Vario-scale visualization of the AHN2 point cloud

TU DELFT
GEOMATICS FOR THE BUILT ENVIRONMENT

Thesis

J.D.N. van der Maaden
4156617
J.D.N.vanderMaaden@student.tudelft.nl

Main mentor:
Prof.dr.ir. P.J.M. van Oosterom
Second mentor:
Dr.ir. B.M. Meijers
Co-reader
Dr. ir. R.Y. Peters
Examiner:
Dr. H.M.H. van der Heijden

April, 2019

Abstract

LiDAR technologies are used to measure point cloud data of the earth's surface. The usage of LiDAR allows for the fast collections of massive data sets. The AHN2 point cloud data set, part of Rijkswaterstaats initiative to map the surface of the Netherlands, contains 639 478 217 460 points.

For efficient visualization in web viewers, these massive point clouds are stored in an octree data structure. Visualization through this method has the downside of discretely visualizing the point cloud. These discrete artefacts are referred to as *density jumps*, and are visible where there is a boundary between blocks retrieved from the octree. These blocks contain different densities because they are retrieved from different levels of the octree. This thesis proposes a continuous visualization method for massive point cloud data sets that aims to eliminate these *density jumps*.

While the continuous visualization of vector data sets has been extensively researched, this is a novel field of research for point cloud data sets. This thesis explores the feasibility of a vario-scale visualization method, and aims to implement it in an existing web viewer architecture. Due to the massive nature of the AHN2 data set, cloud computing and distributed computing techniques are used to imrove the workflow.

The presented methodology removes the *density jumps* by determining an *upper density bound* for the point cloud density relative to the camera position. *Circle packing* theory is used to reinforce the upper bound continuously, thus removing artefacts created by discrete density jumps. A proof-of-concept for this theory is implemented in an existing point cloud web viewer architecture.

Contents

1. Introduction **1**
- 1.1. Problem statement . . . 2
- 1.2. Scientific relevance . . . 4
- 1.3. Research question . . . 4
- 1.4. Scientific scope . . . 5
- 1.5. Thesis structure . . . 5

2. Theoretical background **7**
- 2.1. Vario-scale visualization . . . 7
- 2.2. Point cloud processing . . . 8

3. Theoretical approach **15**
- 3.1. Theoretical vario-scale visualization . . . 15
- 3.2. Density formula . . . 16
- 3.3. Algorithmic implementation . . . 25
- 3.4. Method evaluation . . . 32

4. Framework and Data sets **35**
- 4.1. Framework considerations . . . 35
- 4.2. Proof of concept framework . . . 37
- 4.3. Data sets . . . 38

5. Implementation **41**
- 5.1. Considerations . . . 41
- 5.2. Implementation . . . 41
- 5.3. Resulting approach . . . 42

6. Results **43**
- 6.1. Parallel point cloud indexing . . . 43
- 6.2. Vario-scale frame . . . 46
- 6.3. Density function 1 . . . 50
- 6.4. Density function 2 . . . 55
- 6.5. Density function 3 . . . 60
- 6.6. Density function 4 . . . 65

7. Discussion and Future work **71**
- 7.1. Discussion . . . 71
- 7.2. Findings . . . 72
- 7.3. Integration in scientific body of work . . . 74

8. Conclusion and future work **75**
- 8.1. Conclusion . . . 75
- 8.2. Future work . . . 76

Appendices **82**

A. Discrete levels top down visualization **83**

B. Vario scale top down visualization **87**

C. Config File Entwine **91**

D. Config File Greyhound **93**

E. Results frame original - density **95**

F. Results frame original - perspective **97**

List of Figures

1. SSC for smooth tGAP (Suba, Meijers, & van Oosterom, 2013) . . . 1
2. Vario-scale visualization of point clouds . . . 2
3. Octree recursive storage . . . 3
4. Discrete density jumps in the AHN2 web viewer . . . 3
5. LoD vario-scale visualization (van Oosterom et al., 2015) (left) (MithrandirMage (2012). Retrieved October 12 2017 from https://en.wikipedia.org/wiki/Viewing_frustum 8
6. Octree, R-Tree and k-d tree indexing visualized . . . 10
7. Tile based decomposition (left) vs. Domain based decomposition (right) . . . 10
8. Infrastructure, Platform and Software as a service. (Jamesmbond (2013). Retrieved December 17 2017 from https://mycloudblog7.wordpress.com/2013/06/19/who-manages-cloud-iaas-paas-and-saas-services/) . . . 13
9. B2B IaaS platform adoption. (Skyhigh Networks (2017). Retrieved December 17 2017 from http://www.evontech.com/what-we-are-saying/entry/microsoft-azure-vs-amazon-aws-comparison-between-two-cloud-computing-giants.html) . . . 13
10. 3D (a) vs 4D (b) indexing method . . . 15
11. Discrete density jumps in the AHN2 web viewer . . . 15
12. Density pyramid with (a) and without (b) density jumps . . . 16
13. Example of a top down view (a), and a perspective view (b) of the same data set . . . 17
14. Overlaying a grid on the 2D and 3D spatial extents of a LiDAR data set to clarify how density distribution is affected by the perspective translation . . . 17
15. Translating the 2D spatial extents with a continuous density of three points per $n * m$ units, to a perspective view . . . 18
16. Discrete density jumps in the AHN2 web viewer . . . 18
17. Translating the 2D octree selection method to a perspective view . . . 19
18. Density jumps visible in both the real point cloud viewer (a) and the schematic visualization of the problem (b). . . . 19
19. Octree density jumps in the AHN2 web viewer . . . 20
20. Translating the 2D spatial extents to a vario-scale perspective view . . . 21
21. Vario-scale density function for point cloud data sets . . . 22
22. Mapping from view frustum to the viewing volume. (Scratchpixel 2016. Retrieved April 16 2018 from http://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/projection-matrices-what-you-need-to-know-first) 22
23. Perspective transformation matrix . . . 23
24. Density function of the frame . . . 23
25. Four density formulas visualized . . . 24
26. Filtering bands . . . 27
27. Density function 1 . . . 28
28. Filtering bands using predetermined amount of points per band . . . 29
29. Point radius density . . . 30
30. 2D Point radius density . . . 30
31. Point radius density . . . 31
32. Three evaluated frames . . . 32
33. Basic framework for proof of concept . . . 35
34. The proposed framework . . . 39
35. The data set regions . . . 39
36. Four density formulas visualized . . . 46
37. Density jumps (a) and their representation in the data set (b) . . . 47
38. Original density of frame . . . 48

39. Original perspective view of the frame . . . 49
40. Density function 1 . . . 50
41. Density function 1 density of frame . . . 51
42. Density function 1 perspective view of the frame . . . 52
43. Density function 1 density of the removed points . . . 53
44. Density function 1 perspective view of the removed points . . . 54
45. Density function 2 . . . 55
46. Density function 2 density of frame . . . 56
47. Density function 2 perspective view of the frame . . . 57
48. Density function 2 density of the removed points . . . 58
49. Density function 2 perspective view of the removed points . . . 59
50. Density function 3 . . . 60
51. Density function 3 density of frame . . . 61
52. Density function 3 perspective view of the frame . . . 62
53. Density function 3 density of the removed points . . . 63
54. Density function 3 perspective view of the removed points . . . 64
55. Density function 4 . . . 65
56. Density function 4 density of frame . . . 66
57. Density function 4 perspective view of the frame . . . 67
58. Density function 4 density of the removed points . . . 68
59. Density function 4 perspective view of the removed points . . . 69
60. Merging overhead . . . 71
61. Visual comparison of the four density formula's used . . . 72
62. From data set spatial extents to users screen . . . 72
63. Density function 2 . . . 73
64. Original frame (a) compared to the vario-scale frame (b) . . . 73
65. 4D vs 3D indexing method . . . 77
66. Original density of frame . . . 96
67. Original perspective view of the frame . . . 97

List of Tables

1. IaaS service provided by AWS, Microsoft Azure and Google Cloud 14
2. Computational speed of proposed methods 33
3. Pricing of S3 storage, per year as of December 2018 37
4. Data sets 39
5. AWS instances used for non-parallel benchmark 43
6. Preliminary benchmark 44
7. Cloud computing benchmark 45
8. Comparing benchmark results 71

List of Algorithms

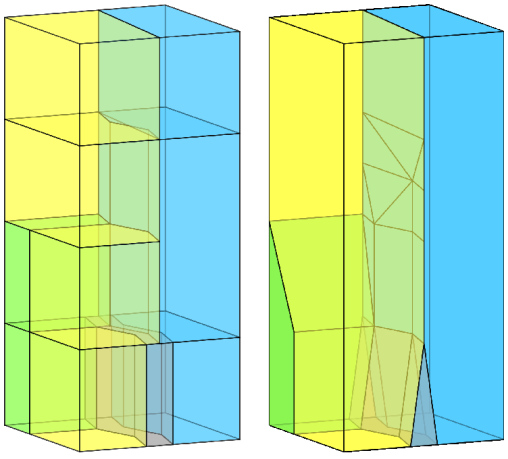
1. Random removal 26
2. Filtering bands 27
3. Point radius density 32
4. Entwine configuration file 91
5. Greyhound configuration file 93

1. Introduction

Point cloud acquisition technologies, such as terrestrial or airborne LiDAR (Light imaging Detection and Ranging), are one of the most promising technologies for surveying and mapping applications. Developed in the 1960s and introduced on airborne platforms on the 1980s, today LiDAR is reaching a level of cost-efficiency where it can be applied to smaller platforms, such as self driving cars or small unmanned aerial vehicles (UAVs). LiDAR systems collect high-density, high-accuracy and very detailed point cloud data about the terrain and surface objects within a relatively short amount of time. Other advantages such as weather and light independence as well as canopy penetration depending on the wavelength make it the technology of choice for many applications within the field of Geomatics.

In 1996 *Rijkswaterstaat* identified these advantages of aerial LiDAR point clouds and started planning the quinquennial collection of a point cloud data set that would cover the Netherlands. AHN1 was collected between 1997 and 2003, and because the major advances in LiDAR scanners within that time span the point density would vary enormously between data sets collected in 1997 and those collected in 2003. The oldest data sets contain 1 point per $16m^2$, while the youngest contain 1 point per m^2. For the AHN2 data set, collected between 2007 and 2012, the goal was set to collect 6-10 points per m^2 (van der Zon, 2013). Currently the AHN3 data set is being collected, and set to be finished in 2020. Because of the incomplete nature of the AHN3 data set, this thesis will focus on vario-scale visualization of the AHN2 data set.

Within the field of cartography, vario-scale visualization of information is still being researched. The idea of showing different *Levels of Detail* would first be introduced as *adaptive zooming* by Cecconi and Galanda (2002) and later this continuous operation would be referred to as *vario-scale visualization* (van Oosterom, de Vries, & Meijers, 2006). Although mention of vario-scale visualization and its potential benefits for point cloud visualization has been made (Poux, Hallot, Neuville, & Billen, 2016), no substantial research has been done into the creation of a vario-scale visualization for point cloud data sets.



(a) SSC for classic tGAP (b) SSC for smooth tGAP

Figure 1: SSC for smooth tGAP (Suba et al., 2013)

Vario-scale vector visualization for cartography is based on the principle of a continuous visualization of the data set. This results in a data index which is best visualized in a 3D cube. Smaller, more detailed polygons are being merged the larger the scale becomes (van Oosterom & Meijers, 2014). This principle is visualized in Figure 1. In the classical approach the polygons do not merge, but rather *jump* from one shape into another once a scale threshold has been reached. This *jump* leaves distinguishable *density jumps*, which can easily be identified when zooming through a data set. Current point cloud visualization methods also contain these *density jumps*.

The vario-scale visualization principle for point clouds that will be introduced in this thesis will eliminate these *density jumps*. This is based upon the same principle as the vario-scale visualization in cartography; the continuous visualization of the data set. For point clouds this is more challenging because there are no semantics; the points in a point cloud are simply X, Y, Z coordinates plotted in the selected Coordinate Reference System (CRS). Whereas cartographic vector data sets have semantic data from which relationships between polygons can be determined. Such as the boundary between polygons, the function of the polygon and the dimensions of the poly-

gon. The principle of vario-scale visualization of point clouds is visualized in Figure 2, where points are being visualized regardless of any scale threshold being reached; points are visualized continuously.

A usability study shows the advantage of using a continuous level of detail, vario-scale, compared to a discrete level of detail when visualizing point clouds (Schütz, Kröstl, & Wimmer, 2019). For a more detailed visualization of a continuous level of detail compared to a discrete level of detail see Appendix A, a discrete visualization, and Appendix B, a vario-scale visualization. In both Appendices a top down visualization is made of zooming through the scale levels. In Appendix A points appear in frame in sudden *density jumps*, whereas in Appendix B the points appear continuously in each frame.

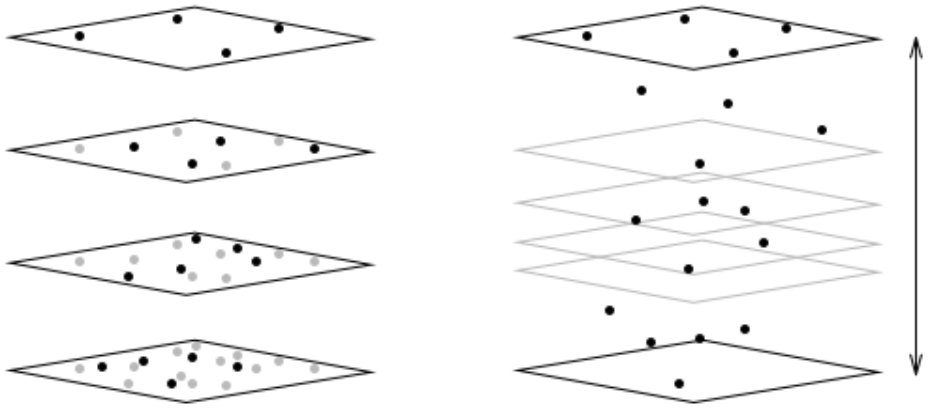


Figure 2: Vario-scale visualization of point clouds

1.1. Problem statement

The problem solved in this thesis is twofold. The first is to create a method for the vario-scale visualization of point clouds. This entails finding requirements for the vario-scale visualization as well as creating possible algorithms for the vario-scale visualisation of point cloud data sets.

Secondly this thesis aims to create a proof-of-concept for the proposed visualization method. The proposed data set to create this method for is the AHN2 data set. The AHN2 data set is one of the largest openly available point cloud data sets.

Vario-scale visualization

Current point cloud visualization methods are heavily reliant on the underlying data structure of the stored point cloud, which has to be efficient for massive data sets.

Currently the most used storage and indexing method for massive point clouds is an octree. An octree is an index structure which stores spatial data, containing XYZ coordinates, in a 3D cube. This 3D cube with sides of length n is recursively split up into eight equal cubes (nodes), each with sides of length $\frac{n}{2}$. This recursion is illustrated in Figure 3.

By storing a set of points in each cube this data structure provides efficient indexing for point cloud because of the spatial relationship that can be derived from the relationship between nodes. This indexing method for point clouds is first proposed by Woo, Kang, Wang, and Lee (2002).

Potree is one of the standards in web visualization of point cloud data sets that uses an indexed octree for web visualisation (Schuetz, 2016). The advantage of this visualization method is that

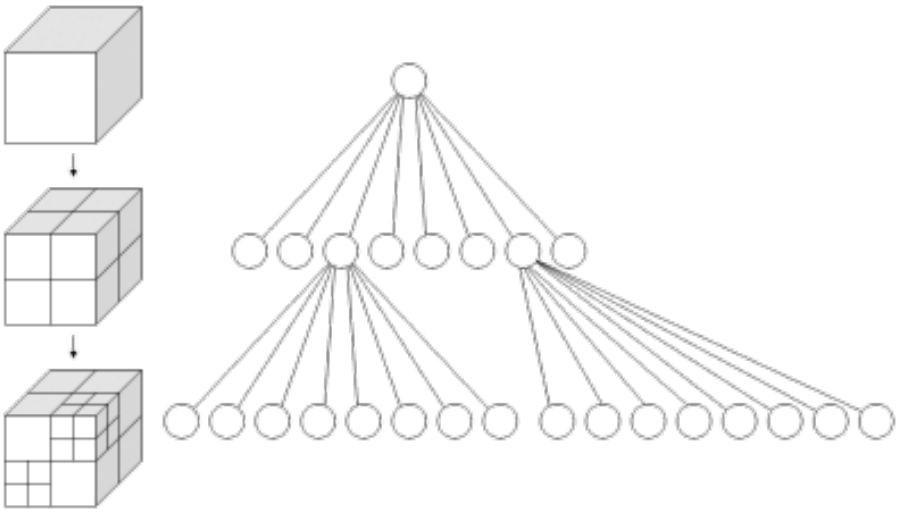


Figure 3: Octree recursive storage

very large data sets can be visualized in real time, because higher levels of detail (the more dense and lower levels of the octree) will not be rendered until required. This limits the amount of data that needs to be visualized, and thus retrieved from the web server.

Another advantage is that this allows regions further from the camera origin to be rendered in lower levels of detail (the less dense and higher levels of the octree). The disadvantage of the underlying octree structure are the discrete *density jumps* in Level of Detail. These *density jumps* occur when moving up or down the octree, in a non-continuous manner. These *density jumps* are especially noticeable during visualization of larger point clouds, when not all data that is rendered on screen is retrieved from the same level of the octree. *Density jumps* in a perspective view of the AHN2 point cloud are illustrated in Figure 4.

The main focus of the vario-scale method proposed in this thesis is to eliminate these *density jumps* from the visualization of the AHN2 point cloud.

Figure 4: Discrete density jumps in the AHN2 web viewer

Massive point clouds

The advantage of fast and cheap acquisition of detailed data face a trade off; point cloud data is very large and has the risk to become unmanageable if not properly stored, processed and visualized.

Because of the increasing ease with which point cloud data can be collected, point cloud data will behave more and more like big data in the future. This means point cloud data systems will face the same defining principles currently faced by other big data systems; Volume, Variety and Velocity. Current local systems for processing point cloud data are ill equipped for handling these future developments, which is why in this thesis a framework will be created that makes use of the scalable advantages of cloud computing.

Many studies have been done on the storage and processing of point cloud data. Most commonly these studies focus on the storage of point cloud data in either files or a relational database. Processing can be done in many ways, it has been commonplace to use open source tools such as LASTools which offer substantial scalability and speed (Hug, Krzystek, & Fuchs, 2004).

These solutions, however, will fall short when in the future point cloud data will be collected not just by specialized teams of surveyors or governments but also by the devices we use every day. Examples of this are self-driving vehicles, artificial reality (AR) / virtual reality (VR) surrounding creation systems and cellular network planning projects. The collection of point cloud data by these applications and others will mean multiple orders of magnitude more point cloud data has to be stored and processed. This is done preferably in the same amount of time as current applications, or faster (real-time). A key application when processing point cloud data is the visualization of the data. This allows users to visually inspect the data, select regions and analyze the data in a clear and orderly manner. Current processing systems are unfit for these applications once the collection of point cloud data will grow even more, with non-real-time rendering of the data being the largest problem.

For these reasons it is important to create a proof-of-concept for the proposed visualization method. This proof of concept should be demonstrated on a massive point cloud, so that it will lay a base for future developments. Processing on massive point clouds requires a careful approach, as well as scalable solutions.

By transitioning the visualization process from local servers to servers on a cloud service it is possible to process large point cloud data sets without needing to own, maintain and service the needed processing power. The ability to up-scale and down-scale the processing power depending on demand have made cloud computing the service of choice for many large and smaller companies performing a variety of data processing tasks with a volatile demand in processing power.

This principle has already been used for point cloud processing, as illustrated by Wang, Hu, Sha, and Han (2017) and Li, Hodgson, and Li (2017). Combining cloud services and distributed computing for data processing will allow for an order of magnitude faster processing of point cloud data and a scalable solution for n users growth (Li, Yang, Liu, Hu, & Jin, 2016).

For these reasons this thesis will use a cloud computing platform, and distributed computing will be used to perform large processing tasks such as indexing of the point cloud.

1.2. Scientific relevance

As mentioned in Section 1.1, the scientific body of research for vario-scale visualization of point clouds is virtually non-existent. The vario-scale visualization of point clouds is an area of research which springs from the vario-scale visualization of vector data sets. This field of research is both larger and older. The basic notion of discrete visualization versus continuous visualization, as visualized in Figure 2, will be built upon.

This thesis will lay the ground-works for further research on vario-scale visualization for massive point clouds, by presenting a theoretical method and practical implementation for vario-scale visualization. This project is open source, and all code is available on GitHub [1]. By choosing the AHN2 point cloud data set this thesis aims to visualize one of the largest LiDAR point cloud data sets currently collected.

1.3. Research question

This research aims to create a vario-scale representation of point clouds by creating both a theoretical method and a practical proof-of-concept. This is done by implementing the method for the vario-scale visualization of the AHN2 point cloud.

The proposed research question for this thesis is:

[1] https://github.com/JippevdMaaden/thesis

To what extent can a vario-scale visualization method be created that eliminates density jumps from the web-based visualization of the AHN2 point cloud?

This research question is supported by multiple supporting research questions:

1. *To what extent is the current body of research done on the vario-scale visualization of vector data sets relevant for the vario scale visualization of point cloud data sets?*
2. *To what extend can a theoretical post-processing approach be created for vario-scale visualization of point cloud data sets?*
3. *Which point-cloud processing framework is best suited to create a proof-of-concept vario scale visualization platform for the AHN2 point cloud?*
4. *To what extend can the theoretical approach be implemented in an existing point cloud web visualization framework?*

The first supporting research question is answered in Chapter 2 and Chapter 3, where the theoretical approach proposed in this thesis is explained. The second supporting question is answered in Chapter 3, where three proposed approaches to enforce vario-scale visualization are explained. The third supporting question is answered in Chapter 4, where the practical requirements for the framework are determined and a choice of components is made for the framework its self. The fourth supporting question is answered in Chapter 5 where the framework and the theoretical approach are combined to form the proof-of-concept for the vario-scale visualization of the AHN2 data set.

1.4. Scientific scope

In the scope of this thesis are two separate objectives:

1. Create a theoretical approach for the vario-scale visualization of point cloud data sets.
2. Create a practical implementation of this approach for the AHN2 point cloud.

Because of the lack of research done on the vario-scale visualization of point clouds data sets, the first objective will mainly be reached by using research from other fields of vario-scale visualization. The theoretical approach for the vario-scale visualization of vector data sets will be analyzed and transferable elements will be used to create a theoretical approach for the vario-scale visualization of the point cloud data sets.

The practical implementation of this theoretical approach will be done for the AHN2 data set. The research objective is to create an initial proof-of-concept for vario-scale visualization of point cloud data sets. This thesis will first create a data set from the AHN2 point cloud that is visualized in the current standard, through octree visualization. Then the proposed method is used to create a vario-scale data set from the same camera angle. The original visualized data set and the vario-scale visualized data-set will then be analyzed both visually and numerically to determine the effectiveness of the implementation.

1.5. Thesis structure

Chapter 2 discusses the current theoretical background, for both vario-scale visualization and point cloud processing. Chapter 3 discusses the proposed theoretical method for the vario-scale visualization of point cloud data sets. Chapter 4 discusses the practical requirements for the proof-of-concept framework, and it describes the decided upon framework as well as the point cloud

data sets used in this thesis. In Chapter 5 implementation of the framework is explained. Chapter 6 presents and analyses the results from the proof-of-concept implementation. Chapter 7 discusses these results and describes what further steps can be made to improve the implementation. Chapter 8 concludes this thesis and reflects on the way in which the main research question has been answered.

2. Theoretical background

The theoretical background of this thesis will lay the ground works for the research done in this thesis. To establish a theoretical framework for vario-scale visualization of point clouds, multiple areas of research will have to be researched. This thesis has two objectives. Firstly to create a theoretical implementation of vario-scale visualization for point cloud data. Secondly to create a proof-of-concept for this implementation. Both of these objectives are introduced in Section 1.1. This Chapter is divided into two Sections, each representing the theory needed to solve an objective.

Section 2.1 will first discuss the current research that has been done on the vario-scale visualization of point cloud data sets, as well as discuss the theory behind vario-scale visualization of vector data sets. This section will support the creation of the theoretical implementation of vario-scale visualization for point cloud data.

Section 2.2 then discusses the current theoretical framework for point cloud processing. Since point cloud processing spans from storing the raw data to processing to visualization, this section is split up into subsections. Each of these subsections will elaborate more upon the specific part of the point cloud processing cycle.

2.1. Vario-scale visualization

Vario-scale vector visualization

One of the first mentions made of vario-scale visualization for vector data-sets, is not referred to with the term 'vario-scale'. But rather with 'adaptive zooming'. As a term introduced by Cecconi and Galanda (2002) it discusses the need to improve the cartographic quality for web mapping and web GIS.

In their paper, Cecconi and Galanda (2002), make the distinction between objects with a predetermined *Level of Detail* that are stored in a multiscale database at different scales. And *temporary visualization* of objects through generalization of topological features for a selected scale at runtime. The objects stored in the multi-scale database are objects that require high computational cost for generalization, and can thus not be done real-time, such as buildings or highways. Objects that are generated on-the-fly require less complex methods and algorithms, objects such as rivers and lakes. The two operations work in tandem and the resulting data-sets are merged to create a temporary map, unique to the scale level and location for which it is created.

The main challenges for this implementation is the real-time generalization of topological objects, and the creation of the temporary map to be viewed client-side. This solution is implemented on the client-side, which is useful for a fast prototype, but leaves some downsides for a real implementation. Implementing the solution on the server-side would cut down the amount of data sent to the client, since the generalization happens before the data is sent. It would allow the solution to utilize the better computational performance on the server for faster generalization or generalization of more computational heavy objects.

This server-side implementation of the adaptive zooming method proposed by Cecconi and Galanda (2002) is introduced by van Oosterom (2005). By this time the method is referred to with the term 'variable-scale'. The server-side implementation is created without redundancy in objects to be stored. All objects are represented by faces and edges. The creation of a data set for unique scale level is done by combining the objects, based on a generalization algorithm that stores relationships between objects in a data structure.

Both the relationships between faces and edges are stored in different data structures, dubbed (Generalized Area Partitioning) GAP-face tree and GAP-edge forest respectively. Faces are assigned an importance value. When zooming through the data set this importance value is used to determine which of the two faces is least important when merging (zooming out), and thus should be removed. The merged face is than assigned the semantics of the most important face and its

importance value is increased. This operation is continued until one face is left at the highest scale. Edges, which represent the topological relationship between faces, can either be removed, merged or changed when faces are merged. These operations are dependant on the GAP-edge forest and the importance ranges assigned to each edge.

Improvements on this method are made by Suba et al. (2013), who implements a tGAP structure (van Oosterom & Meijers, 2014) with a Space-Scale Cube (SSC) for a smooth volumetric representation of gradually changing vario-scale objects. Whereas the GAP structure determines which faces to merge depending on discrete importance values (and thus shows discrete transitions between objects), the tGAP structure in combination with a SSC allows for continuous zoom and gradual change and merging of objects.

Vario-scale point cloud visualization

Mention of vario-scale visualization combined with the visualization of point cloud data sets has been sparse. When these two have been mentioned together (van Oosterom et al., 2014), (Martinez-rubi et al., 2015) is has been referring to vario-scale LoD visualization, which is a discrete solution showing multiple levels of the octree in the same view frustum. This is visualized in Figure 5

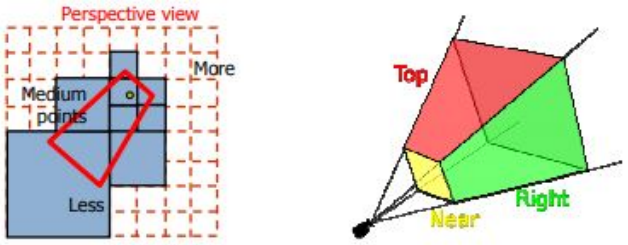


Figure 5: LoD vario-scale visualization (van Oosterom et al., 2015) (left) (MithrandirMage (2012). Retrieved October 12 2017 from https://en.wikipedia.org/wiki/Viewing_frustum

This implementation of LoD vario scale visualization is discussed in Section 1, where the problem of this implementation is visualized in Figure 4. The LoD vario-scale implementation stills shows *density jumps* when visualizing point cloud data sets.

What all research on vector maps has in common, it generalizes based on:

- Semantic data (assign importance value)
- Topological data (edge shape)

And creates a 3D structure based on the 2D levels that are are non-horizontally connected to form a volumetric representation of gradually changing vario-scale objects. What all research on vector maps does not do for point clouds is:

- Take into account 0D objects (no topological data)
- Take into account non-semantic objects

Adding either of the two to a point cloud requires massive amounts of processing power, during the creation of the method it should be determined whether this is needed or not.

2.2. Point cloud processing

This section will give an overview of the current point cloud processing life cycle. This starts at Storage and ends with Visualization.

Storage will discuss current methods used for storing point clouds, both in a file structure and in a relational database. **Indexing** will discuss current indexing methods used for indexing point clouds. **Distributed computing** will go into detail about the current computing methods used to processes massive point clouds. **Cloud computing** will discuss three cloud platforms and their services.

Storage

Currently there are three conventional methods used for storing point cloud data; using an ASCII file, a LAS file or in a relational database. Because of the lack of compression and lack of speed when reading or writing an ASCII file as described by Svalec, Takac, and Zabovsky (2015), using this format is becoming increasingly uncommon. This has left the industry to adapt the LAS file format, and its compressed counterpart LAZ, as the industry standard (ASPRS, 2013). Research into the storage of point cloud data in relational databases has been performed in multiple studies, most recently in a comprehensive benchmark by van Oosterom et al. (2015) where both storage in a LAS file-based solution, storage in a flat relational database model and block storage in a relational database model is bench marked.

While conventionally point cloud data is stored in LAS or LAZ file, a database management system (DBMS) approach has the possibility to enhance the processing of point cloud data by allowing for the querying of not only the point cloud data but other data as well such as vector data to perform spatial queries. In essence storing point cloud data in a LAS file versus storing point cloud data in a DBMS constitutes to the same; both data types facilitate the storage of point cloud data in a block, which is the file its self for LAS files or which are blocks (SDO_PC and SDO_PC_BLK for Oracle) or patches (PCPATCH for PostgreSQL) for DBMS's. The comprehensive benchmark by van Oosterom et al. (2015) shows the differences between both storage methods; a LAS file implementation seems to far outperform all DBMS's when it comes to loading the largest data set (2130s vs. 8490s), while DBMS implementations using the block property of the DBMS compress the data much better (107 GB vs. 440 GB).

For simple queries currently the LASTools point cloud data management system (PCDMS) offers the best performance. Relational databases showed the potential to be more efficient at processing more complicated queries.

For purposes of this thesis distributed storage methods will be discussed, using a LAS file-based storage or a relational database. Since most studies performed in the distributed storage are using Hadoop and the Hadoop Distributed File Storage (HDFS) (Hanusniak, Svalec, Branicky, Takac, & Zabovsky, 2015; Li et al., 2016; Li et al., 2017), this will be used as the main case study. Hadoop is an open source tool for distributed storage and processing of big data (White, 2009). These storage methods can however be applied to other distributed storage and processing frameworks.

The HDFS will store point cloud data similar to storing it on a single location. The advantages is that not all storage locations will have to be queried when only data stored on one node of the HDFS is required. Thus whether a storage method is suitable for the point cloud data is dependent not only on the querying but also on the indexing methods used both across nodes and on the nodes themselves of the HDFS.

Indexing

Conventional indexing methods for point cloud data include octree indexing by Elsenberg, Borrmann, and Nüchter (2013), R-tree indexing by Gong, Zhu, Zhong, Zhang, and Xie (2012), or k-d tree indexing by Sayar, Eken, and Öztürk (2015).

Indexing a point cloud is a necessity when working with point clouds, and different indexing methods are more suitable for different applications. In Figure 6 all three basic indexing methods are illustrated. Indexing point clouds drastically improves query execution time and enables efficient searching and extraction of data.

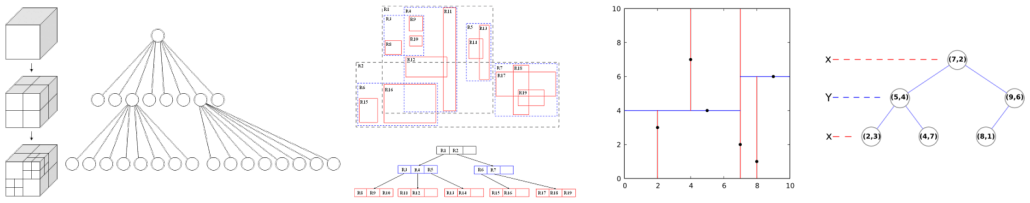


Figure 6: Octree, R-Tree and k-d tree indexing visualized

An octree is an indexing method derived from both binary trees and quadtrees. An octree represents a 3D spatial cuboid which is contains geometry. This cube is then subdivided into eight smaller cubes which each also contain geometry. This subdivision is done until leaf nodes are created; nodes which only contain one geometry and thus do not have to be subdivided further. Because a point cloud, especially very large point cloud data sets, cover much of the earths surface, the higher levels of these octrees function more like quadtree, thus not all the octants of the octree have to be subdivided. To prevent this two stopping rules for octrees are created; maximal depth and a minimal number of points.

A k-d tree is also derived from a binary tree structure and its commonly used to organize points in space, and it is useful for range and k nearest neighbor searches. Creating a k-d tree is similar to the Quick Sort algorithm, where the median point is used to divide the tree further along either the x or y dimension.

An R-tree is similar to an octree because it used geometry to bound the spatial data. In stead of a cube the geometry used is a minimum bounding rectangle, which allows for more freedom when indexing point cloud data. The R-tree is a balanced search tree, which means that unlike an octree all its leaf nodes are on the same level in the tree. Having a balanced tree is an advantage for search queries but has as a pitfall that some minimum bounding rectangles might overlap or cover too much empty space, which results in very bad worst-case performance.

When indexing a distributed data set two types of indexing are important; global and local indexing. Global indexing refers to the indexing method of distributed data sets among nodes on the distributed network. The efficiency of this network depends on multiple factors including the amount of redundancy built into the infrastructure. Local indexing refers to indexing performed on the data that is stored on a single data node.

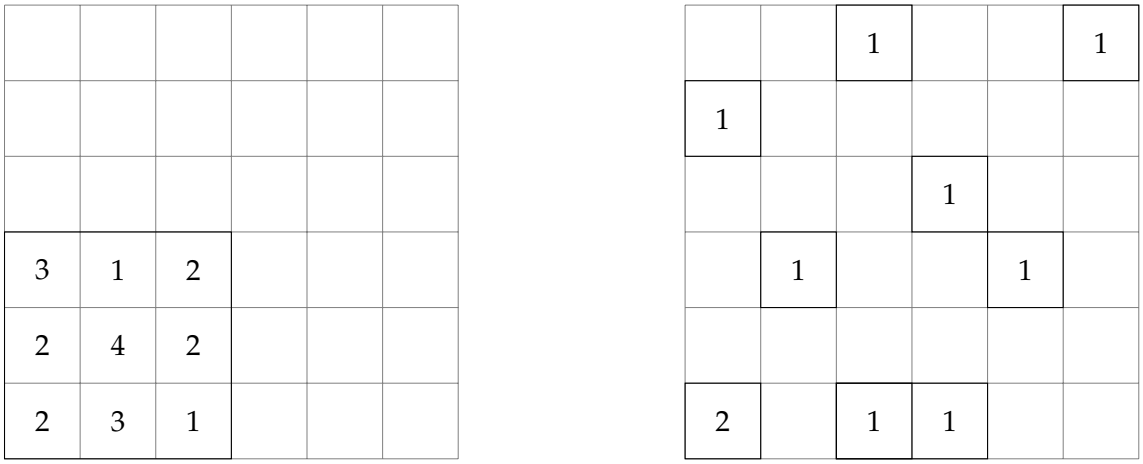


Figure 7: Tile based decomposition (left) vs. Domain based decomposition (right)

Global indexing such as the methods presented by Li et al. (2017) are dependant on the purpose of the point cloud data set but can generally be distributed into two types of decomposition; a

domain-based decomposition and a tile-based decomposition. In a domain-based decomposition the spatial relationship of the data is considered when processing the data on node x. Thus data is first collected according to spatial locality and then processed. A tile-based decomposition does not consider the spatial relationship but minimizes latency by assigning data stored on node x to be processed on node x. In Figure 7 these spatial decompositions are illustrated. The numbers inside the tiles represent the nodes on a distributed file storage infrastructure. It is illustrated that to obtain spatial locality of data within a processing operation the node locality in the distributed file storage infrastructure is compromised. Both methods are suited for different types of processing purposes. For visualization purposes both the lack of latency and spatial locality are important, which implies the assignment of data to specific nodes.

Local indexing indexes the data stored in a single node, which in the case of point cloud data is done by the above mentioned conventional spatial indexing methods.

The indexing methods discussed so far are the most commonly used indexing methods for local indexing. Other indexing methods are derived from the octree, k-d tree or R-tree including the 3DOR-Tree by Gong et al. (2012).

Local indexing methods should improve point retrieval for visualization purposes once the correct node of the distributed system is located.

Distributed computing

Distributed computing, also referred to as parallel computing, is a computation where a single problem is split into multiple sub problems, either calculations or processes, which are all solved simultaneously by multiple computers. Once all sub problems are solved these parallel solutions are combined again to form the final solution. This type of computing allows for enormous problems to be solved relatively fast, if the problem adheres to two requirements. The first requirement is that the problem can be subdivided into sub problems which can be solved in parallel. The second requirement is that the overhead created by splitting and combining the problem should be less than the computational gains. Apache Hadoop, an open-source framework based on the MapReduce framework by Dean and Ghemawat (2008), is currently the most popular and well documented distributed computing framework.

The Apache Hadoop framework is suitable for linear processes, where sub problems are not dependent on other sub problems to be solved. Apache Hadoop does not permit the re-use of data between the computations of sub problems, unless the data is written to temporary or permanent external storage and read again. Unless this storage is permanent, if a sub problem is not solved because of node failure this would break the operation and would mean all the previous sub problems would have to be recomputed. Because of this the distributed computing process is either not very robust or too slow for iterative processes.

The creation of Apache Spark by Zaharia, Chowdhury, J. Franklin, Shenker, and Stoica (2010) changes this with the introduction of resilient distributed data sets (RDD). RDD is the fundamental data structure of Apache Spark (Zaharia, Chowdhury, Das, & Dave, 2012). It is a read-only collection of records that can independently of any RDD before it reconstruct the sub problem if information is lost. An RDD can be explicitly cached in memory and can be reused across parallel operations.

Because Apache Hadoop and Apache Spark function differently, their applications vary as well. Sparks main applications are in machine learning and data science, which because of their iterative nature benefit greatly from having a data structure such as RDD (Zaharia et al., 2012). Well documented packages such as MLlib make Spark easily accessible for these task (Meng et al., 2016).

However, Apache Spark functions the same as Apache Hadoop when used for non-iterative processes, because the underlying MapReduce framework is the same and there is no advantage in using RDDs when performing a linear computation in which data does not have to be reused.

Because many GIS applications are linear, they do not require the re-use of data but simply query the data once to be processed, Hadoop remains the main distributed computing framework used for GIS applications. Illustrated by the recent research into point cloud processing with Hadoop (Wang et al., 2017; Li et al., 2017), and the amount of documentation there is for Hadoop.

Many distributed computing applications have been built for the processing of spatial data and point clouds. Examples of these are HadoopGIS created by Aji et al. (2013), VegaGiStore created by Zhong et al. (2012) and other solutions such as the solution created by Jian et al. (2015) or by leyman Eken and Sayar (2015). Current research in the distributed processing of GIS data focuses mostly on Hadoop as a framework for parallel processing and storage (Wang et al., 2017; Růžička, Orčík, Růžičková, & Kisztner, 2016; Li et al., 2017).

This thesis aims to improve the scalability of the AHN2 point cloud viewer by using a distributed cloud computing solution. Because distributed computing can be a costly solution, and predictions of user traffic are difficult to make, automatic scaling solution as presented by Li et al. (2016) should be researched. This automatic scaling Hadoop cluster is launched on AWS, making it easy to add and remove nodes.

An automatic scaling Hadoop cluster consists of three types of nodes; 1 master node which performs the MapReduce task, n core-slaves which both store and process data and 0 to k compute slaves which are initialized when user traffic becomes too high.

A solution such as the automatic scaling Hadoop cluster shows the advantages of combining distributed computing with cloud computing to form scalable and efficient point cloud processing clusters.

Cloud computing

Cloud computing is a new development that revolutionizes the way both individuals and businesses access computing services. Computing is done on a network of off-site computing resources, which together form a data-center, accessed through the internet (Byrne, Corrado, & Sichel, 2017). The term cloud computing encompasses many services, which are generally divided into three types; Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) (Pancholi, Patel I, Principal, Gandhi, & Patel, 2017). A brief overview of these services and a distinction between what the customer manages versus what the provider manages is given in Figure 8.

As illustrated these services have an increasing level of abstraction, where the customer needs less and less knowledge of the underlying principles. As described by Dutt, Jain, and Kumar (2018) IaaS provides the physical infrastructure and manages these machines. This leaves the customer responsible for the patching and maintaining of the operating system and application software. While this is more work, this allows for greater degrees of freedom. PaaS removes the necessity for the customer to manage the operating system and underlying architecture of the service, the customer merely manages the application software. SaaS allows for the least amount of freedom and comes with the greatest ease of use; the customer does not manage any part of the system. In this service the provider manages everything and the user merely pays for usage.

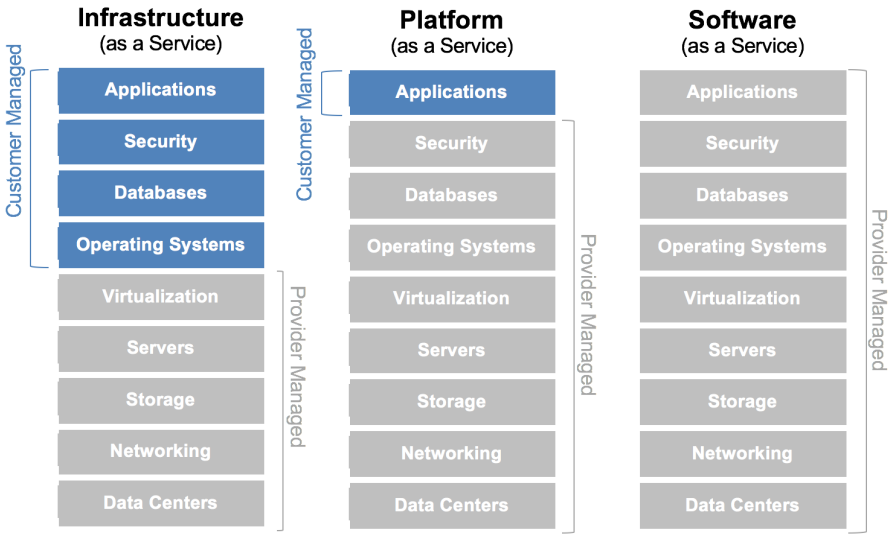


Figure 8: Infrastructure, Platform and Software as a service. (Jamesmbond (2013). Retrieved December 17 2017 from https://mycloudblog7.wordpress.com/2013/06/19/who-manages-cloud-iaas-paas-and-saas-services/)

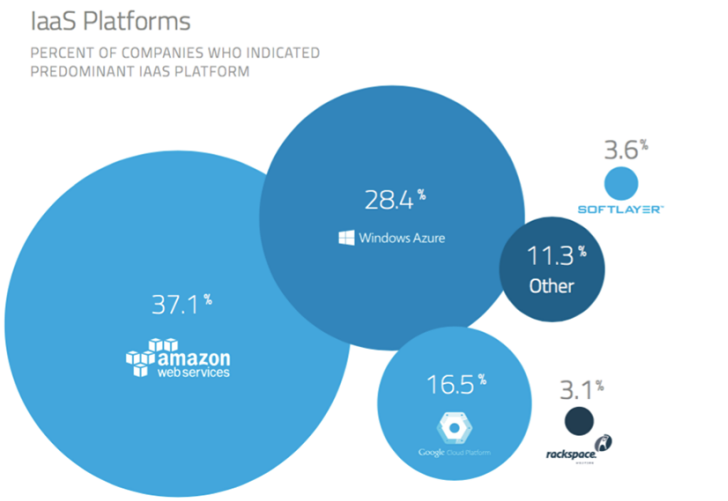


Figure 9: B2B IaaS platform adoption. (Skyhigh Networks (2017). Retrieved December 17 2017 from http://www.evontech.com/what-we-are-saying/entry/microsoft-azure-vs-amazon-aws-comparison-between-two-cloud-computing-giants.html)

Because of the use case and the lack of available ready-made applications for storing, processing and visualizing point clouds, this thesis will focus on IaaS. This will allow for the greatest degree of freedom when processing point cloud data. Currently there are three major providers which provide cloud infrastructure services, these are; Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform, as depicted in Figure 9. Other IaaS providers include companies such as IBM and Salesforce. These platforms however will not be considered for this thesis as Amazon, Microsoft and Google are making incredible progress in the development of their infrastructure, and are widely adopted by businesses and well documented.

All three providers offer a wide range of services, for this thesis three of these services are rele-

vant; Compute services, Database services, Storage services. As seen in Table 1 all three providers offer these services.

	AWS	Microsoft Azure	Google Cloud
Compute services	Elastic Compute Cloud (E2)	Virtual Machines (VMs)	Compute Engine
Database services	Relational Database Service (RDS)	SQL Database	Cloud SQL
Storage services	Simple Storage Service (S3)	Azure Storage	Cloud Storage

Table 1: IaaS service provided by AWS, Microsoft Azure and Google Cloud

AWS started in 2004 and is the longest running cloud service provider. Its services are therefore well documented and AWS is regarded as being the most comprehensive cloud service. Microsoft and Google are quickly catching up though and are rolling out many new services and solutions of their own.

For this thesis AWS will be used as the cloud computing platform. Because AWS is the oldest cloud computing platform its boasts multiple advantages over its competitors. Previous personal experience and the ability to easily and cheaply launch Hadoop or Spark clusters with Elastic MapReduce (EMR) make this the preferred cloud computing platform for this thesis.

3. Theoretical approach

3.1. Theoretical vario-scale visualization

For the creation of a methodology for vario-scale visualization of point cloud data sets, two important design decisions have to be made. The first decision, which framework to use, is made in Chapter 4. An existing visualization framework is chosen and will be modified where needed. The second decision is whether to use either a 4D indexing method, or a 3D indexing method which will require post processing. The difference between these two methods is visualized in Figure 10. A 4D indexing method would use a 4^{th} dimension such as *scale* or *importance* to allow for vario-scale querying of the data set. Whereas a 3D indexing method uses XYZ indexing and post-processing to create the vario-scale visualization method.

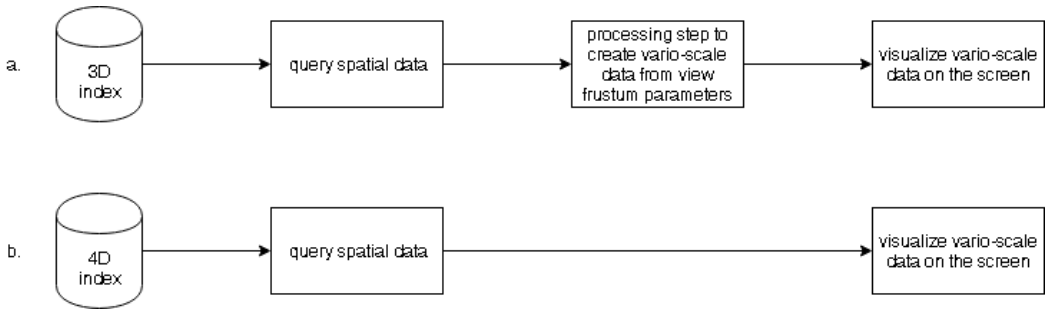


Figure 10: 3D (a) vs 4D (b) indexing method

Creating a 4D indexing method would allow vario-scale point selection to be done by querying the indexing structure; the correct query would return a vario-scale partition of the data set. This implementation is similar to the SSC discussed in Chapter 2. Due to the nature of this thesis, the creation of a proof of concept, and the time constraint in which it is executed, the decision is made to use the 3D indexing method. This method requires a separate processing step to go from the discrete indexed point cloud to a vario-scale partition of the data set.

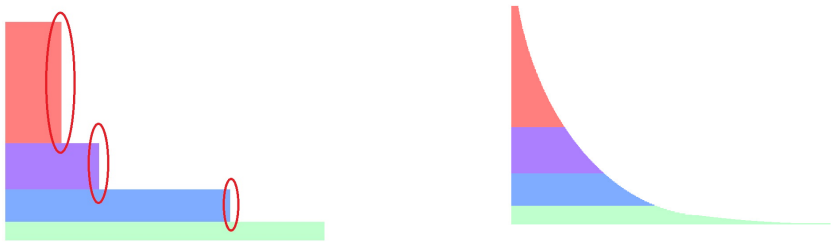
The resulting data should allow the point cloud to be rendered in view without any density jumps. Currently the industry standard for web point cloud visualization is the use of Octree blocks streamed from a web server. This results in density jumps as visible in Figure 11.

Figure 11: Discrete density jumps in the AHN2 web viewer

A good starting point for the analysis of this visualization, is analysing the relationship between density (*points per m2*) and the distance from the camera for current visualization methods using an underlying octree index. This visualization is dubbed *Density pyramid*; where a 2D slice is extracted from the 3D view frustum. Throughout this Chapter density will be referred to as

points per m^2, and not *points per* m^3, because of the 2D nature of aerial LiDAR data sets. These data sets have great point distribution in the X and Y direction, but relatively small point distribution in the Z direction. Because the data set behaves more like a surface than a 3D volume, in this Chapter all calculations will be done using m^2 as opposed to m^3.

In the density pyramid in Figure 12(a) a depiction is made of the relationship between the distance from the camera and the density of the pointcloud. There is a clear visualization of the *density jumps*, marked with red circles, as seen in Figure 11. In Figure 12(b) the same depiction is made, but without the *density jumps*. The relationship between density and distance to camera has been changed to a continuous relationship one. This relationship can be described by a *density formula*. The formula seems to have similarities to a decreasing function such as $f(x) = 0.5^x$ or $f(x) = log_{0.5}x$, with $f(x)$ being the density and x being the point distance from the camera.



(a) Density pyramid with density jumps

(b) Density pyramid without density jumps.

Figure 12: Density pyramid with (a) and without (b) density jumps

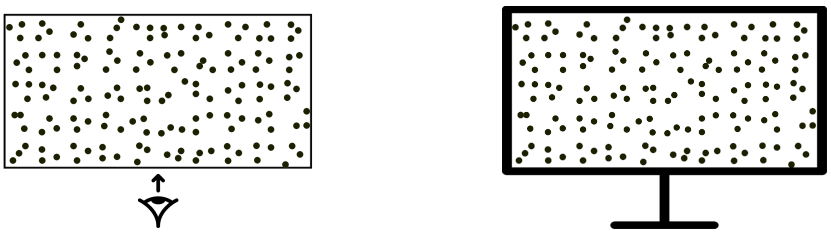
With the creation of this *density formula*, a smooth continuous visualization of a point cloud can be realized. There are two main questions for the creation of a *density formula*:

- What function best describes the density formula of LiDAR data sets?
- How should this density function be reinforced in the point cloud data set?

3.2. Density formula

The density formula is used to describe the distribution of density relative to the camera distance. By using a continuous formula a continuous distribution of points is made, eliminating *density jumps*. This Section will discuss the ideal density formula, without taking into account the implications this has for the Octree indexing structure of the web viewer. This will be done according to three examples, the last one being the vario-scale visualization method. For each example two types of images will be shown, a top-down image of the data-sets spatial extent and a perspective view of the data set as experienced on the computer screen. An example of the top-down image is given in Figure 13(a), and an example of the perspective view is given in Figure 13(b).

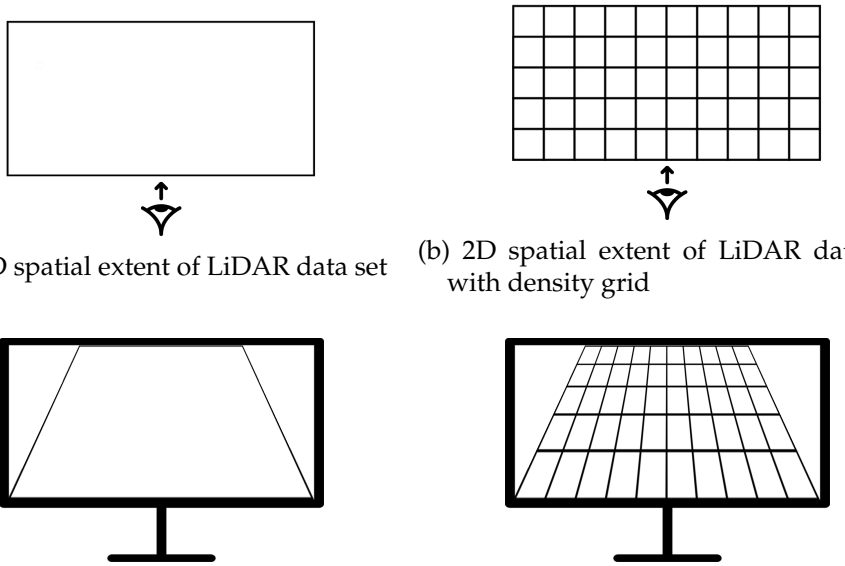
For a continuous viewing experience it is important to note that a pixel is only able to show a single point, thus it does not make sense to render more than 1 point per pixel. Currently the most common screen resolutions is 1 920 x 1 080 pixels, characterized by 1 920 pixels displayed across the screen horizontally and 1 080 pixels down the screen vertically; giving the screen 2 073 600 pixels. In this thesis the amount of 2 073 600 pixels will be used, although in the future it is very likely that 4K screens will become the standard. These screens have a resolution of 3 840 x 2 160 pixels, and thus have 8 294 400 pixels in total. Given that the users screen has 2 073 600 pixels, it should be noted that ideally no more points are requested by the web viewer, since these points can not be rendered on screen. This does not keep into account caching, but for simplicity this will not be regarded as important.



(a) (b)

Figure 13: Example of a top down view (a), and a perspective view (b) of the same data set

For a clear explanation of the *density formula* it will be illustrated in 2D and 3D in increasing steps of complexity. In Figure 14(a) the top-down 2D spatial extents of a LiDAR data set are visualized. In Figure 14(b) we have overlayed this spatial extent with a grid of $n * m$ units. In Figure 14(c) the same spatial extent of the data set is shown, but from the perspective view. The density grid is then also translated to a perspective view, similar to what the end-user might experience in Figure14(d). Throughout this Section we will fill this grid with varying amounts of points, to see what effect this has on the density of point is the data set and the way these points are perceived by the user.



(a) 2D spatial extent of LiDAR data set

(b) 2D spatial extent of LiDAR data set, with density grid

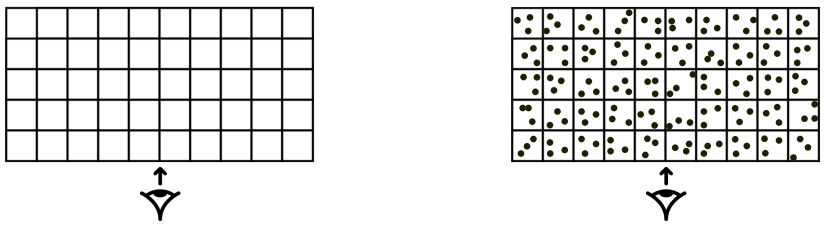
(c) 3D perspective spatial extent of LiDAR data set

(d) 3D perspective spatial extent of LiDAR data set, with density grid

Figure 14: Overlaying a grid on the 2D and 3D spatial extents of a LiDAR data set to clarify how density distribution is affected by the perspective translation

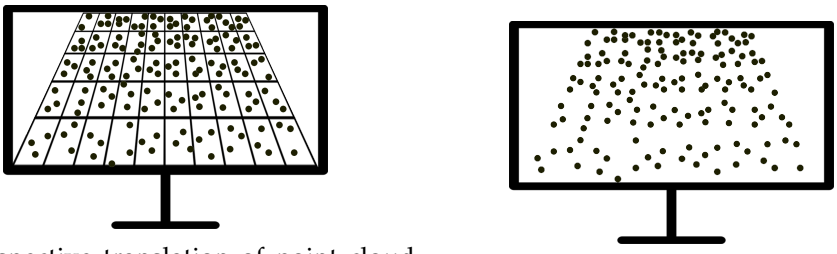
Constant density from camera

First let us illustrate how a constant density in the data set is changed after the perspective translation. The density at any distance from the camera (shown in the top-down view as the eye) is the same. In Figure 15(a) we have created the same spatial extents of the LiDAR data set, with a grid. In each of the grid spaces we will now project an equal amount of points, as seen in Figure 15(b). For this example we use three points. Once the perspective translation is made in Figure 15(c), it becomes evident that this constant density does not provide a constant density of the data set on the screen. In Figure 15(d) it is visualized that the further away from the user, the more dense the data set appears to be. The grid changes shape, while there still remain 3 points in each grid cell. It is evident that the grid closer to the user, ie. in front, is perceived as larger. While the grid in the back is perceived as smaller. This means the larger sections in front are represented by the same amount of points as the smaller sections at the back, making the visualization sub-optimal.



(a) 2D spatial extent of LiDAR data set, with density grid

(b) 2D spatial extents, with a density of 3 points per $n * n$ units



(c) Perspective translation of point cloud with a density of 3 points per $n * m$ units

(d) User's visualization of the point cloud

Figure 15: Translating the 2D spatial extents with a continuous density of three points per $n * m$ units, to a perspective view

The density function of the points in the 2D spatial extent is visualized in Figure 16.

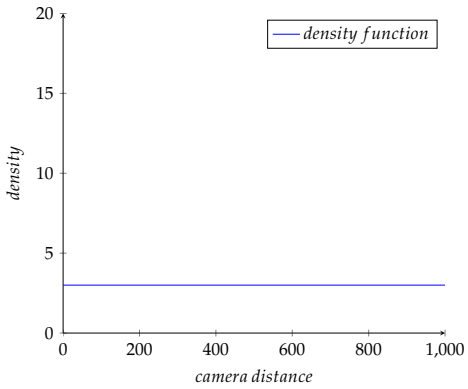
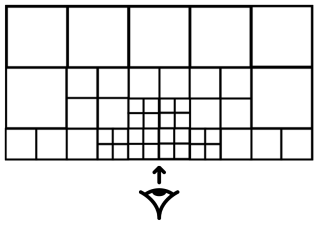


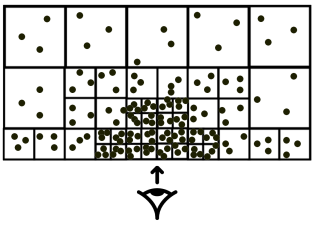
Figure 16: Discrete density jumps in the AHN2 web viewer

Octree blocks density from camera

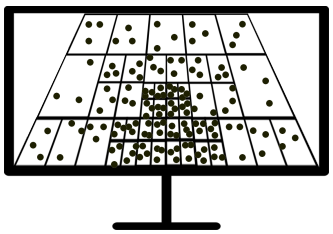
Currently web viewers select points from the octree in a more advanced matter as described in Chapter 4. The blocks are selected depending on their distance to the camera; close to the camera more dense blocks of the octree are selected. An example of what this means for the density grid is given in Figure 17(a). Each block in the octree contains an equal amount of points, the difference being that these are spread out of an increasingly larger spatial extents, as illustrated in Figure 17(b). In Figure 17(c) the perspective translation is done. For the user this means there is a more uniform density across the screen which is visualized in Figure 17(d).



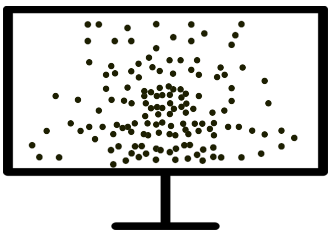
(a) 2D spatial extents, overlayed with an Octree selection grid



(b) 2D spatial extents, with an increased density closer to the camera



(c) Perspective translation of point cloud using octree selection grid

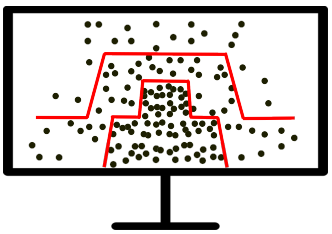


(d) User's visualization of the point cloud

Figure 17: Translating the 2D octree selection method to a perspective view

This however results in the *density jumps* as introduced in Chapter 1, and an illustration of this is made in Figure 18.

(a)



(b)

Figure 18: Density jumps visible in both the real point cloud viewer (a) and the schematic visualization of the problem (b).

The *density formula* will look similar to Figure 19. The density jumps as a result of the octree selection method are highlighted in red.

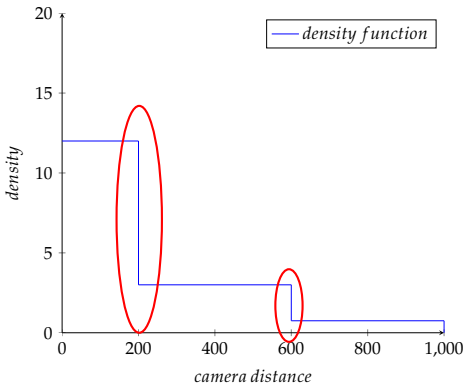


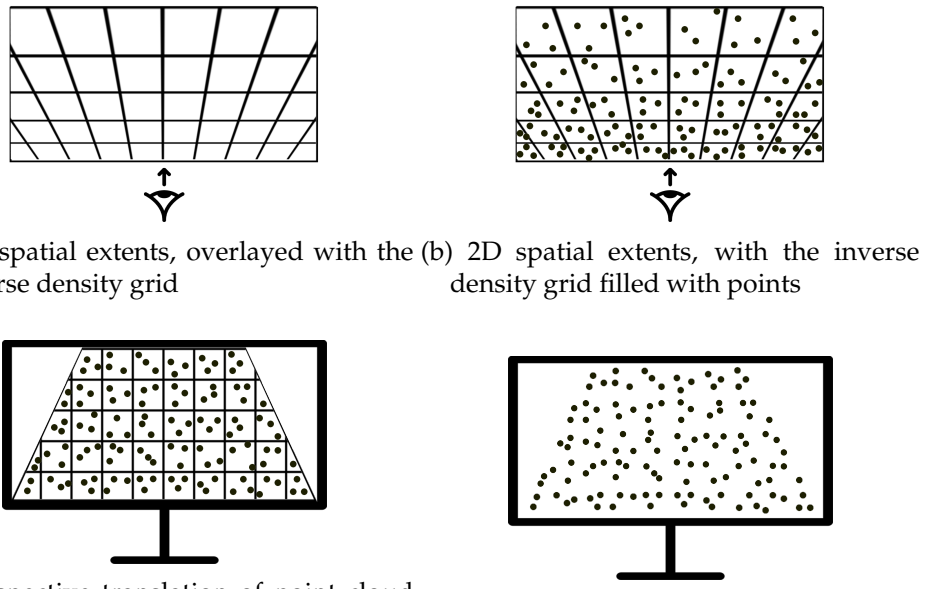
Figure 19: Octree density jumps in the AHN2 web viewer

Vario-scale density from camera

Ideally the density on the screen should be perceived as constant, which means that instead of overlaying the 2D spatial extents with a grid, and translation that to the perspective view, the order should be reversed. The perspective view should be overlayed with a grid, which should then be translated to the 2D spatial extents using an inverse matrix translation, and an improved density selection is realized. This means that vario-scale visualization is tied closely to the perspective matrix translation. The most important variables that contribute to the perspective matrix translation are:

- Field of View (FoV)
- Near plane
- Far plane

A schematic workflow of how to realize a constant density on the computer screen is visualized in Figure 20. In Figure 20(a) the resulting grid from the inverse matrix translation is overlayed on the spatial extent of the LiDAR data set. In Figure 20(b) this grid is filled with a constant number of points per grid space. In Figure 20(c) the translation is made to the screen, which results in the constant density grid. Finally in Figure 20(d) the user is presented with a constant density on the screen, and more importantly no *density jumps* are visible in the data set.



(a) 2D spatial extents, overlayed with the inverse density grid

(b) 2D spatial extents, with the inverse density grid filled with points

(c) Perspective translation of point cloud using the perspective matrix translation

(d) User's visualization of the point cloud

Figure 20: Translating the 2D spatial extents to a vario-scale perspective view

The resulting *density formula* of the data set will look similar to Figure 21.

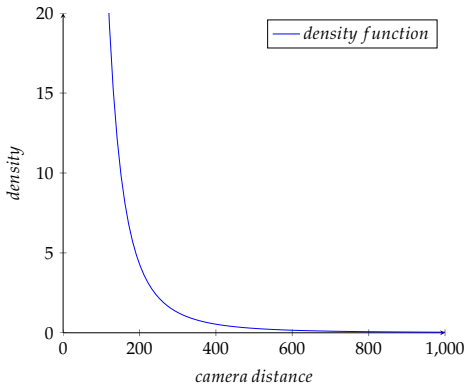


Figure 21: Vario-scale density function for point cloud data sets

This *density function* results to a continuous density on the screen. This means that during the calculation from the view frustum to the viewing volume, visualized in Figure 22, the density is transformed to be uniform across the viewing volume.

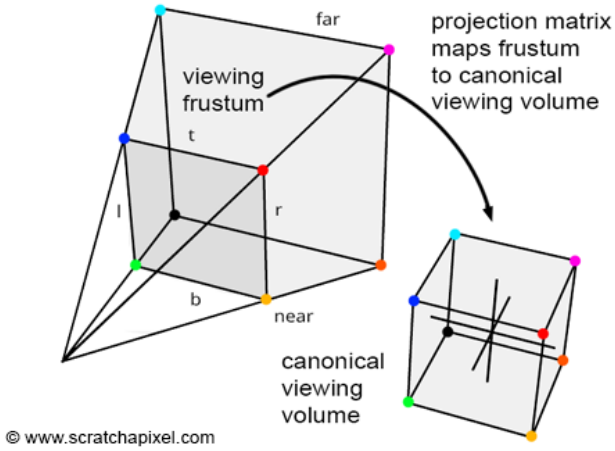


Figure 22: Mapping from view frustum to the viewing volume. (Scratchpixel 2016. Retrieved April 16 2018 from http://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/projection-matrices-what-you-need-to-know-first)

Vario scale density using perspective transformation matrix

This translation from truncated square pyramid to the viewing volume of $2x2x2$ is done using the perspective transformation matrix, for which a simplified version is shown in Figure 23. Where f represents the far plane, n represents the near plane and S represents the field of view (FoV), as shown in equation 1

$$S = \frac{1}{tan(\frac{FoV}{2} * \frac{\pi}{180})} \tag{1}$$

In Figure 23 x_s, y_s and z_s are the coordinates in the viewing volume, derived from x, y and z coordinates in the view frustum. Each points XYZ coordinates, inside the view frustum, is translated to $X_sY_sZ_s$ coordinates, inside the viewing volume. From this we can deduct that the density is also translated through the perspective transformation matrix.

In the viewing frustum there are six planes; the plane closes to the camera is referred to as the near plane, and the plane furthest away from the camera is referred to as the far plane. The four other planes are determined by the FoV. According to this, the density function is a product of the

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ w_s \end{bmatrix} = \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & -\frac{f}{(f-n)} & -1 \\ 0 & 0 & -\frac{f*n}{(f-n)} & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Figure 23: Perspective transformation matrix

parameters that define the view frustum, the *near plane*, *far plane* and the *FoV*.

Finding the *global density function* that, dependant on these three variables, will describe the vario-scale relationship for every screen is outside of the scope of this thesis. This will be discussed in Chapter 8 as future work.

Density function

In this thesis, for each frame that is evaluated, a *density function* will be determined depending on the octree selection procedure that is used by the proposed framework. This allows the experimentation with multiple *density functions* per frame, and will produce results that are both numerically and visually analysed to propose a vario-scale solution. In Chapter 6 a frame is visualized in a vario-scale manner, using four *density formulas*. The frames density function is shown in Figure 24. Originally the frame's density formula is similar to:

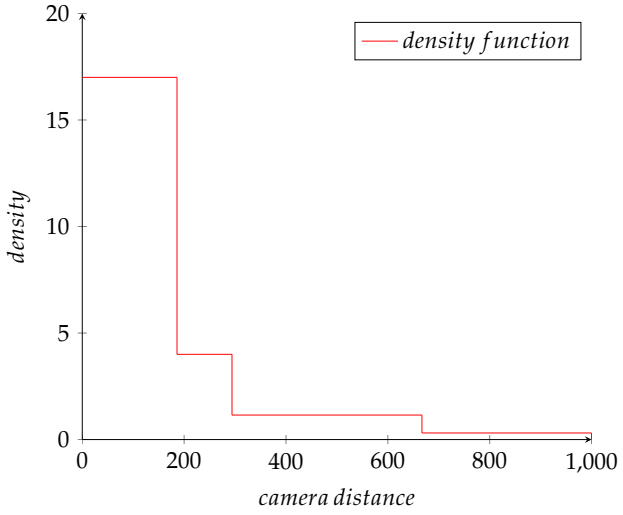


Figure 24: Density function of the frame

Four *density functions* are created for comparison, each of with preserves a different percentage of points from the original frame. Algorithm 2 shows the *density function* that preserves 75% of the original points. Algorithm 3 shows the *density function* that preserves 75% of the original points. Algorithm 4 shows the *density function* that preserves 75% of the original points. And Algorithm 5 shows the *density function* that preserves 75% of the original points.

$$\frac{1}{0,6789(0,0035 * camera\_distance)^3} \tag{2}$$

$$\frac{1}{0,6789(0,005 * camera\_distance)^3} \tag{3}$$

$$\frac{1}{0,6789(0,007 * camera\_distance)^3} \tag{4}$$

$$\frac{1}{0,6789(0,01 * camera\_distance)^3} \tag{5}$$

All four of these *density functions* are visualized in Figure 25, and the resulting vario-scale data sets are presented and analysed in Chapter 6.

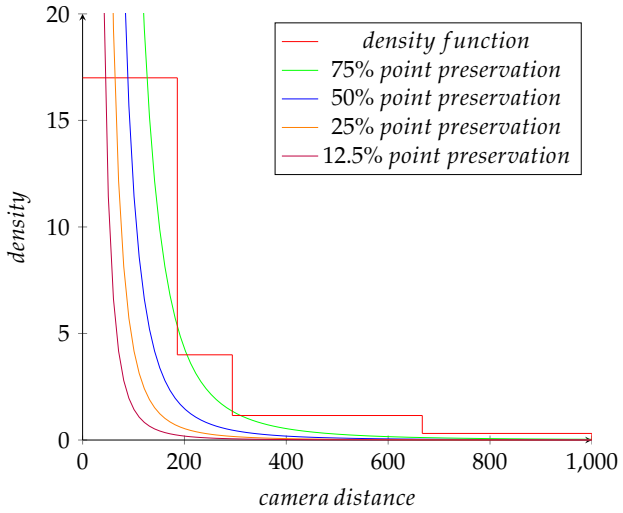


Figure 25: Four density formulas visualized

3.3. Algorithmic implementation

For this thesis three possible reinforcements of the density formula have been explored.

- Random removal
- Filtering bands
- Point radius density

These three methods all have advantages and disadvantages in terms of produced result, difficulty of implementation and computing time.

Random removal
Random point removal works on the basis of generating a random value for each point, and either keeping or discarding the point according to a threshold. This threshold is determined by the *density formula*. Python code for the Random Removal algorithm is found in Algorithm 1. For each point four variables have to be computed; distance from the camera d, random value r, local density l and threshold t.

The distance from the camera for point p is calculated using equation 6, where cp is the camera position.

$$d = \sqrt{(p.X - cp.X)^2 + (p.Y - cp.Y)^2 + (p.Z - cp.Z)^2} \tag{6}$$

Local density is determined for a circle with radius of 0.5642 meters around a point, which gives the density per m^2 locally, since $Area = \Pi r^2$ so for $r = 0.5642$ the Area will be $1m^2$.

Upside of this method is that random selection is not affected by factors such as spatial point order. The downside to this method is that it requires the determination of the *local density* per m^2 for each point in the point cloud data set. Many of these computations are redundant because they are done for points that will not be selected, but is needed because the percentage of points to remove (threshold between 0-1) can only be determined if the complete density is known.

Any assumptions of uniform density could significantly speed up this process. Although it is known that for the AHN2 data sets, which should have a density of at least 9 points per m^2, the density varies from 8 points per m^2 to 40 points per m^2 locally. This is dependant on many factors such as flight path and surface structure. For this reason no assumptions are made on the density of the point cloud as a whole.

Algorithm 1 Random removal

```
import numpy as np
import random
import scipy

def random_removal(points, camera_parameters, density_function):
    """

    """
    camera_origin = camera_parameters.origin
    selected_points = set()
    kdtree = scipy.spatial.KDTree(points)

    for point in points:
        d = distance_from_camera(point, camera_origin)
        r = random(0,1)
        local_density = len(kdtree.query_ball_point(point, 0.5642))
        t = density_function(d) / local_density

        if r >= t:
            continue
        else:
            selected_points.add(point)

    return selected_points
```

Filtering bands

Filtering bands works on the basis of splitting the data set into smaller partitions, and enforcing the *density formula* for these smaller partitions. It essentially makes the *density jumps* so small they appear to be continuous. In Figure 26 these bands are visualized, where in the distance d from the camera an n amount of bands is created with a width of w.

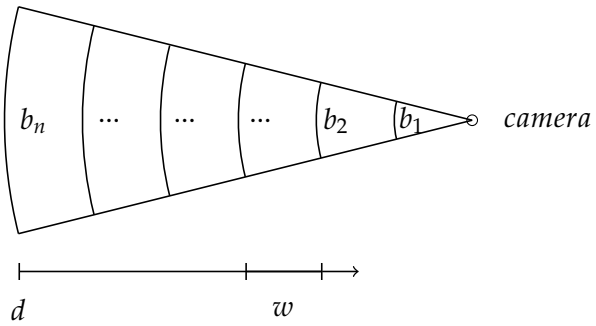


Figure 26: Filtering bands

Removing points in the pointcloud by filtering bands is done by first producing the bands used to filter on. This splits up the data set into parts that each have an allowed point threshold. Circular bands have been chosen for this implementation, extending outwards from the camera origin. The first band b_1 starts at the camera origin and is a circle with radius w. Each next band $n_i + 1$ will also have have width w and start at a radius of$i * w$ from the camera origin and end at a radius of $i + 1 * w$ from the camera origin. In Algorithm 2 the *filtering bands* method is implemented in Python.

Algorithm 2 Filtering bands

```
def filtering_bands(points,
                    camera_parameters,
                    density_function,
                    width = 1.0):
    """

    """
    distance = camera_parameters.distance
    origin = camera_parameters.origin
    selected_points = set()

    for i in range(distance / width):
        band_radius_center = i * width + 0.5 * width
        local_points_array = points_in_band(origin,
                                            band_radius_center,
                                            points)
        allowed_points = density_function(band_radius_center)

        selected_points.add(local_points_array[allowed_points]

    return selected_points
```

Between bands the density changes using the *density formula*. The density is calculated for the

centre of the band. This allows the density to gradually decrease between bands. Inside bands the density is enforced using arbitrary selection, up to a threshold t determined by the *density formula*. This selection can be made non-arbitrary, random for example, which would increase computing time.

The advantage of this approach is the ability to keep local features visually attractive while simultaneously gradually decreasing the density. It is however not truly continuous because of the discrete nature of the bands. This method requires two functions, the *density formula* that determines the density descent as a local threshold t. And the calculation of local points in the band derived from the *camera origin*, *band radius center* and the total *points*.

For the Filtering bands approach an implementation has been tested whereby a set amount of points for each band is selected, removing the need for local density calculations. This would in theory allow a significant speedup. However due to the data sets spatial extents intersecting with the outer spatial bands this would result in a *density formula* depicted in Figure 27. Towards the spatial extents the spatial bands are intersected, and thus cover a smaller area. Selecting a set amount of points for each filtering bands would then increase the density towards the edges of the data set spatial extents. This problem is schematically depicted in Figure 28, where due to the spatial extents only the orange area of the outer band is filled with points, compared to the grey area in the smaller band. This results in a smaller area being filled with the same amount of points towards the spatial extent of the data set, resulting in a density spike towards the edges of the data set.

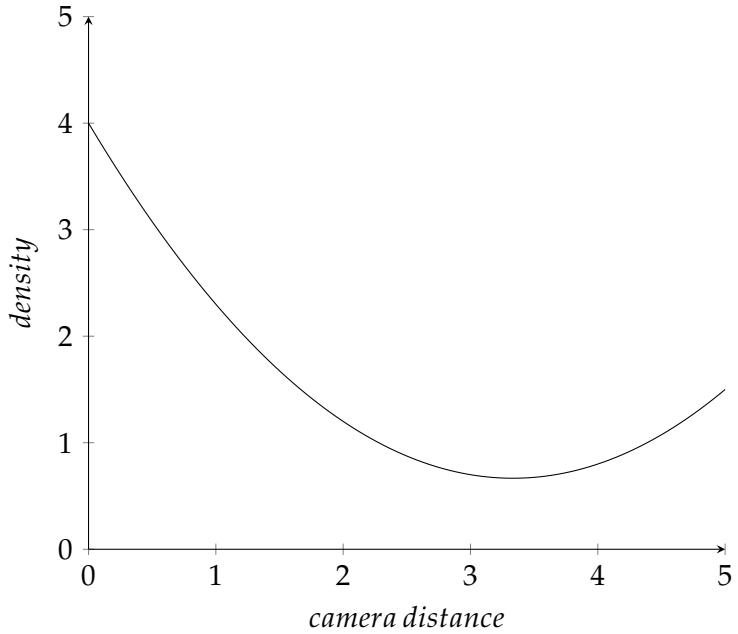


Figure 27: Density function 1

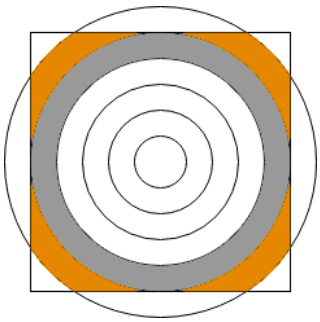


Figure 28: Filtering bands using predetermined amount of points per band

Point radius density

Point radius density works on the basis of plotting circles around points in the data set, where only 1 point is allowed in each circle. This way an *upper density bound* is determined; a limit of the maximum amount of points for any given area. By linking the radius of the circles to the *density function* a continuous *upper density bound* can be created. In Figure 29 an example is given of the circles being projected on the data set, in each of which only 1 points is allowed.

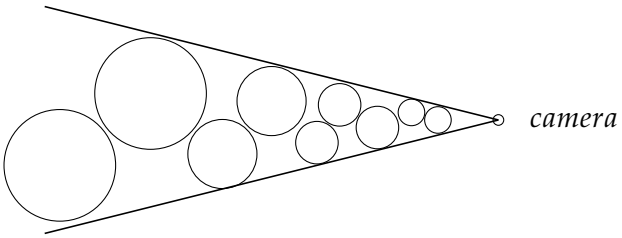


Figure 29: Point radius density

The creation of circles with a continuously increasing radius is based on the packing problem, a mathematical problem of packing a single container as densely as possible. Circle packing is a sub field of study. It studies fitting circles, sometimes of equal size, into a 2D or 3D euclidean space. The fitting must be done so no overlap occurs, and thus guarantees a maximum amount of circles for each given area or maximum amount of sphere for each given volume. As discusses, we will limit this research to 2D euclidean space. Circle packing is the choice for this problem, since the 2D neighbourhood of a 0D element is represented as a circle.

In Figure 30 an example is given of the most optimal 2D circle packing, a hexagonal distribution.

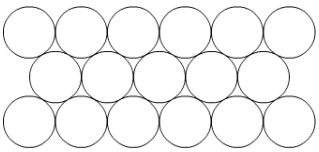


Figure 30: 2D Point radius density

For fitting circles of equal size in a 2D euclidean space the optimal ratio is shown in equation 7 using the hexagonal packing distribution (Chang & Wang, 2010). This indicates that for each square meter a maximum of 90% of the surface can be covered by circles.

$$\frac{\Pi\sqrt{3}}{6} \approx 0.9069 \tag{7}$$

This information is critical for determining the formula for the *upper density bound*. Knowing that at most 90% of the space will be occupied by circles creates a relationship between the neighbourhood (radius) in which there can be no other points. For a given radius, the area occupied by the circle is shown in equation 8. In equation 9 it is deducted that there can be a maximum of 1 point in each $3.5r^2$ unit of area. This allows us to determine the amount of points in any area, by specifying the radius in which there can be no other points.

$$A = \Pi r^2 \tag{8}$$

$$\frac{A}{0.9} = \frac{\Pi r^2}{0.9} \approx 3.5r^2 \tag{9}$$

Knowing the relationship between circle radius and the area it covers for a given point, we can determine how to apply this to density. If we want n points per square meter this means we

want 1 point per $\frac{1}{n}$ square meter. This means we can deduct the relationship between density and radius, which is shown in equation 10. In equation 12 we deduct how to determine the radius from the density.

$$\frac{1}{n} = 3.5r^2 \tag{10}$$

$$r^2 = \frac{1}{3.5n} \tag{11}$$

$$r = \sqrt[2]{\frac{1}{3.5n}} \tag{12}$$

If we wish to enforce a density of 5 points per square meter, this means the radius to determine the area in which there can be no other point is $r = \sqrt[2]{\frac{\frac{1}{5}}{3.5}} \approx 0.24 meter$. Since the *density function* determines the amount of allowed points for a given area, relative to the distance from the camera, these functions can be combined to produce a *radius function* relative to the camera distance. This gives the *radius function* in equation 13:

$$radius = \sqrt[2]{\frac{1}{3.5 * density\ function}} \tag{13}$$

This function specifies the radius in which no point can be closer than radius r to another point, to reinforce the *upper density bound*. Quick execution of this function requires a fast lookup of point neighbours. A KD-tree is used for this purpose. Using a Nearest Neighbour algorithm, the neighbouring points inside this radius r can be discarded from the data set. What is left is a data set with a density that is never above the plotted upper limit of the density.

Python code for the Point radius density algorithm is found in Algorithm 3. For each point three variables have to be computed; distance from the camera d, allowed radius r and the nearest neighbours nn. If a point has already been used as a neighbouring point inside the circle radius of a point, no computations have to be done for that point. In Figure 31 these points are shown in red, while the points for which all three computations have to be done are shown in green. Depending on the amount of points that are removed this can significantly speed up the computation.

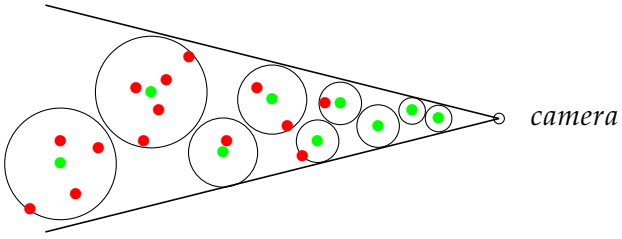


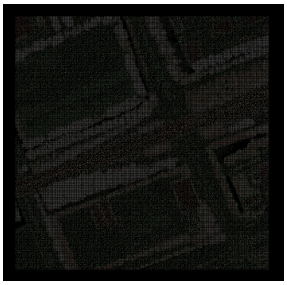
Figure 31: Point radius density

Algorithm 3 Point radius density

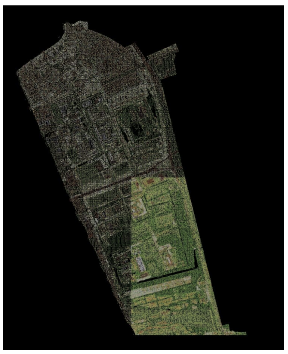
```
import scipy

def circle_packing(points, camera_parameters, density_function):
    """

    """
    used_points = [False] * len(points)
    selected_points = set()

    kdtree = scipy.spatial.KDTree(points)

    for j, point in enumerate(points):
        if used[j] == True:
            continue

        d = distance_from_camera(point, camera_parameters)
        r = radius_function(density_function(d))
        nn = kdtree.query_ball_point(point, r)

        for i in nn:
            used[i] = True

        selected_points.add(point)

    return selected_points
```

3.4. Method evaluation

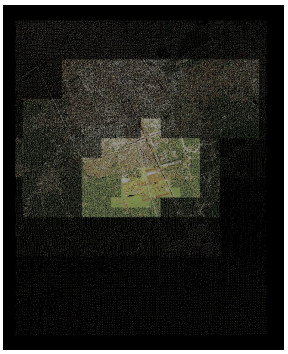
This Section will evaluate the proposed methods in terms of speed per frame. All three implementations have been tested for three frames, shown in Figure 32. The three methods will be evaluated in terms of computational speed. Table 2 shows the results of computational speed for the three proposed methods.



(a) Frame A with 73 241 points



(b) Frame B with 453 438 points



(c) Frame C with 1 992 453 points

Figure 32: Three evaluated frames

	Frame A (s)	Frame B (s)	Frame C (s)
Random removal	1.3	5.4	24
Filtering bands	0.6	3.6	15.1
Point radius density	0.01	0.08	0.36

Table 2: Computational speed of proposed methods

The Point radius density is a clear improvement over the other 2 methods, showing improvements of 800% over the Random removal method and 500% over the Filtering bands method. This improvement is due to the fact that only the points that are kept in the vario-scale result are processed, while the discarded points are not.

4. Framework and Data sets

The creation of a web-based point cloud visualization framework requires solutions for each stage of the framework. The framework has four major stages, for each of which this Chapter will discuss the best options. These four stages are *storage*, *indexing*, *processing* and *visualization*. How these four stages interact with each other in the framework is depicted in Figure 33. In Section 4.1 a consideration is made between the most used or prevailing industry standards for each stage. Section 4.2 will describe which solution is chosen, and will present the framework used for the proof of concept.

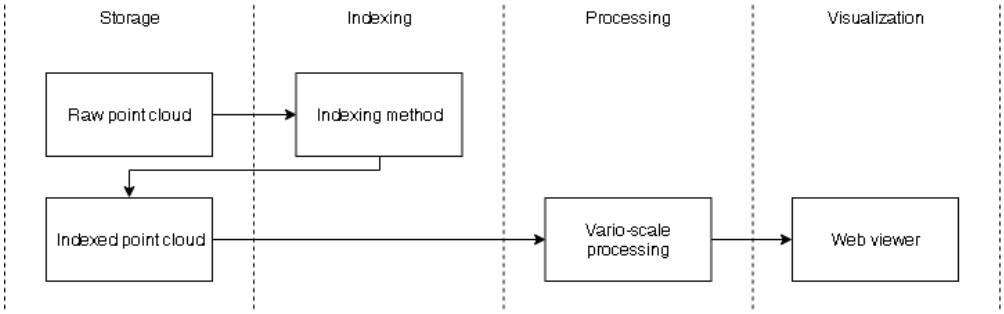


Figure 33: Basic framework for proof of concept

4.1. Framework considerations

This Section will evaluate the available option for each of the four major stages of the framework; *storage*, *indexing*, *processing* and *visualization*. In this Section the available options will be elaborated upon. No choice of framework option will be made, since all stages of the framework interact with each other. It is thus important to evaluate the options not as isolated choices, but in the context of the bigger framework.

Storage

For storage there are two main options to choose from, which are a file-based storage method and a database storage method. For each of these methods there are multiple implementations, for example a file-based storage method could be comprised of ASCII-files, LAS-files or LAZ-files, and for database storage there is a wide variety of systems to choose from. For this consideration the LAS/LAZ-file system will be evaluated against the PostgreSQL database. The LAS/LAZ-file system is currently the most optimal file-bases system for point clouds, and PostgreSQL has the pgpointcloud extension for storing point clouds.

There are two considerations for the storage method, which are data set size and i/o operation speed. Good compression will reduce the data set size, but might hinder i/o operations due to the need for decompression. For this proof of concept the emphasis will lie on reduced data set size, since the AHN2 data set in LAS form is approximately 12 TB. In terms of i/o operations a PostgreSQL database solution is significantly slower compared to a LAS-file system, depending on the performed operation (van Oosterom et al., 2015). LAS files perform the best for i/o operations because, compared to LAZ files, these do not need to be decompressed. The performance difference between LAS and LAZ files is most notable in smaller queries, where the decompression overhead is relatively large. PostgreSQL database storage offers a reduced compression rate compared to LAZ files. LAZ files offer a $x10$ compression over LAS files, whereas PostgreSQL offers a $x4$ compression rate over LAS files.

Indexing

For point cloud indexing there is a wide variety of indexing methods. For web-based visualization

there is one indexing method that is the most widely used; an Octree. The use of an octree is based on the fast querying capabilities that it provides. The data is stored in blocks that are queried depending on their position in the view frustum. This offers advantages in terms of caching, and it limits data transfer per user input; when the camera position changes, additional blocks will be queried from the back-end as needed. The previously loaded blocks that are still visible in the view frustum do not have to be queried, saving requests to the web-server.

For this section two of the most widely used octree indexing methods for file-based point cloud data sets will be discussed, PotreeConverter and Entwine. Both of these indexing methods offer Octrees that are compatible with the web viewers discussed later.

The PotreeConverter builds an Octree with a limit of 20 000 points per block. Each point is selected and evaluated prior to being inserted. It is determined if there are already points within n units of the point, and if so the point would be sent down to the next level. This threshold n is 250 units for the first level of the octree, and is halved for subsequent each level in the Octree. This check is done for each block, until an empty space in a block is found or a new block is created. It is important to note that the point insertion order is arbitrary, and dependant on the first point in the first object that is read.

Entwine builds an Octree with a soft limit of 32 768, or 2^{15}, which is the maximum value for a signed 16 bit integer. This value is calculated using the $span$ and $overflowThreshold$ parameters. These values can be changed in the config file and by default $span = 256$ and $overflowThreshold = 0.5$. For each block a voxel grid is created of $span^3$, or $256^3 = 16\ 777\ 216$ voxels by defult. To get an even distribution of points throughout a block in the octree, during insertion each point is evaluated against this voxel grid. If the voxel is empty, the point is inserted. If the voxel is already filled, both points are evaluated against each other. The point closest to the voxel's center is kept. Points that are not insert into the block, are put in one of 8 overflow buckets depending on its location. Each of these overflow buckets will become a child block once the amount of points in the voxelgrid as well as the points in the overflowbuckets is more than 32 768.

Processing

For the processing of point cloud data there are two options; the use of open source libraries or the creation of custom tools in a programming language. The main advantage of open source tools is that a large part of the framework is created, and the tools have been optimized. These tools, such as PDAL or LAStools, provide multiple extensions for reading, writing and a multitude of different operations on LAS files. Creating a custom solution in a programming language such as Python or C++ offers the ability to exactly control the implementation of the vario-scale algorithms, and quickly adapt to new findings.

The custom nature of the vario-scale visualization implementation requires high degrees of control over the processing environment. Thus a programming language to produce a custom solution is required. There are multiple options to choose from, each with its own advantages and disadvantages. For comparison, Python and C++ will be briefly discussed because they both offer extensions for LAS/LAZ files. The main difference being that Python is an interpreted language and C++ is a compiled language. This creates an obvious advantage for C++, which has more compact code and faster runtime speeds. On the other side advantages for Python include adoption rate, writability and readability.

Visualization

For point cloud web-visualization there are two main frameworks that are widely used; Potree and Plas.io. In terms of compatibility these do not differ greatly. Both of these frameworks use an Octree indexed file-based point cloud data set to query from, such as the index created by PotreeConverter or Entwine.

In Potree point selection is done in two actions. Firstly a 2D profile of the required spatial

extents of the visualized region is created. The required Octree nodes are selected based on their spatial extents intersecting the 2D profile. A second constraint ensures no nodes deeper than the maximum allowed depth of the Octree are selected. Requests are made to the web server to retrieve the selected nodes and these nodes are stored in a priorityqueue. Secondly a view frustum is created, and for each node in the priorityqueue a determination is made whether or not the node is inside the view frustum, does not exceed the point limit and does not exceed the allowed Octree depth.

For Plas.io no information is available on the point selection. A request for an explanation is currently pending. In Plas.io the main advantage is the fact that camera parameters are passed as parameters during request handling. These five parameters allow the determination of the camera orientation within that data set CRM. The five camera parameters that are passed are:

- azimuth
- distance
- max-distance
- target
- elevation

4.2. Proof of concept framework

So far multiple options for each part of the framework have been discussed. This section will choose an implementation method for each of the four parts: Storage, Indexing, Processing and Visualization. Each decision made will be made with integration of systems in mind to create an adequate, easy to work with, web-based work flow.

Storage

For Storage the decision has been made to pick the most cost effective, thus highest compression, method available. Since the complete AHN2 data set in LAS format would be approximately 12TB it can be stated that compression and cost reduction are of high importance. As discussed in Section 4.1 the LAZ file system offers the best compression rates. The resulting AHN2 data set to be stored would be approximately 1TB. Table 3 shows the cost for storing the AHN2 data set on different cloud platforms.

	Warm Storage	Cold Storage	Long term storage
AWS	$20.70	$9.00	$3.60
Google	$18.00	$9.00	$6.3
Azure	$16.74	$7.65	$1.53

Table 3: Pricing of S3 storage, per year as of December 2018

Cloud storage is chosen because of fast communication with other framework stages hosted in the cloud environment, typically $10Gbps$, and a 99.99% up-time of data service. Microsoft's cloud service, Azure, is the cheapest in all three categories when comparing cloud services. Although not the cheapest option, storage on AWS is chosen because of the interoperability with other framework stages.

Indexing
For indexing Entwine is chosen. The choice for Entwine is made because of its integration with the existing workflow; it seamlessly reads data from the Amazon S3 storage and its indexed structure can be read by most web-viewers.

Comparatively Entwine also provides faster indexing than PotreeConverter. This can be explained by the selection method. The PotreeConverter evaluates whether there are points within threshold n units in the block, and keeps or discards the points based on that. This requires multiple vector calculations to determine the distances between points. Entwine on the other hand places each point in a voxel grid. If the block is already occupied, a single calculation suffices to determine whether or not the new block is closer to the voxel's center.

Processing
For processing the most important factors include computational speed and custom algorithm implementation. Because the decision is made to use LAZ files in both Storage and Indexing, it is critical that the processing framework has a workable extension for LAZ files.

Python is chosen to build the processing framework over other programming languages such as C++. Python offers the benefit of producing readable algorithms and an understandable implementation. In combination with Python multiple extensions will be used to make up for the normally slower speed of an interpreted language; the Laspy extension will be used for LAS/LAZ file operations and the Numpy extension will be used for fast N-dimensional array operations (uses C for speed).

Visualization
For visualization Plas.io is chosen. The most important factor is the accessibility of camera parameters, which are crucial for vario-scale visualization. By using the camera parameters this allows us to calculate the camera origin, which is crucial for the implementation of the vario-scale visualization method.

Resulting framework
The resulting framework stores the AHN2 point cloud data set as LAZ files in an AWS S3 bucket. This bucket is queried by Entwine, running on AWS, for indexing. The indexed data set is then stored in another AWS S3 bucket. This bucked can be queried by Greyhound, also running on AWS, which queries the original data set (with density jumps). This data set run through the Python implementation, and the vario-scale implementation is performed to serve the client a vario-scale point cloud data set. This is illustrated in Figure 34.

The indexing operation, shown in red, is performed once for the entire data set (pre-processing). The visualization operation, shown in green, is performed every time the client queries the data set (post-processing).

4.3. Data sets

The data to be processed and visualized is the AHN2 data set. The AHN2 data set consists of 639 478 217 460 points in total. The complete data set is available for download via the PDOK (Publieke Dienstverlening Op de Kaart).

Multiple subsets of the complete AHN2 data set will be created for testing and scaling purposes. All data sets will be stored in the LAZ file format on AWS S3. These files are accessible through every server that has been given the proper credentials. Figure 35 illustrates the data sets. Table 4 shows some specifications of the data sets.

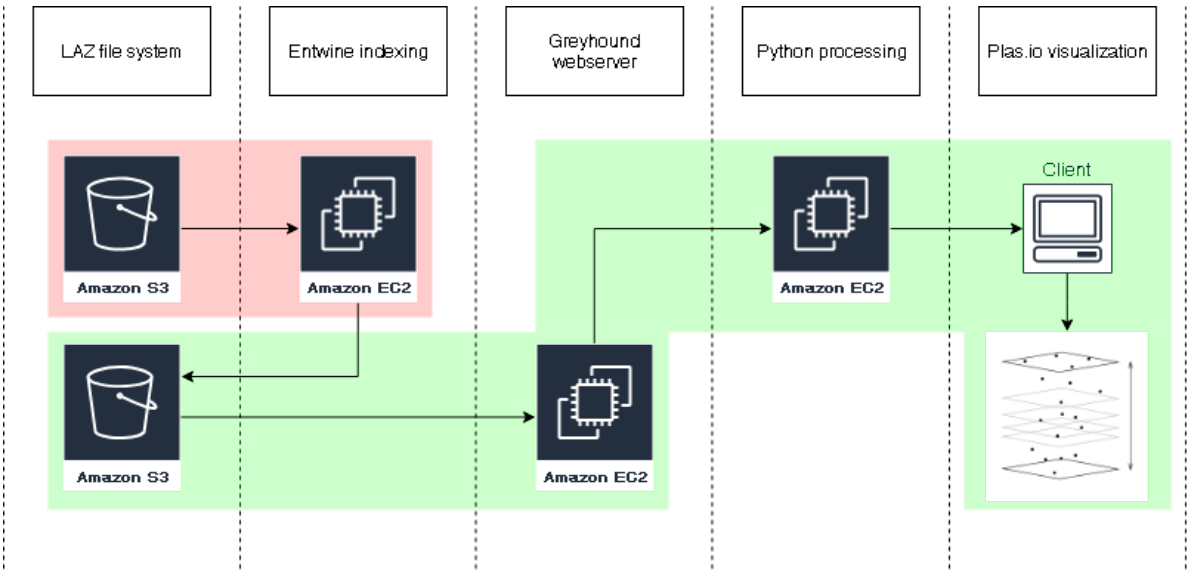


Figure 34: The proposed framework

Name	Format	Total Size	Points
Delft Wippolder	LAZ	10,7MB	5.00×10^6
TU Delft Campus	LAZ	162MB	1.00×10^8
Municipality Delft	LAZ	3,2GB	2.20×10^9
Province Zuid Holland	LAZ	84,7GB	6.03×10^{10}
AHN2 Full	LAZ	987GB	6.39×10^{11}

Table 4: Data sets

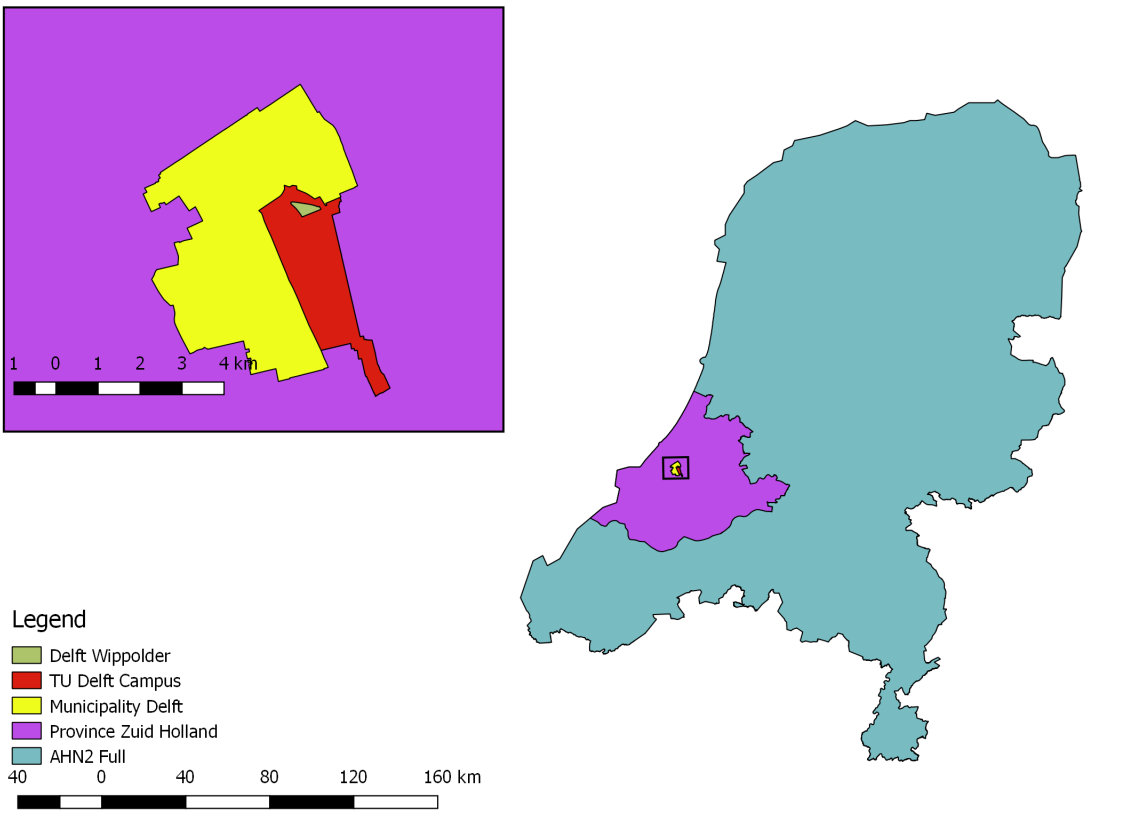


Figure 35: The data set regions

5. Implementation

5.1. Considerations

For the proof of concept for vario-scale pointcloud visualization, the Point radius density method as described in Chapter 3 is chosen. As shown in Section 3.4 the computational speedup is a massive improvement over the other two methods. The hypothesis is that, while removing a large part of the *density jumps*, some dense features will still be distinguishable visually. Because only the upper bound of the density is set, all features less dense than this upper bound will be fully distinguishable. This might include some of the existing *density jumps*. This hypothesis will be both visually and numerically tested in Chapter 6.

5.2. Implementation

In Chapter 4, a framework has been chosen to implement the proof of concept. The framework is implemented on AWS instances, in this Section are more in depth description will be given of how each part of the framework is implemented.

Storage - LAZ file system
Storage of both the raw data set and the indexed data set is done on AWS S3 storage. The raw data buckets are made publicly available through:

- http://tu-delft-campus.s3.amazonaws.com/
- http://municipality-delft.s3.amazonaws.com/
- http://province-zuid-holland.s3.amazonaws.com/

The indexed data sets are not public, these can privately be accessed using the AWS Cli and the authors private credentials.

Indexing - Entwine
Indexing is done using Entwine, publicly available on *https://github.com/connormanning/entwine*. Entwine is run using Docker, and requires a config file with parameters. The configuration that is used is for for example the Province Zuid Holland data set is added in Apendix **??**.

Web server - Greyhound
The web server is run using Greyhound, publicly available on *https://github.com/hobu/greyhound*. Greyhound is run using a custom the AWS image (AMI) which is not made publicly available because it contains the authors private AWS credentials. Via these permissions, and using the following configuration file in Apendix D, Greyhound will query the indexed data sets. It is important to allow inbound and outbound traffic on port 8080 for the instance, otherwise Greyhound will be unable to either receive or send requests.

Processing - Python
The Python implementation is run on the same AMI, and uses a Flask API to behave similar to the Greyhound web server. This implementation is also publicly available on *https://github.com/JippevdMaaden/thesis*. Note that the Greyhound web server url will have to be edited manually in the *utils.py* file. It is important to allow inbound and outbound traffic on port 5000 and 8080, which will allow requests to Greyhound and Flask.

Visualization - Plas.io

The Plas.io webserver has not been modified, and is run by visiting *http://speck.ly/*. Two parameters are passed as variables in the url, the Flask web server url as *s* and the resource name as *r*. A Speck.ly URL looks like this:

http://speck.ly/?s=ec2-18-195-51-3.eu-central-1.compute.amazonaws.com:8080/r=tu-delft-campus

5.3. Resulting approach

Make sure that on all the instances that are used the appropriate credentials are supplied to the AWS Command Line Interface (AWS CLI). The configuration file for Greyhound should be altered, a custom path should be added to the resource that will be made available for the implementation. The security settings on all instances should allow traffic on the ports mentioned above. Once the instances are set up properly Greyhound and Flask should be started by running:

docker run -d -it -p 8080:8080 -v ∼/greyhound:/root/greyhound -v ∼/.aws:/root/.aws connormanning/-greyhound -c /root/greyhound/config.json

and

python ∼/thesis/flask/speckly_app.py

This allows requests to be made to the instance running the Flask app on port 5000, the default port for Flask. These requests are then directed to Greyhound which returns a bitstream of a LAS file. This bitstream is then turned in to the LAS file in the Flask app, where it is processed and turned into a bitstream again. This bitstream is then sent to speck.ly where the points are visualized in the web browser.

As of writing a bug is encountered when turning the LAS file into a bitstream in the Flask app, which results in no points being visualized in the web browser.

6. Results

In this Chapter two results will be presented. In Section 6.1 a small benchmark is presented on the parallel indexing of massive point cloud data sets using cloud computing. In Section 6.2 a single frame will be presented that has been created using an Entwine indexed point cloud queried by the Plas.io web viewer.

6.1. Parallel point cloud indexing

Because of the decision to choose a distributed cloud computing solution, a small benchmark has been made in which this decision is put into perspective by showing the speed-up compared to performing the indexing computations on a single machine.

Two benchmarks are done, the first on a single machine to test which configuration is best for non-parallel indexing. The second on a cluster to determine the speedup parallel indexing provides compared to non-parallel indexing. For this benchmark four data set are used, presented in Section 4.3. The AHN2 data set is not used, due to the cost of the combined computational power required.

The preliminary, non-parallel, benchmark is done on four different AWS instances. These AWS instances are instances optimized for compute-intensive workloads. These deliver a cost-effective price per compute ratio. The properties of these four instances are shown in Table 5.

Model	CPU	Mem (GiB)	Network Performance (Gbps)
c5.2xlarge	8	16	up to 10
c5.4xlarge	16	32	up to 10
c5.9xlarge	36	72	10
c5.18xlarge	72	144	25

Table 5: AWS instances used for non-parallel benchmark

The preliminary benchmark results are illustrated in Table 6. These show that the *c5.4xlarge* instance on AWS is the most suitable for a scalable distributed approach. For a data set the size of the TU Delft Campus it is able to process the most points per hour, with 1.68×10^9 points per hour. This would mean that indexing the entire AHN2 data set would take 380 hours. In the preliminary benchmark we see the tendency for data set with more points to be processing slower, already hinting to the possible improvements a distributed cloud computing approach could offer. Notably there seems to be a drop in performance for the *c5.9xlarge* and the *c5.18xlarge* instances. The nature of this drop in performance is not researched. There could be multiple possible explanations, such as the locality of hardware that make up these instances or i/o collisions due to the large amount of CPU's. No research is done into this since it is outside the scope of this thesis.

c5.2xlarge	Time (s)	Points per hour
Delft Wippolder	20	1.02×10^9
TU Delft Campus	410	8.82×10^8
Municipality Delft	11 200	7.09×10^8
Province Zuid Holland	389 452	5.57×10^8
c5.4xlarge	Time (s)	Points per hour
Delft Wippolder	20	1.02×10^9
TU Delft Campus	215	1.68×10^9
Municipality Delft	6 427	1.24×10^9
Province Zuid Holland	224 762	9.65×10^8
c5.9xlarge	Time (s)	Points per hour
Delft Wippolder	20	1.02×10^9
TU Delft Campus	249	1.45×10^9
Municipality Delft	6842	1.16×10^9
Province Zuid Holland	235 276	9.22×10^8
c5.18xlarge	Time (s)	Points per hour
Delft Wippolder	21	9.73×10^8
TU Delft Campus	303	1.19×10^9
Municipality Delft	7 800	1.02×10^9
Province Zuid Holland	250 202	8.67×10^8

Table 6: Preliminary benchmark

The second benchmark, using distributed cloud computing for parallel processing, has been performed for two data sets. The first two data sets are not included in this benchmark because of their size. These data set are too small to offer a significant speedup compared to indexing on a single instance. The largest data set, the AHN2 data set, has not been used due to economic considerations; indexing the entire data set would take 380 instance hours and in the process would cost $260. In Table 7 the results are presented.

c5.4xlarge	Instances used	Points per hour	Merging overhead (s)
Municipality Delft	20	3.26×10^{10}	210
Province Zuid Holland	100	1.49×10^{11}	1 186

Table 7: Cloud computing benchmark

In total the Municipality Delft data set is indexed in 249 seconds of processing, and 210 seconds of merging. In total the indexing takes $249 + 210 = 459$ seconds. Compared to the 6 427 seconds the indexing takes on a single instance, this offers a 1 400% improvement. The Province Zuid Holland data set is indexed in $1456 + 1186 = 2642$ seconds, which is an improvement of 8 500% over the indexing on a single instance.

6.2. Vario-scale frame

Four different *density formulas* have been used to create four different vario-scale frames. An analysis will be made into the main goal of this thesis, removing the *density jumps* from the frame. The method used to enforce the *density formula* is the *Point radius density* method proposed in Chapter 3. The framework used to create these results is determined in Chapter 4 and its implementation is explained in Chapter 5.

The four density formulas used to visualize the vario scale frame have been introduced in Section 3.2. These are:

- Density formula 1: $0,6789(0,0035 * camera\_distance)^3$
- Density formula 2: $0,6789(0,005 * camera\_distance)^3$
- Density formula 3: $0,6789(0,007 * camera\_distance)^3$
- Density formula 4: $0,6789(0,01 * camera\_distance)^3$

And each preserve a percentage of points in the data set, from 75% to 12.5% respectively. These *density formulas* are visualized in Figure 36.

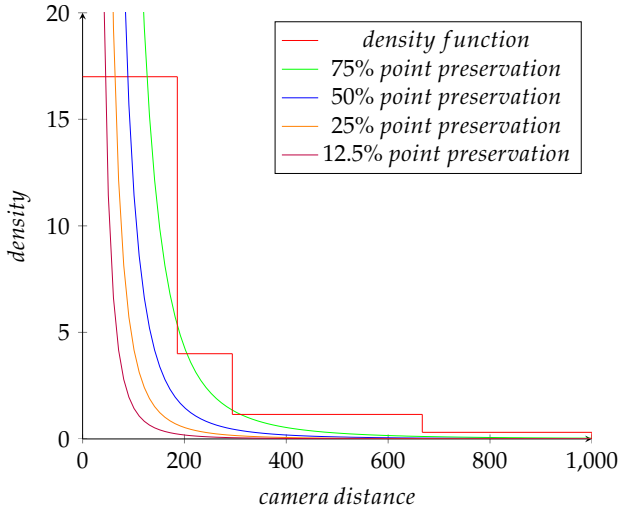
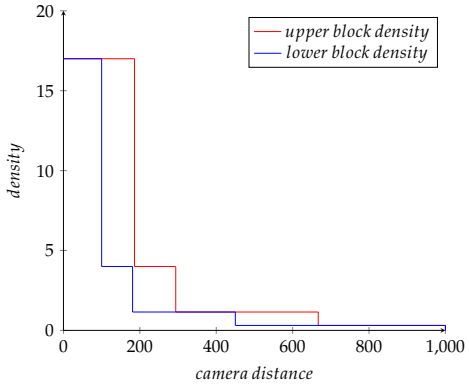


Figure 36: Four density formulas visualized

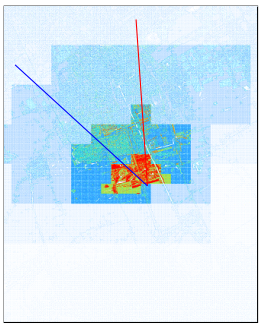
The results will be presented in a similar fashion as the examples given in Section 3.2. Firstly the graph with the density jumps is presented, similar to the graph in Figure 37. In addition to this graph, the *density function* that is used will be plotted, showing which regions of the point cloud have been removed, and which regions are still visible. In Figure 37(b) both the *upper block density* is depicted in red, and the *lower block density* is depicted in blue. It is important to realize that in the 3D spatial extents of the data set there are infinite density formulas for every line from the camera outward. The *upper block density* and *lower block density* merely show the highest *density function* and lowest *density function* respectively that occur in this data set.

Four images in total will be shown for each result; a top down view of the pointcloud density will be show and an image of the perspective view from the camera position will be presented. This will be accompanied by two images that show the top down view of the point cloud density and a perspective view, for the removed points. These images will be guided by a graph showing the density formula that is used to create the image. For reference, on the next page Figure 38 and Figure 39 are presented, the original top down and original perspective view of the

frame respectively. These two Figures are also added as Appendix E and Appendix F respectively, which should be used to visually compare the results presented here to the original top down and perspective view.



(a) Density jumps currently in the frame



(b) Density function plot top down

Figure 37: Density jumps (a) and their representation in the data set (b)

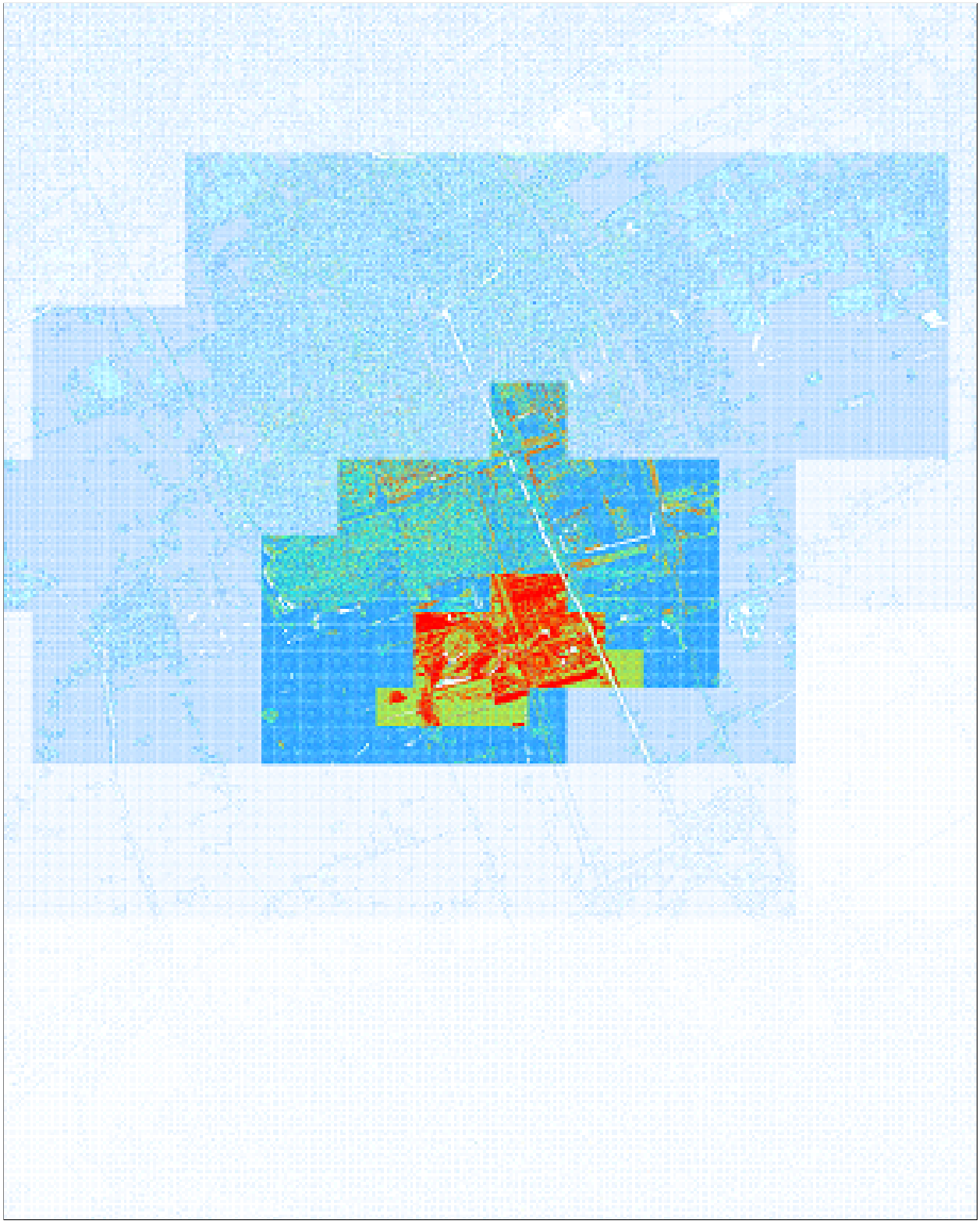


Figure 38: Original density of frame

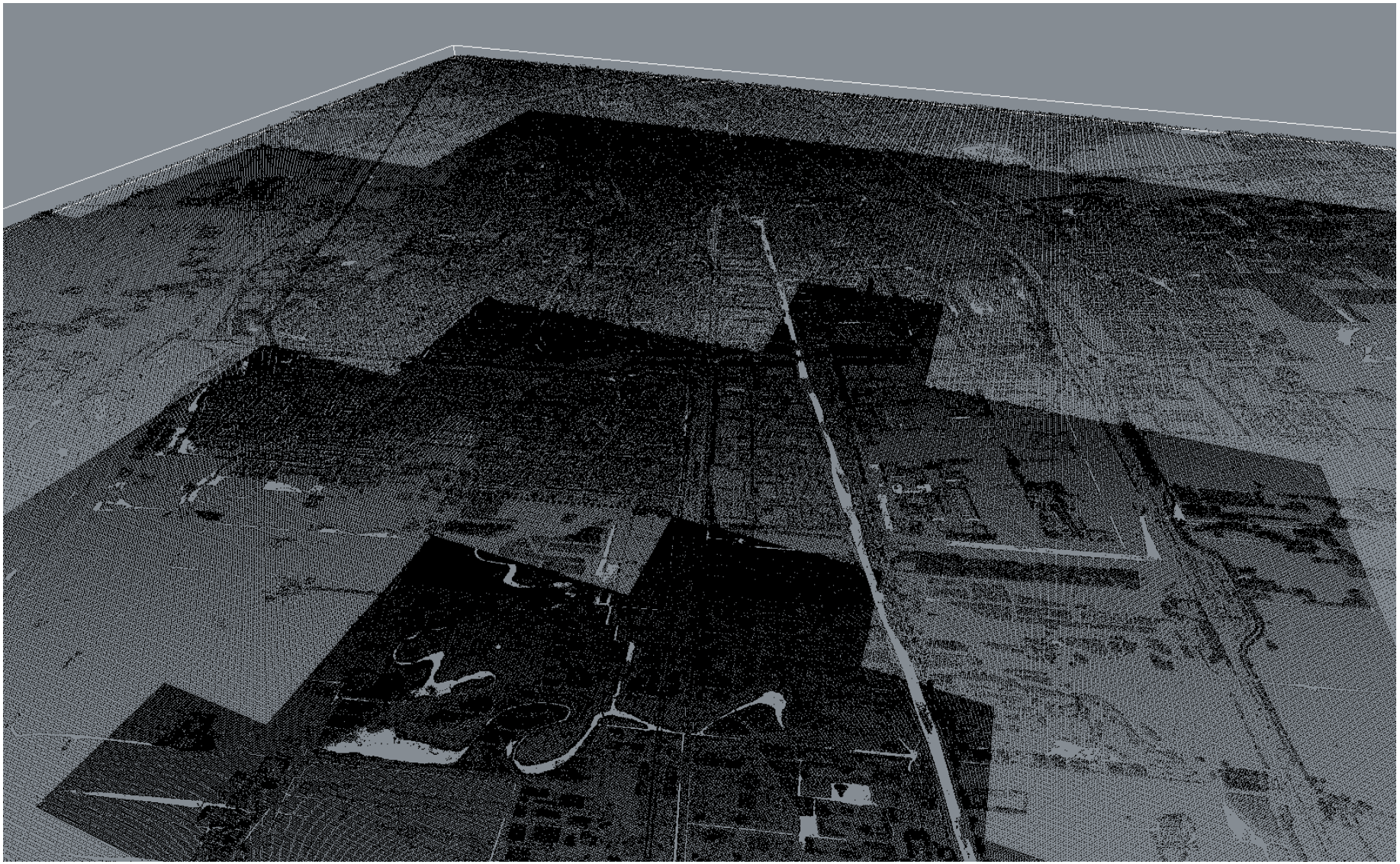


Figure 39: Original perspective view of the frame

6.3. Density function 1

This *density function* used in this section, is presented in Figure 40. This function keeps 75% of the original data set, the area marked in green. Some of the *density jumps* are removed, although a large part of the *density jumps* still remain in the data set. Figure 41 shows these dense regions still exist. In Figure 42 it is visible that from the perspective view there are still large *density jumps* marking the transition between levels in the octree.

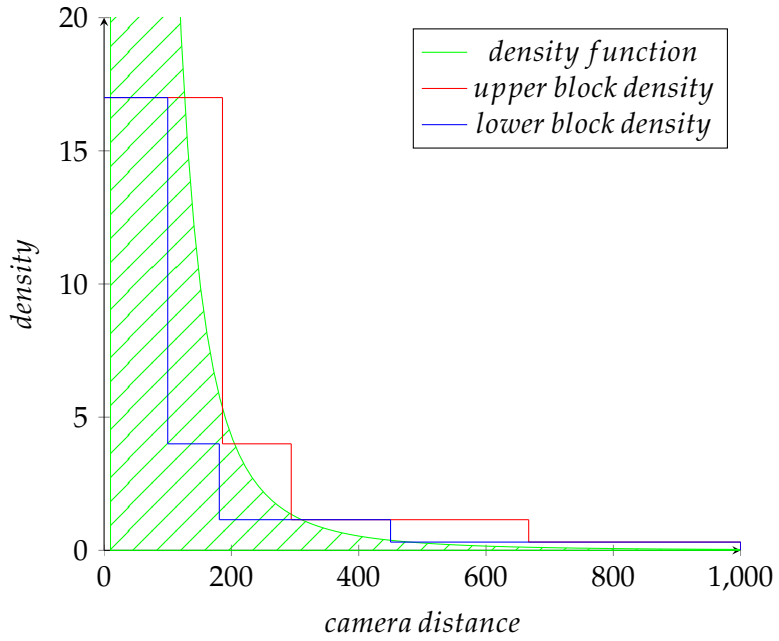


Figure 40: Density function 1

The Figures showing both the removed points density and perspective views, Figure 43 and Figure 44 respectively, show that the points that are being removed are mainly around the *density jumps* that were previously visible. Not enough points seem to have been removed to completely eliminate the *density jumps* however.

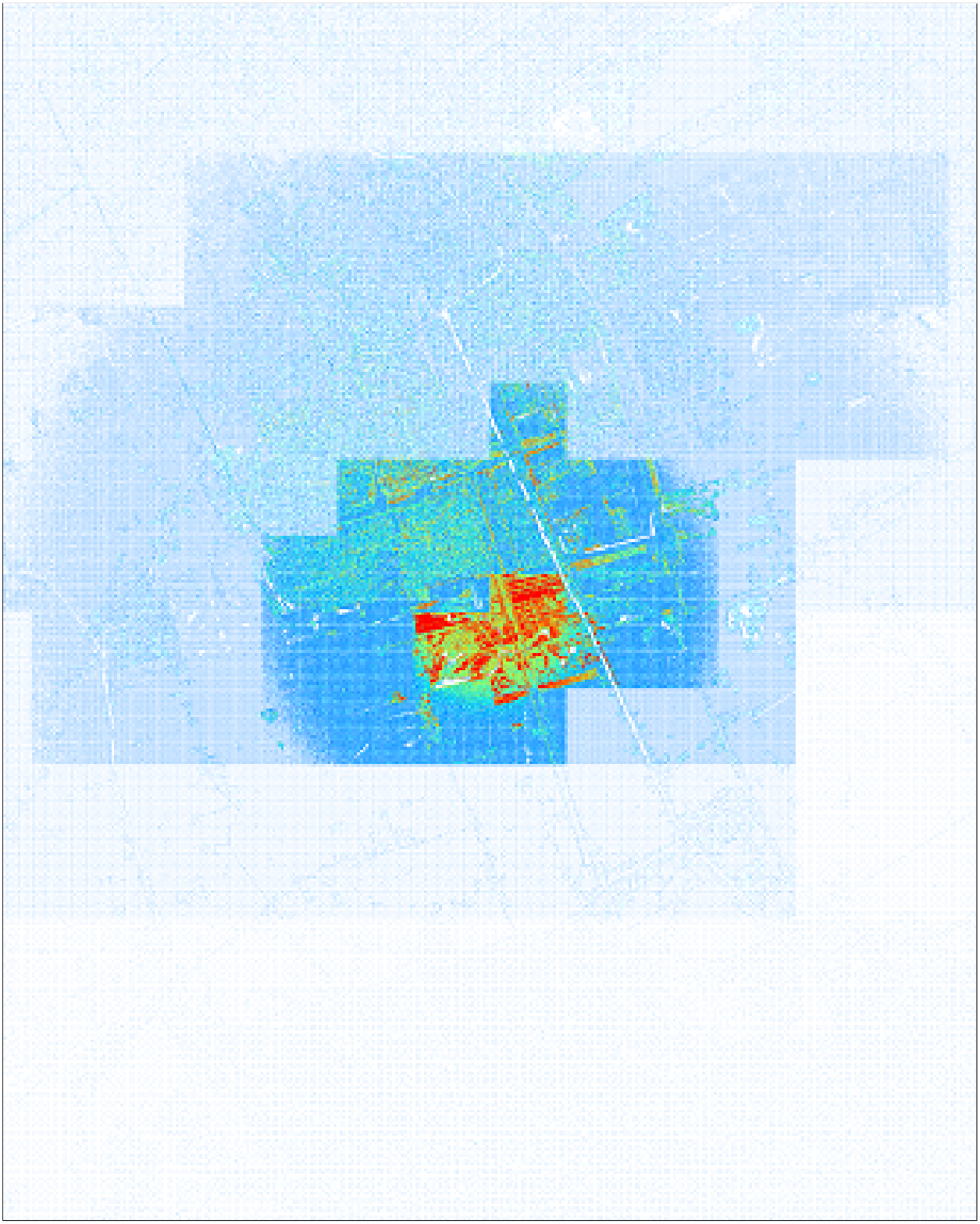


Figure 41: Density function 1 density of frame

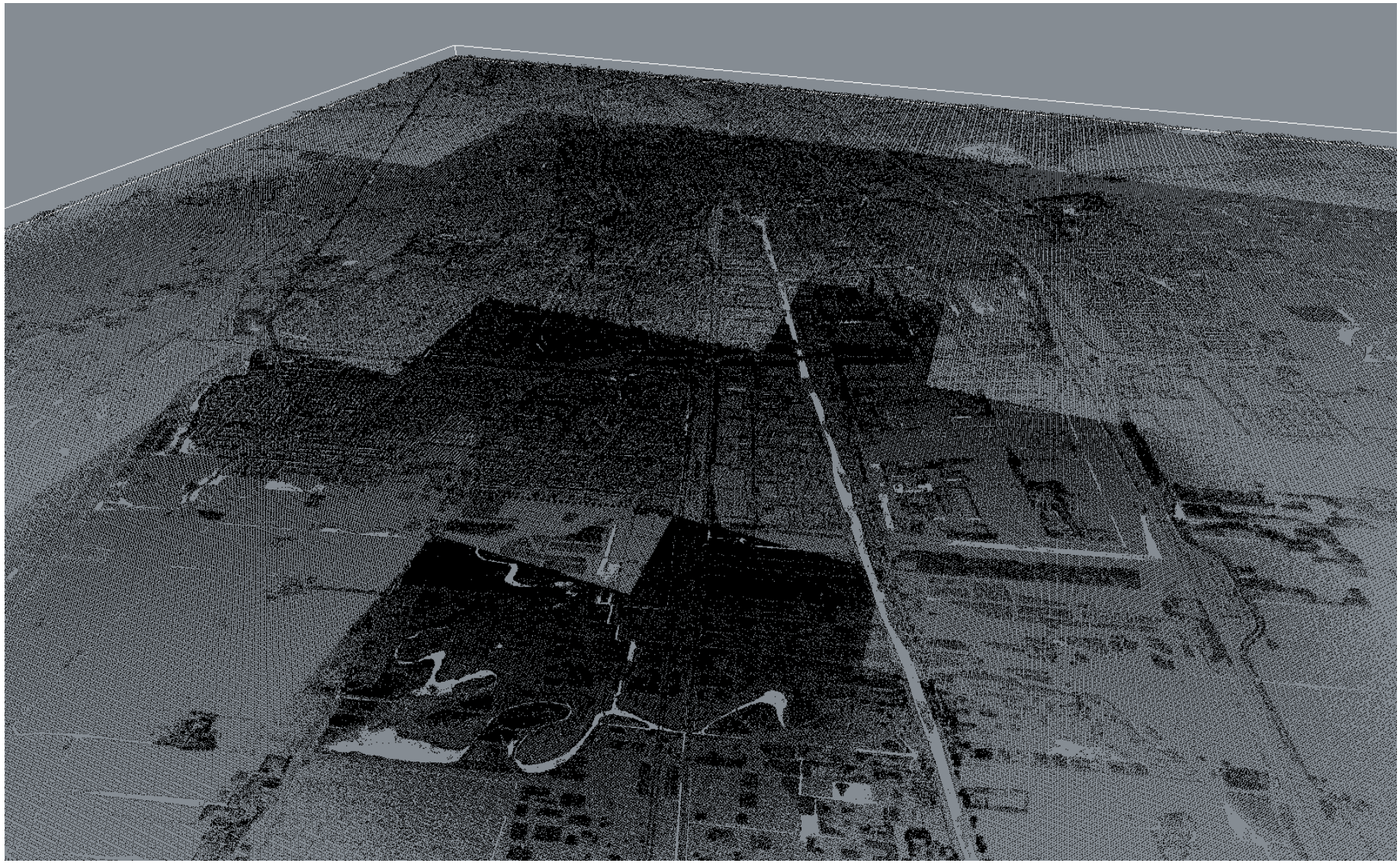


Figure 42: Density function 1 perspective view of the frame

Figure 43: Density function 1 density of the removed points

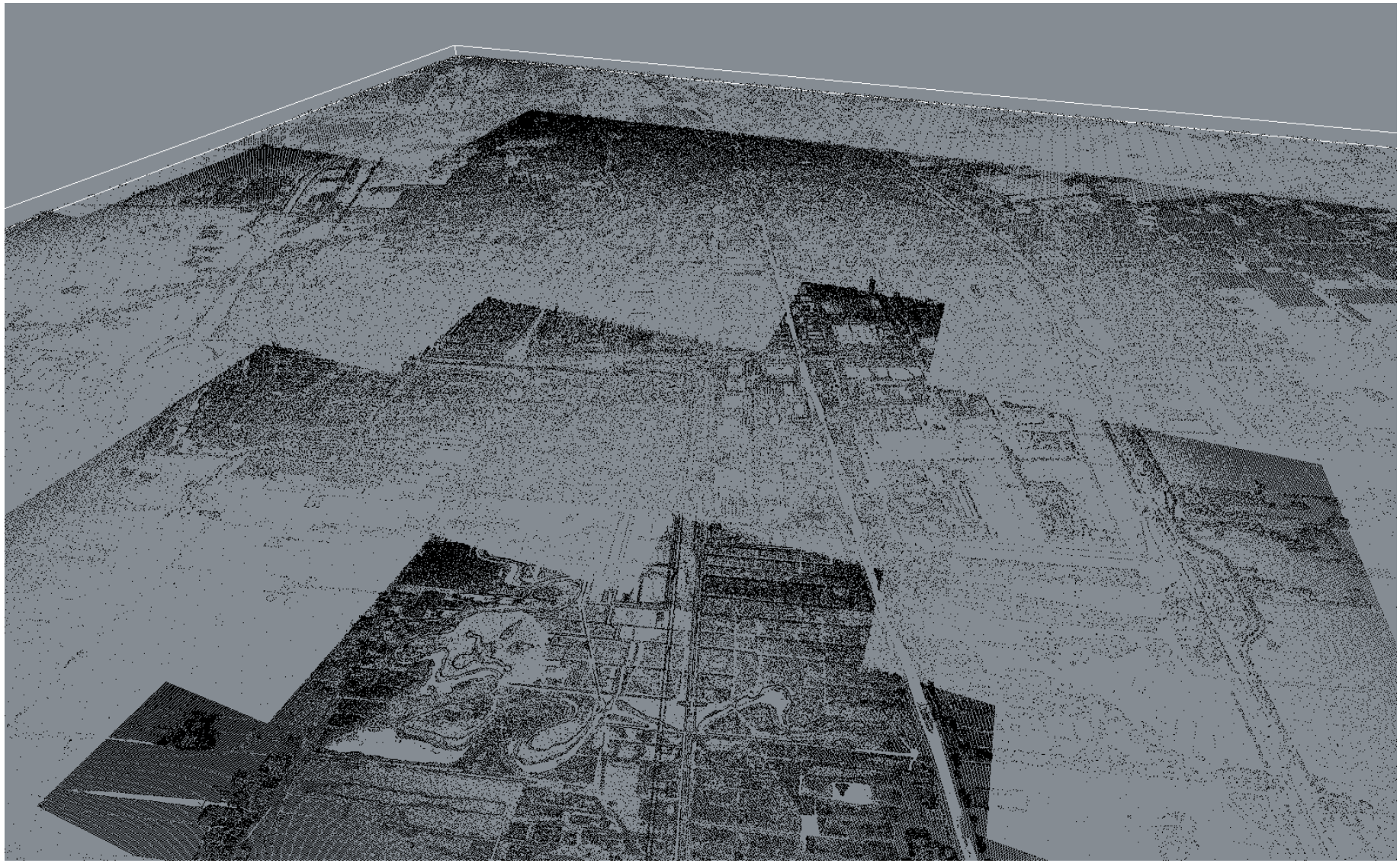


Figure 44: Density function 1 perspective view of the removed points

6.4. Density function 2

This *density function* used in this section, is presented in Figure 45. This function keeps 50% of the original data set, the area marked in green. Most of the density jumps are removed, with the exception of a region in the middle of the frame where a *density jump* is still visible in the data set. Figure 46 shows some of these dense regions. In Figure 47 it is visible that from the perspective view there is a region in the middle of the frame where a *density jumps* marks the transition between levels in the octree.

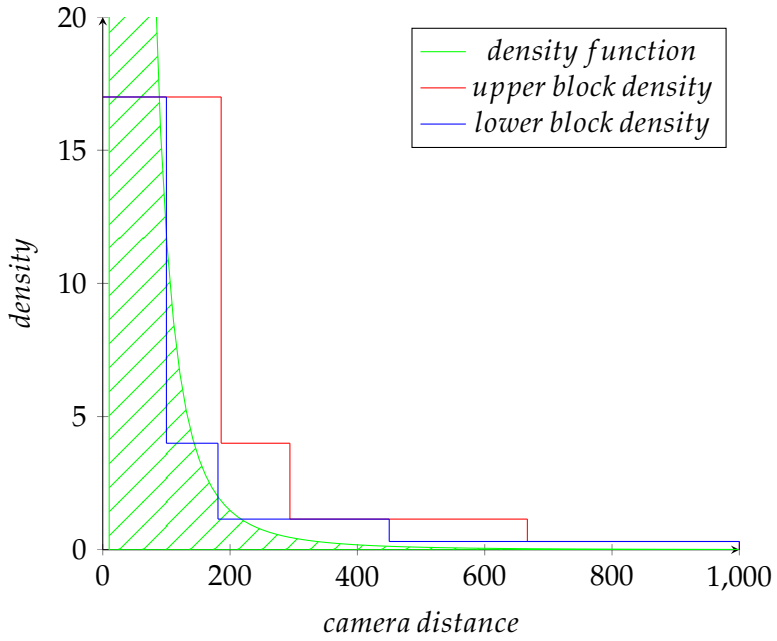


Figure 45: Density function 2

The Figures showing both the removed points density and perspective views, Figure 48 and Figure 49 respectively, show that a large amount of points around the *density jumps* are being removed.

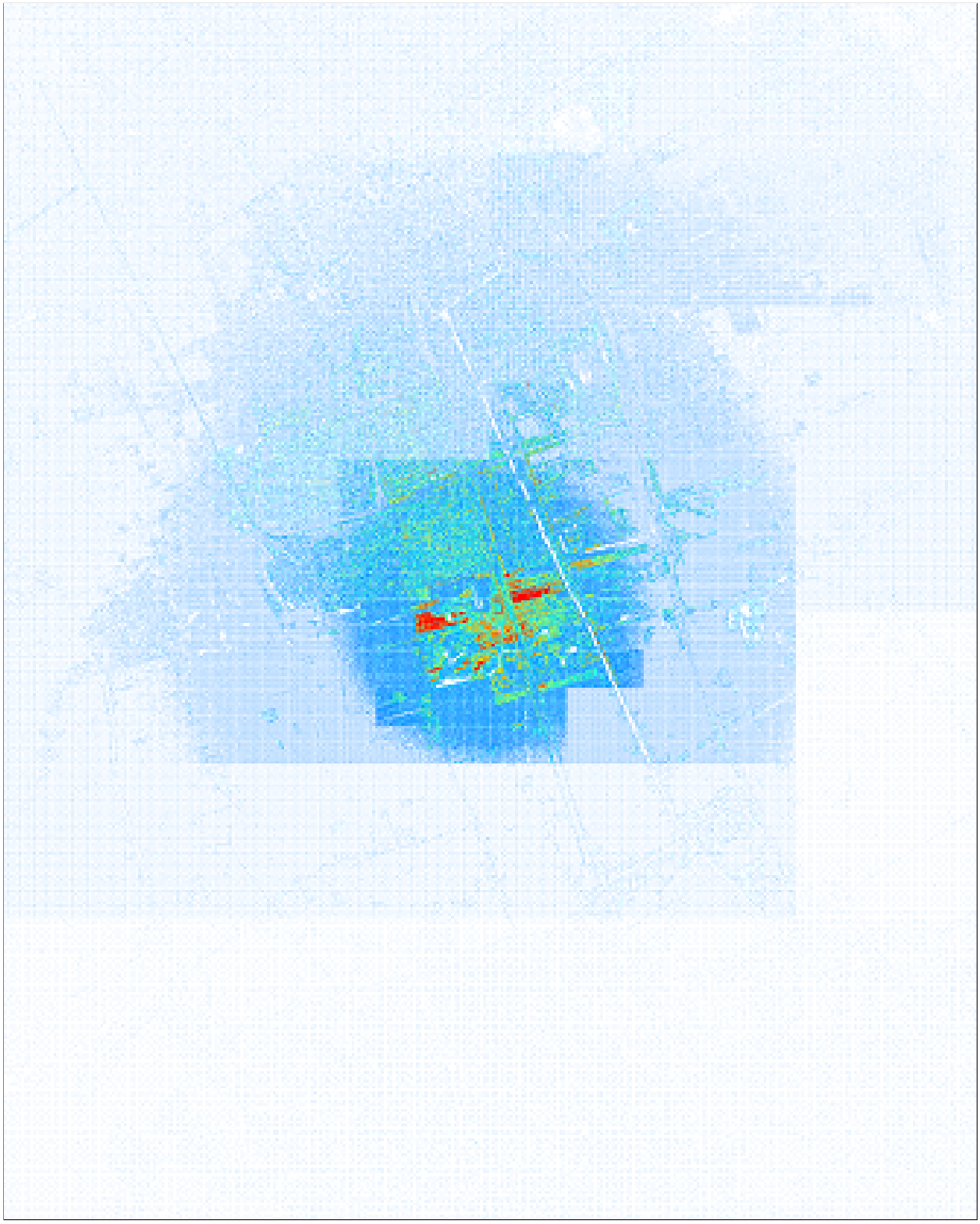


Figure 46: Density function 2 density of frame

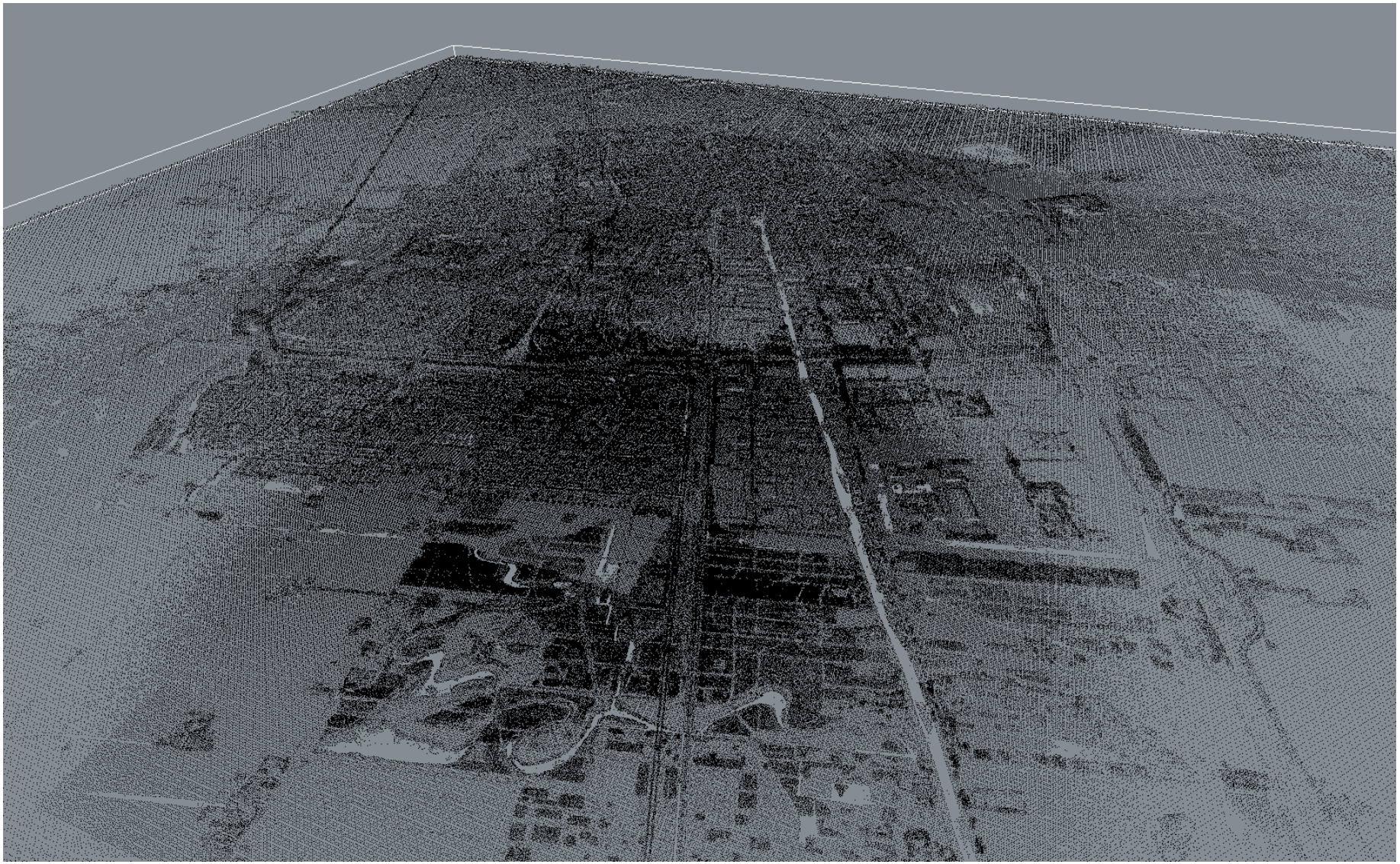


Figure 47: Density function 2 perspective view of the frame

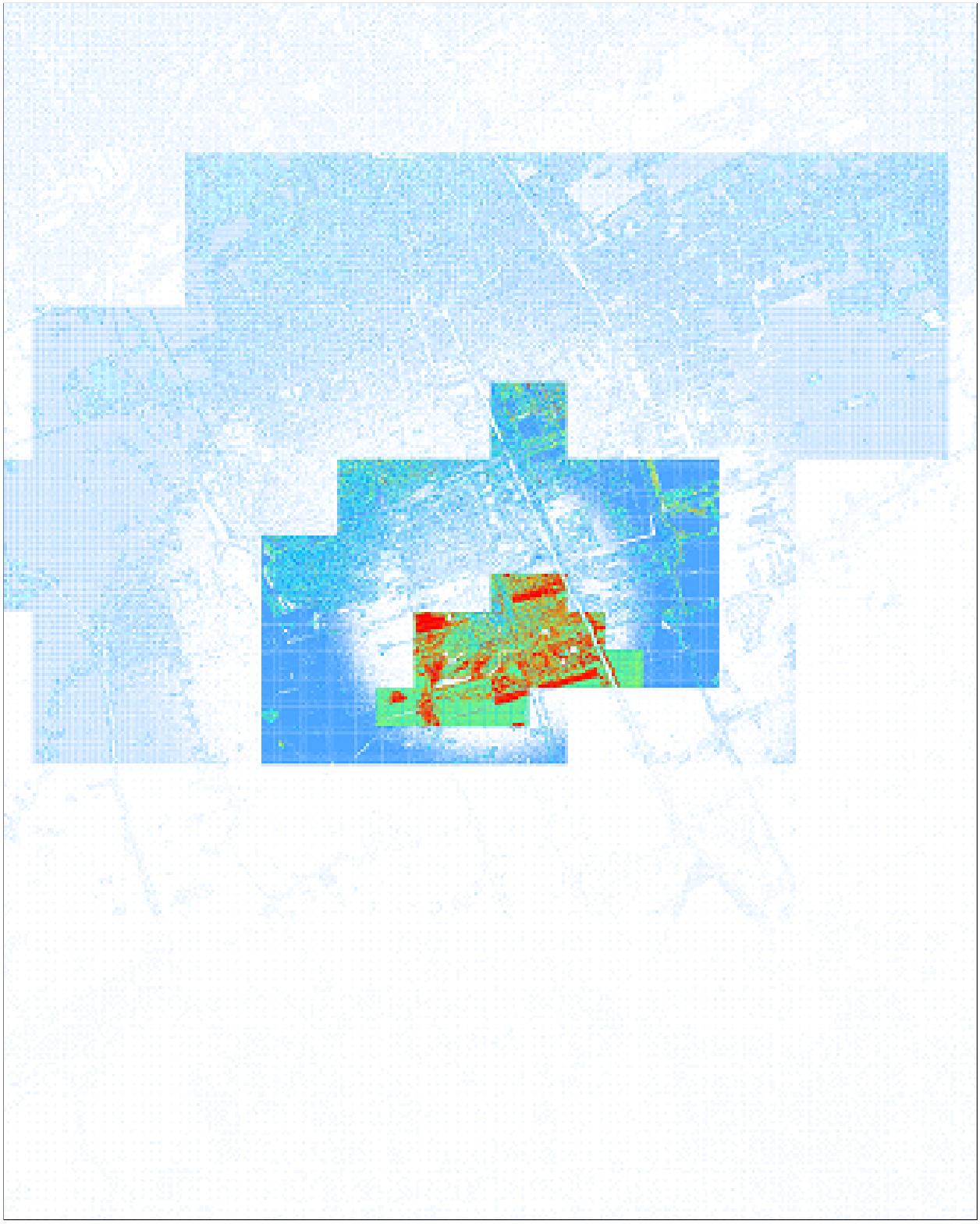


Figure 48: Density function 2 density of the removed points

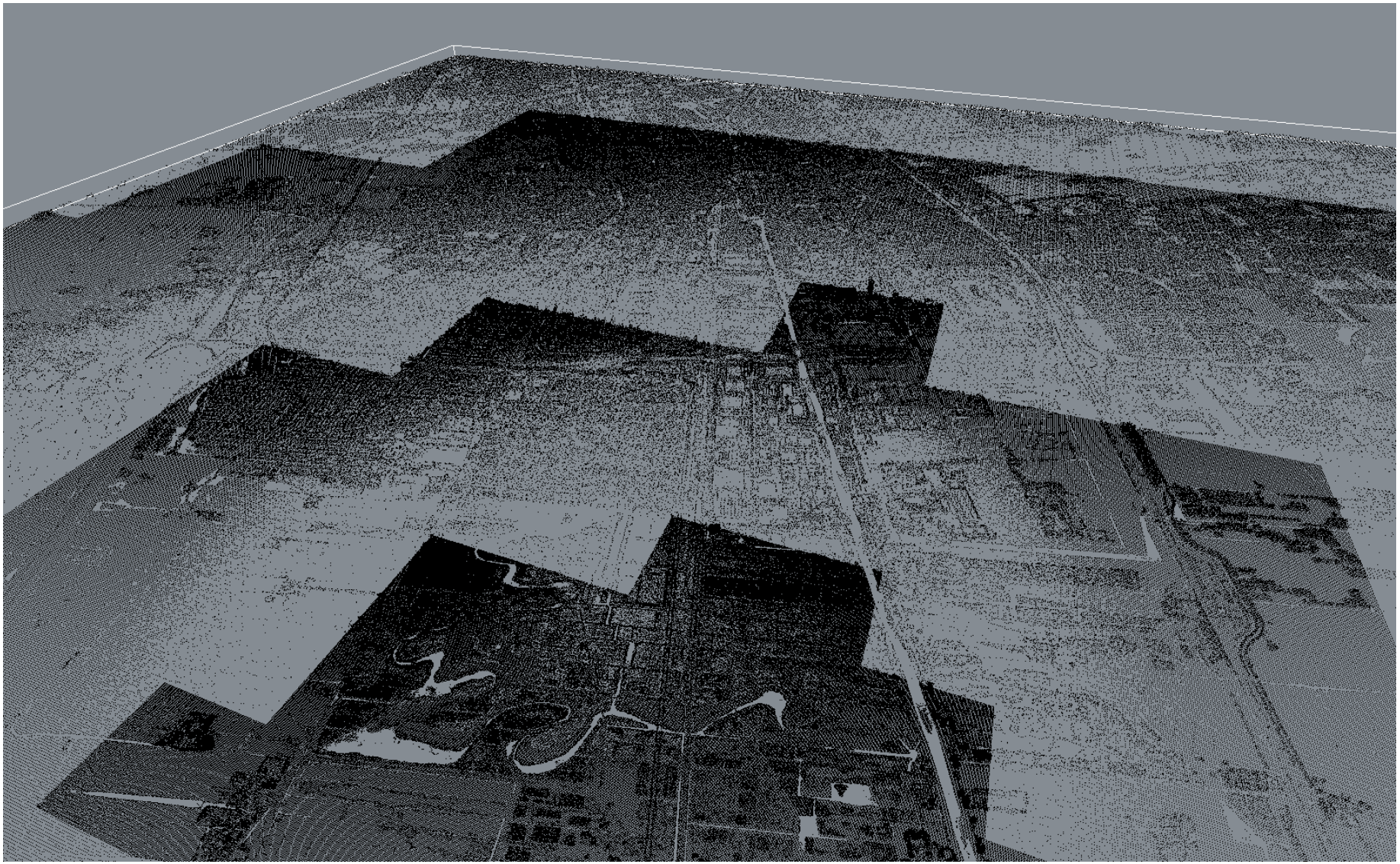


Figure 49: Density function 2 perspective view of the removed points

6.5. Density function 3

This *density function* used in this section, is presented in Figure 50. This function keeps 25% of the original data set, the area marked in green. All of the visible *density jumps* are removed, with the exception of *density jumps* created by octree blocks behind the camera view frustum. These *density jumps* do however become increasingly visible due to the lack of density in the overall data set. Figure 51 shows these *density jumps*, that stand out due to the lack of context. In Figure 52 it is visible that from the perspective view no density jumps are discernible.

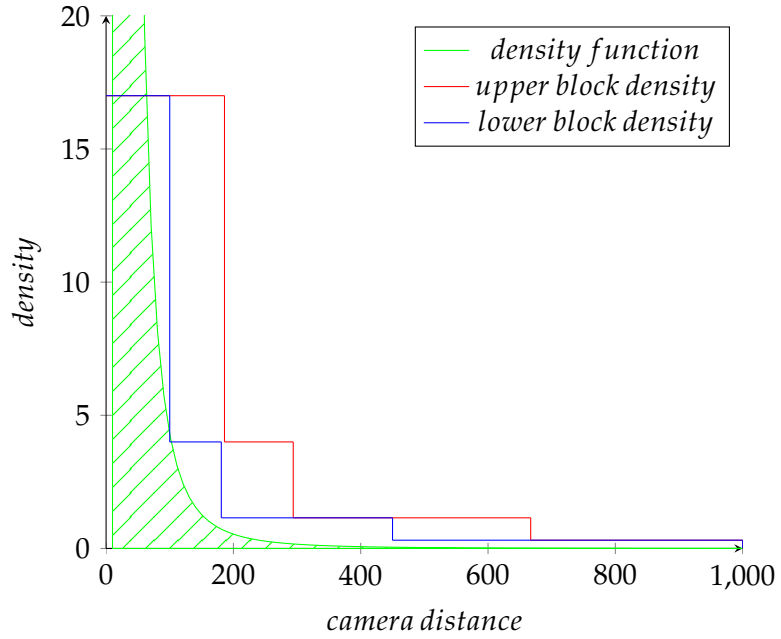


Figure 50: Density function 3

The Figures showing both the removed points density and perspective views, Figure 53 and Figure 54 respectively, show that all of the points around the *density jumps* are being removed.

Figure 51: Density function 3 density of frame

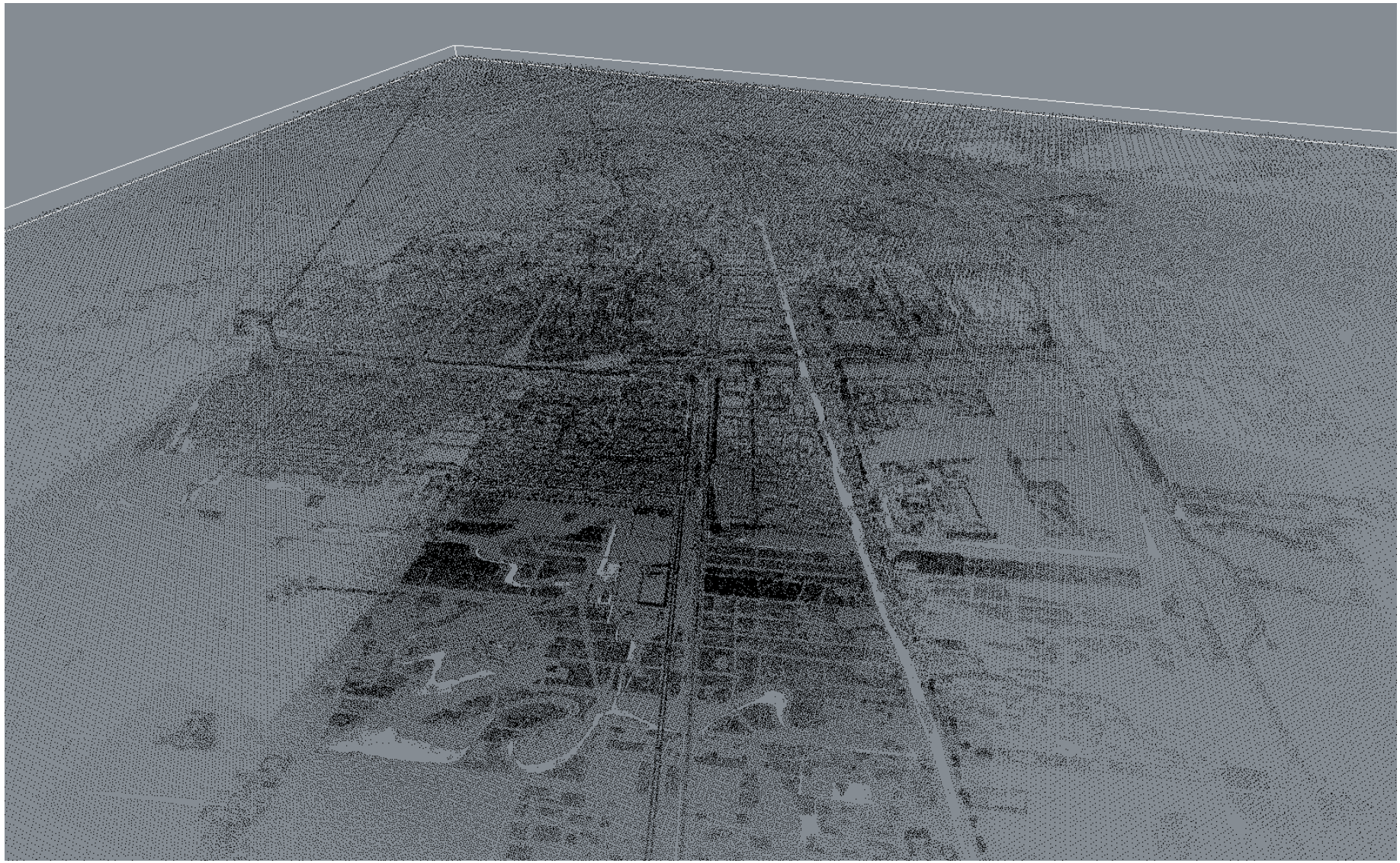


Figure 52: Density function 3 perspective view of the frame

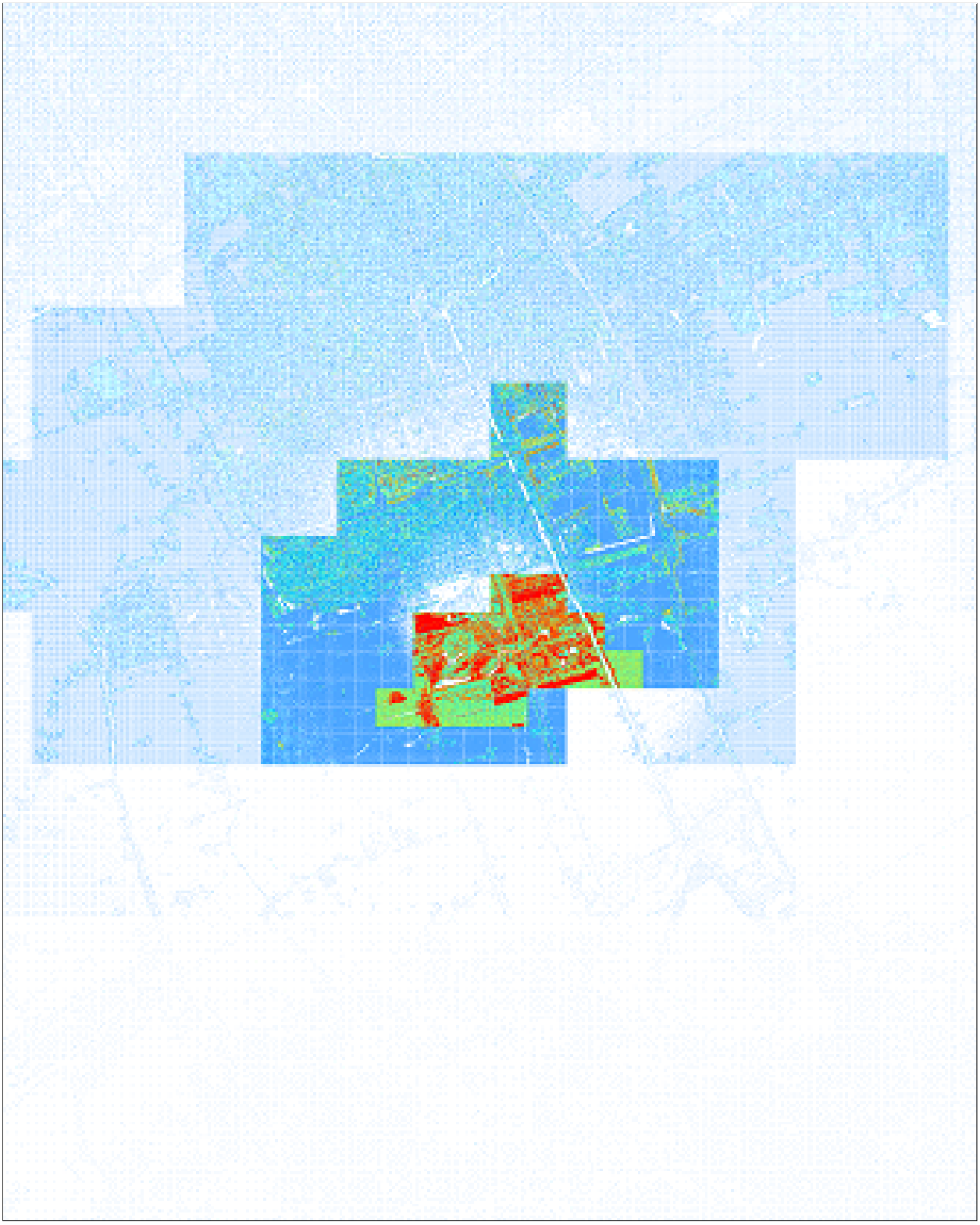


Figure 53: Density function 3 density of the removed points

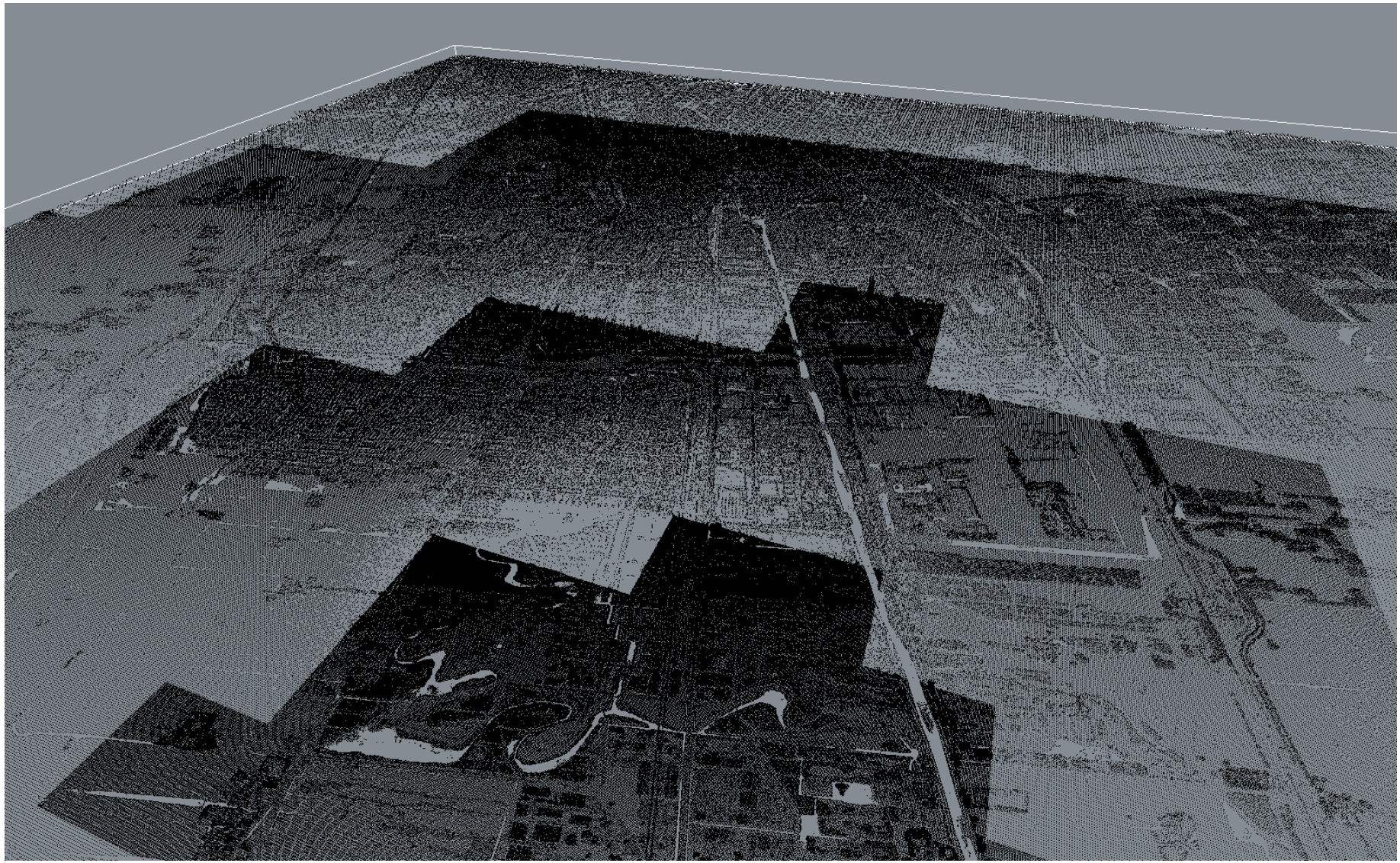


Figure 54: Density function 3 perspective view of the removed points

6.6. Density function 4

This *density function* used in this section, is presented in Figure 55. This function keeps 12.5% of the original data set, the area marked in green. All of the *density jumps* are removed. Figure 56 shows the lack of *density jumps*. In Figure 57 it is visible that there are no *density jumps* in the data set, but the context of the data set is difficult to read due to the lack of points in general.

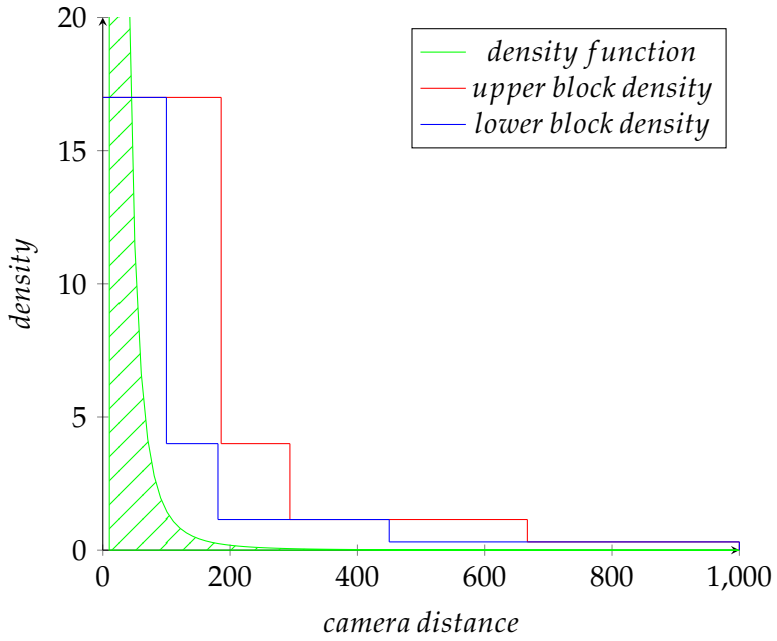


Figure 55: Density function 4

The Figures showing both the removed points density and perspective views, Figure 58 and Figure 59 respectively, show that a large part of the data set has been removed.

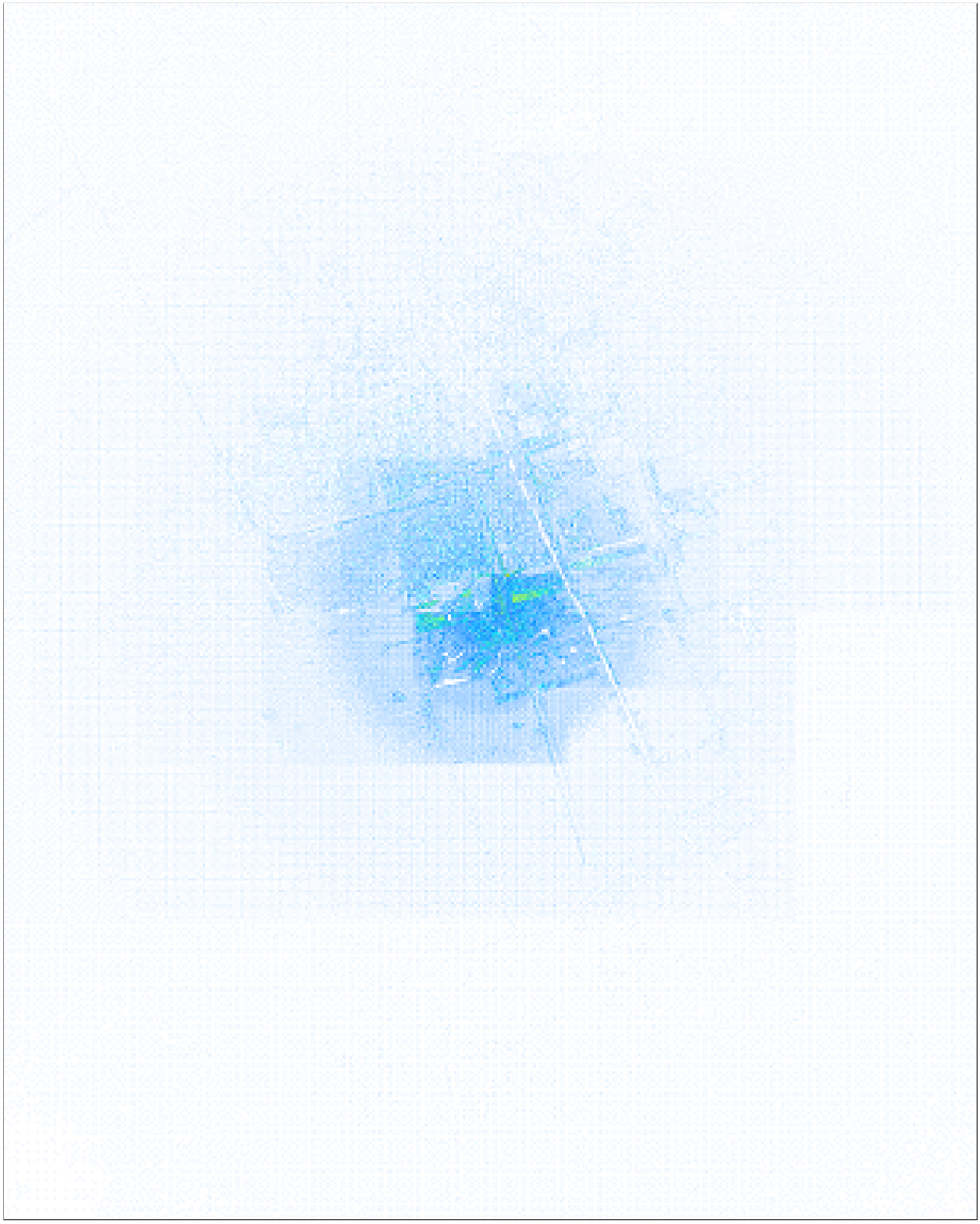


Figure 56: Density function 4 density of frame

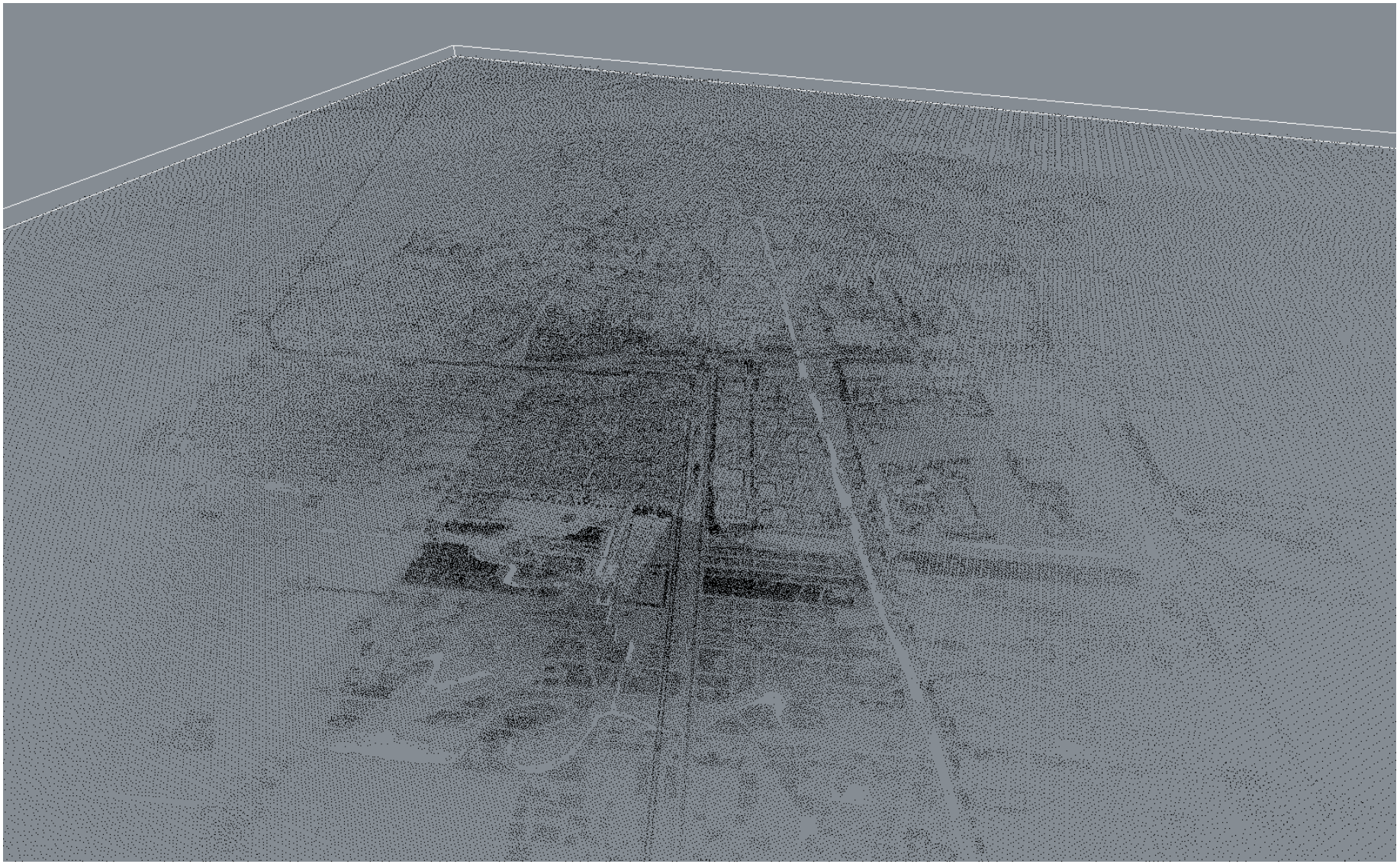


Figure 57: Density function 4 perspective view of the frame

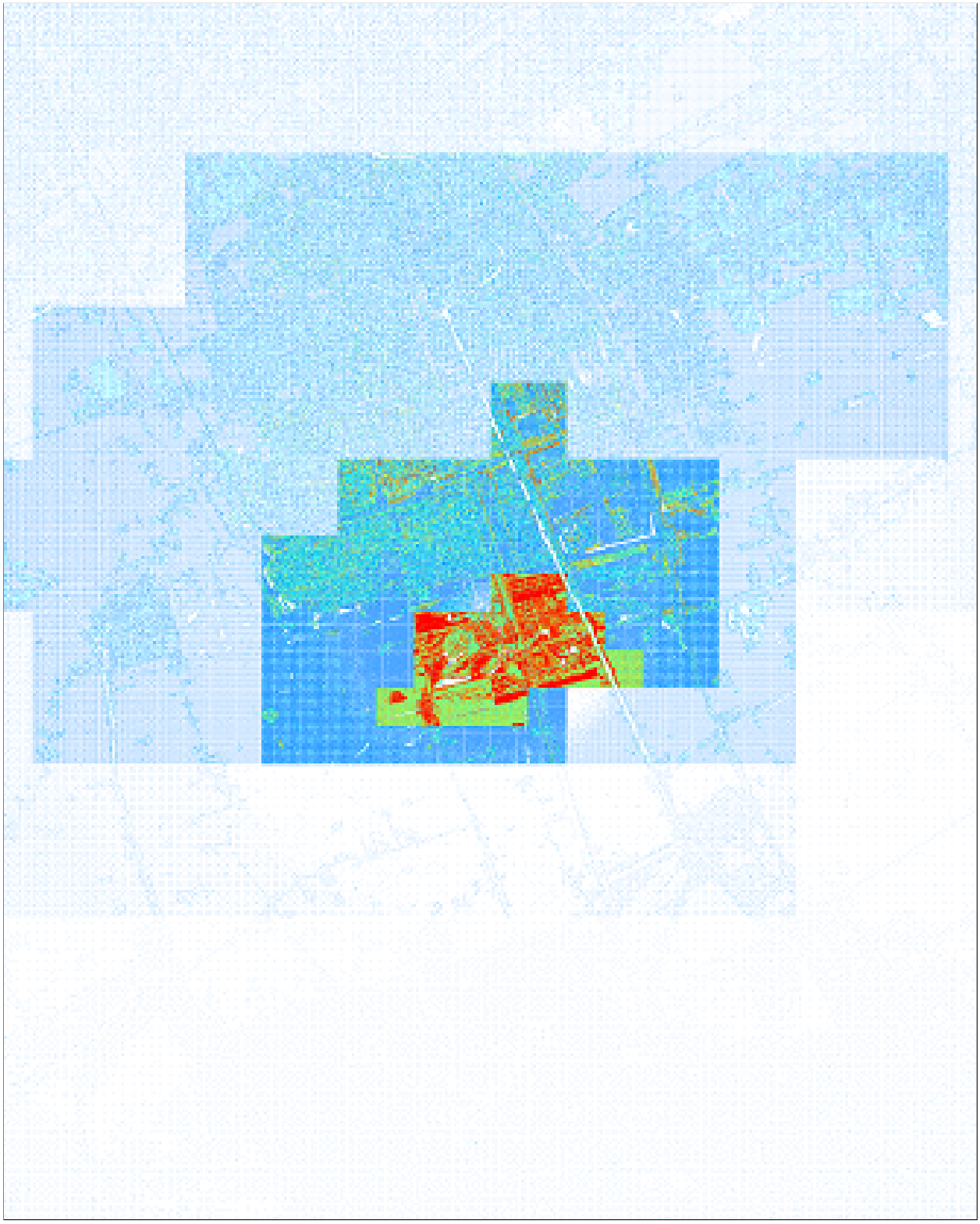


Figure 58: Density function 4 density of the removed points

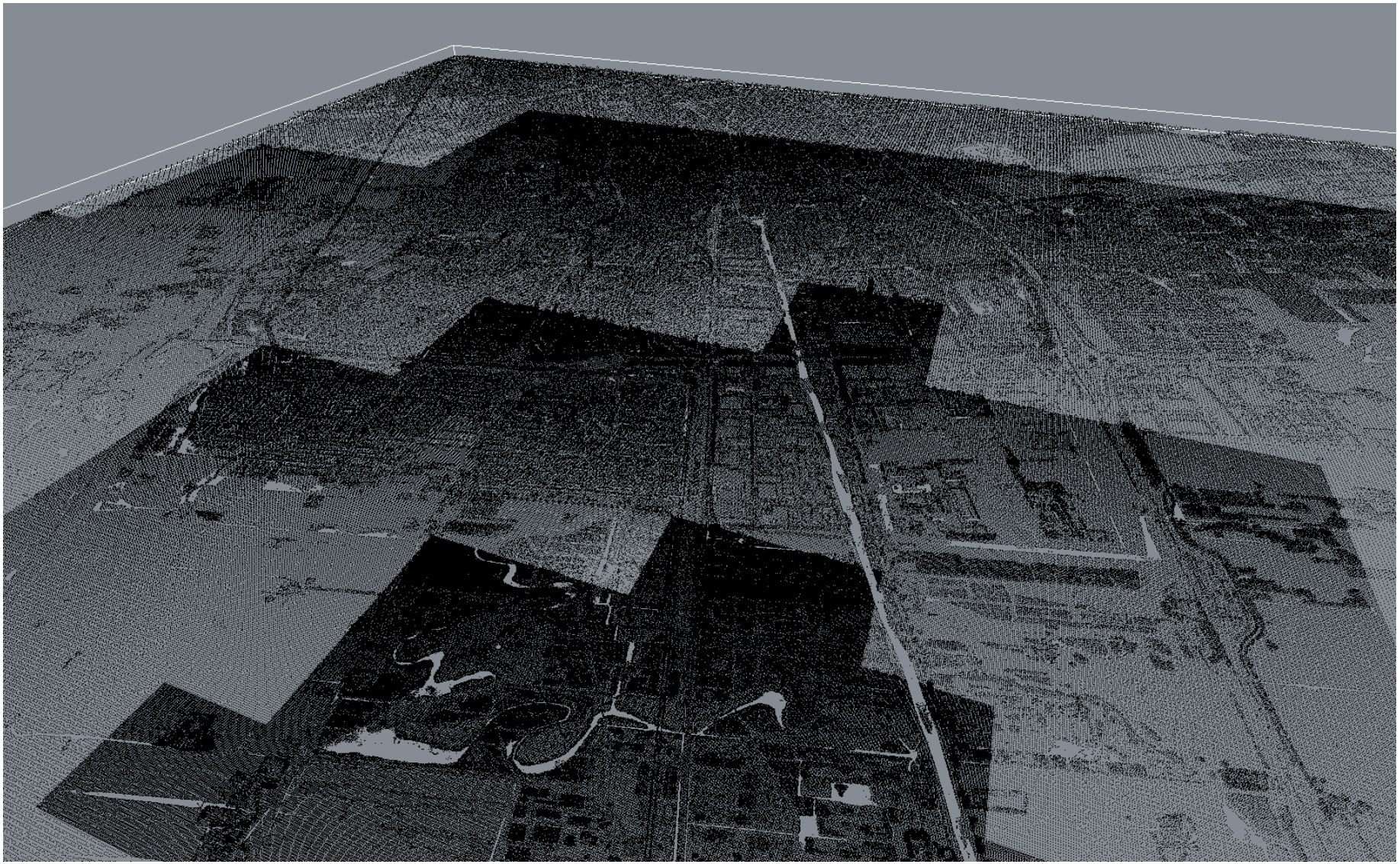


Figure 59: Density function 4 perspective view of the removed points

7. Discussion and Future work

This thesis has two objectives. Firstly to create a theoretical implementation of vario-scale visualization for point cloud data. Secondly to create a proof-of-concept for this implementation. Both of these objectives are introduced in Section 1.1. In this Chapter the results will be discussed in Section 7.1, and the main findings will be presented in Section 7.2.

7.1. Discussion

Benchmark
In the parallel point indexing benchmark two benchmarks are run, first a preliminary benchmark to determine which AWS instance is best suited for the indexing. Secondly a distributed cloud computing benchmark for parallel processing. The results are presented in Table 8

	single instance processing (s)	parallel processing (s)	speedup (%)	instances used
Municipality Delft	6 427	459	1 400	20
Province Zuid Holland	224 762	2642	8 500	100

Table 8: Comparing benchmark results

Using parallel processing shows significant speed up in indexing time for both data sets. The advantage of using multiple instances becomes clear, with an 8 500% speedup using 100 instances compared to a 1 400% speedup when using 20 instances. Where an n number of instances is used, an n00% speedup is expected. This is not the case due to the merging overhead per operation, which can not be performed in parallel and thus does not enjoy the speedup received from processing across multiple instances. The merging overhead is visualized in Figure 60. Without this merging overhead the speedup would be 2 581% for 20 instances and 15 436% for 100 instances.

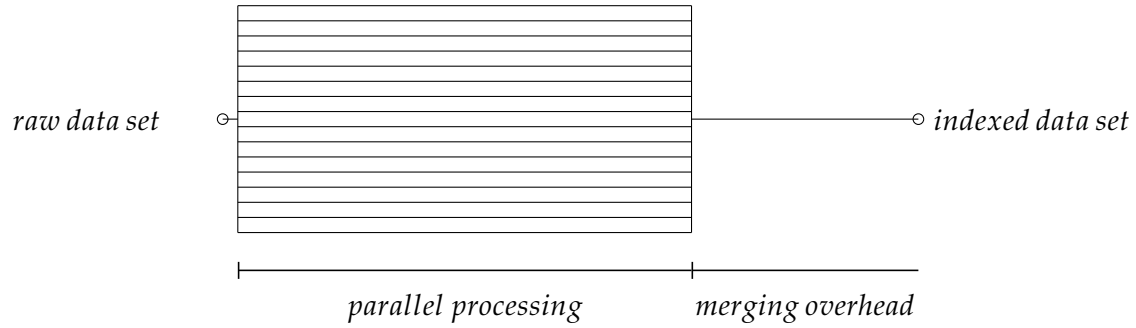
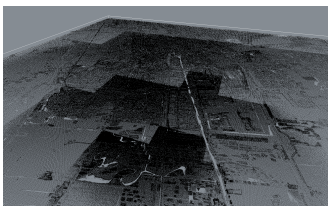
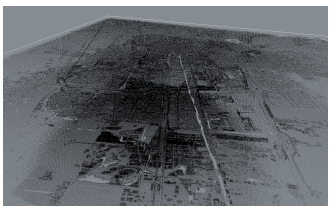


Figure 60: Merging overhead

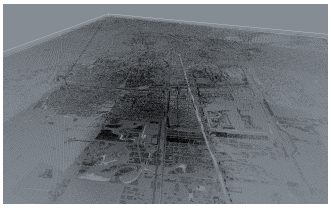
Single frame vario-scale
A visual comparison of the multiple frames, as seen in Figure 61, shows that density formula 2 is the most successful in removing *density jumps* while retaining the spatial context of the data set. Both density formula 3 and 4 remove too much of the spatial context, while density formula 1 retains too many *density jumps*. There are however slight *density jumps* visible in the furthest edges of the data set, in line with the hypothesis proposed in Chapter5.



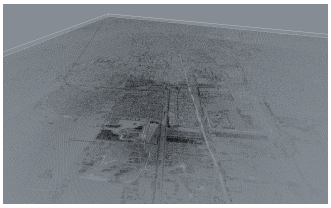
(a) Density formula 1



(b) Density formula 2



(c) Density formula 3

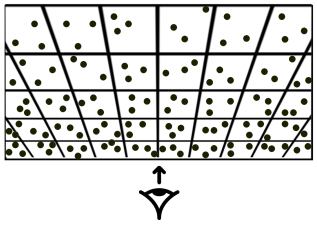


(d) Density formula 4

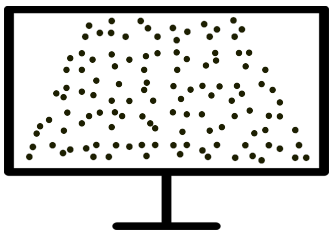
Figure 61: Visual comparison of the four density formula's used

7.2. Findings

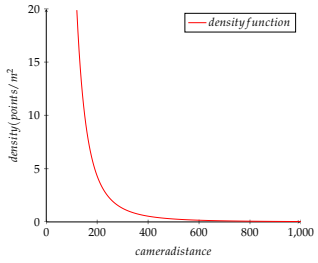
Theoretical vario-scale implementation
In Chapter 3 the theoretical reasoning behind the vario-scale implementation is explained. It follows the translation of a continuous declining density in the data set, to a homogeneous density presented on screen. In Figure 62 this transition is visualized. A declining *density formula* shown in Figure 62(c) results in a constant density on screen, shown in Figure 62(d)



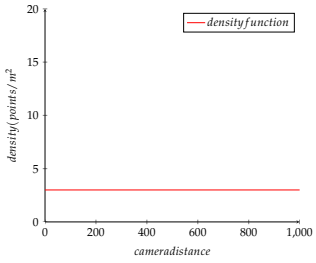
(a) Data set spatial extents



(b) User's visualization of the point cloud



(c) Density in data set



(d) Density on screen

Figure 62: From data set spatial extents to users screen

Furthermore three theoretical approaches are presented to enforce this vario-scale visualization during the proof-of-concept practical implementation. Out of these three implementation methods the *Point radius density* method is most suitable in terms of simplicity and computational complexity. The *Point radius density* method allows the enforcement of a continuous density formula through the relationship between the camera position and the projected circle around a point in

which no other point can exist, for visualization purposes of the frame.

Practical vario-scale implementation

The practical implementation of vario-scale visualization for point cloud data is based on existing web-server architecture that serves point cloud data to web-viewers. This is done through an octree index, which allows for little querying due to its spatial indexing structure. The proposed implementation in Chapter 5 removes points from the queried data set to create a point cloud with a continuous declining density.

As discussed in Section 7.1 the implementation with the best results is the implementation of *density function 2*, which removes 50% of the data set. This *density function* is presented in Figure 63, where the area marked in red is the regions of the point cloud data set that are removed.

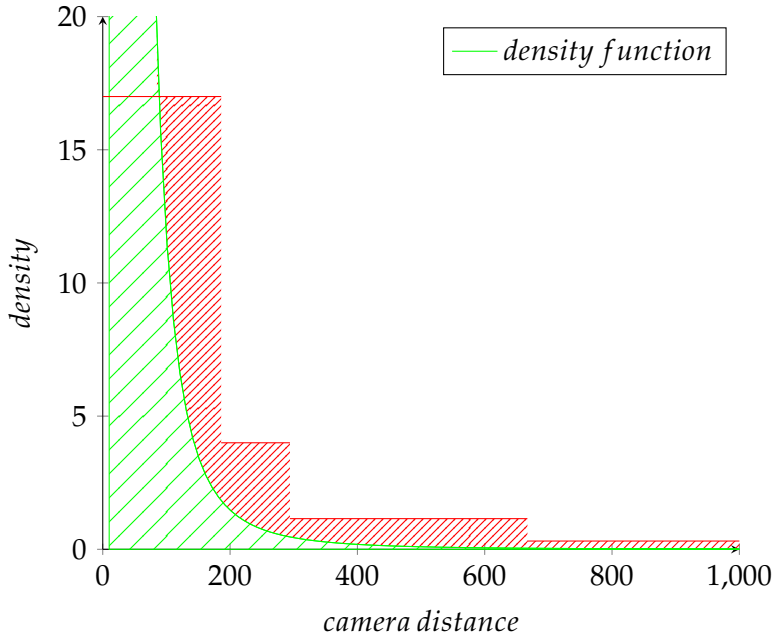
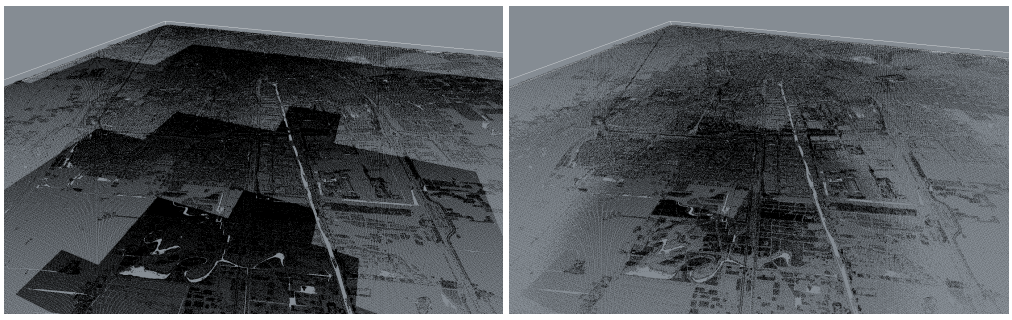


Figure 63: Density function 2

When comparing the original frame, visualised in Figure 64(a), to the vario-scale frame, visualized in Figure 64(b), it is apparent that a large part of the density jumps are removed.



(a) (b)

Figure 64: Original frame (a) compared to the vario-scale frame (b)

Main findings

The main finding of this thesis is twofold. For the theoretical implementation of vario-scale point

cloud visualization the importance of the view frustum perspective translation matrix is proven. There is a direct relationship between the parameters that create the view frustum, and the density function that should be created for a frame. The *Point radius density* method as presented is the most efficient method to enforce the required *density formula* for a given frame. For the practical implementation it is proven that it is possible to visualize massive point clouds in a vario scale way using current web viewer architecture. This method does however have its shortcomings in terms of computational speed; it is not yet possible to compute 30 frames per second. The required minimum for visualizing the point cloud in a web viewer. In terms of *density functions*, four functions have been researched, with varying results. Some *density functions* either show too many points, leaving *density jumps*, or show too little points, removing too much of the context.

7.3. Integration in scientific body of work

Currently there is a very limited body of research done on the vario-scale visualization of point cloud data sets. This thesis is one of the first to contribute to the area of research. In Section **??** recommendations will be made as to where further research should focus.

8. Conclusion and future work

This Chapter will discuss the extent to which the research question, and its sub-questions, have been answered in Section 8.1 and recommendations will be made for future works in Section 8.2

8.1. Conclusion

First the four supporting research questions will be answered, after which the main research question will be answered.

1. *To what extent is the current body of research done on the vario-scale visualization of vector data sets relevant for the vario scale visualization of point cloud data sets?*

   The current body of research for vector data sets is described in Section 2. For the vario-scale visualization of vector data sets a semantic data set (such as GAP-face tree and GAP-edge forest) is created to complement the existing vector data. This semantic data set acts as a look-up when determining the scale level and the related vario-scale visualization.

   Attempts have been made in this thesis to enrich the massive point cloud data set with such semantic information. The major difference when determining semantic information for point cloud data sets vs. vector data sets is that for the point cloud data set every point will have to be enriched. Making the amount of semantic information for the same area orders of magnitude larger for a point cloud data set. In some attempts this has proven to increase the data set size by up to 40%.

   Because indexing the data set with a fourth dimension has proven unfruitful, there is no main takeaway from the research done on the vario-scale visualization of vector data sets.

2. *To what extend can a theoretical post-processing approach be created for vario-scale visualization of point cloud data sets?*
   Three methods are proposed for the reinforcement of a per-frame vario-scale visualization of the point cloud data set. The most computationally efficient, *Point radius density*, is chosen to implement. The idea of enforcing density through the radius around a point is novel in its simplicity and the way in which it allows a continuous density to be reinforced throughout the point cloud data set.

3. *Which point-cloud processing framework is best suited to create a proof-of-concept vario scale visualization platform for the AHN2 point cloud?*

   Chapter 2 gives an introduction into the point cloud processing life-cycle, from storage to visualization. This creates an introduction to Chapter 4 where multiple frameworks for each phase of the point cloud processing life-cycle are considered.

   The choice is made for a point cloud processing framework that unitizes the possibilities of cloud computing as well as web servers. With all instances and storage running on AWS, the decision has been made to use Entwine for indexing, Greyhound for processing and Plas.io for visualization. These frameworks are compatible and thus allow this thesis to focus on the novel implementation of vario-scale visualization for point cloud data sets.

4. *To what extend can the theoretical approach be implemented in an existing point cloud web visualization framework?*

The implementation of the theoretical method is done through a second web server, which retrieves the point cloud data from the Greyhound web server and processes it as a vario-scale solution. As discussed in Section **??** future research should look into implementing the solution in the Greyhound web server.

Lastly the main research question will be answered:

To what extent can a vario-scale visualization method be created that eliminates density jumps from the web-based visualization of the AHN2 point cloud?

The creation of a vario-scale visualization method is successful, with the creation of a method that is developed on top of an existing point cloud visualization technology stack.

A theoretical approach of enforcing a density throughout the data set so that the density on screen is perceived as continuous is created. For the practical approach, removing *density jumps*, multiple *density formulas* are created that remove points close to transition areas between Octree levels. This is enforce using the *Point radius density* method, where a formula is developed whereby point density is linked to circle radius relative to the distance to the camera, in which there can be no neighbouring point. Allowing for an enforcement of the *upper density limit* in the point cloud data set.

This theoretical approach is implemented in an existing point cloud visualization technology stack, where it is evident that a significant amount of points need to be removed to achieve relevant results in removing the *density jumps*.

8.2. Future work

This Section will discuss improvements on the current implementation and future work for further research on the vario-scale visualization of point cloud data sets.

4D index implementation

In Chapter 3 a brief mention is made into the 4D indexing of the point cloud data set, something which has been researched in this thesis but which proved to be too time intensive. The general goal of creating this 4D index is to have a 4th value which assists in the retrieval of points in a vario-scale manner. This, in theory, removes a computational load for each frame and displaces the computational load to the indexing of the point cloud. This will prove to be a major advantage, since it will be a big step towards reaching the ability to handle 30 vario-scale frames per second.

The 4th dimension on which the point cloud is indexed should be an attribute which assigns a weight to the point, determining whether or not it should be rendered. This weight is similar to the Generalized Area Partitioning implementation discusses in Chapter 2 introduced by van Oosterom (2005). This would remove the need to compute any point neighbourhood, thus removing a large part of the computational load. Removing this processing step is visualized in Figure 65.

Determine a global formula, and query blocks accordingly

In Chapter 3 the relationship between the camera parameters and the required *density function* for vario-scale visualization is proven. For the results presented in Chapter 6 four different *density functions* have been tested. Determination of a *global density formula* which, dependant on the camera parameters, would create a vario-scale visualization method independent of data set or screen parameters.

This would require a new point cloud visualization framework to be created, since multiple factors such as indexing and Octree block selection method would have to be adjusted for this *global density function*. This would lay the foundation for a truly vario-scale web framework for

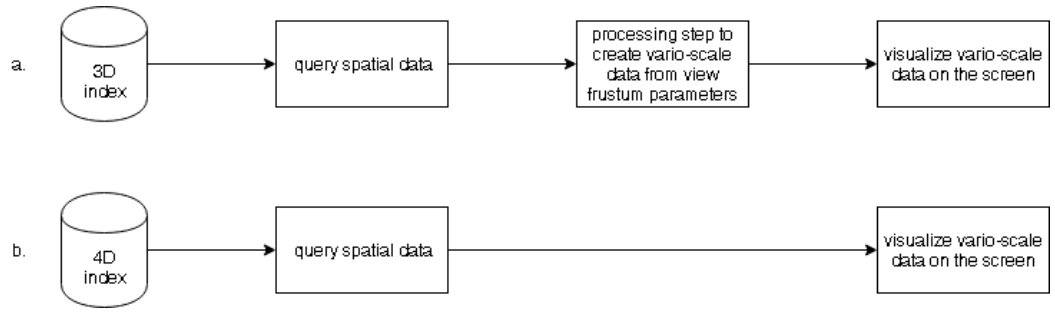


Figure 65: 4D vs 3D indexing method

point cloud visualization.

Implementation in Greyhound
In Chapter 5 the framework is introduced, and there are two web servers introduced. The first is the existing Greyhound web server, the second is the web server used for the vario-scale processing. Moving the implementation to the Greyhound web server would remove a web server from the workflow, and altogether simplify the implementation. This would improve the speed with which a frame can be computed, because sending the point cloud data between web servers is a time consuming endeavour.

Implementations on client-side
The complete implementation introduced in Chapter 5 is based on a server-side architecture. Because no data is send to the user before all the processing is done, there can be a substantial waiting time. By moving the implementations to the client side the need to perform per-frame calculations on the server side can be eliminated. This allows points to be rendered on screen faster. Before a client side implementation is realised, other optimizations such as the **4d indexing structure** should be realised first. These remove computational complexity, which would allow the computation to be done client-side.

Process only the affected octree blocks
In Chapter 6 the presented results show density jumps, although these occur in limit areas. By predicting these locations, and only processing the point cloud blocks that are neighbouring the *density jump*, this implementation would be faster. Research will have to be done in determining whether or not the time saved by processing fewer blocks is larger than the overhead needed to compute which blocks are neighbouring a *density jump*.

Research into addition of Octree blocks
The results presented in Chapter 6 are created by removing points from the existing web viewer architecture using an indexed Octree. There is a need to remove a large part of the point cloud data set before the *density jumps* are removed. By using addition of points instead of removal of points the implementation has a better control of which blocks are queried. The addition of Octree blocks works on the basis of 'filling' the space underneath the *density function* until it is filled. This removes much of the computational load needed to remove individual points. The addition of Octree blocks requires a new block selection strategy to be created.

References

Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., & Saltz, J. (2013). Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, *6*(11), 1009–1020. doi:10.14778/2536222.2536227. arXiv: NIHMS150003

ASPRS. (2013). LAS Specification Version 1.4-R13. *The American Society for Photogrammetry & Remote Sensing*, *July*(November 2011), 1–28.

Byrne, D. M., Corrado, C., & Sichel, D. (2017). The Rise of Cloud Computing: Minding Your P's and Q's. (March). Retrieved from https://bea.gov/about/pdf/acm/2017/the-rise-of-cloud-computing-minding-your-ps-and-qs.pdf

Cecconi, A. & Galanda, M. (2002). Adaptive Zooming in Web Cartography. *Computer Graphics Forum*, *21*(4), 787–799. doi:10.1111/1467-8659.00636

Chang, H.-c. & Wang, L.-c. (2010). A Simple Proof of Thue ' s Theorem on Circle Packing. arXiv: 1009.4322v1

Dean, J. & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, *51*(1), 107. doi:10.1145/1327452.1327492. arXiv: 10.1.1.163.5292

Dutt, A., Jain, H., & Kumar, S. (2018). Providing Software as a Service : a design decision (s) model. *Information Systems and e-Business Management*, *16*(2), 327–356. doi:10.1007/s10257-017-0356-9

Elsenberg, J., Borrmann, D., & Nüchter, A. (2013). One billion points in the cloud - an octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, *76*, 76–88.

Gong, J., Zhu, Q., Zhong, R., Zhang, Y., & Xie, X. (2012). An Efficient Point Cloud Management Method Based on a 3D R-Tree. *Photogrammetric Engineering & Remote Sensing*, *78*(4), 373–381. doi:10.14358/PERS.78.4.373

Hanusniak, V., Svalec, M., Branicky, J., Takac, L., & Zabovsky, M. (2015). Exploitation of Hadoop framework for Point Cloud Geographic Data Storage System.

Hug, C., Krzystek, P., & Fuchs, W. (2004). Advanced LiDAR data processing with LasTools. *XXth ISPRS Congress*, (July), 12–23.

Jian, X., Xiao, X., Chengfang, H., Zhizhong, Z., Zhaohui, W., & Dengzhong, Z. (2015). A hadoop-based algorithm of generating DEM grid from point cloud data. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, *40*(7W3), 1209–1214. doi:10.5194/isprsarchives-XL-7-W3-1209-2015

leyman Eken & Sayar, A. (2015). Big Data Frameworks for Efficient Range Queries to Extract Interested Rectangular Sub Regions. *International Journal of Computer Applications*, *119*(22), 36–39. doi:10.5120/21372-4423

Li, Z., Hodgson, M. E., & Li, W. (2017). A general-purpose framework for parallel processing of large-scale LiDAR data. *International Journal of Digital Earth*, (October), 1–22. doi:10.1080/17538947.2016.1269842

Li, Z., Yang, C., Liu, K., Hu, F., & Jin, B. (2016). Automatic Scaling Hadoop in the Cloud for Efficient Process of Big Geospatial Data. *ISPRS International Journal of Geo-Information*, *5*(10), 173. doi:10.3390/ijgi5100173

Martinez-rubi, O., Verhoeven, S., van Meersbergen, M., Schutz, M., van Oosterom, P., Gonclves, R., & Tijssen, T. (2015). Taming the beast : Free and open-source massive point cloud web visualization. *Capturing Reality Forum 2015*, (March 2016), 23–25. doi:10.13140/RG.2.1.1731.4326

Meng, X., Bradley, J., Street, S., Francisco, S., Sparks, E., Street, S., … Franklin, M. J. (2016). MLlib : Machine Learning in Apache Spark. *17*, 1–7. doi:10.1145/2882903.2912565. arXiv: arXiv:1505.06807v1

Pancholi, V. R., Patel I, B. P., Principal, C., Gandhi, M. L., & Patel, B. P. (2017). A Study on Services Provided by Various Service Providers of Cloud Computing. *Advances in Computational Sciences and Tech*, *10*(6), 1725–1729. Retrieved from http://www.ripublication.com

Poux, F., Hallot, P., Neuville, R., & Billen, R. (2016). Smart Point Cloud: Definition and Remaining Challenges. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *IV-2/W1*(October), 119–127. doi:10.5194/isprs-annals-IV-2-W1-119-2016

Růžička, J., Orčík, L., Růžičková, K., & Kisztner, J. (2016). Processing LIDAR data with Apache Hadoop. doi:10.1007/978-3-319-45123-7

Sayar, A., Eken, S., & Öztürk, O. (2015). Kd-tree and quad-tree decompositions for declustering of 2D range queries over uncertain space. *Frontiers of Information Technology & Electronic Engineering*, *16*(2), 98–108. doi:10.1631/FITEE.1400165

Schuetz, M. (2016). *Potree : Rendering Large Point Clouds in Web Browsers* (Doctoral dissertation).

Schütz, M., Kröstl, K., & Wimmer, M. (2019). Real-Time Continuous Level of Detail Rendering of Point Clouds. In *Ieee vr 2019* (pp. 1–8).

Suba, R., Meijers, M., & van Oosterom, P. (2013). 2 D vario-scale representations based on real 3 D structure.

Svalec, M., Takac, L., & Zabovsky, M. (2015). Performance enhancing of storage system for point cloud geographic data. *2015 10th System of Systems Engineering Conference, SoSE 2015*, 434–438. doi:10.1109/SYSOSE.2015.7151919

van der Zon, N. (2013). Kwaliteitsdocument AHN2.

van Oosterom, P. (2005). Variable-scale Topological Data Structures Suitable for Progressive Data Transfer: The GAP-face Tree and GAP-edge Forest. *Cartography and Geographic Information Science*, *32*(4), 331–346. doi:10.1559/152304005775194782

van Oosterom, P., de Vries, M., & Meijers, M. (2006). Vario-scale data server in a web service context. *Proceedings of the ICA Commission on Map Generalisation and Multiple Representation*, 1–14.

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

van Oosterom, P. & Meijers, M. (2014). Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. *International Journal of Geographical Information Science*, *28*(3), 455–478. doi:10.1080/13658816.2013.809724

van Oosterom, P., Ravada, S., Horhammer, M., Marinez Rubi, O., Ivanova, M., Kodde, M., & Tijssen, T. (2014). Point cloud data management. *IQmulus Workshop on Processing Large Geospatial Data*, (July).

Wang, C., Hu, F., Sha, D., & Han, X. (2017). Efficient LiDAR point cloud data managing and processing in a Hadoop-based distributed framework. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*, 121–124.

White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly Media, Inc.

Woo, H., Kang, E., Wang, S., & Lee, K. H. (2002). A new segmentation method for point cloud data. *International Journal of Machine Tools and Manufacture*, *42*(2), 167–178. doi:10.1016/S0890-6955(01)00120-1

Zaharia, M., Chowdhury, M., Das, T., & Dave, A. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Nsdi*, 2–2. doi:10.1111/j.1095-8649.2005.00662.x. arXiv: EECS-2011-82

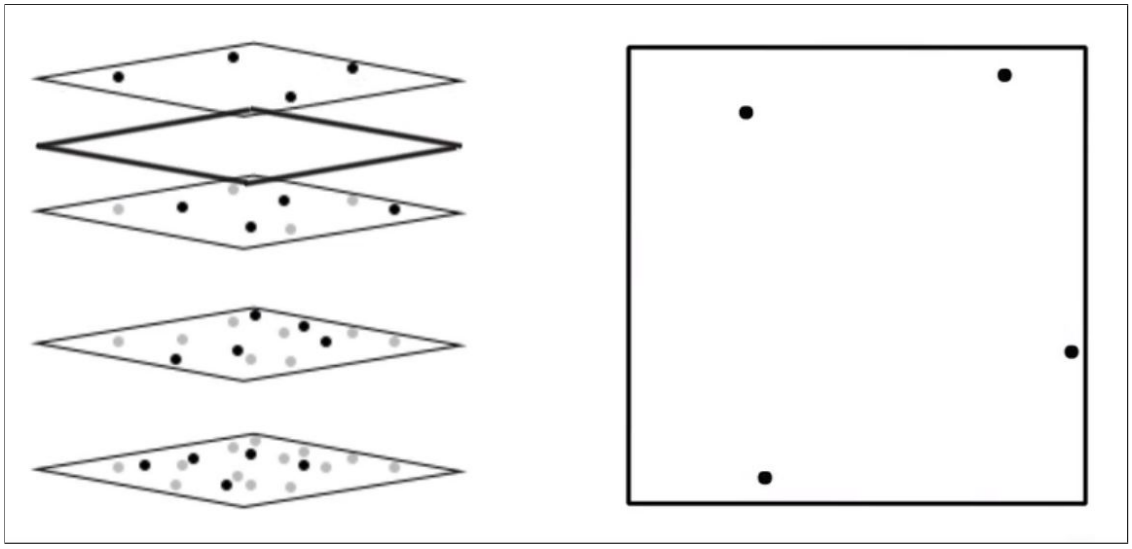
Zaharia, M., Chowdhury, M., J. Franklin, M., Shenker, S., & Stoica, I. (2010). Spark: Cluster Computing with Working Sets. *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, *10*, 10.

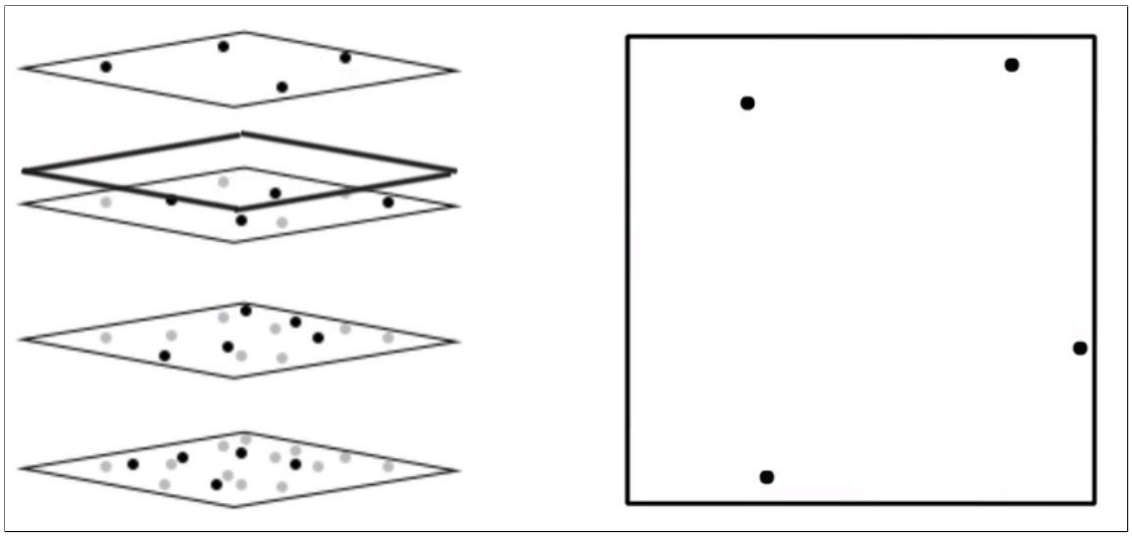
Zhong, Y., Han, J., Zhang, T., Li, Z., Fang, J., & Chen, G. (2012). Towards parallel spatial query processing for big spatial data. *Proceedings of the 2012 IEEE 26th International Parallel and*

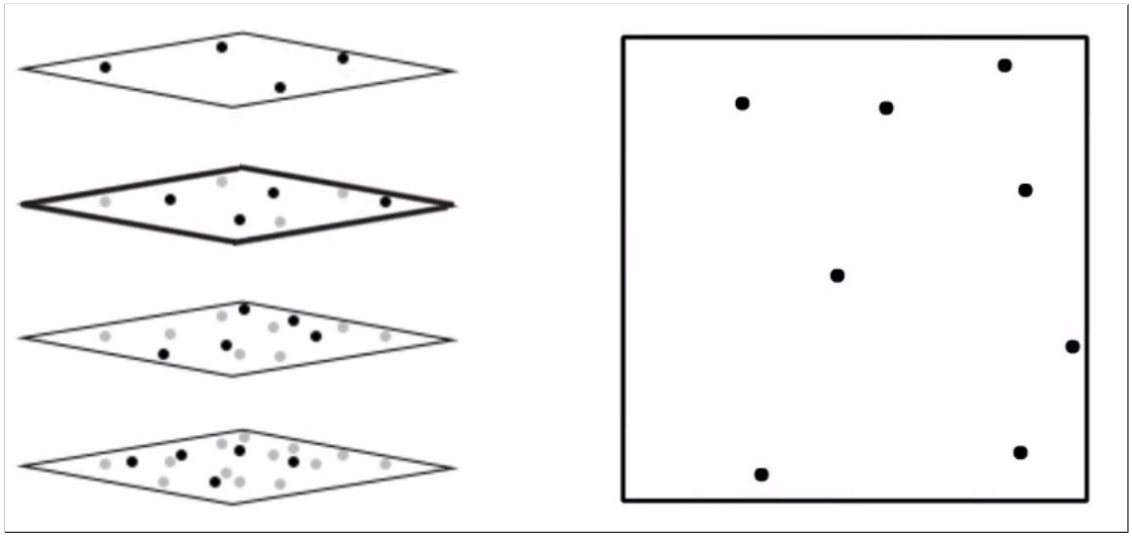
Distributed Processing Symposium Workshops, IPDPSW 2012, 2085–2094. doi:10.1109/IPDPSW.2012.245

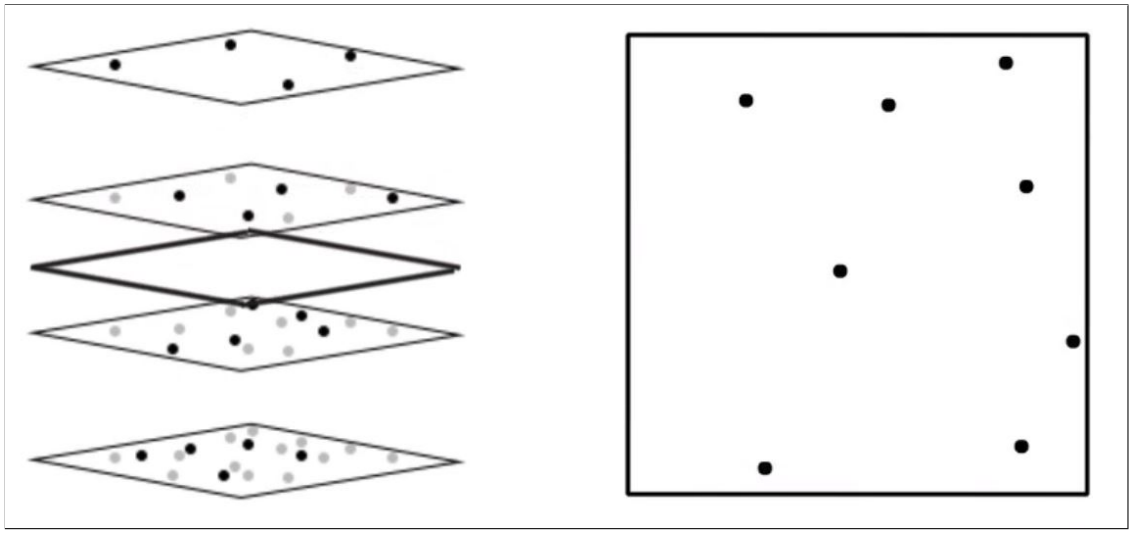
Appendices

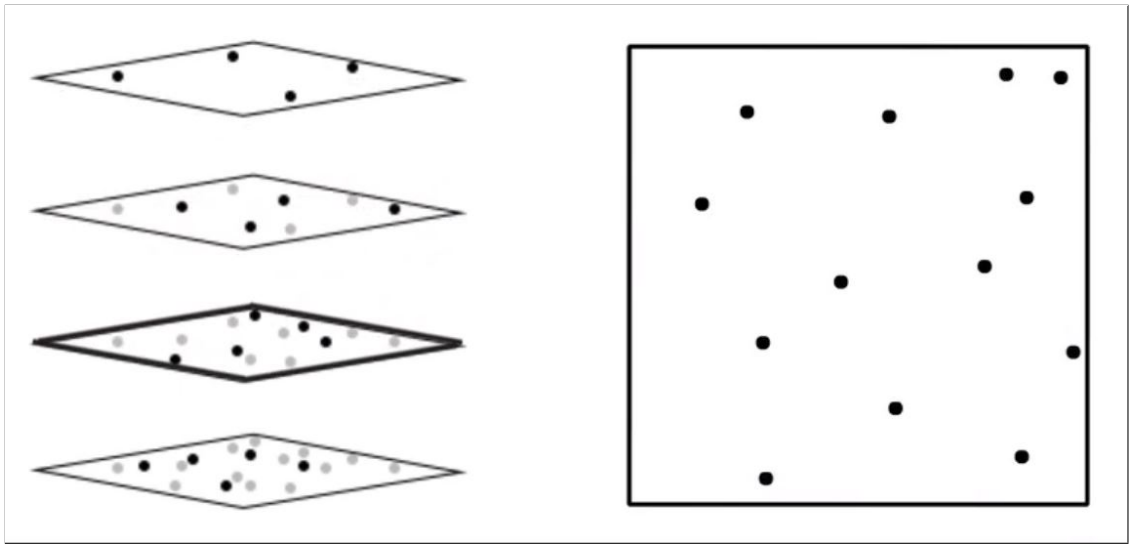
A. Discrete levels top down visualization

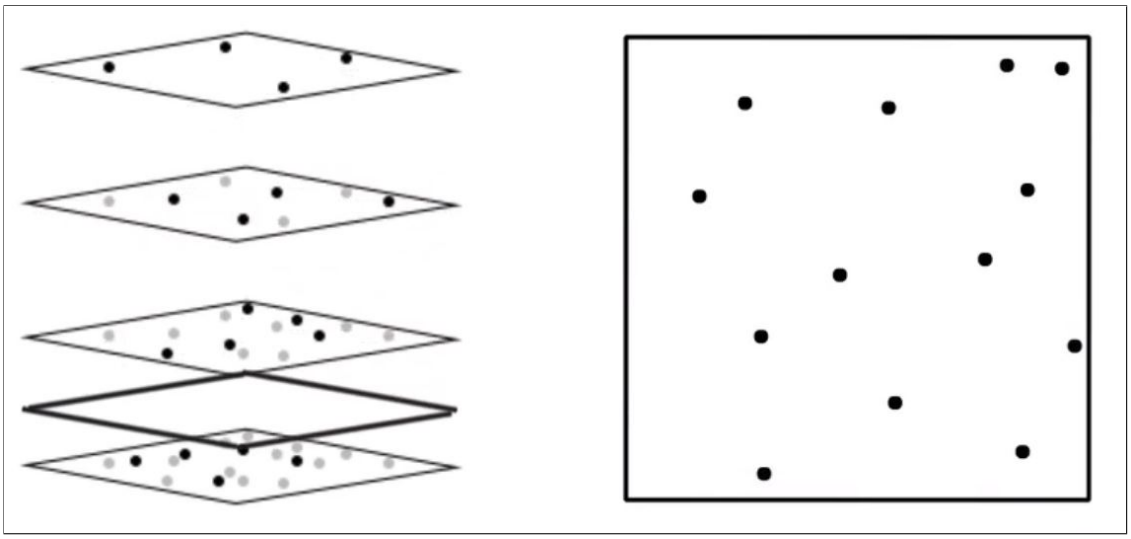


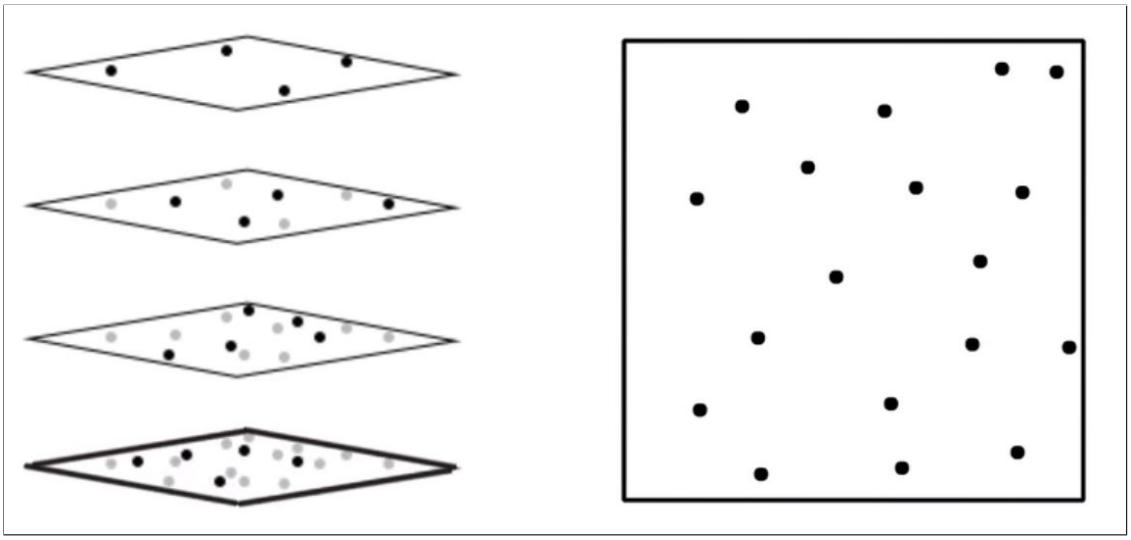




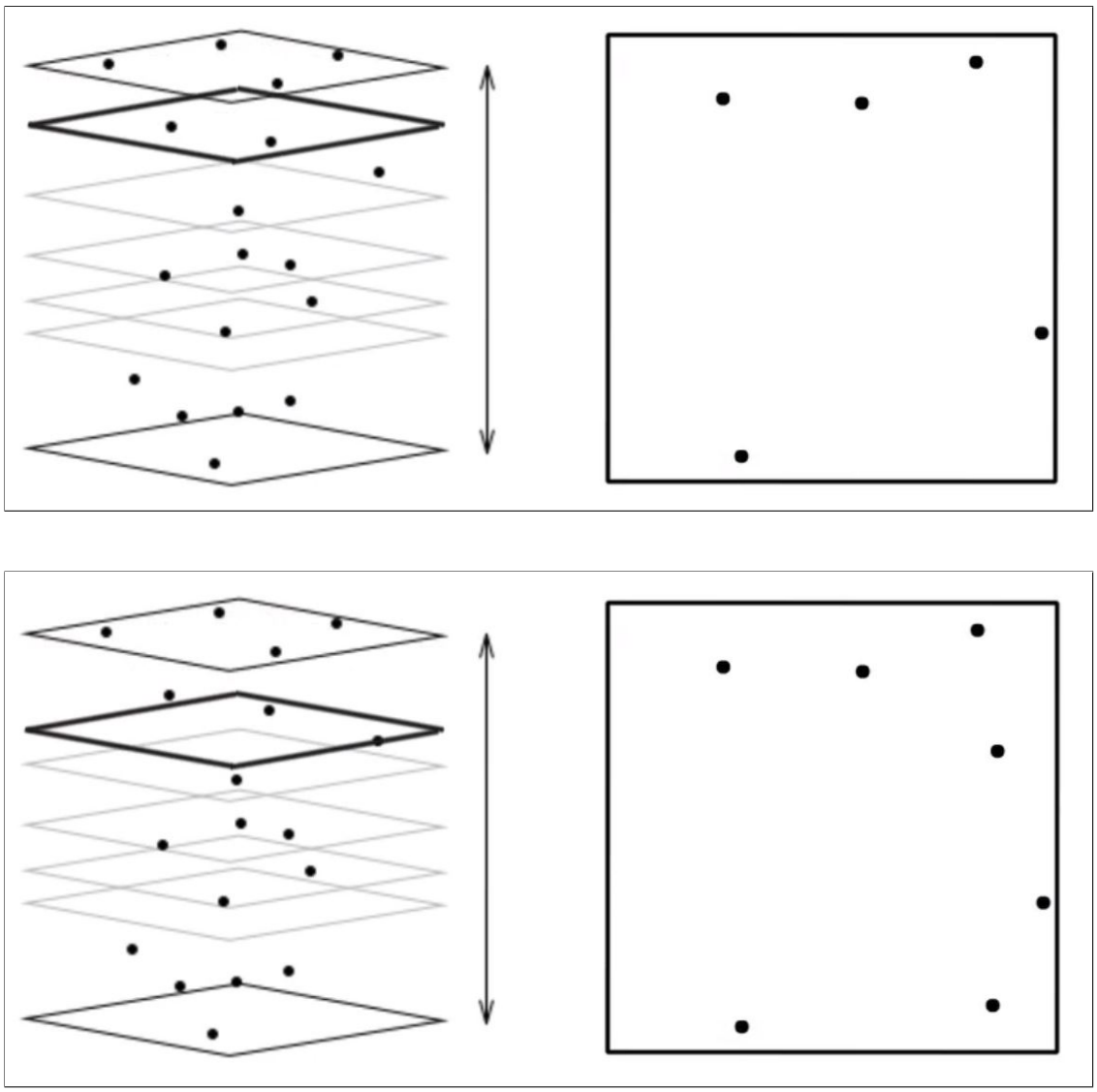


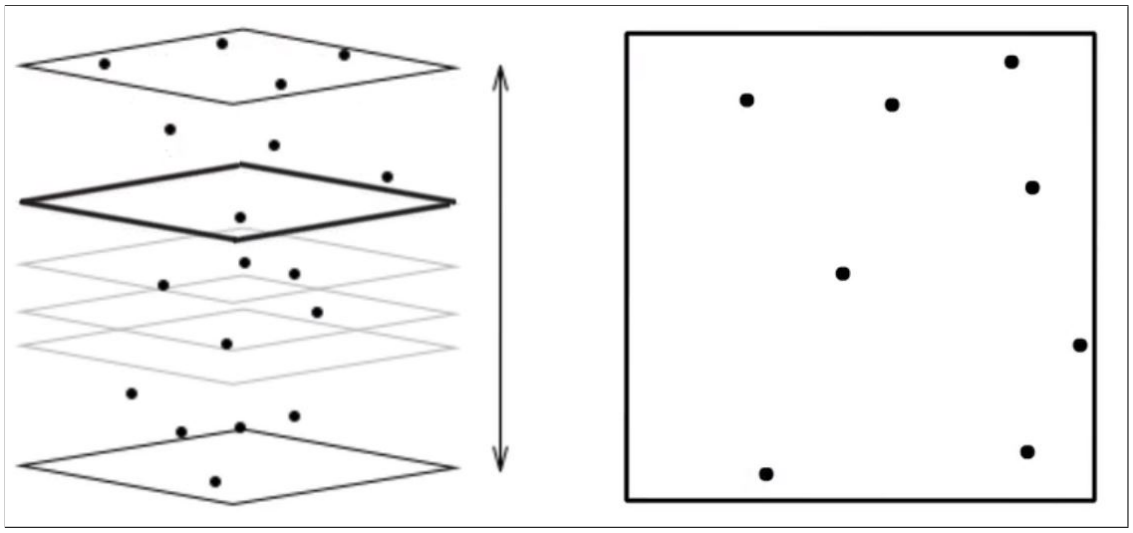


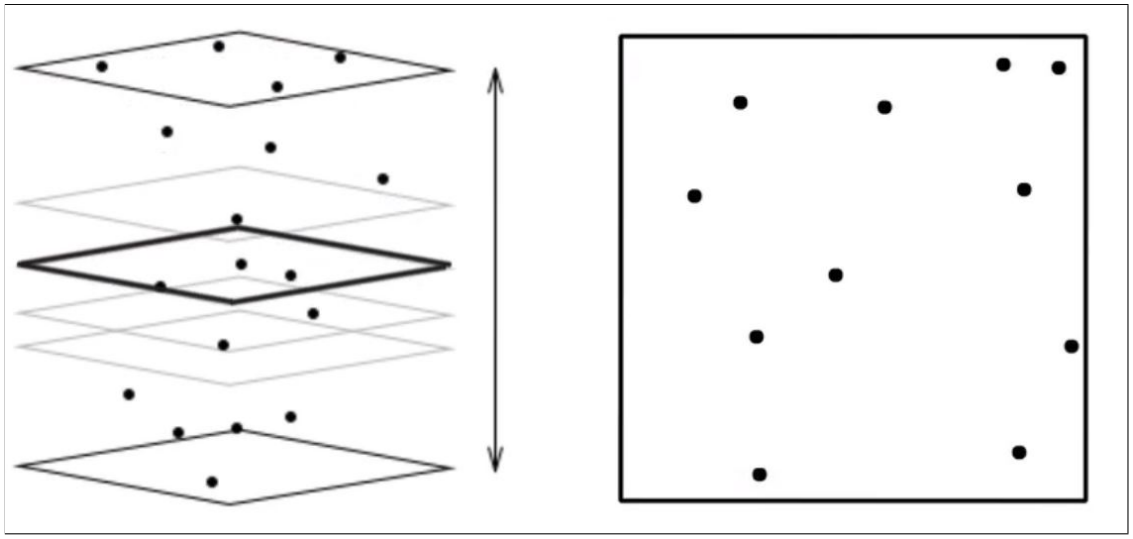


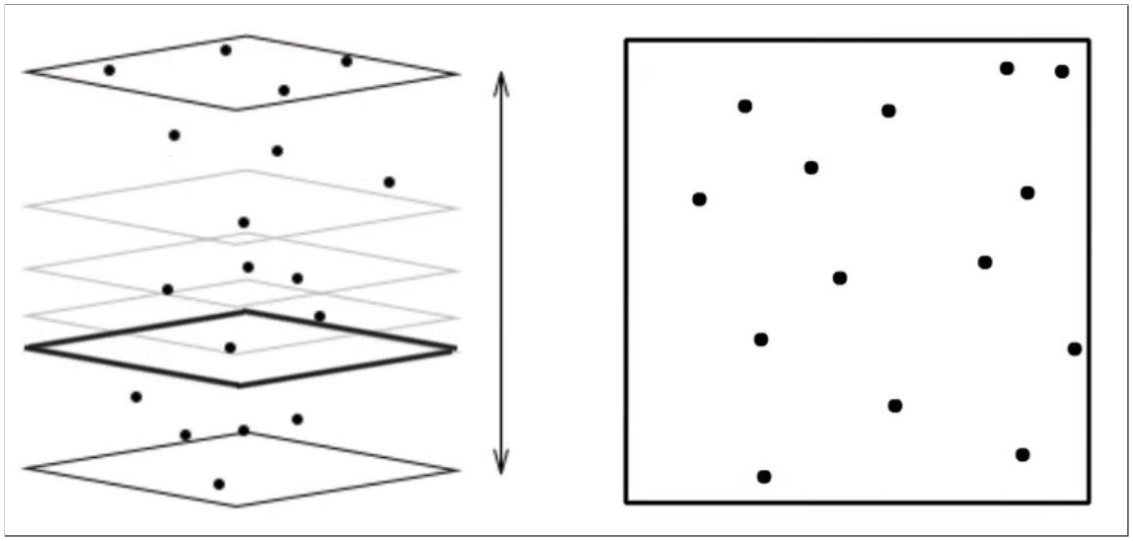


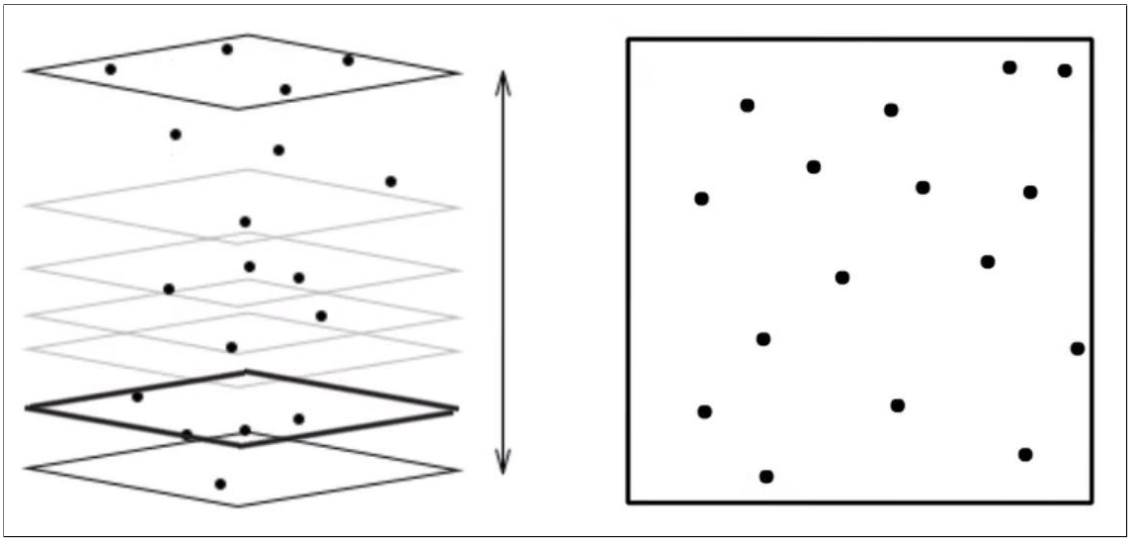
B. Vario scale top down visualization

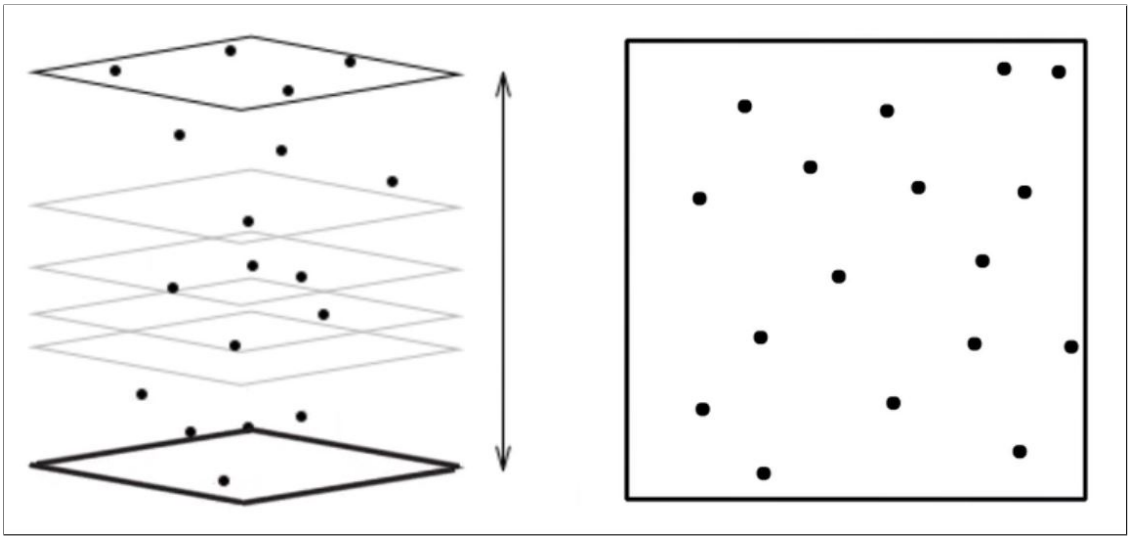












C. Config File Entwine

Algorithm 4 Entwine configuration file

```
{
    "input": "s3://province-zuid-holland",

    "output": "s3://jippe-test/greyhound/province-zuid-holland"
       ,

    "schema": [
        { "name": "X", "type": "floating", "size": 8 },
        { "name": "Y", "type": "floating", "size": 8 },
        { "name": "Z", "type": "floating", "size": 8 }
    ]

    "threads": 14

    "subset": { "id": 1, "of": 16 }

}
```

D. Config File Greyhound

Algorithm 5 Greyhound configuration file

```
1   {
2       "cacheSize": "1 GB",
3       "paths": ["s3://jippe-test/greyhound"],
4       "resourceTimeoutMinutes": 30,
5       "http": {
6           "port": 8080,
7           "headers": {
8               "Cache-Control": "public, max-age=300",
9               "Access-Control-Allow-Origin": "*",
10              "Access-Control-Allow-Methods": "GET,PUT,POST,
                    DELETE"
11          }
12      }
13  }
```

E. Results frame original - density

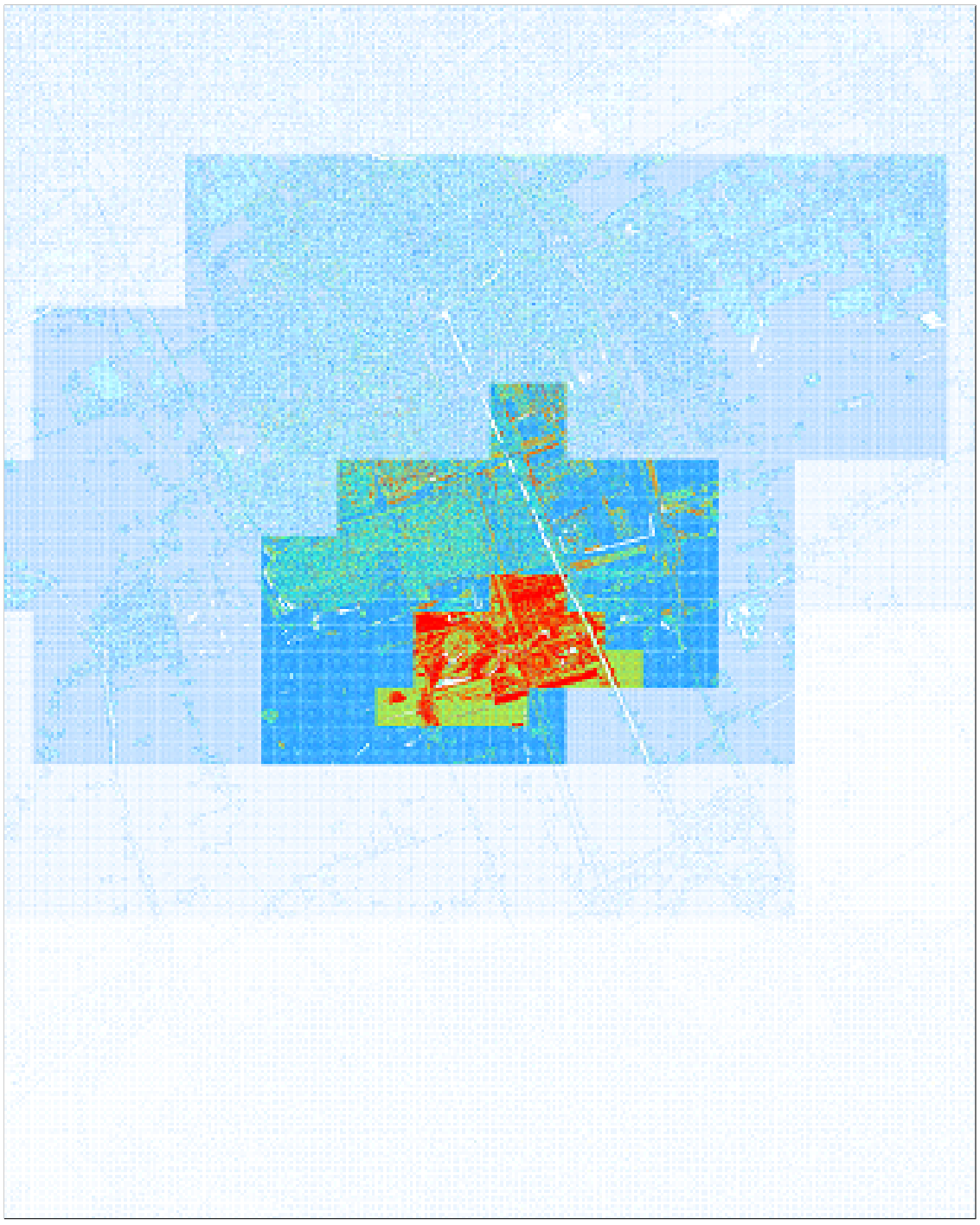


Figure 66: Original density of frame

F. Results frame original - perspective

Figure 67: Original perspective view of the frame