and it
requires manual tuning. Ideally, a metric is resilient to differing densities, either in or
between data sets, producing orderings that still guarantee that each level-of-detail will
have a relatively uniform density. This likely requires some form of automated parameter
tuning, possibly in multipass preprocessing steps.

At the same time, more thought can be put in the theory behind the metric. Perception
depends on many variables, and for our metric we simply worked from the assumptions
that edges and vertical surfaces are more important. Proper perception heuristics, as
provided by the field of visual saliency, may provide conceptually better metrics.

In any case, replacing the metric is one avenue that definitely warrants future research.
The evaluation tools can be used to evaluate any replacement metrics.

Data access

Another step of interest is the data access side of point cloud visualizations. Currently,
a data structure is stored as a collection of files, optimized for out-of-core access. This
file access is completely one-sided, with the client is doing all the work. A possible
optimization is to store the data structure in a data base instead, and have a server serve
out the data structure from this data base. The server can then implement common,
possibly parametrized parts of the point cloud visualization pipeline that it can efficiently
perform and perhaps even apply common server optimization techniques to, such as
caching. What is possible, and what the effects are on the performance of the point
cloud visualization pipeline, warrants more research.
CHAPTER 6. CONCLUSIONS AND FUTURE WORK

Data sets

Related to this is the issue of scalability and diversity. As described in section 1.3, more and more point cloud data sets become available, and their size grows exponentially. It is important to evaluate if the toolchain can handle different types of data set, and if it scales well enough. The evaluation tools that are a part of the toolchain can assist in further research on this subject. These researchers are particularly interested in interactively visualizing the full AHN2 data set.

Evaluation

The results gathered in section 5.1 are initial results, and it is important to perform better subjective studies, and on a much larger scale to gain a good sample size. The human visual system aspect of the optimization is very hard to measure properly, and how to do this definitely warrants future research. In particular, a slight extension to our original research question is of note: how much can we reduce the complexity of a point cloud visualization while minimizing the impact on the effectiveness of that visualization? In other words, what can we get away with? Finding this limit will warrant intensive subjective studies.

One possible large scale subjective study to get started with is an online survey. This research originally intended to include such a survey, but we have instead opted to provide the toolchain and describe the process for future researchers, mostly for resource reasons. The subjective study videos one can make with the toolchain (see chapter 5) can be stored online and included in a web based survey. If services such as YouTube and Google Spreadsheets are used, a survey can be put together without much effort. Such a survey could include just one question per video, simply asking: which side of the video do you prefer (or do they look the same)? However, it may also include more in-depth questions, for example asking which parts of the video constituted good or bad visualizations. Since the survey is web based, it should hopefully be easier to find and attract more subjects to take it. Alternatively, a service such as Amazon Mechanical Turk allows one to pay people to take surveys, making it easier to achieve an acceptable sample size.

Performance

Moving away from the more theoretical steps, there are a number of practical steps that warrant improvement or even change. For example, as it is implemented now, the toolchain is a prototype that contains a number of applications implemented in Python that communicate with C++ through a software layer. While perfect for rapid prototyping, this communication is slow, and it hurts performance. Implementing these applications directly in the lower level language will massively increase performance, most notably when it comes to preprocessing. It is of note that one then loses the rapid

---

1 At the time of writing the recording of AHN2 was still in progress.
2 http://www.youtube.com
3 http://docs.google.com
4 http://www.mturk.com

---

92
prototyping aspect, although that is not necessarily a bad thing: the applications have matured to a level where they only need to be changed in specific places. A proper implementation will even allow this seamlessly and efficiently, against the cost of more software design efforts.

**Improvement**

There are some bugs in the current implementation of some of the steps as well. For example, the PID controller stops working correctly when zooming out to a high degree. In fact, the implementation of many of the steps is far from perfect, as the primary goal was to complete an initial cycle. Naturally, the prototype toolchain will benefit from having its steps improved upon. Of course, steps can also be replaced with better working alternatives altogether.

**Deployment**

Finally, we hope to move the software out of the laboratory, and into the real world. Currently the toolchain is available on one workstation, and part of it is available on a server for performance purposes. Preferably, it should be easier to set up on other machines, with the visualization steps perhaps even available at consumer level. Ideally, in the end, we would like the visualization steps of the toolchain to be deployable on any machine, being able to visualize a point cloud given the budget of that machine. For example, these researchers dream of interactive visualization of points clouds on mobile devices.
List of Figures

1.1 Point cloud visualizations of various industrial parts. ........................................ 1
1.2 Point cloud visualizations of Lucy, a statue of an angel, and David, the well-known statue by Michelangelo. ................................................................. 2
1.3 Point cloud visualization of the aerosol pollution over China, India, and Bhutan. ......................................................... 2
1.4 Point cloud visualization of the Stephansdom church in Vienna, Austria. ........ 3
1.5 Point cloud visualization of the Domitilla Catacomb in Rome. ....................... 3
1.6 Point cloud visualization of a part of a city. ...................................................... 4
1.7 A point cloud of a child’s face. ........................................................................... 6
1.8 Point cloud visualization of a small part of the AHN2 aerial LiDAR point cloud data set. ...................................................................................... 8
1.9 Comparison of the resolution in the AHN1 and AHN2 data sets. ..................... 9

2.1 A point cloud of a person’s head and shoulders. .............................................. 12
2.2 A ground-based and an aerial LiDAR scanner. .................................................. 13
2.3 Part of the Stephansdom ground-based point cloud data set. ......................... 13
2.4 The aerial LiDAR scanning process. ................................................................... 14
2.5 The Domitilla Catacomb ground-based point cloud data set. ......................... 15
2.6 Part of the AHN2 aerial LiDAR point cloud data set. ....................................... 15
2.7 Point cloud visualizations using surface elements (surfels). ......................... 17
2.8 A simple point cloud visualization pipeline. ..................................................... 19
2.9 Example client-server architecture for a point cloud visualization pipeline. .... 20
2.10 Quadtree space decomposition of the space of a node. ................................... 21
2.11 Octree space decomposition of the space of a node. ....................................... 21
2.12 A two dimensional kd-tree constructed on a data set of 6 points. ............... 22
2.13 A two dimensional R-tree constructed on a point data set. ......................... 23
2.14 A three dimensional R-tree constructed on a point data set. ....................... 23
2.15 The bottlenecks in a simple point cloud visualization pipeline when dealing with massive data sets. .................................................. 25
2.16 Three levels of detail. .................................................................................... 26
2.17 Level-of-detail providing multiple resolutions in a point cloud visualization. 26

3.1 Discrete level-of-detail. .................................................................................... 31
3.2 Hierarchical level-of-detail. ............................................................................. 32
3.3 Discrete boundaries in hierarchical levels-of-detail. ....................................... 33
3.4 Continuous level-of-detail on top of hierarchical level-of-detail. ................... 34
3.5 Distribution of points over levels-of-detail. .................................................... 35
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6</td>
<td>The random shuffle</td>
<td>35</td>
</tr>
<tr>
<td>3.7</td>
<td>The weighted random shuffle</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>The prototype toolchain</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Our research methodology</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>The data structure of the original toolchain</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>The space mouse is a device for navigating in three dimensional space</td>
<td>46</td>
</tr>
<tr>
<td>4.5</td>
<td>A frame from a flythrough visualization done by the original visualization tool</td>
<td>46</td>
</tr>
<tr>
<td>4.6</td>
<td>The empty cells in an octree data structure when used on a predominantly two dimensional data set</td>
<td>48</td>
</tr>
<tr>
<td>4.7</td>
<td>The bounding sphere as it applies to a cell near the bottom of a quadtree</td>
<td>48</td>
</tr>
<tr>
<td>4.8</td>
<td>The bounding spheres for the cells in the octree for the Kop van Zuid data set before (left) and after (right) the optimization</td>
<td>50</td>
</tr>
<tr>
<td>4.9</td>
<td>Example of an index in a data structure</td>
<td>51</td>
</tr>
<tr>
<td>4.10</td>
<td>Example of an out-of-core hierarchical superstructure</td>
<td>52</td>
</tr>
<tr>
<td>4.11</td>
<td>The data structure of the new toolchain</td>
<td>52</td>
</tr>
<tr>
<td>4.12</td>
<td>The sum of squared distances metric</td>
<td>54</td>
</tr>
<tr>
<td>4.13</td>
<td>The grid optimization used for the SSD metric</td>
<td>56</td>
</tr>
<tr>
<td>4.14</td>
<td>The effect of the very low point importances assigned by the SSD metric</td>
<td>57</td>
</tr>
<tr>
<td>4.15</td>
<td>The application of the SSD metric to a typical section of an aerial LiDAR point cloud data set</td>
<td>57</td>
</tr>
<tr>
<td>4.16</td>
<td>Importance as assigned by the SSD metric, with increasing range parameter</td>
<td>59</td>
</tr>
<tr>
<td>4.17</td>
<td>Nonuniform density distribution in an aerial LiDAR point cloud data set</td>
<td>60</td>
</tr>
<tr>
<td>4.18</td>
<td>Importance as assigned by the SSD metric, with a range parameter of 3 meters</td>
<td>61</td>
</tr>
<tr>
<td>4.19</td>
<td>Importance distribution as resulting from the SSD metric</td>
<td>62</td>
</tr>
<tr>
<td>4.20</td>
<td>Continuous level-of-detail using random ordering and importance ordering</td>
<td>67</td>
</tr>
<tr>
<td>4.21</td>
<td>Visualization on a budget</td>
<td>71</td>
</tr>
<tr>
<td>5.1</td>
<td>An excerpt of the flythrough for the Kop van Zuid data set</td>
<td>75</td>
</tr>
<tr>
<td>5.2</td>
<td>An excerpt of the flythrough for the AHN2 Urban data set</td>
<td>76</td>
</tr>
<tr>
<td>5.3</td>
<td>An excerpt of the flythrough for the AHN2 Rural data set</td>
<td>76</td>
</tr>
<tr>
<td>5.4</td>
<td>The recording setup for the videos</td>
<td>77</td>
</tr>
<tr>
<td>5.5</td>
<td>The behaviour over time of all performance statistics, for each configuration of the method individually, for the objective studies on the Kop van Zuid data set</td>
<td>78</td>
</tr>
<tr>
<td>5.6</td>
<td>The behaviour over time of all performance statistics, for each configuration of the method individually, for the objective studies on the AHN2 Urban data set</td>
<td>79</td>
</tr>
<tr>
<td>5.7</td>
<td>The behaviour over time of all performance statistics, for each configuration of the method individually, for the objective studies on the AHN2 Rural data set</td>
<td>79</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.8</td>
<td>The behaviour over time of each statistic individually, for all four config-</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>urations of the method, for the objective studies on the Kop van Zuid data</td>
<td></td>
</tr>
<tr>
<td>5.9</td>
<td>The behaviour over time of each statistic individually, for all four config-</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>urations of the method, for the objective studies on the AHN2 Urban data</td>
<td></td>
</tr>
<tr>
<td>5.10</td>
<td>The behaviour over time of each statistic individually, for all four config-</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>urations of the method, for the objective studies on the AHN2 Rural data</td>
<td></td>
</tr>
<tr>
<td>5.11</td>
<td>An example of the “split-screen” visualization used for the subjective study</td>
<td>83</td>
</tr>
<tr>
<td>5.12</td>
<td>The information sheet used to explain the subjective study process</td>
<td>84</td>
</tr>
<tr>
<td>5.13</td>
<td>The results of the subjective study as a box plot</td>
<td>86</td>
</tr>
</tbody>
</table>
List of Tables

5.1 The three data sets used for the evaluations ........................................ 73
5.2 The four configurations of the method that are evaluated .................. 74
5.3 Technical specifications of the preprocessor server and the visualization client .......................................................... 74
5.4 Properties of the data sets ................................................................. 74
5.5 Time taken by the tools in the preprocessing toolchain for our data sets .................. 74
5.6 Memory used by the tools in the preprocessing toolchain for our data sets ........ 74
5.7 The experiences of the subjects in the subjective study ..................... 85
5.8 The videos used in subjective study .................................................. 85
5.9 The ratings of the subjects in the subjective study ............................ 85
Bibliography


