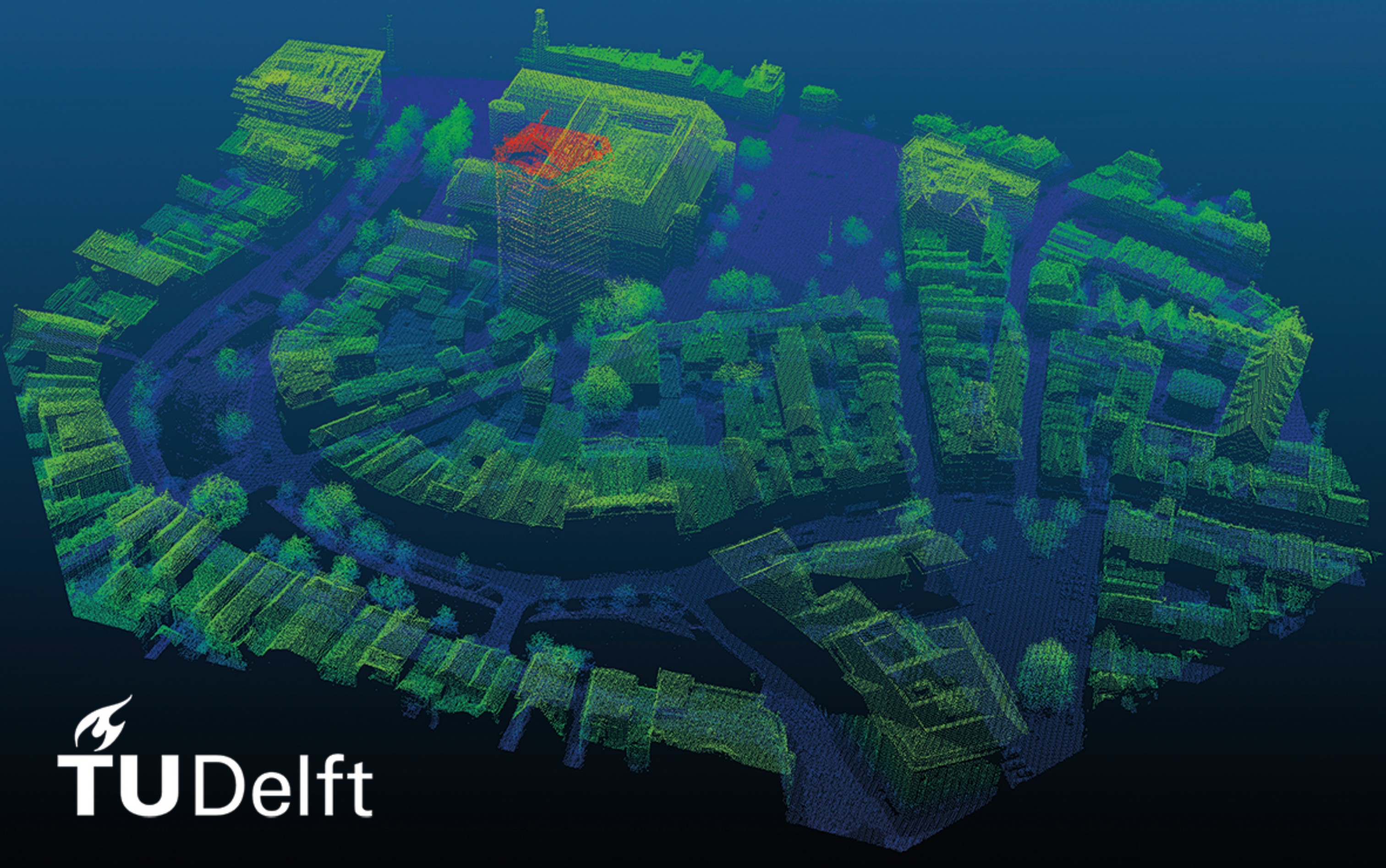
MSc thesis in Geomatics

SplitSFC: A database solution for massive point cloud data management

Yuduan Cai
2024



TU Delft

MSc thesis in Geomatics

SplitSFC: A database solution for massive point cloud data management

Yuduan Cai

January 2024

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

Yuduan Cai: *SplitSFC: A database solution for massive point cloud data management* (2024)
🅭🅯 This work is licensed under a Creative Commons Attribution 4.0 International License.
To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.

The work in this thesis was carried out in the:

Geo-Database Management Centre
Delft University of Technology

Supervisors:	Dr.ir. B.M. Meijers
	Prof.dr.ir. P.J.M. van Oosterom
Co-reader:	G.A.K. Arroyo Ohori

Abstract

Point cloud data are gaining more importance in the Geomatics domain, with the development of modern sensing technologies like LiDAR and photogrammetry. The size of massive point clouds are growing, and the performance of traditional database solutions for its data management become insufficient. It remains a huge challenge for researchers to come up with a data management solution that handles the huge volumes of data, while providing standardized functionalities (Van Oosterom et al. [2015]).

Space Filling Curve (SFC) has been explored as a good spaital access techniques for organizing point clouds. Existing SFC-based database solutions include PlainSFC (Van Oosterom et al. [2015]), HistSFC(Liu [2022]), etc. However, they use a flat table to store point records and it is not compact for massive point clouds. In this thesis, a SFC-based database solution that manages point cloud in blocks is proposed. The purpose is to improve the performance of current point cloud database solutions, especially with storage space. This model organizes the point clouds based on Space Filling Curve, and innovatively splits each SFC key to a head and a tile. The points with the same SFC head are placed in the same block. SFC tails and other property dimensions are stored as arrays in other columns.

Compared with the pgPointCloud and Oracle SDO_PC, the intermediate SplitSFC prototype does not show significant advantage in storage and data retrieval efficiency so far. However, it is fair to believe that with the improvement of algorithms and implementations, it has the potential to be an approximate and efficient point cloud data management solution.

Acknowledgements

Two and half years ago, I arrived in Delft, with the enthusiasm to dive in GIS and explore European cultures. As time goes by, Delft reveals its beauty. The duck family is walking on campus peacefully. The geese are swimming in the canal. The giant bird, Mr Czapliński, is standing elegantly in the graveyard. The river reflects the sunlight. People running. People walking. People cycling. The poetry, elegance and ease of Delft is composed everyday. Sincerely, a stream of profound happiness stems from the bottom of my heart. I realize it is another golden time in life.

My story in Delft is about Geomatics. The journey is full of obstacles, and there are countless struggling moments. However, the pain and the growth provide profound nutrition. The deeper joy comes from seeking knowledge. I am grateful to receive such a good education at TU Delft and be in such a fascinating and dynamic domain. From Geomatics, I step further towards my dream, combining the wisdom of geography and technology. Meanwhile, Geomatics connects me with my friends and my teachers. They are the flowers and fragrance of my life in Delft. I find my class united and lovely. We have close ties, help each other out and share our enthusiasm for Geomatics. I carefully freeze the memories in my mind. The teachers, on the other hand, guide us towards the palace of Geomatics and provide wonderful lessons. Among them, I want to give special thanks to Martijn Meijer and Peter van Oosterom.

The most challenging task for me at TU Delft is the master thesis. The hardest part is to manage myself. Unlike the first year, I was walking alone in the darkness during the thesis period. I took countless detours. Very often, I distracted myself with digital devices, social activities or anything reachable. I had trouble concentrating, although I like my research topic very much. Sometimes I feel ashamed to meet the supervisors because of little progress. Thankfully, Martijn and Peter are two lighthouses in the dark ocean. They are kind and patient every time, giving me a lot of encouragement and guidance. Now the research is coming to an end, I feel the sunlight is coming out again.

The Animal Farm, my beloved study group during the final year, is another key element of my university life. It consists of Arsenijs Nitijevskis the cat, Boris Berman the dog, Jarno Vegting the lion, Yehor Ruban the tree and me the tiger. We spend the final year studying together, playing together. We laugh out loud. We have become each other's teaser, precious friends, and role models.

There's more to thank, Mum and Dad, the clouds in the sky, the snow that melts in the garden, Yueyue and Xixi the border collies, Jumbo Supermarket, Xior Student Housing, my Philips vacuum cleaner, the rain and the raincoat…

Finally, I want to quote from Rabindranath Tagore: "I leave no trace of wings in the air, but I am glad I have had my flight."

Contents

List of Figures

List of Tables

1. Introduction

1.1. Problem Statement

A point cloud is a discrete set of points in 3D space, and is one of the most widely used formats for representing 3D shapes and objects. The most significant advantage is its simplicity and efficiency in representing 3D data. It stores the coordinates of individual points in space, rather than the complex geometry of surfaces and shapes, and may contain other attributes like intensity, classification, etc. This makes it easy to process and analyse large amounts of 3D data, such as from LiDAR sensors or 3D scans. Additionally, point cloud data can be easily visualised, shared, and integrated with other software and systems. With the developments in the point cloud acquisition technologies, like terrestrial and airborne laser scanning, mobile mapping, image matching and multi-beam echo-sound techniques etc., it becomes easier and easier to acquire data. Therefore, it is widely used in industries such as surveying, architecture, engineering, construction, and robotics for applications such as Building Information Modelling (BIM), 3D scanning, and autonomous navigation. The popularity is expected to continue to grow in the coming years as the technology becomes more widely available and the cost of data acquisition decreases.

The majority of the application of point cloud data is currently done at the file level, but an efficient database solution is still crucial. Database Management System (DBMS) has advantages in functionalities, optimised disk IO strategies and automatic parallelization in the query executions [Van Oosterom et al., 2015]. There are many commercial or open-source DBMSs available and suitable to manage spatial data, like Oracle, PostgreSQL, MonetDB, etc. Researchers and engineers have explored this field for many years. However, the database solutions are still not mature yet, and the specific approach should depend on the use case, the size and the structure of the data, and the type of queries frequently performed. The most significant challenge is the size of the point cloud data. Point clouds can have large volume, with billions of points. Traditional relational databases are not well suited for handling this amount of data. Other challenges include the need for fast querying and indexing, and the need to support multi-dimensional data. To improve the performance, common techniques include compression, sampling, indexing, partitioning, database optimization and so on. One appropriate approach is to use Space Filling Curve (SFC) to group and encode the points. Existing soluions include PlainSFC [Van Oosterom et al., 2015], HistSFC [Liu, 2022], etc., but they do not employ a block-based storage and are not compact for massive points.

In this research, we come up with an improved storage model to increase the efficiency of storage and manipulation of point clouds in databases. The approach is to split the SFC key in a head and tail part and make blocks of points inside the database based on the SFC head, resulting in a more compact storage. The effect on the manipulation of the data will also be investigated, like the efficiency and of retrieving points. The expected outcome includes a data model, a Python-based software for importing, querying and exporting, and a scientific benchmark.

1.2. Use cases

Point cloud, as a representation of three-dimensional objects or environments created by gathering a large number of points in 3D space, have numerous applications across various industries. The ease to collect data and the richness, accuracy and versatility of point cloud data contribute to its widespread use and growing popularity in various industries, supporting diverse applications and enhancing decision-making processes.

A storage solution of massive point clouds is crucial, because it is the foundation of almost all applications of point clouds. One major obstacle is to have a storage solution that is both compact for storage space and efficient for data retrieval. The goal of our research, is to come up such a solution and provide foundations for rich point cloud applications.

1.2.1. Data collection techniques

We can compare the data generation of 3D objects in CityGML and point clouds. CityGML uses geometry to describe the shape of each 3D object, while point clouds use the location of each point. The sensors, however, cannot directly distinguish the objects from each other. Therefore, the sensor can first scan the 3D city objects and output the data as point clouds. If the user wants CityGML to represent the landscape, the point clouds can be converted to CityGML format later on, with some advanced algorithms.

The ease to collect massive point cloud data has improved significantly thanks to the development of modern sensing technologies. Point clouds can be generated from various data sources, such as LiDAR (Light Detection and Ranging), photogrammetry, navigation system, laser scanning, or other 3D scanning technologies. The data collection technologies enable point clouds to have applications in diverse fields and for different purposes.

LiDAR (Light Detection and Ranging)

The data collection process of the LiDAR system is simple and convenient. The LiDAR system emits laser pulses towards the target area. The laser beams hit objects and return to the sensor, measuring the time taken for the round trip. This information, combined with GNSS (Global Navigation Satellite System) and IMU (Inertial Measurement Unit) data, helps determine the precise 3D location of points. In the end, a large set of points will be collected.

Photogrammetry

Photogrammetry is an image-based 3D reconstruction method. It uses a camera to capture multiple 2D images of an area from different angles. The algorithm analyzes the overlapping images to triangulate and create 3D point clouds by identifying common features and reconstructing the geometry based on visual information.

1.2.2. Applications

The most significant advantage of point clouds is its accurate and rich details. It can describe the shape and geometry of 3D objects in detail, and it can also include the details of texture, colour and intensity, providing a foundation for comprehensive analysis and visualization. For example, Virtual Reality (VR) and Video Gaming needs a highly detailed scene

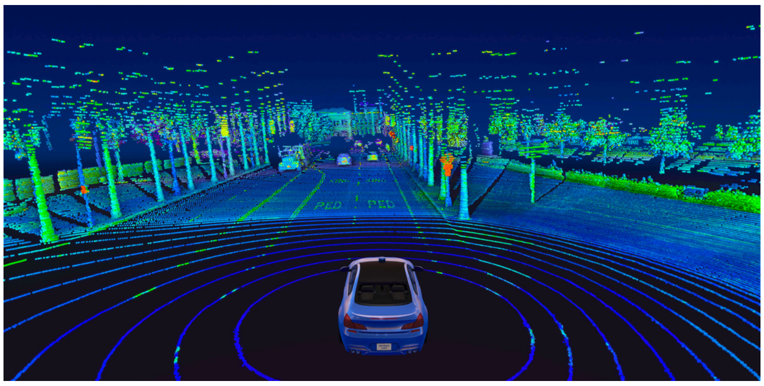


Figure 1.1.: Real-time 3D vision on the car using LiDAR system
Source: Velodyne Lidar technology, 2020

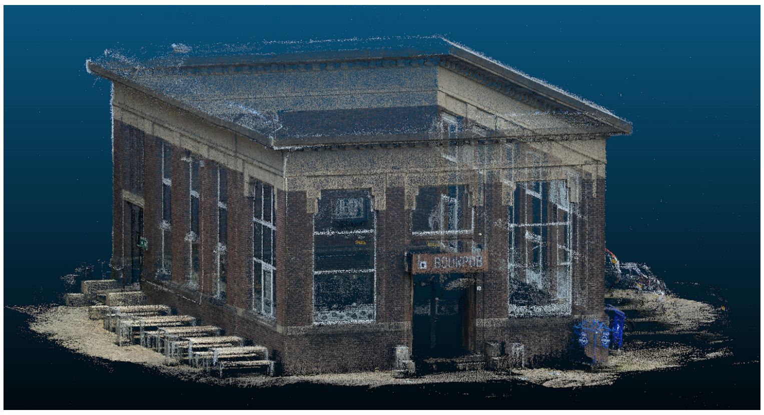


Figure 1.2.: Using photogrammetry technique to reconstruct a pub with 102 images

to enhance the visual quality and realism of virtual reality experiences. Point clouds are very suitable for this kind of application. Other possible applications include construction of architecture and built environments, asset management, environmental change detection, autonomous vehicle detection, and so on. In the Geomatics domain, point clouds can also be integrated with Building Information Modeling (BIM) and Geographic Information Systems (GIS), enhancing data interoperability and enabling more comprehensive modelling and analysis.

Digital Terrain Modelling (DTM)

A Digital Terrain Model (DTM) represents the bare ground of earth surface, removing any above-ground object like buildings and vegetation. In theory, point cloud is not considered valid representations of a terrain, because the rules to reconstruct the surface at unsampled locations are not defined. Still, a DTM is usually generated using point cloud. Commonly-used operations for point clouds that represent terrains include thinning, ground filtering, outlier detection, shape detection and so on.

The Actueel Hoogtebestand Nederland (AHN) dataset is the Dutch national elevation dataset. It is also the test dataset of our research. The AHN2 dataset contains 639 478 217 460 points.

There are many applications, such as urban planning, civil engineering and hydrological and flood risk Assessment. Take hydrological and flood risk assessment for example, the elevation and gradient of the ground level are used to determine flow direction of each grid, and the amount of water can be calculated in the area, so that the experts can predict the water level in the ditches and rivers, whether floodplains and ditches can be sufficiently drained and whether the dikes are still high and strong enough.

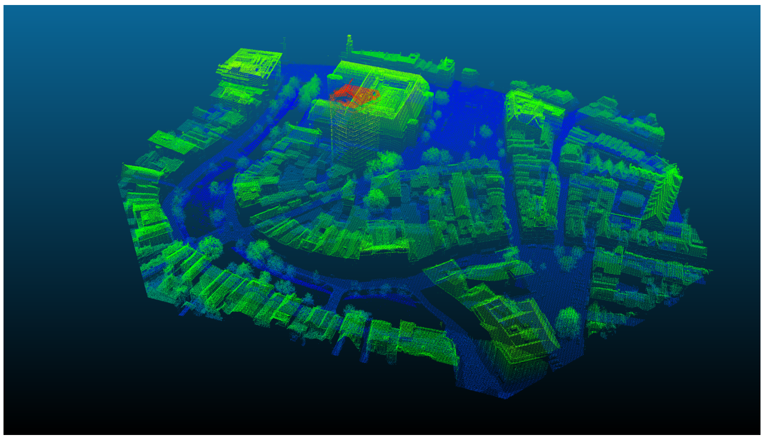


Figure 1.3.: The point clouds of a custom shape area from the AHN3
Source: rAHNextract, 2020

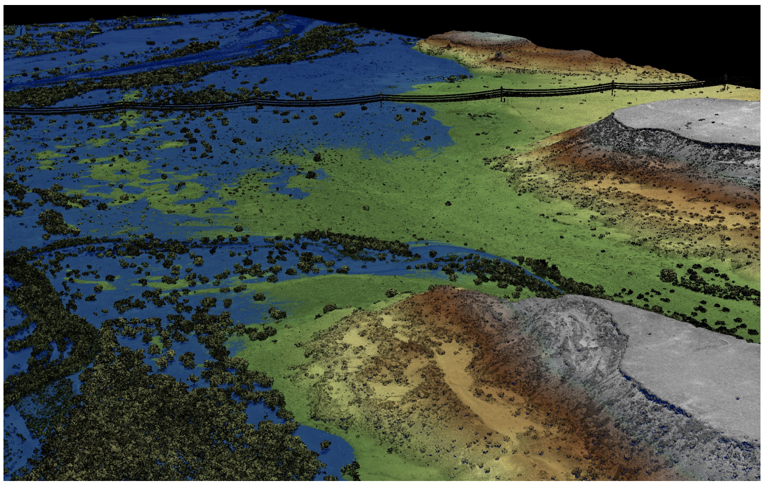


Figure 1.4.: Simulation of a flood event in New Mexico
Source: USGS

Online Point Cloud Service

Online point cloud service is easy and convenient for users, and can accommodate rich functions to meet various requirements, from visualizing and downloading point cloud data to implementing complex point cloud analysis.

Online point cloud service usually has a multi-layer and complex structure. Liu [2022] proposed a 5-layer architecture on the cloud computing platform (Figure 1.5), combining the inspiration from previous work [Richter and Döllner, 2014; Wang et al., 2018] and various use cases. Due to the nature of handling massive datasets, online massive point cloud services face several technical challenges. The performance is crucial, as users often require quick and seamless access to specific portions or segments of point cloud data. As the volume of point cloud data continues to increase, scalability becomes a critical factor. The data storage and retrieval are a core part of the architecture.

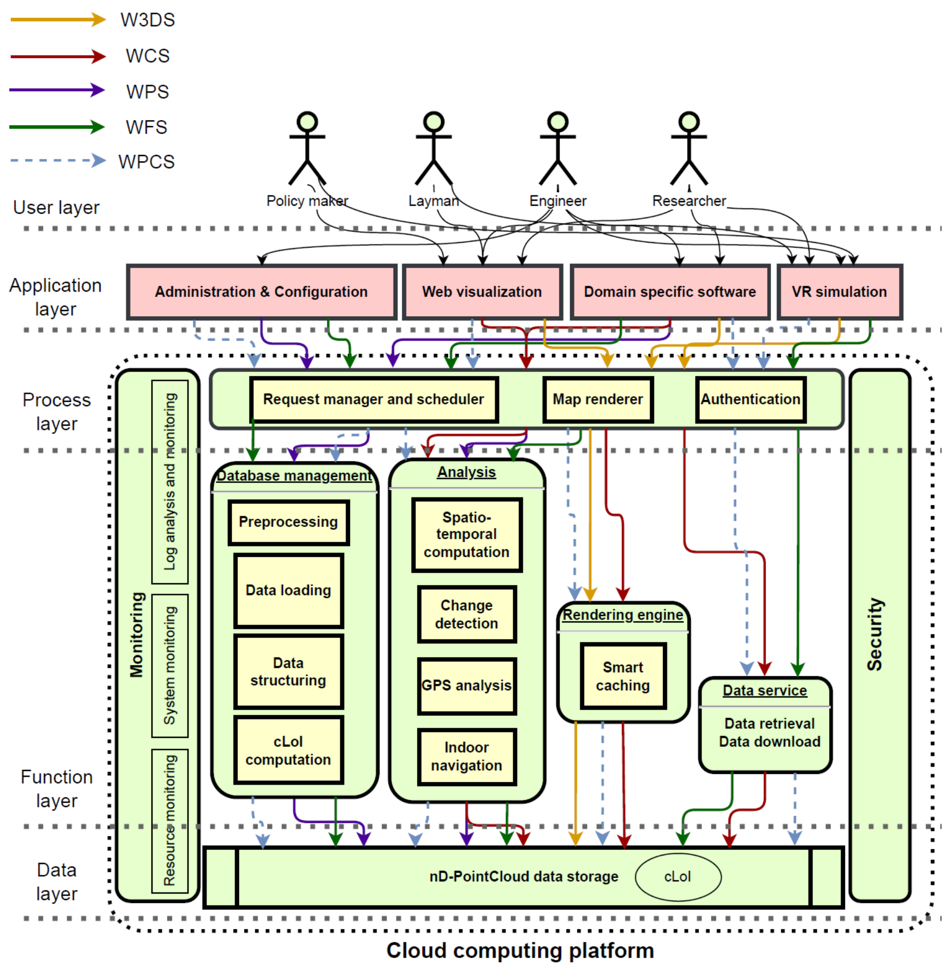


Figure 1.5.: A 5-layer design of architecture of point cloud computing platform
Source: Liu, nD-PointCloud Data Management, 2022

A good example is the AHN2 3D viewer and download tool [van Oosterom et al., 2017; Van der Maaden, 2019]. Previous AHN2 viewer only provide 2D visualizations for seletion, but it cannot provide rich details from 3D point cloud, because the large volume of data make it very demanding for the data storage, processing speed, and bandwidth and network constraints. To solve the problem, the storage of AHN2 3D viewer is based on an octree structure proposed by Schütz [2016], which took around 15 days with processing distributed in different machines and processes. Since the surface of the Netherlands is very flat, this octree structure of AHN2 is quite similar to a quadtree structure.

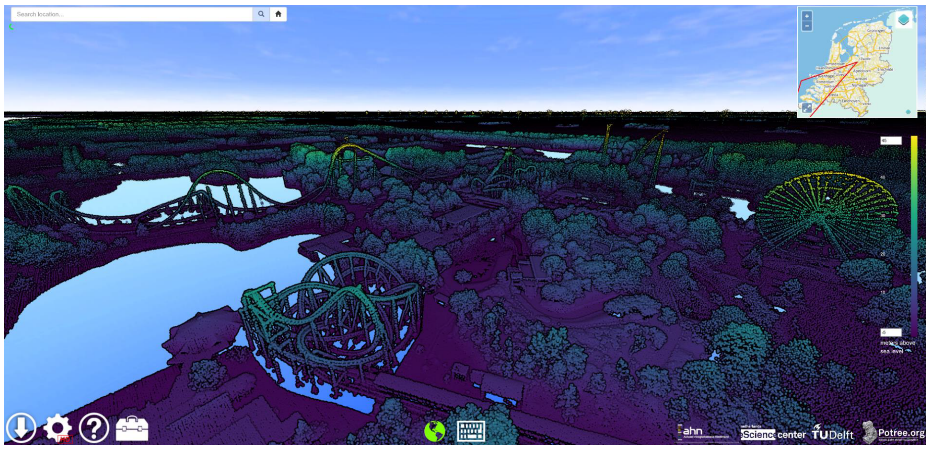


Figure 1.6.: A screenshot of AHN2 3D viewer

1.3. Research questions

1.3.1. Research questions

In the research of literature and existing methods, we find that one major issue of point cloud database solutions is the huge storage space because of the large volume of point cloud data. Some existing solutions have employed Space Filling Curves to improve data organization and querying performance. Noticing the characteristics of Space Filling Curve (SFC) keys, such as the relationship between Morton Curve and 2^n-tree structure, we can consider designing a new storage model called SplitSFC, where SFC keys are split into two parts, and the points with the same SFC head are stored in same block. We will study its effect in storage and operational efficiency. Therefore, the main research question is addressed as follows:

Does employing the split Space Filling Curve approach constitute an effective strategy for managing massive point clouds in a relational database?

In order to answer the main question, the following sub-questions are proposed:

- How to efficiently use SFC to organise the points in blocks?
- How to split the SFC keys, i.e. by fixed-length or adaptive algorithm?
- How to query the points based on the suggested method?
- How does the proposed method work compared to the current solutions?

1.3.2. Research scopes

The main purpose of the research is to propose an improved storage model to efficiently store and manage massive point clouds in the relational database. The methodology will focus on splitting the SFC keys and making point blocks based on it. Eventually, a scientific benchmark and comparison of different strategies will be made to validate the practicality of the new methodology.

On the other hand, the following aspects are not studied in this research:

1. The performance of various Space Filling Curves. We will only use the Morton curve in this research.
2. Various selections of organizing dimensions. We will only encode X and Y coordinates. Other dimensions like Z coordinate, time component, LoD and other attributes will not be encoded into SFC key.
3. Compression of property dimensions. The research focuses on encoding and compression of the organizing dimensions, notably X and Y coordinates.

1.4. Thesis outline

The outline of the thesis is structured as follows:

Chapter 1 introduces the research topic about the data management of point cloud, the potential use case of the study, the research questions and scopes and the thesis outline.

Chapter 2 explains the theoretical background of the research, including point cloud and its data management and the spatial access methods. The data management system includes file-based system and database management system. The most important spatial access method is the Space Filling Curve.

Chapter 3 summarizes the existing database solutions for point clouds, especially the SFC-based solutions. The state-of-the-art block-based storage models are pgPointCloud and Oracle SDO_PC, and the SFC-based solutons include PlainSFC, DynamicSFC and HistSFC.

Chapter 4 describes the methodology of SplitSFC approach, including the motivation for the design, storage model, the loading and querying procedure, and the benchmark framework. The main idea of the methodology is the splitting of the SFC keys and making blocks based on split SFC.

Chapter 5 details the used tools and the implementation, including hardware and software environment, datasets and the details of constructing the pipelines.

Chapter 6 describes and analyses the results of the benchmark. From the mini-benchmark, we gain more knowledge about the effects of parameters. From the medium benchmark, we evaluate the performance of the SplitSFC prototype and compare it with that of pgPointCloud and Oracle SDO_PC.

Chapter 7 draws the conclusions extracted from this benchmark, together with limitations and future work.

Finally, the appendices contain the details, including the descriptions of the tested queries, the specific data of the benchmark results and the usage of the prototype.

2. Theoretical background

This chapter aims to provide the relevant theoretical knowledge for the subjects that will follow in the rest of the thesis. With these introduced, the readers can understand the main problem and approaches in this field.

The chapter is organised as follows: Section 2.1 introduces what the point cloud is, and the term of point cloud data management system. Section 2.2 introduces Spatial Access Methods in the database, namely how to index and cluster the points. Section 2.3 introduces the most widely-used file system, including LAS / LAZ format and relevant softwares. Section 2.4 introduces the three types of model in point cloud DBMS solutions.

2.1. Point Cloud

A point cloud is a set of points in three-dimensional space. Point clouds are multi-dimensional data. Each point usually contains XYZ coordinates and other attributes, like intensity, classification, RGB colour, etc. The major sources of the point clouds include the Light Detection and Ranging (LiDAR) systems, photogrammetry, 3D scanning, navigation systems and conversion from other types.

Van Oosterom et al. [2015] proposed that point clouds should be recognized as a distinct form of spatial data representation, alongside vectors and rasters. While point clouds exhibit similarities with vector-based structures due to their point-based nature, they also share characteristics with raster data owing to their sampling nature. However, point clouds are irregularly scattered, and differ from individual points or multi-points in vectors. Point clouds are gaining more popularity and importance in the industry and academia. Hence, it is necessary to acknowledge them as the third spatial data type and build standardization for wide usage.

2.1.1. Dimensions and design aspects

The nature of the dimensions of the point cloud (Liu [2022]) is distinguished as follows:

- **Spatial dimensions (XYZ)**: It is the most basic dimension for point clouds, as point clouds are spatial data. XY or XYZ can be organized together in some spatial queries, but Z dimensions may be queried less compared to X and Y dimensions, as query is more often based on the footprint area, and XY indicates the location in the footprint while Z indicates the height. The spatial extent of the point cloud is often stored in the metadata of the point cloud.
- **Temporal dimension**: It is the acquisition time of the point, usually in GPS time. It is also a key dimension, especially in spatio-temporal applications. With temporal information, we can perform change detection with the point clouds.

- **LoD (Level of Detail)**: It delineates the level of intricacy within the 3D points. The Level of Detail (LoD) holds significant importance in balancing data precision, storage capacity, and computational demands. Certain systems or software platforms have the capability to adjust LoD according to either the user's preferences or the viewing distance. For instance, during zoomed-out views, a lower LoD could be presented to enhance efficiency, whereas zooming in prompts a transition to higher LoD to provide more intricate details.
- **Classification**: It shows the type of object that the point belongs to. It enables point cloud data to perform semantic analysis, e.g. the attributes of trees or buildings in a certain region. The classification codes of LAS files are 0-never classified, 1-unclassified, 2-ground, 6-building, 9-water, etc.
- **Other property dimensions**: Other dimensions are relatively less important than other dimensions, and they may be not used in data organization or indexing, but they may still be useful in some specific tasks. For example, RGB colour is useful in visualization, and intensity can be used in 3D object detection and segmentaton.

Apart from the nature of the dimensions above, Van Oosterom et al. [2015] present more relevant aspects to consider during the design phase of massive point cloud data management systems:

- **Spatial Access Method**: It involves organizing points effectively, utilizing strategies such as efficient blocking, clustering (employing Space Filling Curves), and various indexing techniques within multidimensional space. A significant factor impacting the clustering facet involves selecting dimensions for the calculation of clustering keys, such as utilizing 2D (X, Y) or 3D (X, Y, Z) configurations.
- **Compression technique**: It plays a vital role in reducing storage space and facilitating dissemination through wireless networks.
- **Operation**: It encompasses tasks such as loading, selecting, and complex algorithms on point clouds, such as computing normal vectors, nearest neighbor search, and more.
- **Parallel processing technique**: It can maximize the computational capabilities of computers. Parallel processing should in general have a better performance than single-process systems, enabling more efficient utilization of computing resources.

2.1.2. Point Cloud Data Management System (PCDMS)

Point Cloud Data Management System (PCDMS)[Van Oosterom et al., 2015] refers to a system that stores, organizes, processes and visualizes large volumes of 3D point clouds. It can be categoried into file-based systems, database management systems, and hybrid systems. The key challenge of PCDMS is to handle massive data and offer standardized functionalities at the same time [Van Oosterom et al., 2015].

File-based systems use a file or a set of files to store the point cloud data, and rely on softwares to edit, process, analyse and visualize them. Regardless of the format, a point cloud file can often be seen as an array of point records, each of which contains the spatial coordinates and potentially other dimensions. The file formats include ASCII format (plain text file), PLY format (a standardised ASCII format) and LAS format. File-based systems are flexible and efficient, and functionalities can be supported by the softwares. Therefore, currently most applications are based on file-based systems. However, it lacks native functionality support, and the data retrieval has to go over many files, which is not convenient. The LAS format and corresponding softwares are described in Section 2.3.

The Database Management System, on the other hand, manages the storage and the retrieval both based on the DBMS itself. Softwares or programming languages are usually used for complex algorithms only. For example, Structured Query Language (SQL) can be used to query and update the data. The storage and querying can be more efficient with native support, like parallel process techniques. Therefore, it is of high practical value to perform research on the DBMS solutions for massive point clouds. The existing database solutions are summarized in Section 3.

2.2. Spatial Access Methods

Spatial Access Methods are crucial in Point Cloud Database Management Systems (PCDBMS) for efficient spatial operations. It has two aspects, spatial indexing and spatial clustering [van Oosterom, 1999]. Point clouds are usually large and distributed in an irregular manner, and the Spatial Access Method can speed up the spatial queries significantly.

Spatial indexing enables the system to know storage location fast when searching for certain objects. Otherwise, a 'full table scan' has to be done in a relational database, which means that every record in the database has to be checked whether it meets the spatial selection criterion. And Spatial Clustering stores neighbouring points physically together.

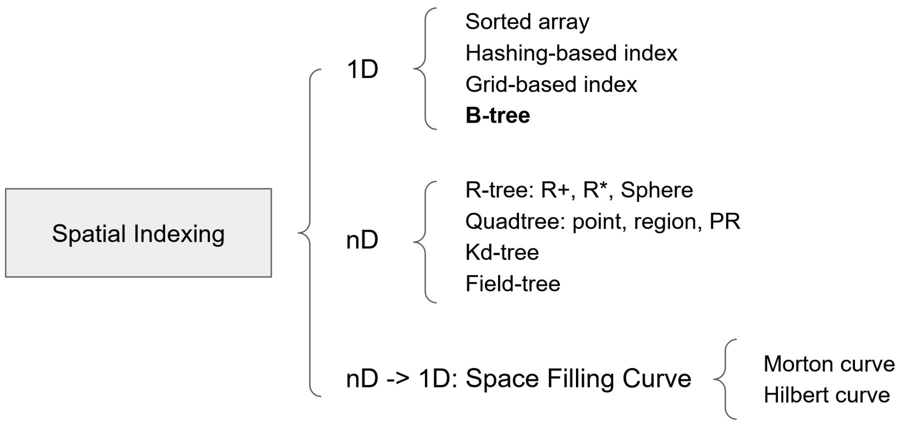


Figure 2.1.: Common spatial indexing methods

2.2.1. Importance of spatial indexing

Spatial Indexing is a key technique in the Point Cloud DBMS solutions. L Li et al. [2023] summarized the two factors that have determined the spatial indexing techniques that have been proposed.

First, the index can filter and exclude a large number of spatial targets unrelated to the spatial operation, and improve the efficiency of spatial query. It enables the system to know storage location fast when searching for certain objects. Without a spatial index, a 'full table scan' has to be done in a relational database. As spatial data sets are usually very large, this is unacceptable in practice for most applications. Therefore, a spatial index is required, which enables the system to find the required object addresses efficiently without looking at every object.

The second is that the multidimensional nature of the point cloud data makes traditional database indexing techniques unsuitable. Point clouds have the following properties:

1. Point cloud has spatial dimensions and property dimensions, and the spatial dimensions have a large amount of data;
2. The spatial operations of point clouds (such as spatial selection) require special calculations and it costs a large amount of calculations;
3. Point cloud data is multi-dimensional and needs to be specifically defined with a reasonable order, making it impossible to apply common sorting techniques.

Therefore, a suitable spatial indexing method is vital in the database solution design. In the storage process, the spatial indexing technique maps multi-dimensional data to a linear space and has a clustering effect of storing spatially adjacent and query-related points physically together. In the querying process, it is a bridge connecting spatial query operations and point cloud objects, and improves query speed significantly by eliminating a large number of irrelevant targets.

Commonly used spatial indexes include grid index, B-tree index, R-tree index, Quadtree index, Space Filling Curve, etc. They are introduced in the following sections.

2.2.2. One-dimensional indexing

Binary Search Tree

The Binary Search Tree (BST) is a rooted binary tree data structure with the key of each internal node being greater than all the keys in the respective node's left sub-tree and less than the ones in its right sub-tree. This property makes it possible to efficiently search, insert, and remove items in the tree. However, it is impractical for massive data due to poor performance and high memory usage.

BST was discovered independently by several researchers. The algorithm is attributed to Conway Berners-Lee and David Wheeler for the problem of storing labeled data in magnetic tapes [Windley, 1960].

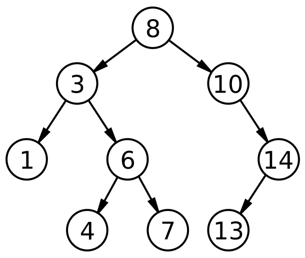


Figure 2.2.: A Binary Search Tree of size 9 and depth 3
Source: Wikipedia

B-tree and its variants

B-tree [Bayer and McCreight, 1970], short for Balanced Tree, generalizes the Binary Search Tree and allows for nodes with more than two children. It is a self-balancing tree where all the leaf nodes are at the same level which allows for efficient searching, insertion and deletion of records.

B-tree has a guaranteed time complexity of O(log n) for basic operations like insertion, deletion, and searching, which makes it suitable for large data sets and real-time applications. However, B-tree can have a high disk usage, and it does not suit all cases.

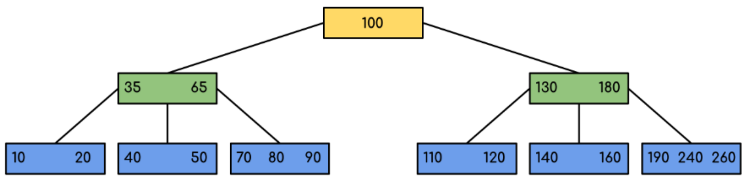


Figure 2.3.: A B-tree of depth 3
Source: Introduction of B-Tree, Geeks for Geeks

Apart from B-tree, sorted arrays and hash-based indexing can also be used to organize one-dimensional data. For example, in our proposed data model, B-tree index is created in SFC heads, and SFC tiles are organized as sorted arrays. Notably, these 1D indexing methods can also be applied to multi-dimensional data if they are mapped to one-dimensional, with the help of methods like Space Filling Curve. This topic will be further introduced in Section 2.2.4.

2.2.3. Multi-dimensional indexing

Multi-dimensional indexing poses more challenges due to the increase in complexity. In high-dimensional spaces, data tends to become sparser, meaning that the available data points are located farther apart from each other. This sparsity negatively impacts the effectiveness of traditional indexing methods.

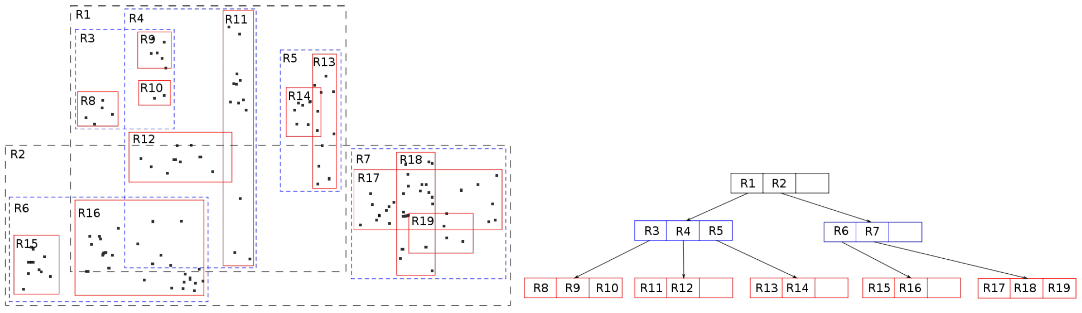
Multi-dimensional indexing mainly consists of hashing and the hierarchical methods [Gaede and Günther, 1998]. In this section, we introduce two hierarchical indexing methods, R-tree and 2^n-tree, and their variants.

R-tree and its variants

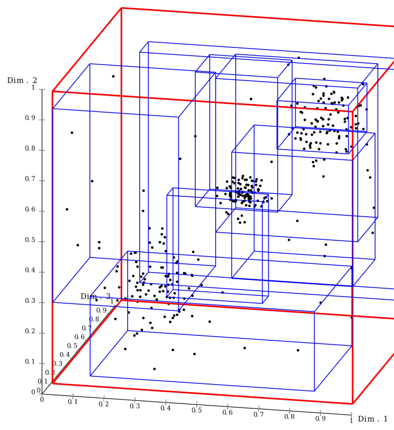
R-tree [Guttman, 1984] is the one of the most widely used spatial indexing structures. The 'R' in R-tree refers to rectangle. Its fundamental design revolves around the organization of spatial objects within a balanced tree structure. Each node in the R-tree encompasses a bounding rectangle that encapsulates spatial objects or child nodes.

The root node of R-tree encompasses all the spatial objects in the dataset. Intermediate nodes represent bounding rectangles that enclose their child nodes' spatial extents, and leaf nodes contain references to the actual spatial objects or further node levels. R-trees dynamically adjust their structure by splitting nodes or merging nodes to maintain balance and optimize query performance. These adjustments occur during insertions or deletions to ensure efficient search operations.

R-trees efficiently handle spatial queries, such as range searches, nearest neighbor searches, and intersection queries. They exploit the hierarchical nature to prune irrelevant branches of the tree during searches, reducing the search space and improving query performance. R-trees have many variations, such as R*-tree. R+-tree, Sort-Tile-Recursive-tree, and Hilbert R-tree. R*-tree employs more sophisticated node splitting and reinsertion techniques compared to the traditional R-tree, aiming to enhance the tree's balance and reduce overlap among bounding rectangles. And R+-tree reduces overlap among bounding rectangles and refines the splitting strategies in order to minimize unnecessary space overlap between rectangles while maintaining efficient query performance.



(a) 2D

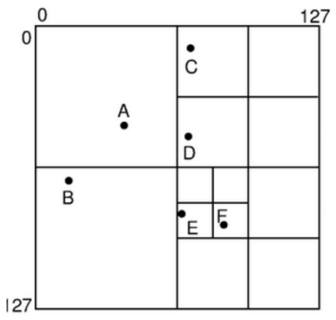


(b) 3D

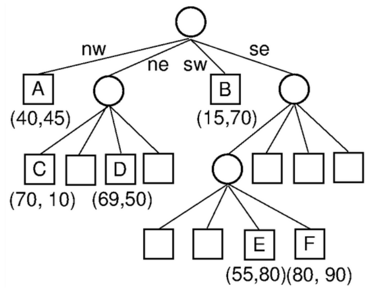
Figure 2.4.: R-trees for point clouds
Source: Wikipedia

2^n-tree and its variants

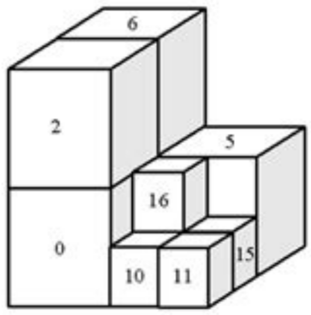
2^n-tree is based on recursive decomposition of multi-dimensional space into 2^n equal sub-regions [Finkel and Bentley, 1974]. It starts with a root node representing the entire area, which is then recursively divided into 2^n equal sub-regions, called child nodes. Each sub-region is represented by 2^n children in a 2^n-tree, leading to a tree-like structure where each node either has 2^n children or is a leaf node (a node with no children). The 2^n-tree is called Quadtree in 2D space and Octree in 3D space.



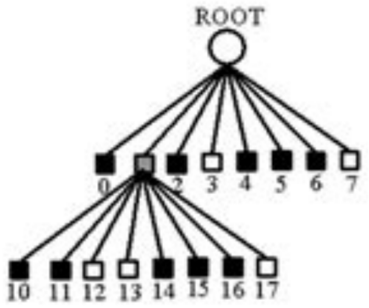
(a) Space division of Quadtree



(b) Tree structure of Quadtree



(c) Space division of Octree



(d) Tree structure of Octree

Figure 2.5.: A Quadtree and a Octree structure
Source: OpenDSA & Sung et al. [2001]

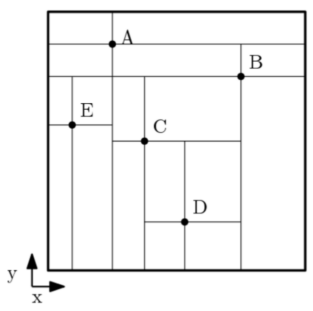
The Quadtree family encompasses variations like Point Quadtree, which stores points at the leaf nodes, and Region Quadtree, which stores information about regions in the leaf nodes. Point-region (PR) Quadtree [Orenstein, 1982] stores a list of points that exist within a region in the leaf nodes. These variations allow for flexibility in handling different types of spatial data efficiently.

Morton curve is a type of Space Filling Curve and it can represent the materialized path in 2^n-tree. This property will be introduced in Section 2.2.4.

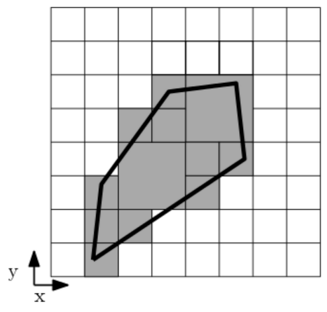
2.2.4. Space Filling Curves

Space filling curve is a continuous, self-repeating curve that traverses through every point in a given space. It was initially proposed in the 1890s. From a mathematical point of view, the space filling curve can be regarded as a method of mapping a multidimensional data space into a one-dimensional data space. As a curve, the space filling curve passes through each discrete grid in the multi-dimensional space, and only once through each grid cell, and finally the grids of the multi-dimensional space are numbered uniformly in a linear order.

There is no natural ordering in the multi-dimensional data space in the first place. The fact that storage disks are one-dimensional storage devices complicates the problem of storing multi-dimensional data in a one-dimensional space. Therefore, a mapping function from a



(a) Point Quadtree



(b) Region Quadtree

Figure 2.6.: A Quadtree and a Octree structure
Source: Psomadaki [2016]

multi-dimensional space to a one-dimensional space is needed. This mapping function is distance-invariant, so that adjacent elements in the space are mapped to points on a straight line and correspond one to one. Among all space filling curves, Morton curve (Z-order curve or Peano curve) [Morton, 1966] and Hilbert curve [Hilbert, 1935] are most widely used with spatial data.

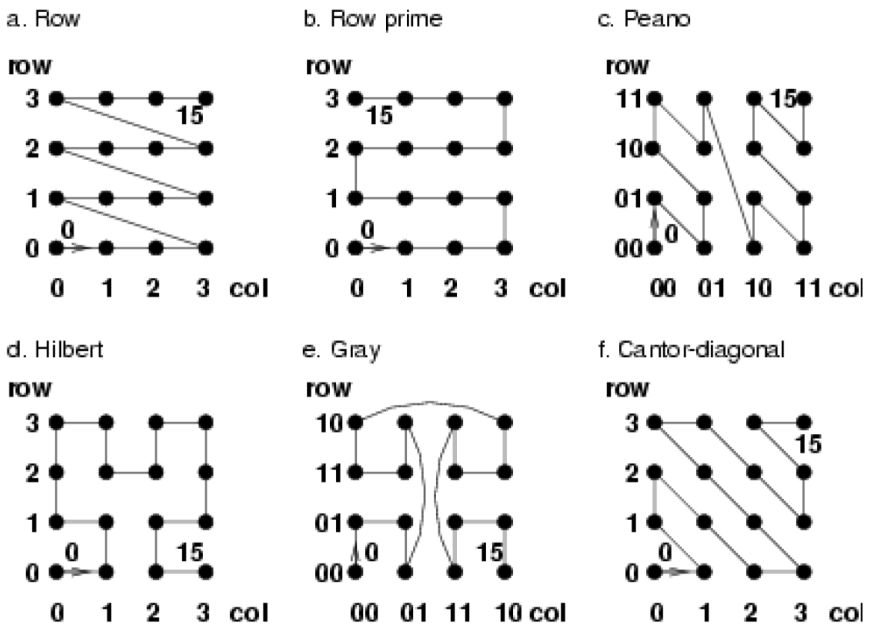


Figure 2.7.: Common Spatial Filling Curve
Source: van Oosterom [1999]

Hilbert Curve

The Hilbert curve is named after mathematician David Hilbert. The Hilbert curve exhibits a structured, fractal-like shape. It is constructed by recursively dividing a space-filling square or hypercube into smaller squares or hypercubes and then connecting these smaller elements in a specific order to form a curve.

The Hilbert curve is good at locality preservation. Nearby points in multidimensional space tend to be close to each other along the curve. This property makes it useful for applications where preserving the spatial relationship between data points is essential.

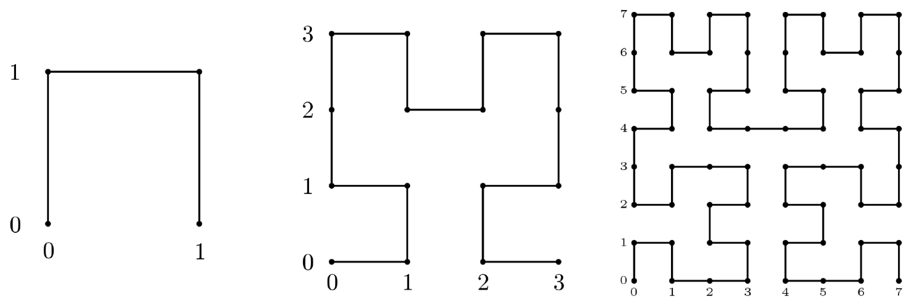


Figure 2.8.: The Hilbert Curve for 2, 4 and 8 bit representation
Source: Psomadaki [2016]

Morton Curve

The Morton curve, named after its creator, G. Morton, also referred to as the Z-order curve due to its zigzagging pattern, is a method used for interleaving the bits of multiple dimensions to create a one-dimensional curve. This curve maintains spatial proximity among adjacent points in multiple dimensions by combining the coordinates of points along different axes into a single value, where nearby points in space tend to have nearby values on the curve. However, it results in 'jumps' in the curve. In other words, two consecutive keys may not always be neighbouring to each other.

The Morton curve works by partitioning the space into smaller squares or hypercubes and then recursively subdividing these squares or hypercubes into smaller regions. This process generates a curve that visits each subdivision in a specific order, ensuring that nearby points in space are also close along the curve.

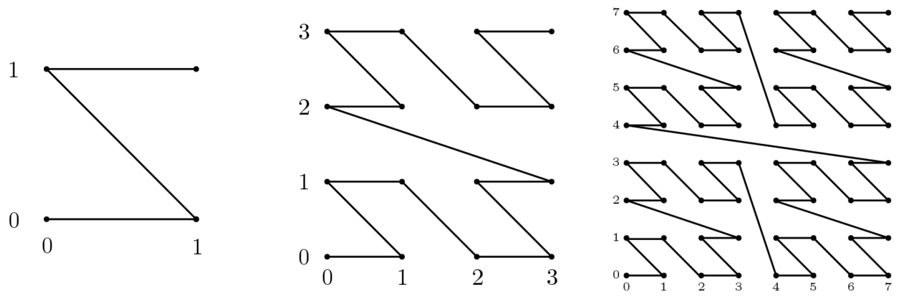


Figure 2.9.: The Morton curve for 2, 4 and 8 bit representation
Source: Schütz [2016]

If we compare Morton Curve and Hilbert Curve, we can tell that Morton Curve is easier to implement, and Hilbert curve excels in explicitly preserving spatial locality. The choice between Morton and Hilbert curves often depends on the trade-offs between simplicity, locality preservation, and the specific needs of the application at hand.

Morton-Quadtree hierachy

The Morton Curve has a strong connection with the 2^n-tree [Van Oosterom et al., 2015]. It can represent the hierarchical structure of 2^n-tree. Figure 2.10 illustrates the node and the range in 2D. All points have integer coordinates. By truncating the last n bits of the Morton codes of the points recursively, Morton codes at upper levels are derived. That is to say, the Morton codes of points implicitly have a hierarchy which is equivalent to a Quadtree structure. A branch node covers the nodes on the level below, and represents the spatial extent of a quadrant. Thus, the branch node also indicates a range of Morton codes starting from the lower-left corner to the upper-right. The property is extensible to higher dimensions because of the Quadrant recursive properties of space filling curves.

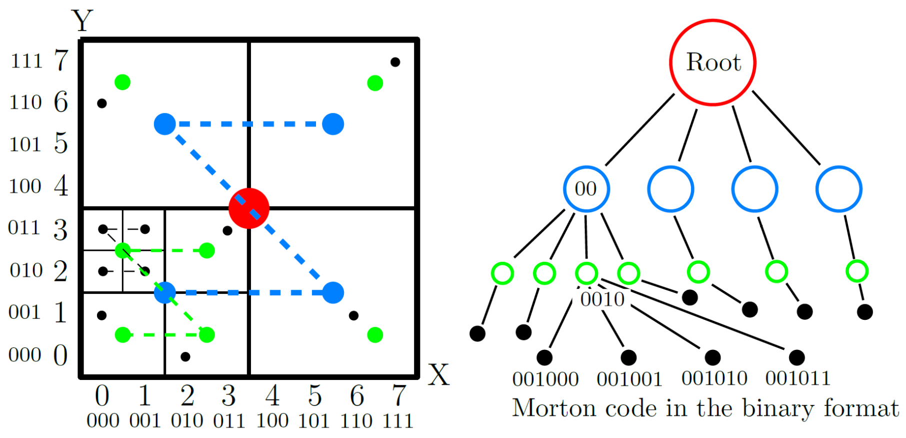


Figure 2.10.: The hierarchical structure of Morton Curve and Quadtree
Source: Haicheng Liu [2020]

2.3. Point Cloud File-based System

The majority of point cloud data management is currently done at the file level. Regardless of the format, a point cloud file can often be organized as an array of point records, each of which contains the coordinates and attributes of one point. The common file formats include ASCII plain text files, PLY (Polygon File Format) format and the LAS format. In this section, only LAS / LAZ format and related tools will be introduced.

2.3.1. LAS / LAZ format

The LAS (LiDAR Data Exchange Standard) format, currently at version 1.4, is the most widely used standard for point cloud data. As the name implies, it was designed for datasets that originate from LiDAR scanners. The American Society for Photogrammetry and Remote Sensing (ASPRS) is responsible for the maintenance of this data standard. The schema of LAS format [ISPRS, 2019] contains three main parts:

- **Header** contains information about the data such as its version, the point format (which tells the different dimensions stored for each point) and other metadata.

- **Variable Length Records (VLRs)** store additional information such as the SRS (Spatial Reference System), description on extra dimensions added to the points.
- **Point Records** contain the point records, and the dimensions may include X, Y, Z, intensity, return number, classification, GPS time, etc.

LAZ is a lossless compression of LAS format [Isenburg, 2013]. The schema is shown in Figure 2.11. It keeps the header and VLRs directly from LAS, but the point record part is stored as blocks with scaling and offsetting techniques for compression. Because of lossless compression and built-in support for LAZ in point cloud processing softwares, the trade-off between LAS and LAZ lies in the storage size and the processing time, as the compression and decompression operations on LAZ files do take extra time.

name of atomic item		**size**	point type and size					
			0	**1**	**2**	**3**	**4**	**5**
point attributes	format	size	20	28	26	34	57	63
POINT10		**20 bytes**	x	x	x	x	x	x
X	int	4 bytes	x	x	x	x	x	x
Y	int	4 bytes	x	x	x	x	x	x
Z	int	4 bytes	x	x	x	x	x	x
Intensity	u_short	2 bytes	x	x	x	x	x	x
Return Number	3 bits	3 bits	x	x	x	x	x	x
Number of Returns of Pulse	3 bits	3 bits	x	x	x	x	x	x
Scan Direction Flag	1 bit	1 bit	x	x	x	x	x	x
Edge of Flight Line	1 bit	1 bit	x	x	x	x	x	x
Classification	u_char	1 byte	x	x	x	x	x	x
Scan Angle Rank	u_char	1 byte	x	x	x	x	x	x
User Data	u_char	1 byte	x	x	x	x	x	x
Point Source ID	u_short	2 bytes	x	x	x	x	x	x
GPSTIME10		**8 bytes**		x		x	x	x
GPS Time	double	8 bytes		x		x	x	x
RGB12		**6 bytes**			x	x		x
Red	u_short	2 bytes			x	x		x
Green	u_short	2 bytes			x	x		x
Blue	u_short	2 bytes			x	x		x
WAVEPACKET13		**29 bytes**					x	x
Wave Packet Descriptor Index	u_char	1 byte					x	x
Bytes Offset to Waveform Data	u_int64	8 bytes					x	x
Waveform Packet Size in Bytes	u_int	4 bytes					x	x
Return Point Waveform Location	float	4 bytes					x	x
X(t)	float	4 bytes					x	x
Y(t)	float	4 bytes					x	x
Z(t)	float	4 bytes					x	x

Figure 2.11.: The schema of LAZ format
Source: Isenburg [2013]

2.3.2. Softwares

Softwares and programming libraries are essential to process, analysis and visualize file-based point clouds. The most popular point cloud processing software, LASTools, was developed by Martin Isenburg around the early 2000s. According to their official website, it provides a set of command-line tools that can efficiently classify, tile, convert, compress, filter, raster, triangulate, contour, clip and polygonize LiDAR data.

Laspy and PDAL are a Python and a C++ library for translating and manipulating point cloud data respectively. PDAL also includes an abstraction layer for solutions using LAS files, Oracle and PostgreSQL.

2.4. Point Cloud DBMS

The ongoing discourse surrounding the applicability of Database Management Systems (DBMS) in managing point cloud data remains a topic of continuous investigation [Liu, 2022]. While file-based solutions maintain popularity among users, database solutions offer distinct advantages such as enhanced functionalities, optimized disk IO strategies, and automated parallelization during query executions [Van Oosterom et al., 2015].

However, the development of a DBMS solution is consistently confronted with challenges. Among these challenges are the management of huge data volumes, high dimensionality of the data, and the imperative need for efficient querying and indexing mechanisms tailored to point cloud datasets. Furthermore, researchers and developers keep on seeking improved solutions, indexing structures, and query languages for point cloud data.

Within point cloud DBMS solutions, three types of storage model can be distinguished [Van Oosterom et al., 2015; Psomadaki, 2016]:

- **Flat table model**: Points are directly stored in a table, one row per point. It is simple and flexible, but it usually occupies a lot of storage space. Therefore, it is more suitable for small point clouds.
- **Block-based model**: Blocks of points are stored in the table, one row per block. This storage model is compact and has better scalability, thus suitable for massive point clouds.
- **Hierarchical model**: Points are stored in a tree structure.

3. Existing database solutions

The state-of-the-art database solutions for point cloud are introduced in this chapter. The storage models can be categorized into flat table model (Section 3.1), block-based model (Section 3.2) and hierarchical model. The SFC-based model has been studied in the past ten years and will be introduced in Section 3.3.

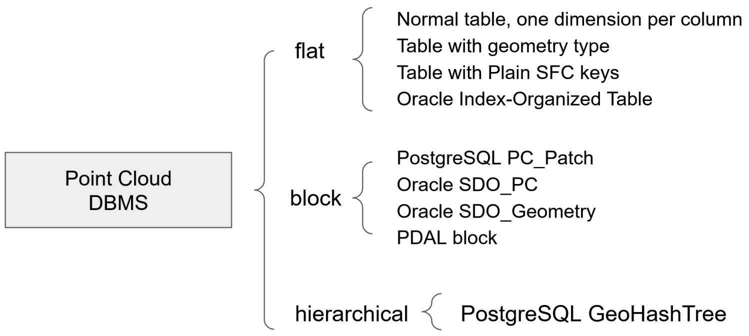


Figure 3.1.: The category of database solutions for point clouds

3.1. Flat table model

In a flat table, each record stores a single point. The possible solutions are:

Normal table with numberal data type and B-tree index

Each dimension stores an individual column, and B-tree indexes can be created on frequently searched columns. However, as spatial querying is usually multi-dimensional,this solution is very inefficient for spatial data retrieval and analysis.

Table with geometry data type and R-tree index

The spatial dimensions XYZ are stored as a geometry type and the R-tree index can be created on the geometry column. Other attributes are stored in individual columns. This data organization supports more spatial functions and speeds up spatial queries significantly. The disadvantage is that the R-tree index may occupy large storage space, and the point geometry type can support at most three dimensions in Oracle and four dimensions in PostgreSQL.

Table with 1D key and B-tree index

It is also called the PlainSFC approach (Van Oosterom et al. [2015]). The organizing dimensions are stored not by their original values, but their 1D key value (e.g. SFC key) , and this column is set as the primary key. The property dimensions are attached to the key.

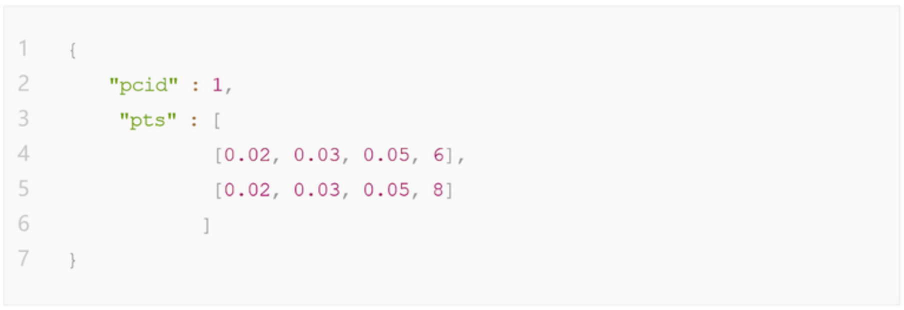
In Oracle, this can be relatively easier to implement with Index-Organized Table (IOT), which is a specialized type of table that stores data primarily within an index structure. Unlike traditional heap-organized tables where data is stored in an unsorted manner and indexes are separate structures, an IOT organizes the data and stores it directly within the B-tree index structure.

3.2. Block-based model

The block-based models in Oracle and PostgreSQL differ and are based on different data types.

3.2.1. pgPointCloud

pgPointCloud (Ramsey, 2013) is a PostgreSQL extension for storing point cloud (LIDAR) data. It is efficient and robust. It has two point cloud data types, PC_Point and PC_Patch. The former one is for a flat table, and the later one stores blocks of points. A PC_Patch record stores the ID of the point block and a list of points with all the dimensions (Figure 3.2). The records are stored in a TOAST (The Oversized-Attribute Storage Technique) table. Just like PostGIS extension, pgPointCloud also provides functions to query and manipulate the points.



```
{
    "pcid" : 1,
     "pts" : [
                [0.02, 0.03, 0.05, 6],
                [0.02, 0.03, 0.05, 8]
               ]
}
```

Figure 3.2.: An example of PC_PATCH schema

3.2.2. Oracle SDC_PC

The block-based model in Oracle usually stores point blocks as Binary Large OBject (BLOB) types in a table. The most widely used approach is based on SDC_PC data type(Oracle, 2019). There are two tables in the schema. One is the Base PC Table. It contains SDO_PC objects (Oracle, 2019) and their identifiers. The other table is the Block Table. Its columns contain SDO_PC_BLK object (point blocks), block ID, the number of points in the block, etc. A R-tree or a Hilbert-R tree can be built as an index. However, the organization of blocks only supports XY dimensions, and all other dimensions can only be stored as property dimensions.

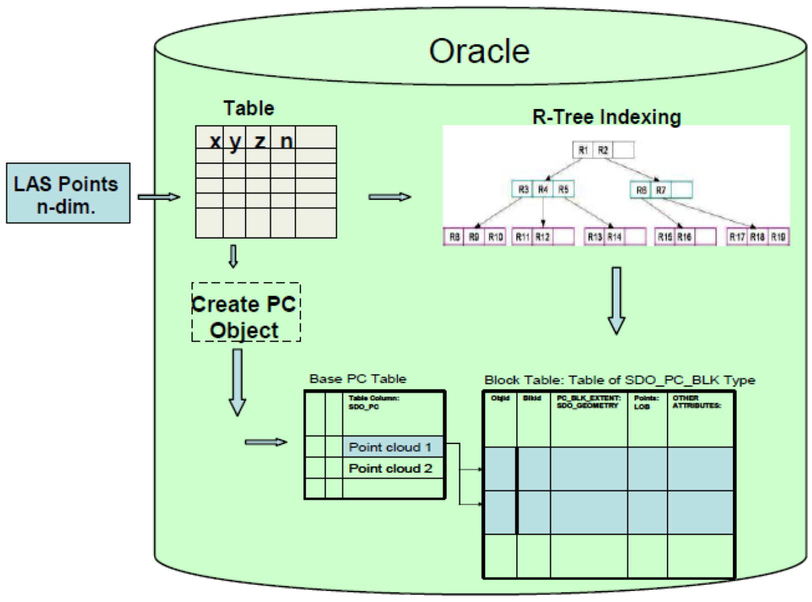


Figure 3.3.: The schema for Oracle SDO_PC object
Source: Alfarrarjeh et al. [2020]; Liu [2022]

3.3. SFC-based data organization

Due to the size of point cloud data, efficient data organization with techniques like spatial indexing and compression are crucial. In order to sort and create index on multiple organizing dimensions of the points, a Space Filling Curve (SFC) encoding is often applied to the points.

There are mainly three advantanges of using SFC to organize points:

1. **Dimension reduction**. It reduces the dimensions to one, and the points can be sorted and a B-tree index can be easily created on the SFC key. If a full-resolution curve is applied, the SFC key has uniqueness and can also be set as the primary key.

2. **Spatial clustering**. SFCs especially Morton Curve and Hilbert Curve have good clustering effects, making the points clustered on the physical level.

3. **Lossless compression**. If the full-resolution SFC key is computed, there is no need to store the original values of the organizing dimensions. The original values can be obtained by decoding the SFC keys. This can potentially save huge storage space.

In this way, the multiple dimensions are reduced to one and the points can be sorted or indexed with ease. On the other hand, some SFCs like Morton curve has a deep connection with 2^n-tree, which can potentially increase the querying efficiency.

3.3.1. PlainSFC

Van Oosterom et al. [2015] first designed a flat table model with Morton keys of points. It is named as PlainSFC by Liu [2022].

The main table contains columns of x, y, z and the Morton code. If the Morton code is a full-resolution encoding, the organizing dimensions can be removed. A B-tree index is created on the Morton code and the points are reordered by the index. The query algorithms are based on the relationship between Morton curve and 2^n-tree.

As to querying, PlainSFC adopts two filters, shown in Figure 3.4. The first filter uses the Morton-Quadtree hierarchy to approximate the query geometry and derive the ranges. Then a second filtering will be conducted in a following step to complete the query. Other query geometries can also be addressed.

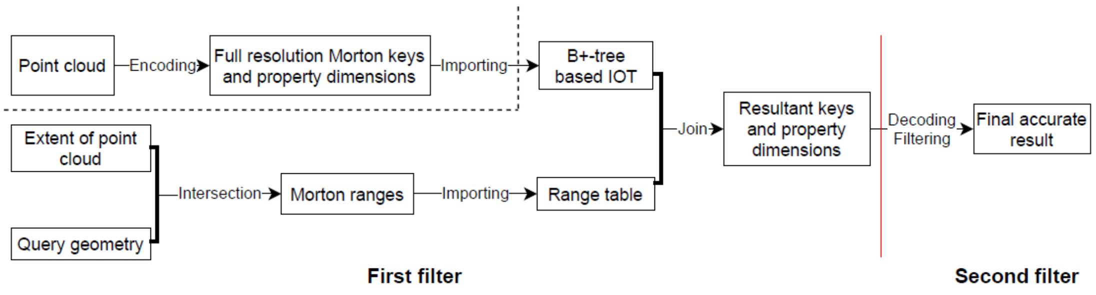


Figure 3.4.: The loading and querying procedure of PlainSFC
Source: Liu [2022]

Its performance was compared with a normal flat table with a XY B-tree index.The benchmarking results show that loading of PlainSFC method took longer time due to the computation of the Morton code of each points, and the queries of the PlainSFC approach are much faster and scale well than a normal flat table thanks to better spatial data organization and smarter data accessing strategy.

3.3.2. DynamicSFC

Psomadaki [2016] designed a database solution for dynamic point clouds using Space Filling Curve. I call it DynamicSFC. Dynamic point clouds are commonly used for identifying changes during a certain period or performing spatio-temporal analysis. Unlike static point clouds, the time component is a key dimension for dynamic point clouds. Therefore, the integration of time and space is a key problem in the research.

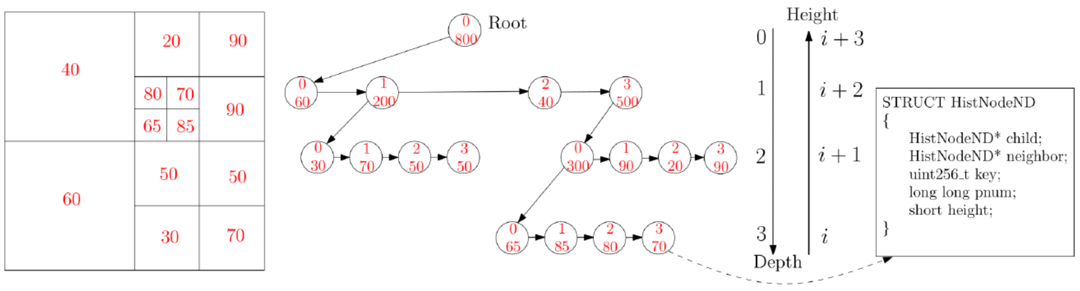
The encoding of time and space is based on the Space Filling Curve. Psomadaki investigated two approaches of integrating space and time (integrated and non-integrated), treatments of z (as an organizing dimension or a non-organizing dimension), and made implementations in Oracle.

From the benchmarking results, it can be inferred that the integrated approach has a better performance in managing and querying dynamic point clouds. From the perspective of data organization, this approach offers a very compact storage model, which in the case of z as organizaing dimension, includes an IOT with only one column. From the perspective of querying, this approach offers scalable query response times and has much less false hits than a Minimum Bounding Rectangle approximation, which is commonly in spatial databases nowadays.

3.3.3. HistSFC

Due to the non-uniform distributions of point clouds, PlainSFC may generate many empty ranges with no points inside during the querying process, and it significantly increases the time cost of the filtering process. To solve this problem, an improved distribution-aware method called HistSFC is introduced by Haicheng Liu [2020], and it uses HistogramTree for range computation.

HistogramTree is an additional pointer-based structure besides the main table of PlainSFC. It is compact and is stored in a flat table. It presents data distribution and guides range computation in the filtering process. As shown in Figure 3.5, it records the point count for each SFC node at different level. If the count exceeds the threshold of the tree, it will be partitioned into SFC nodes in a lower level. From the pointer of the parent SFC node, all its children can be visited. A height field is used in a HistTree node to distinguish different nodes, because branch nodes at different levels may possess identical keys. The HistTree can be built either during or after the data loading process or afterwards, and the update is automatic when there is change in the main table.



```
STRUCT HistNodeND
{
    HistNodeND* child;
    HistNodeND* neighbor;
    uint256_t key;
    long long pnum;
    short height;
}
```

Figure 3.5.: A 2D HistTree example, with threshold being 100
Left: point counting, middle: pointer structure of HistTree, with each node storing a SFC key and number of points, right: structure of a HistTree node
Source: Haicheng Liu [2020]

The query is executed with the aid of HistTree. Unlike PlainSFC which adopts a fixed depth for recursive decomposition of nodes, HistSFC employs a flexible strategy which is adaptive to point density, as shown in Figure 3.6. The main querying process lies in an overlap detection between HistTree nodes and the query window. There are three types of nodes according to their spatial relationship with the query window, i.e. outside, inside and boundary. The boundary nodes influence the False Positive Rate (FPR) of selection significantly, which then impacts the performance of a secondary filtering. An adaptive range generation method is developed to refine boundary nodes and thus optimized the whole querying process.

The performance of HistSFC is evaluated by comprehensive benchmarks with two use cases, AHN2 and flood risk querying. By comparing PlainSFC and HistSFC to the state-of-the-art solutions, HistSFC is the most efficient solution among them.

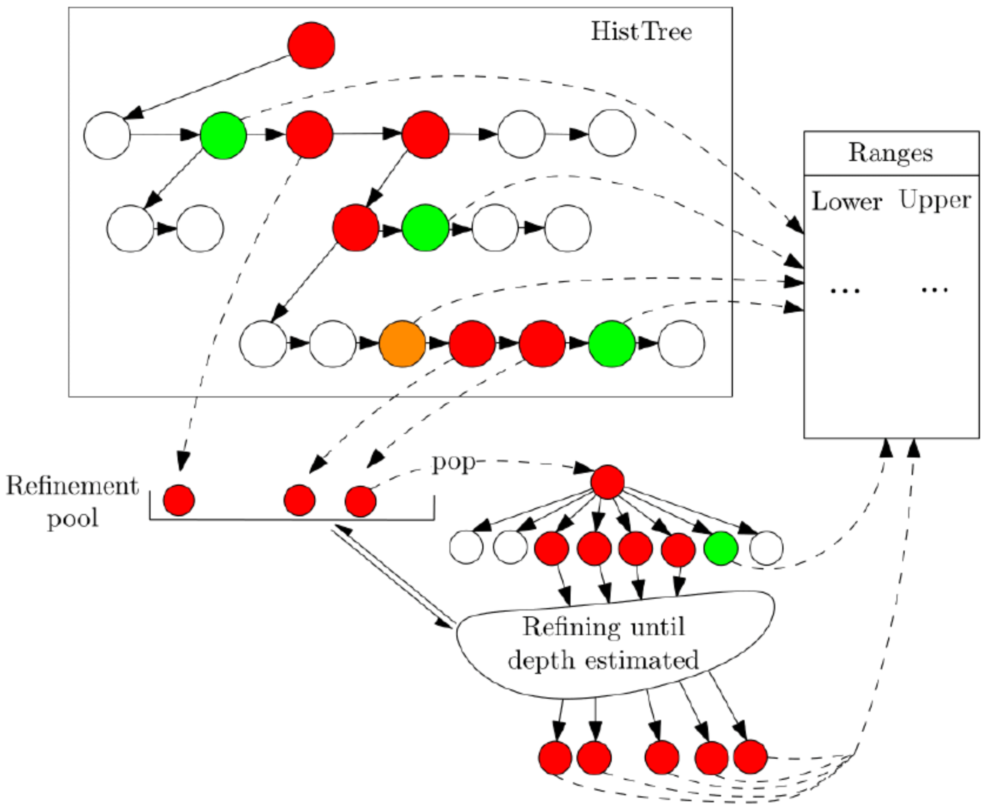


Figure 3.6.: Range generation using a 3D HistTree
Green nodes: within query geometry, red nodes: intersecting query geometry; white nodes: outside query geometry
Source: Liu [2022]

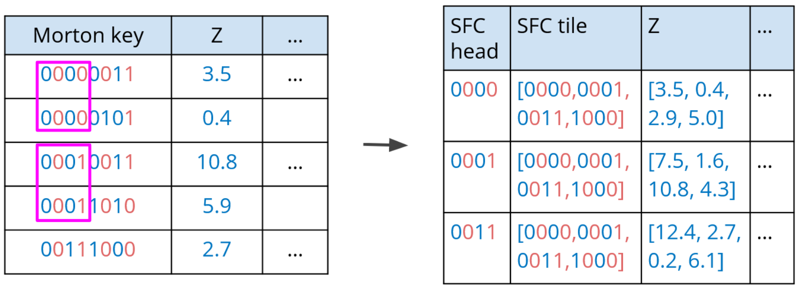
4. Methodology

The methodology chapter describes the proposed SplitSFC approach. Section 4.1 gives the motivation and inspiration for using a split SFC key for the managing massive point clouds in relational database. Section 4.2 provides the description of the storage model in the database. Section 4.3 gives an overview of procedures and algorithms, including data preparation, loading and querying. Finally, Section 4.4 illustrates the benchmarking strategy to validate and improve the methodology.

4.1. Motivation

SplitSFC is an improved block-based method proposed in this thesis, inspired by the PlainSFC approach and the hierarchy of Morton code. The advantages of using SFC to organize points have been discussed in Section 3.3. They are **dimension reduction**, **spatial clustering effect** and **lossless compression**. The PlainSFC approach has shown promising performance improvement in the benchmark. However, there is a significant drawback in its data organization, as a flat table is not a compact structure for massive point clouds. Therefore, it can be a possible significant improvement to design a block-based model using Space Filling Curve. If we can store the common property for each block rather than storing it repeatedly for all points, the data can be hugely compressed.

The numerical observation in a PlainSFC table is the initial inspiration of the SplitSFC approach. It can be noticed in the purple box of Figure 4.1 that the same head parts are shared with many points. If we group the points with the same SFC head together in a block, then each SFC head is stored once and shared with multiple points. In this way, the SFC keys have a potential to be highly compressed.



Morton key	Z	...
00000011	3.5	...
00000101	0.4	
00010011	10.8	...
00011010	5.9	
00111000	2.7	...

SFC head	SFC tile	Z	...
0000	[0000,0001, 0011,1000]	[3.5, 0.4, 2.9, 5.0]	...
0001	[0000,0001, 0011,1000]	[7.5, 1.6, 10.8, 4.3]	...
0011	[0000,0001, 0011,1000]	[12.4, 2.7, 0.2, 6.1]	...

Figure 4.1.: From PlainSFC to SplitSFC: grouping points with SFC head

In addition, the geometric meaning of the split is related to the hierarchical structure of 2^n-tree and Morton Curve, as introduced in Section 2.2.4. Based on this structure, the point clouds are decomposed naturally. Each block corresponds to a Morton node in a certain level. In other words, the points within the same quadrant are stored in the same block. The spatial coordinate of each point can then be decomposed into two parts, (a) the spatial extent of the quadrant, and (b) the specific location of the point inside this quadrant. Correspondingly, when the Morton code of each point is split into two parts, the SFC head corresponds to the spatial extent of the quadtrant while the SFC tile corresponds to the specific location of the point within the quadrant. Therefore, the points with the same SFC head are in the same blocks, and the SFC head is stored as the main property of the block.

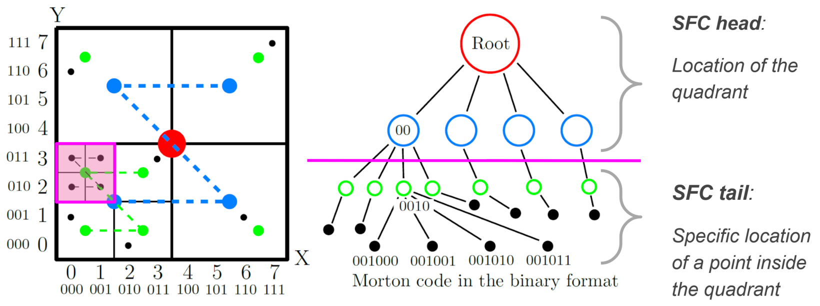


Figure 4.2.: The geometric meaning of the split, adapted from Liu [2022]

The most commonly used SFCs are Morton Curve and Hilbert Curve. We use Morton Curve because of its simplicity and close relationship with 2^n-tree. The algorithm of generating Morton curve is bitwise interleaving, therefore the encoding and decoding are rather efficient. In comparison, Hilbert curve has a relatively complex algorithm. On the other hand, the connection between Morton Curve and 2^n-tree structure makes it ideal for speeding up the querying process.

4.2. Storage Model

The SplitSFC schema contains three components: two tables and one index (Table 4.1).

- **Metadata table** describes the metadata of the point cloud dataset, providing context and helping the user to understand its characteristics. It stores the dataset name, Spatial Reference ID, the number of points, the spatial extent, the length of the SFC head and tile, etc.
- **Points table** stores the specific dimensions of the points. The original values of the organizing dimensions are converted into a SFC head and a SFC tail. The points with the same SFC head are grouped into one block. The table stores one block per record. The SFC_head column is integer, and the other columns are arrays. The points in one block are sorted by their SFC tails, and the SFC tail and the other property dimensions of each point are stored in SFC_tile column and the remaining columns.
- **A B-tree index** is created on the SFC_head column for efficient data retrieval.

Spatial Reference System

The Spatial Reference Identifier (SRID) is specified in the metadata table, together with other geometry information such as the total number of the points and the spatial extent of the

Table	Column	Data Type	Description
Metadata	name	string	The name of the dataset
	srid	integer	The Spatial Reference ID of the dataset
	point_count	(big) integer	The number of points of the dataset
	ratio	double	The fixed ratio of SFC head and tile ranges
	scales	array of double	The scale factors for organizing dimensions
	offsets	array of double	The offset factors for organizing dimensions
	bbox	array of double	The spatial extent of XYZ coordinates
Points	sfc_head	integer	The SFC head ranges
	sfc_tail	array of integer	A sorted array of SFC tails of the points in the block
	other dimensions	array	An array of values of points in the block

Table 4.1.: The schema of the SplitSFC approach

point cloud. By default, the used SRIDs are defined by the European Petroleum Survey Group (EPSG).

The values in X, Y, and Z fields are scaled and shifted from the original coordinates. Similar to LAS format, the values in the database need to be multiplied by a scaling value and added to an offset value to convert to the actual coordinates, i.e.:

$$X_{\text{coordinate}} = X_{\text{record}} \cdot X_{\text{scale}} + X_{\text{offset}}$$

$$Y_{\text{coordinate}} = Y_{\text{record}} \cdot Y_{\text{scale}} + Y_{\text{offset}}$$

The scaling factors X_{scale}, Y_{scale} and the offsets X_{offset}, Y_{offset} are also stored in the metadata table. To encode the organizing dimensions to Morton code, the values should be converted into integer data type. Notice that the number of decimals that can be stored are determined by the scaling factor. For example, the scaling factors 0.1, 0.01, and 0.001 would give us 1, 2, and 3 decimals respectively.

To simplify the data model, all points in one database should use the same Spatial Reference System, including SRID, scales and offsets.

The organizing and non-organizing dimensions

The dimensions of the points include spatial coordinate XYZ, time component, LoD (Level of Detail) compoment, classification, intensity, etc., as introduced in Section 2.1.1. The dimensions can be divided into organizing dimensions and associate dimensions (non-organizing dimension) based on whether they are encoded into SFC keys or not. The possible organizing dimensions usually include XYZ coordinates, time components, and Level of Details (LoD) components, depending on the user need.

In our research, We will not investigate time and LoD component, as they are not within the scope of the research. We will only encode X and Y coordinates. The remaining attributes will be stored in their original forms in the table. As we use the Dutch National Digital Surface Model for experiments, there are two reasons for encoding only X and Y: (a) the Netherlands is very flat; (b) the tested queries are mainly based on 2D geometry. So we treat the Z dimension as an associate attribute, rather than a spatial dimension.

4.3. Procedure and Algorithms

4.3.1. Pre-processing and Loading

The purpose of the preparation and loading procedure is to generate the SFC keys of the point clouds, physically order and organize the points and load them into the database.

The whole procedure (Figure 4.3) is divided into three phases: the pre-processing, the loading and the post-processing.

- **Pre-processing**: The raw point clouds are converted into point blocks during the processing. The steps in the pre-processing procedure are (a) reading the LAS/LAZ file, (b) encoding the organizing coordinates into the Morton key, (c) splitting the key into two parts, (d) grouping the points with the same SFC head into blocks and (e) sorting the points in each block based on the SFC tail.
- **Loading**: This step is the bulk loading of the point blocks into a normal heap table.
- **Post-processing**: A B-tree index is built upon the sfc_head column, and the connection with the database is closed.

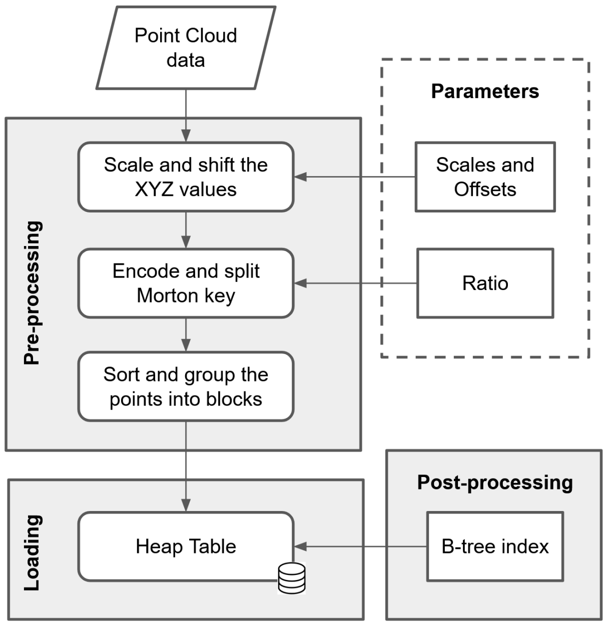


Figure 4.3.: The preparation and loading procedure for the SplitSFC approach

Morton encoding and decoding algorithm

To make a morton code, we need to interleave the bits of organization dimensions. For-loop method, Magic Bits method and Look-Up Table (LUT) method are three Morton encoding algorithms. Baert [2013] compared the performance of three algorithms. The results in Figure 4.4show that the Look-Up Table Method is much more efficient than other algorithms.

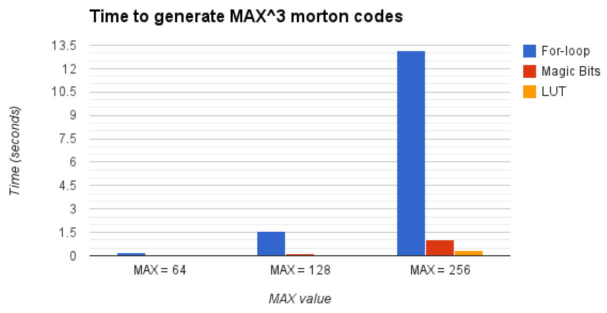


Figure 4.4.: The performance comparison of three Morton encoding methods
Source: Baert [2013]

Therefore, we use the Lookup Table (LUT) method to encode and decode the Morton codes. The algorithm is a divide-and-conquer method. The main idea is to pre-compute splitting a certain subset of bits, and then split the input integers byte by byte, and shift the results in place. For example, if there are only two organizing dimensions i.e. XY coordinates, we can first make the LookUp table for X values, and for Y values, just shift them to the left by 1 bit.

Splitting algorithm

The splitting algorithm is based on fixed-length, and the length is computed by the ratio of the head length and the tail length:

$$ratio = \frac{length(SFChead)}{length(SFCtail)} \cdot 100\%$$

4.3.2. Querying

We mainly focus on the 2D geometry querying with XY coordinates as organizating dimensions. The geometry query can be rectangle, circle or polygon.

The querying procedure has two steps, filter step and refinement step. In the first step, we only filter out the points within the bounding box of the given geometry, and the output is a set of decoded points. In the refinement step, we apply a specific geometry filter on the points.

Filter step

The filter step is to perform a range query based on SFC keys. The steps are:

1. Compute the bounding box of the given query geometry;
2. Perform range query in the SFC heads, and obtain a set of the fully contained SFC head ranges and a set of overlapping SFC head values;
3. Import the fully contained SFC head ranges into a temporary table, and join the range table and the point block table, and the obtained points are all within the bounding box;

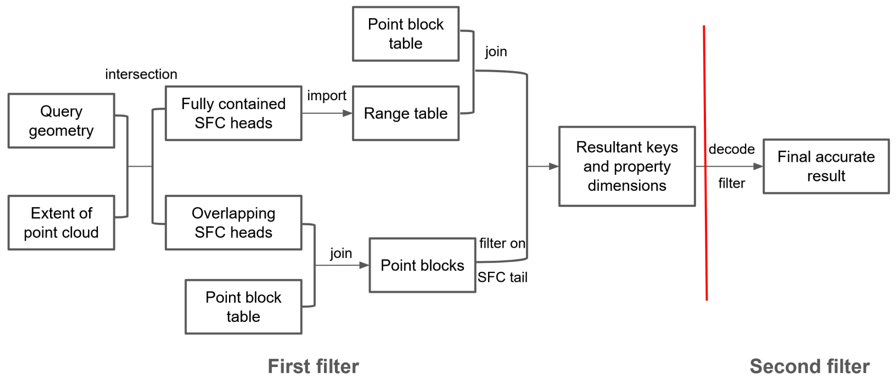


Figure 4.5.: The querying procedure of SplitSFC approach

4. Select the records from the point block table that are within the set of overlapping SFC heads, unpack each block, and perform range query in the SFC tails;
5. Merge the points from step 3 and step 4, and they are all points within the given bounding box;
6. Decode the points.

Refinement step

The points obtained from the filter step are all points within the bounding box, not the given 2D geometry. The goal in the refinement step is to filter out the points outside the geometry. Oracle and PostgreSQL/PostGIS both offer this kind of geometry functions.

Morton range search algorithm

The query algorithm needs to be modified for the SFC data organisation.

The algorithm uses the characteristics that Morton Curve is strongly associated with 2^n-tree. When querying with SFC keys, the geometry region needs to be converted into Morton key ranges, then the query becomes a search problem in one dimension.

For an multi-dimensional Morton Curve, each shift of n bits to the left is equivalent to moving one level deeper in the 2^n-tree. Here we will explain the algorithm in a situation with 2D Morton Curve and Quadtree. We will iterate each bit from left to right, moving two bits at a time. In each iteration, we need to determine the containment relationship of the four quadrants with the query range. There are three possible containment relationships: refine possible, no overlap and fully contained. To determine the relationship, we compare the bounding box in the quadrant to the query range. The bounding box of the quadrant can be obtained by decoding the largest and smallest Morton key in the quadrant, i.e.:

quadrant_x_min, quadrant_y_min = DecodeMorton2D(Morton_key_min)

quadrant_x_max, quadrant_y_max = DecodeMorton2D(Morton_key_max)

Next, we compare x and y ranges of quadrant and query geometry to determine their containment relationship. If it is not contained, then no further query is performed on that level. if it is full containment, all Morton ranges within that quadrant are the target range. If refinement is possible, we will continue to explore the sub-regions of that quadrant, i.e.,

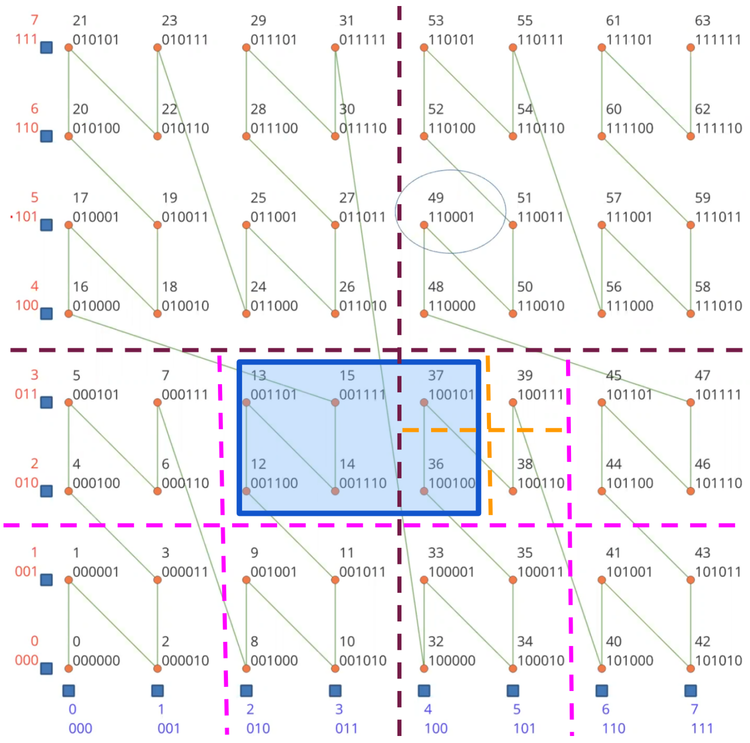


Figure 4.6.: An example of range search using 2D Morton Curve, adapted from the presentation of PCServe [Meijers, 2022]
Input query range is [2, 4] for x, [2, 3] for y; output Morton range [12, 15] and [36, 37]
Brown line: Quadtree depth of 1. Purple line: Quadtree depth of 2, orange line: Quadtree depth of 3

for that quadrant, we continue to move two bits to the right to determine the containment relationship.

4.4. Benchmark

Benchmark is crucial in the research and development process of designing database solutions. The assessment of the performance of the prototype guides for the direction of improvement. We use the benchmark framework designed by Van Oosterom et al. [2015] to evaluate our prototype for mainly three reasons:

- The tested dataset Actueel Hoogtebestand Nederland 2 (AHN2) is open data and easily accessible. Therefore, the results are reproductive.
- The benchmark framework is well-designed. It is based on user requirement analysis, and considers the general-purpose functionalities comprehensively. In addition, to explore the scalability of the Point Cloud DBMS, it scales up the input dataset 10 times in each stage.

- Several state-of-the-art database solutions have been tested with this framework, therefore we can directly compare the performance of SplitSFC with that of other solutions in PostgreSQL.

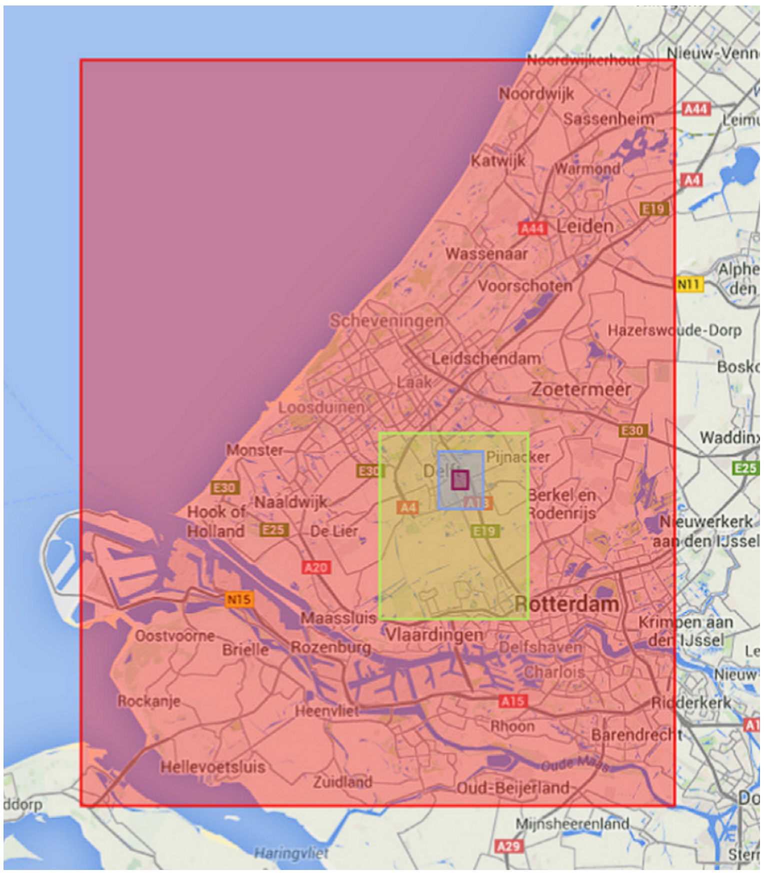


Figure 4.7.: Approximated projection of the extents of the used datasets in Google Maps: Purple area is for 20M dataset, blue area is for 210M dataset, green area is for 2201M dataset, read area is for 23090M dataset
Source: Van Oosterom et al. [2015]

4.4.1. Scalability

The benchmark uses the Actueel Hoogtebestand Nederland 2 (AHN2) dataset. The test datasets are the subsets of it, and they are tailored for the three stages of the benchmark (Van Oosterom et al. [2015]):

- **Mini-benchmark**: It starts with a small dataset of about 20 million points in Delft Campus. In this stage, we can explore the effects of different parameters, and the results serve as a basis to decide on parameter settings in the next stage benchmark.
- **Medium-benchmark**: In this stage, the benchmark uses several dataset up to 23090M with about 20 billion points. It explores the scaling behaviour of the database solution,

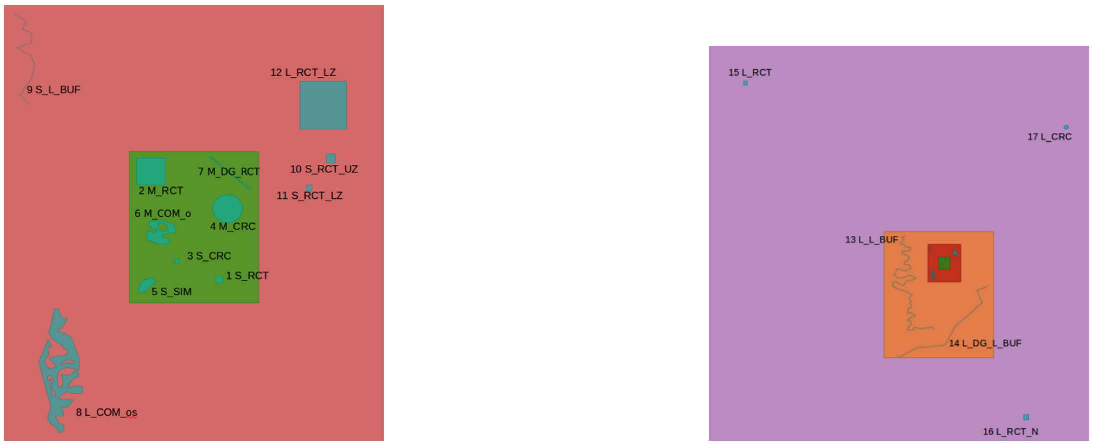
and extends the querying functionalities of the mini-benchmark.

- **Full-benchmark**: In this stage, the complete AHN2 dataset is loaded into the database and queried. However, we only perform mini-benchmark and medium-benchmark in our experiments.

4.4.2. Querying Functionalities

In the benchmark framework (Van Oosterom et al. [2015]), all queries are selection of points. Most are selected within a query region, and the types of regions include rectangles (axis-aligned or diagonal), circles, simply polygons, polygons with holes, buffer of polylines, etc. The geometries are shown in Figure 4.8. We also perform nearest queries and height queries. See Appendix A for detailed queries.

The query return method is Create Table As Select (CTAS), which means the returned points are all stored in a new table (Van Oosterom et al. [2015]).



(a) Up to 210M dataset

(b) Up to 23090M dataset

Figure 4.8.: Queries used in the medium benchmark
Source: van Oosterom et al., Massive point cloud data management: Design, implementation and execution of a point cloud benchmark, 2015

5. Implementation

The implementation chapter contains a description of the prototype implemented to show whether the SplitSFC approach is an effective solution for massive point cloud data management. For the implementation of SplitSFC, a set of Python scripts have been developed. The scripts can import LAS/LAZ files into PostgreSQL in the designed schema, perform basic spatial queries and output LAS files from the database.

This chapter is organized as follows: Section 5.1 presents the tools and datasets used to develop the prototype and perform the experiments. After that, Section 5.2 gives the implementation details of the prototype, including the importer, the querying tool and the exporter.

5.1. Tools

5.1.1. Hardware

In order to make the results comparable with other solutions being tested within the same benchmark framework, we use the same hardware environment as that in the point cloud data management research by Van Oosterom et al. [2015]; van Oosterom et al. [2017]. The supercomputer is called pakhuis, and the settings are:

A HP DL380p Gen8 server with 128 GB RAM and 2 × 8 Intel Xeon processors E5-2690 at 2.9 GHz, RHEL 6 as operative system and different disks directly attached including 400 GB SSD, 5 TB SAS 15 K rpm in RAID 5 configuration (internal), and 2 × 41 TB SATA 7200 rpm in RAID 5 configuration (in Yotta disk cabinet).

5.1.2. Software

The main softwares for the implementation are PostgreSQL and Python.

- **PostgreSQL and PostGIS** are the DBMS used for the storage and manipulation of point cloud data. It is chosen because it is the most advanced open-source relational database management system, and has good spatial support with PostGIS. The versions are PostgreSQL 9.3.2 and PostGIS 2.1.1.
- **Python** handles the data preparation outside the database because SQL is unsuitable for complicated algorithms like SFC encoding and decoding. It was chosen for its simplicity, but it also poses a significant disadvantage in time efficiency. The Python version that we use is the 3.

- **Psycopg2** is used for connecting Python and PostgreSQL, and it allows execution of raw SQL commands in Python. In the older version of the implementation, SQLAlchemy was used because of its simple ORM model and beauty in grammar, but it was replaced by psycopg2 due to its advantage in efficiency.

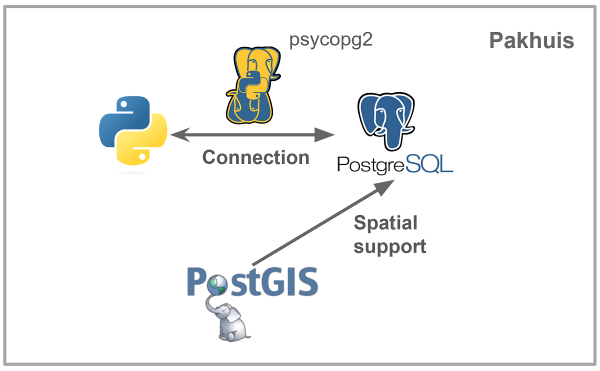


Figure 5.1.: The hardware and software environment for the SplitSFC prototype

5.1.3. Datasets

As mentioned in Section 4.4, we employ the Actueel Hoogtebestand Nederland 2 (AHN2) dataset to evaluate performance. This dataset comprises a total of 640 billion points, with a sample density ranging from 6 to 10 points per square meter across the country. The Spatial Reference System for AHN2 is the Amersfoort/RD New, identified by the Spatial Reference System Identifier (SRID) 28992. All datasets involved in these experiments are formatted in LAS and contain solely XYZ dimensions.

We only perform the mini-benchmark and the medium benchmark from the original benchmark framework. Four datasets have been tailored to perform the benchmark, varying from 20 million points in TU Delft campus to 20 billion points in South Holland Province. Each dataset stores ten times more points than the previous dataset. Table 5.1 and Figure 4.7 show the details of the dataset and an approximate spatial extent projection.

Name	Points	Files	Size (GB)	Area (km2)	Description
20M	20,165,862	1	0.4	1.25	TU Delft Campus
210M	210,631,597	1	4	11.25	Major part of Delft city
2201M	2,201,135,689	153	42	125	City of Delft and surroundings
23090M	23,090,482,255	1492	440.4	2000	Major part of South Holland

Table 5.1.: Dataset description
Source: Van Oosterom et al. [2015]

5.2. Implementation

Three Python tools have been developed to implement the functionalities of the SplitSFC approach. All these tools require input parameters.

- **Importer**: It can convert LAS / LAZ files into the designed format and load them into the database.
- **Querying tool**: It can perform 2D geometrical query and elevation query on the point records in the tables, and save the results in a new table.
- **Exporter**: It can convert the point records in the query returned table into a LAS file.

5.2.1. Importer

The purpose of this tool is to convert the LAS files into desired format and load them into the database. The given parameters include the path, name, SRID of the dataset, the split ratio of the SFC key, and the configuration of database connection. The importer script has two modes. One is single-file importing, and the other is multiple-file importing. In the multiple-file mode, the point clouds are processed and imported per file, otherwise the program may be terminated due to memory limits if the dataset is too big.

The data preparation process is in Python, and outputs the data in CSV files. In the loading process, Python connects to the database using psycopg2, creates the tables, copies the CSV files into the tables, and creates the B-tree index.

1. Data pre-processing

Calculate metadata

To begin with, we use laspy to read the metadata and the point records in the LAS files. The metadata table has six columns, including name, srid(Spatial Reference ID), point_count (the number of points), head_length, tail_length and bbox (bounding box). The name and SRID is given in the parameters. There are two situations, regarding whether the input are one file or multiple files:

If there is only one file, then the number of points and bounding box can be directly obtained from the header of the file.

If there are multiple files, iterate the header of each file, sum up the point count and merge the bounding box. Next, we calculate the length of the SFC head and SFC tail by the given ratio. We take the maximum value of the X and Y coordinates, encode them into a Morton key. Based on the length of this key, we can calculate the length of the SFC head and SFC tail with the ratio.

Calculate and split the Morton key

The implementation of spliting uses the bitwise operators in Python. We encode the XY coordinates of each point to Morton key using the LookUp Table (LUT) algorithm introduced in Section 4.3.1, and split the Morton key into a head and tile based on the length computed in the previous step.

Group the points based on SFC head

Both Python and PostgreSQL can sort and group the points. If we use Python, we can use the sorted and groupby method from the itertools. If we use PostgreSQL, we need to transfer all points with their Morton codes and non-organizing dimensions into a flat table in the database, and use the native groupby support of PostgreSQL and load the processed data into a new table. In the experiment, we find the Python implementation faster, probably because the data transfer with ungrouped points creates a huge overhead. Therefore, the final implementation employs the sorting and grouping in Python.

For large data, it is more efficient to use 'copy' method rather than 'insert' method to load data into the database. Therefore, we write the pre-processed data into CSV files. The points in each block are sorted by the value of SFC tail. SFC tail and the other attributes are nested into lists.

2. Loading

Connect to the database, create table

In this step, Python connects to the database and creates a metadata table and a point record table. The number of the columns of the point record table actually depends on the number of dimensions that we want to keep. In this prototype, we only keep XYZ coordinates, and XY coordinates are converted into the SFC key, therefore we have three columns in this case: sfc_head, sfc_tail and z. The points in each block are sorted by the value of their SFC tail, and other dimensions are stored in separate columns of the array.

```
CREATE TABLE pc_metadata (
    name TEXT,
    srid INT,
    point_count BIGINT,
    head_length INT,
    tail_length INT,
    scales DOUBLE PRECISION[],
    offsets DOUBLE PRECISION[],
    bbox DOUBLE PRECISION[]
    );
CREATE TABLE pc_record (
    sfc_head INT,
    sfc_tail INT[],
    z DOUBLE PRECISION[]
    );
```

Import data into the database

After the tables are established, the metadata values are inserted into the pc_metadata table and the point blocks are copied into the pc_record table in the form of CSV file. Copy method is used because it is more efficient than insertion, and our data volume is huge.

```
COPY pc_record FROM stdin WITH CSV HEADER
```

Post-processing

Build a B-tree index on SFC_head column.

The B-tree index can improve the querying performance.

```
CREATE INDEX btree_index ON pc_record USING btree (sfc_head)
```

5.2.2. Querying tool

The querying tool allows the users to perform 2D geometry query for points in the database. The input 2D geometry can be . The given parameters include the source dataset name (i.e. 20M), the querying mode and the geometry. The query mode means the type of the 2D geometry, and it supportInrectangular, circular and polygonal addition, attribute query is also supported. the user s can put parameters as minz or maxz to query by elevation.

The query return method in the selection queries is CTAS (Create Table As Select) so for each query the returned points are saved in a new table. If the user wants to visualize the points, the data can be exported to LAS files with the exporter (See Section 5.2.3).

2D Geometry Search (rectangular, circular, polygonal)

Calculating Bounding Box

To simplify the querying process, the rectangular, circular and polygonal searches are searched with the bounding box of the given geometry first, and then we search the points within the bounding box with the specific geometry. For rectangular search alongside the axis, the given geometry is the bounding box. For circular search, the bounding box is the center point plus or minus the radius. For polygon search, just take the maximum and minimum value of the vertices of the given polygon.

Filter search

In this quad-code search, first the fully contained and overlapping SFC head ranges are taken out. All rows with the fully contained SFC head ranges are taken out and the points are decoded to XYZ coordinates. The rows with overlapping SFC head ranges need to go deeper with the search depth, and only the tails that are within the given ranges are taken out and the points are decoded back to XYZ coordinates. The results are created as a new table in the database, and the format is the PostGIS data type Geometry(PointZ).

Refinement search

Now the points are all within the bounding box of the given geometry, and they are in the original format: XYZ coordinates. The next step is to perform a query based on the specific geometry. This can be achieved by using ST_DWithin or ST_Within method from PostGIS for circle or polygon respectively.

For example, the SQL for refinement step of circular query is:

```
DELETE FROM {pc_record}
WHERE NOT ST_DWithin(point, ST_MakePoint({center_x}, {center_y}),
{radius});
```

The SQL for refinement step of polygonal query is:

```
DELETE FROM {pc_record}
WHERE NOT ST_Within(point, ST_GeomFromText('{wkt_string}'))
```

Attribute Search

After performing the geometry query, an additional attribute query can be performed, if requested by the user. The corresponding SQL for minz search is (minz is the value of parameter) :

```
DELETE FROM {pc_record}
WHERE ST_Z(point) < {minz}
```

5.2.3. Exporter

The exporter cannot be used alone. It needs to used with a querying tool. The querying tool first creates a table that contains all the points of the given query, then the exporter outputs a LAS file that contains all points in the querying result table.

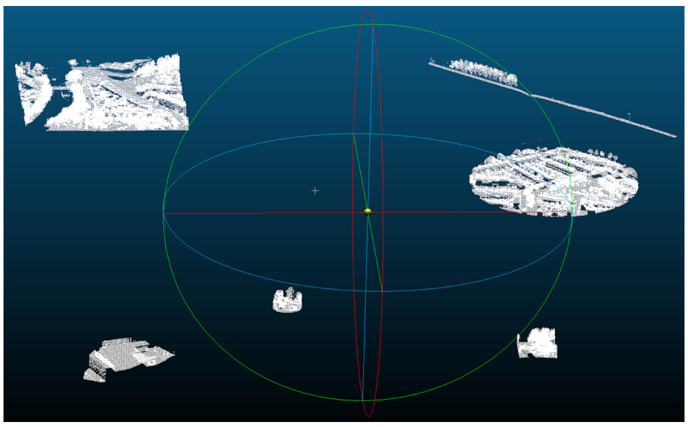


Figure 5.2.: Visualization of the query results of 20M dataset in CloudCompare

6. Results and analysis

Chapter 6 illustrates the benchmark results and analysis of the SplitSFC prototype and other state-of-the-art block models for PCDBMS. In Section 6.1, the effects of parameters like ratio, scales and offsets were explored, and the benchmark was done at mini-level with 20M and 210M dataset. In Section 6.2, the medium benchmark is executed. It shows the performance of the SplitSFC prototype, in comparison with other solutions.

6.1. Mini benchmark

In order to explore the effects of ratio, scales and offsets, three questions have been investigated:

1. How much storage space is saved by dropping two decimal places of precision?
2. Can shifting the points to the corner of their bounding box potentially save the storage space?
3. What is the ratio for the optimized storage size in each set of scales and offsets?

To answer these questions, we imported the 20M dataset and the 210M dataset into the database with three sets of scales and offsets. With each set of scales and offsets, we test different ratios, starting with a 50% split, and then shift the SFC head either two bits to the left or right.

	Explanation	**Scales**	**Offsets**
Solution 1	Default value	[0.01, 0.01]	[0, 0]
Solution 2	The points are shifted to the corner of the bounding box	[0.01, 0.01]	[min_x, min_y]
Solution 3	The resolution is reduced to 1m	[1, 1]	[0, 0]

Table 6.1.: Scales and offsets for the test

6.1.1. Scales

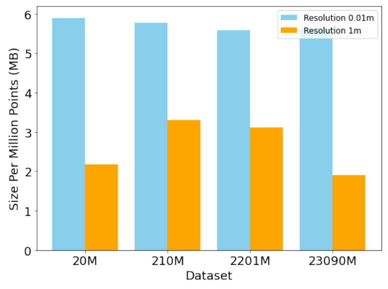
To explore the effect of scales, we compare the import time and storage size of Solution 1 and Solution 3 in Table 6.2.

From Table 6.2 and Figure 6.1, we can infer that:

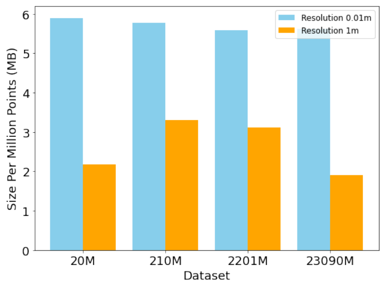
1. Reducing the precision does not significantly decrease the importing time.
2. Reducing the precision dramatically saves the storage space. In the case of the tested dataset, when the precision is reduced by two decimal points, the total storage size is reduced by almost half.

Datasets	Resolution (m)	Import Time(s)	Size (MB)	Points/s	Size Per Million Points (MB)
20M	0.01	105.21	119	191672	5.90
	1	215.56	44	93551	2.18
210M	0.01	1349.6	1215	156070	5.77
	1	1422.31	696	148091	2.30
2201M	0.01	11443.59	12288	192347	5.58
	1	11857.9	6854	185626	3.11
23090	0.01	120778.07	130048	191181	5.63
	1	102824.69	44032	224562	1.91

Table 6.2.: Importing and storage performance with a 0.01m resolution and 1m resolution



(a) Imported points per second



(b) Size per million point

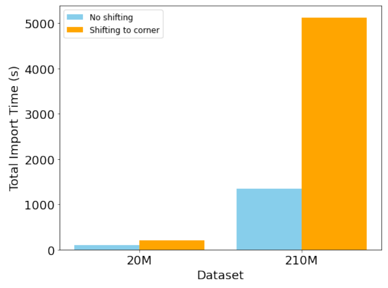
Figure 6.1.: The comparison of importing and storage efficiency between Solution 1 and 3

6.1.2. Offsets

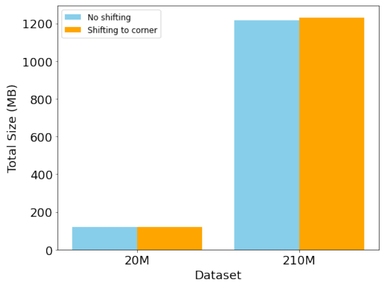
To explore the effect of offsets, we compare the import time and storage size of Solution 1 and Solution 2 in Table 6.3.

Datasets	Offset (m)	Import Time (s)	Size (MB)	Points/s	Size Per Million Points (MB)
20M	[0, 0]	105.21	119	191672.48	5.90
	[85000, 446250]	200.63	120	100512.70	5.95
210M	[0, 0]	1349.6	1215	156069.65	5.77
	[84000, 445000]	5126.94	1232	41083.30	5.85

Table 6.3.: Importing and storage performance with no shifting and shifting to the corner



(a) Total import time



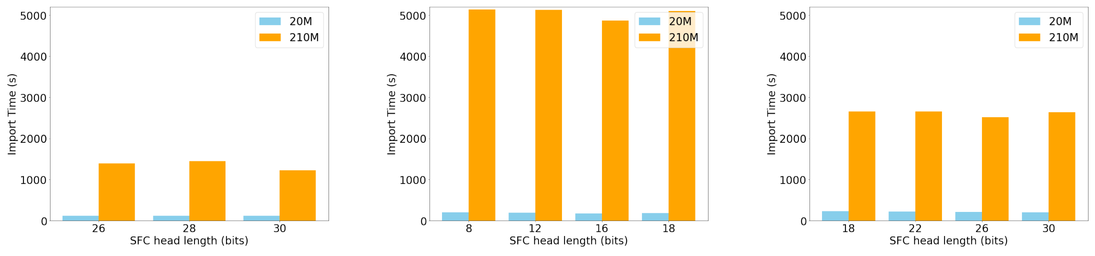
(b) Total size

Figure 6.2.: The comparison of importing time and storage size between Solution 1 and 2

From Table 6.3 and Figure 6.2, we can infer that shifting the points to the corner may not help much with either increasing importing efficiency or storage space. More discussion about the possible reasons are discussed in Section 6.1.3.

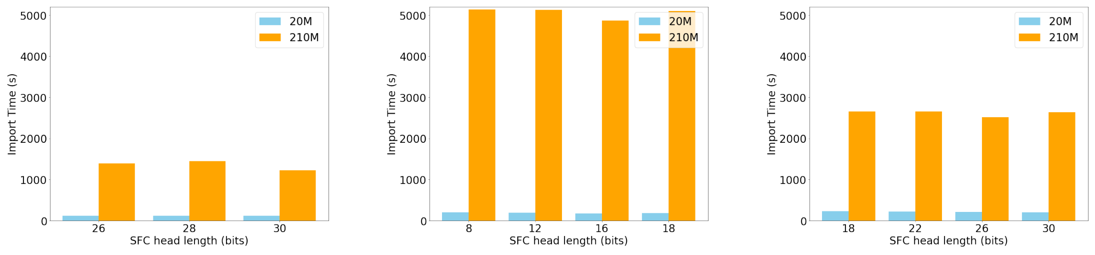
6.1.3. Ratio

For each set of scales and offsets, we have already tested several possible head-tail splits and recorded their performance. In this section, a detailed experiment on the effect of ratio is carried out. See Appendix B for detailed data.



(a) Default mode (b) Shifting mode (c) Resolution = 1m

Figure 6.3.: The comparison of import time for different ratios in all three modes



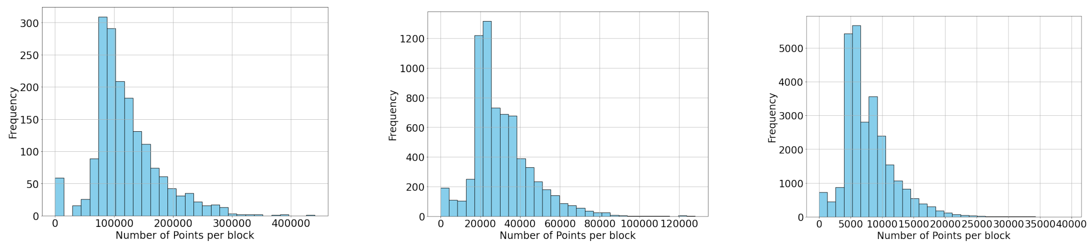
(a) Default mode (b) Shifting mode (c) Resolution = 1m

Figure 6.4.: The comparison of storage size for different ratios in all three modes

As shown from Figure 6.3 and Figure 6.4, we can infer that (a) the most suitable ratio for different situations can be different; (b) slight change in splitting ratio does not significantly improve the efficiency or the storage size.

We draw scatter plots and histogram of the number of points in each block for the importing of 210M dataset in Solution 1 (default mode), with an input ratio of 50%, 55% and 60%. The frequency of the number of points per block seems to follow the Gaussian distribution. As the ratio increases, the SFC head becomes longer, and the number of blocks increases, and the number of points per block decreases.

As mentioned in the previous subsection, the shifting does not save storage space than the default mode. This phenomenon may be related to the effect of ratio. My initial hypothesis is that it might help to save some storage space, because the shifting makes the resolution of X and Y smaller, and the length of the SFC keys is shorter. Take the 210M dataset for example, the total length of the SFC key shrinks from 52 bits to 38 bits, but the reality is that the total database size does not decrease at all.



(a) ratio = 50%   (b) ratio = 55%   (c) ratio = 60%

Figure 6.5.: The distribution of the number of points in each block, tested with 210m dataset scales = [0.01, 0.01, 0.01], offsets = [0, 0, 0]

We retrieve the number of blocks in the table and calculate the distribution of the number of points in each block (See Figure 6.5). As we can see, in Solution 2 (shifting mode), the number of points in each block dramatically decreases for each level of splitting ratio, which means the grouping is not so effective. This is probably because the spatial partitioning of the quadtree changes. Without shifting, the original quadtree extent covers a large area. After shifting the altered spatial extent of the quadtree shrinks dramatically.

6.2. Medium benchmark

In the medium benchmark stage, we import the 20M, 210M, 2201M and 23090M datasets into three databases. We compare the performance of the SplitSFC prototype with that of other block-based solutions in PostgreSQL and Oracle. The flat table solution takes up much more space, therefore it is not meaningful to compare with it. The results of other solutions were benchmarked by Van Oosterom et al. [2015]. The other two solutions are:

- **pgPointCloud**: The point blocks are stored as PC_Patch objects. See Section 3.2 for details. The blocking and loading process are done with an external tool called PDAL.
- **Oracle SDO_PC**: In this solution, the points are stored in a BLOB, and the blocks stored as SDO_PC_BLK objects. The metadata is stored as SDO_PC objects in a different table. 2D Hilbert key is computed for each point, and is stored as the primary key in an IOT table.

6.2.1. Importing and storage

From Table 6.4 and Figure 6.6, we can tell that:

- **Importing time**: The SplitSFC prototype is faster than Oracle SDO_PC, but slower than pgPointCloud. They also show different scalability when the imported dataset grows. When the size of the dataset grows ten times larger, the importing time of the SplitSFC prototype and Oracle SDO_PC grows almost ten times as well, but the importing time of pgPointCloud grows less dramatically.
- **Storage size**: The SplitSFC prototype occupies slightly more than pgPointCloud, but much less than Oracle SDO_PC.

Dataset	Approach	Time(s)				Size (MB)		Points/s
		Total	Initial	Load	Close	Total	Index	
20M	pgPointCloud	153.41	22.03	130.53	0.85	101.77	0.69	131451
	Oracle SDO_PC	296.22	0.35	228.17	67.7	226.5	0.2	68077
	pg SplitSFC	105.21	87.99	17.22	0	119	0.03	191672
210M	pgPointCloud	129.14	0.81	121	7.33	1009.13	5.18	1631033
	Oracle SDO_PC	1246.87	0.25	557.7	688.92	2244.5	1.44	168928
	pg SplitSFC	1349.6	1189.04	160.56	0	1215	0.05	156070
2201M	pgPointCloud	754.22	0.86	687.49	65.87	10245.61	53.05	2918427
	Oracle SDO_PC	16737.02	0.86	7613.39	9122.77	21220.5	13.25	131513
	pg SplitSFC	11443.59	9796.2	1647.24	0.15	12288	1.75	192347
23090M	pgPointCloud	12263.05	0.87	7450.1	4812.08	106781.48	552.77	1882931
	Oracle SDO_PC	**192612.07**	0.31	96148.06	96463.7	220085.5	165.55	119881
	pg SplitSFC	**120778.07**	104319.59	16457.31	1.17	130048	37	191181

Table 6.4.: Importing results, ratio = 50%

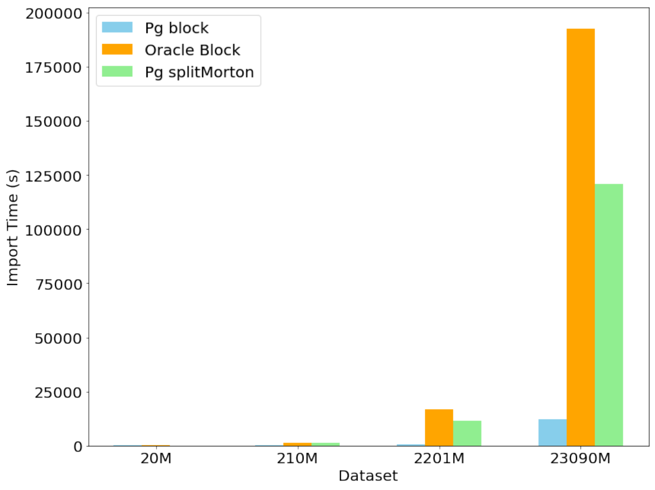


Figure 6.6.: The comparison of import time of three PC-DBMS solutions

6.2.2. Data retrieval

The complete results are shown in Appendix C. Apart from query response times, the number of points in each query is also recorded, to evaluate the accuracy and reliability of the querying algorithm. The queried points are further exported to LAS files and visualized in CloudCompare software to validate the reliability of the results (See Figure 5.2).

From Figure 6.7 and 6.8, we can infer:

- SplitSFC is slower compared to other solutions, but the querying response times are acceptable in real-world applications.
- The query response time remains almost constant for the same query geometry when we increase the coverage of the dataset.
- The number of returned points have slight differences, probably due to the nature of our range query algorithm with Morton keys. Also, other data models seem to use

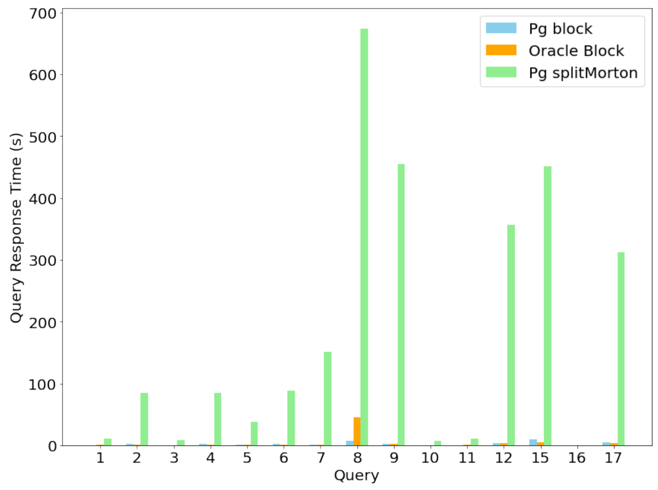


Figure 6.7.: The comparison of query response time for all queries

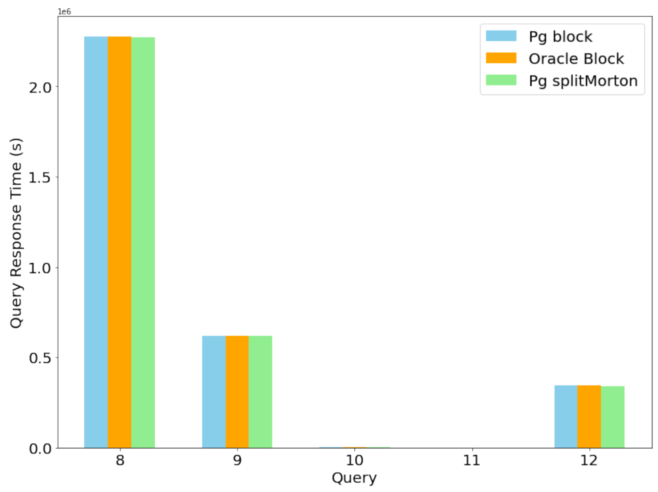


Figure 6.8.: The comparison of the number of points for query in 210M dataset

a polygon approximation of the circle, but we use the distance method for the circle search to test the SplitSFC approach.

- The factors that affect the querying response time include the querying geometry itself and the size of the dataset. As the experiments show, the spatial extent and the complexity of query geometry itself matters much more than the size of the dataset. Query 13 and 14 are complicated polylines and the program crashes when the PostGIS is performing the polygon query.

7. Conclusions and discussion

In Chapter 7, conclusions and discussion were drawn from the results, and it offers a comprehensive summary of the whole research. We will first present the conclusions, contributions and limitations in Section 7.1, and illustrate the future work in Section 7.2.

7.1. Conclusions and Discussion

7.1.1. Conclusions

The objective of this research is to come up with an efficient point cloud DBMS solution using split Space Filling Curve. The main research question is to find out whether employing the split Space Filling Curve approach constitutes an effective strategy for managing massive point clouds in a relational DBMS.

In order to draw an answer to the main question, we will discuss the sub-questions first.

How to efficiently use SFC to organise the points in blocks?

A block-based model is used as it is more compact than a flat table and occupies less space. We do the following to organize the points more efficiently:

- **Scales and offsets**. Since the data type of the values for Morton encoding is integer, we need to scale the original float values to integers. To convert these values to the actual coordinates on the ground, they need to be multiplied by a scaling factor and added to an offset value. The scaling factor determines the number of decimals that can be stored. For example, the factors 0.1, 0.01, and 0.001 would give us 1, 2, and 3 decimals respectively.

- **Morton curve**. The points are encoded with Morton keys. Morton keys can index and cluster the points and map the multi-dimensional data into a 1D space, therefore it may optimize the data retrieval performance. On the other hand, as a full resolution curve, the Morton keys are stored and the original values are dumped, this can potentially save storage space.

- **Splitting and grouping**. Each Morton key is splitted into a head and a tile, and the points with the same head are grouped into one block. The SFC head implies the rough region of the points. It can be stored once, and shared with many points inside the same region. Therefore, it can not only clustering the points, but also potentially save storage space.

In summary, this kind of point organization will not only index and cluster the points to improve the querying performance, but also has potential to compress data and save storage space. Therefore, the approach theoretically has the potential to become an effective strategy for data management of massive point clouds.

How to split the SFC keys, i.e. by fixed-length or adaptive algorithm?

In this thesis, we split the SFC keys with a fixed-length approach. We include 'ratio' as the parameter, and it determines the length of the SFC head and SFC tail. The ratio should either be too big or too small, otherwise it loses the effects of clustering and compression. Different sets of scales and offsets have been tested alongside the ratios, and it shows that selection of ratio is related to many factors, such as the point distribution. The recommended parameter tuning approach is to test ratios around 50%-60% on a small dataset, and compares the results. Once a suitable ratio is found, use this ratio for a bigger dataset with similar patterns.

How to query the points based on the suggested method?

The querying algorithm contains two steps, filter step and refinement step. In the filter step, the points within the bounding box of the given geometry are retrieved. The range search algorithm is based on the hierarchy of Morton Curve that represents the 2^n-tree structure. In the refinement step, the points outside the given geometry are filtered out.

How does the proposed method work compared to the current solutions?

In this research, the performance of SplitSFC prototype is compared with those of pgPointCloud and Oracle SDO_PC. The metrics of performance includes importing time, storage size and query response time. The benchmarking results show:

- SplitSFC takes longer time to import, because of the extra time in Morton key computation, sorting and grouping and the nature of Python implementation.
- pgPointCloud and SplitSFC occupy comparable storage space, and Oracle SDO_PC occupies more storage space. Notably, if the algorithm and the implementation are improved, there is a chance that SplitSFC can have a better performance compared to other state-of-the-art database solutions.
- SplitSFC is significantly slower in querying. However, this may also be due to the nature of Python implementation and code quality.

Having given answers to the sub-question above, the main research question can also be answered: **Does employing the split Space Filling Curve approach constitute an effective strategy for managing massive point clouds in a relational database?**

While the split Space Filling Curve (SplitSFC) approach presents potential advantages for managing massive point clouds in a relational database, my study's findings suggest that its immediate implementation may not yield a definitive improvement.

The results of the research show that SplitSFC approach has longer import time and querying response time, and occupies a little more storage space than pgPointCloud. However, the observed performance drawbacks might be attributed to specific factors such as the programming language used (Python) and the need for optimizations, including an adaptive SFC key splitting algorithm and implementation in C++. Thus, while theoretically promising, the current empirical evidence indicates a 'probable' rather than a conclusive 'yes' or 'no' regarding the effectiveness of SplitSFC in its present form for this particular application.

7.1.2. Contributions

In this research, we design, implement and benchmark an improved block-based model namely SplitSFC approach to manage massive point clouds in relational database. The purpose is to improve the performance of current point cloud database solutions, especially with storage space. This model organizes the point clouds based on Morton Curve, and innovatively splits the SFC keys to heads and tiles. The model groups the points with the same SFC head into blocks, and stores SFC tails and other property dimensions as arrays in other columns.

Although the SplitSFC model does not show significant advantage in storage and data retrieval efficiency so far, it is fair to believe that with the improvement of algorithms, schemas, implementations, it has the potential to be an approximate and efficient point cloud database solution.

7.1.3. Limitations

A few major criticisms are made about the methodology and implemented prototype.

Fixed-length SFC key splitting

In this research, we only experiment with a fixed-length split, which means each block is a quadrant of the same level or same size. However, the nature of the point clouds determines that the point distribution is irregular, therefore the quadrants should not be the same size. In dense areas, the quadrant usually contains too many points, and it can be divided further, which means the length of the SFC head can be longer. On the contrary, in a sparse area, a quadrant may contain too few points, and it can be combined with neighbouring quadrants to form a bigger quadrant, which means the length of the SFC head should be shorter.

Programming language

Python is used for most algorithms. However, Python itself is much slower than lower-level languages like C or C++ due to its interpreted nature. And the algorithms like Morton encoding and decoding are expensive. Therefore, the implemented prototype is not very efficient in its current state.

It also causes trouble for the benchmark. Other data models were implemented with tools like C++. Our solution shows a significantly longer loading time and querying time, but it is hard to tell whether it is due to the nature of our data model , or due to the nature of Python.

Dimensions in the tested dataset

The imported datasets only have XYZ dimensions. This is because the original benchmark experiments [Van Oosterom et al., 2015] use the datasets with only XYZ dimensions. The benefit is that we focus on studying the effects of compressing the XYZ dimensions, and the results are easier to compare. However, we also lack knowledge about these models' performance when there are many dimensions.

Moreover, in the implemented prototype, only XY dimensions are encoded into Morton keys. Ideally, it would be better to explore the effects of encoding XYZ dimensions into Morton keys, or even time components and Level of Details in the SFC keys.

Limited functionalities

Current implementation of the SplitSFC approach only supports 2D geometry query and height query. More functionalities should be developed to support real-world application, i.e. nearest neighbour search, complex spatial analysis.

Parallel Loading

In our implemented prototype, the file is loaded one after one. The allocation for computational resources is not optimized. If the pre-processing and loading process can perform parallelly, the importing efficiency will be significantly improved.

Data transfer

Whether in data importing or retrieval process, a large volume of data is transferred back and forth in LAS files, Python and PostgreSQL. The data transfer and I/O itself is very expensive, therefore it would be better to develop some native database functions to support more algorithms.

7.2. Future Work

Adaptive splitting algorithm

The point cloud is irregularly distributed, with dense points in some places and sparse points in other places. Therefore, splitting with a fixed-size quadrant is inappropriate. The next step in the improvement of algorithm is to replace the fixed-length splitting algorithm to an adaptive splitting algorithm.

We can learn from the data structure and algorithm of the HistSFC approach to compute the distribution of the points. For example, a HistogramTree structure can be computed in the beginning to provide a guide for Morton key splitting.

3D Morton key

In this research, the splitting is only explored with a 2D Morton curve. The splitting effect of the 3D Morton curve is unknown. On the other hand, how to encode the 3D Morton key is also a question. Take AHN2 dataset for example, since the Netherlands is very flat, the data range in the Z dimension is much smaller than the X and Y dimensions. In this case, maybe the Z dimension is only encoded into the head part of the Morton key, which means it can result in a 3D Morton head and a 2D Morton tail.

Multi-split algorithm

In our research, the Morton key has a head/tail split, which means it is splitted into two parts. A more imaginative splitting is to split it into multiple sections, i.e. Morton head, Morton body, Morton tail. This can potential save more storage space.

Implementation with C++

Python is unsuitable to process massive point clouds due to its interpreted nature. To significantly improve the performance of the prototype, a C++ implementation is cruicial.

A. Description of the queries

There are seventeen queries in the benchmark. All queries are from Van Oosterom et al. [2015]. The original geometries of the queries are stored in the GitHub repository:

`https://github.com/NLeSC/pointcloud-benchmark`

Dataset	ID	Key	Area (km2)	Description
20M	1	S_RCT	0.0027	Small axis-aligned rectangle
	2	M_RCT	0.0495	Medium axis-aligned rectangle
	3	S_CRC	0.0013	Small circle, radius 20 m
	4	M_CRC	0.0415	Medium circle, radius 115 m
	5	S_SIM	0.0088	Small, simple polygon
	6	M_COM_o	0.0252	Medium, complex polygon, 1 hole
	7	M_DG_RCT	0.0027	Medium, narrow, diagonal rectangular area
210M	8	L_COM_os	0.1341	Large, complex polygon, 2 holes
	9	S_L_BUF	0.0213	Small polygon (10 m buffer in line of 11 pts)
	10	S_RCT_UZ	0.0021	Small axis-aligned rectangle, cut in max. z
	11	S_RCT_LZ	0.0051	Small axis-aligned rectangle, cut in min. z
	12	L_RCT_LZ	0.1419	Large axis-aligned rectangle, cut in min. z
2201M	13	L_L_BUF	0.0475	Large polygon (1 m buffer in line of 61 pts)
	14	L_DG_L_BUF	0.0499	Large polygon (2 m buffer in diag. line of 8 pts)
23090M	15	L_RCT	0.2342	Large axis-aligned rectangle
	16	L_RCT_N	0.1366	Large axis-aligned rectangle in empty area
	17	L_CRC	0.1256	Large circle

Table A.1.: Query description

B. Importing results for ratio experiment

Here are the benchmarking results of the ratio experiment shown in Section 6.1.3.

Solution 1: Default mode

Scales = [0.01, 0.01]

Offsets = [0, 0]

Dataset	Input ratio	Length (bits)		num_block	mean num_pts in blocks	Import Time(s)	Size (MB)
		SFC head	SFC tail				
20M (52 bits)	0.5	26	26	75	268878	119.48	116
	0.6	30	22	3097	6511	121.39	119
	0.65	32	20	12111	1665	122.48	127
210M (52 bits)	0.5	26	26	3496	60249	1395.72	2405
	0.55	28	24	6895	30548	1445.88	1205
	0.6	30	22	27089	7776	1226.33	1230

Table B.1.: The importing results of solution 1, default mode

Solution 2: Shifting mode

Scales = [0.01, 0.01]

Offsets = [min_x, min_y]

Dataset	Input ratio	Length (bits)		num_block	mean num_pts in blocks	Import Time(s)	Size (MB)
		SFC head	SFC tail				
20M (34 bits) local test)	0.15	4	30	16	1260366	185.37	119
	0.2	6	28	56	360105	201.56	120
	0.25	8	26	208	96951	200.74	120
	0.35	10	24	775	26020	200.63	120
	0.4	12	22	3036	6642	192.89	124
	0.45	14	20	11988	1682	183.22	132
	0.5	16	18	47346	426	179.14	139
	0.55	18	16	186141	108	183.87	241
210M (38 bits)	0.25	8	30	120	1755263	5141.34	1229
	0.35	12	26	1702	123755	5126.94	1232
	0.45	16	22	26903	7829	4865.18	1264
	0.5	18	20	105943	1988	5105.57	**1340**

Table B.2.: The importing results of solution 2, shrifting mode

Solution 3: resolution = 1m

Scales = [1, 1]

Offsets = [0, 0]

Dataset	Input ratio	Length (bits)		num_block	mean num_pts in blocks	Import Time(s)	Size (MB)
		SFC head	SFC tail				
20M (34 bits) local test)	0.5	18	20	3	6721954	232.26	44
	0.6	22	16	24	840244	220.96	44
	0.7	26	12	336	60017	212.28	44
	0.8	30	8	5042	4000	206.2	48
210M (38 bits)	0.5	18	20	15	14042106	2655.55	435
	0.6	22	16	180	1170176	2659.81	435
	0.7	26	12	2831	74402	2513.27	438
	0.8	30	8	43818	4807	2636.74	467

Table B.3.: The importing results of solution 1, default mode

C. Querying results for medium benchmark

The detailed querying results in the medium benchmark are described here. There are three database solutions tested for comparison. They are pgPointCloud, Oracle SDO_PC and SplitSFC in PostgreSQL. See Section 6.2 for details of the database storage model. For each query, we record the number of returned points and the query response time. Notably, the results of Q13-Q14 are not shown, because it got killed in the refinement step.

Approach	**Dataset**	**Number of points**							**Time (s)**						
		1	**2**	**3**	**4**	**5**	**6**	**7**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
pgPointCloud	20M	74947	718131	34697	563108	182930	387142	45821	0.32	2.14	0.2	1.69	0.61	1.72	0.41
	210M								0.32	2.15	0.2	1.65	0.64	1.62	0.46
	2201M								0.31	2.19	0.21	1.67	0.67	1.63	0.41
	23090M								0.32	2.19	0.21	1.68	0.68	1.68	0.44
Oracle SDO_PC	20M	74947	718131	34697	563110	182930	387145	45821	0.41	1.38	0.34	1.21	0.62	1.38	0.53
	210M								0.38	1.28	0.36	1.22	0.62	1.29	0.54
	2201M								0.39	1.36	0.36	1.23	0.6	1.33	0.5
	23090M								0.4	1.3	0.34	1.21	0.6	1.4	0.53
SplitSFC	20M	77604	720700	34623	562743	182759	387104	45848	10.75	84.34	8.1	85.19	37.74	87.87	151.15
	210M								10.82	84.1	8.14	85.33	38.12	87.5	151.35
	2201M								10.88	83.28	8.19	85.65	38.75	87.3	151.36
	23090M								31.96	82.74	8.3	83.82	39.07	87.7	151.66

Table C.1.: The querying results of Q1-Q7 queries within Delft Campus

Approach	**Dataset**	**Number of points**					**Time (s)**				
		8	**9**	**10**	**11**	**12**	**8**	**9**	**10**	**11**	**12**
pgPointCloud	210M	2273141	620392	2434	591	342952	7.03	2.26	0.19	0.22	3.31
	2201M	2273141	620392	2434	591	342952	7.13	2.17	0.19	0.23	3.32
	23090M	2273152	620932	2434	591	342952	7.23	2.33	0.19	0.22	3.32
Oracle SDO_PC	210M	2273290	620394	2434	591	342952	44.76	1.7	0.34	0.37	3.59
	2201M	2273290	620394	2434	591	342952	45.12	1.67	0.34	0.37	3.48
	23090M	2273290	620394	2434	591	342952	44.79	1.73	0.36	0.39	3.6
SplitSFC	210M	2271841	620540	2366	629	341809	673.41	454.54	7.17	10.26	356.59
	2201M	2271841	620540	2366	629	341809	677.07	451.87	9.73	10.31	357.31
	23090M	2271841	620540	2366	629	341809	670.18	452.13	9.56	10.52	362.08

Table C.2.: The querying results of Q8-Q14 in Major part of Delft

Approach	**Dataset**	**Number of points**			**Time (s)**		
		15	**16**	**17**	**15**	**16**	**17**
pgPointCloud	23090M	3992425	0	2201330	9.2	0.09	5.25
Oracle SDO_PC	23090M	3992513	0	2201333	4.52	0.29	3.13
SplitSFC	23090M	3981188	0	2202932	450.94	0.18	311.96

Table C.3.: The querying results of Q15-Q17 in South Holland

D. Software usage

The source code is stored and maintained in GitHub repository:

https://github.com/cynthiacai56/splitSFC/

The usage of all tools is simple. First, put the parameters in a json file. Then, execute the corresponding Python script.

Importer

The importer has two modes. One for single file loading, and the other for loading all files in one folder.

The command:

```
python3 importer.py --input import.json
```

The parametes in query_20m.json file:

```
{
  "config": {
    "dbname": "splitsfc",
    "user": "admin",
    "password": "aaa",
    "host": "localhost",
    "port": 5432
  },
  "imports": {
    "210m": {
      "mode": "file",
      "srid": 28992,
      "path": "/data/210m/ahn_bench000210.las",
      "ratio": 0.55
    }
    "2201m": {
      "mode": "dir",
      "srid": 28992,
      "path": "/data/2201m",
      "ratio": 0.55
    }
  }
}
```

Querying tool

The command:

```
python3 query.py --input query_20m.json
```

The parametes in import.json file:

```
{
  "config": {
          "dbname":"splitsfc",
          "user": "admin",
          "password": "aaa",
          "host": "localhost",
          "port": 5432
  },
  "queries":{
          "A1_S_RCT": {
                  "source_dataset": "20m",
                  "mode": "bbox",
                  "geometry": [85670, 85721, 446416, 446469]
          },
          "A3_S_CRC": {
                  "source_dataset": "20m",
                  "mode": "circle",
                  "geometry": [[85365, 446595], 20]
          }
  }
}
```

Exporter

The exporter cannot be used alone. You need to run querying tool with the same json file first, and then run the exporter. The querying tool first creates a table that contains all the points of the given query, then the exporter outputs a LAS file that contains all points in the querying result table.

The command:

```
python3 exporter.py --input query_20m.json
```

Bibliography

Abdullah Alfarrarjeh, Seon Ho Kim, Vinuta Hegde, Cyrus Shahabi, Qingyun Xie, Siva Ravada, et al. A class of r*-tree indexes for spatial-visual search of geo-tagged street images. In *2020 IEEE 36th international conference on data engineering (ICDE)*, pages 1990–1993. IEEE, 2020.

Jeroen Baert. Morton encoding/decoding through bit interleaving: Implementations. 2013. URL https://www.forceflow.be/2013/10/07/morton-encodingdecoding-through-bit-interleaving-implementations/.

Rudolf Bayer and Edward McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, pages 107–141, 1970.

Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331, 1990.

Howard Butler, Bradley Chambers, Preston Hartzell, and Craig Glennie. Pdal: An open source library for the processing and analysis of point clouds. *Computers & Geosciences*, 148:104680, 2021.

Jiafeng Chen, Lu Yu, and Wenyi Wang. Hilbert space filling curve based scan-order for point cloud attribute compression. *IEEE Transactions on Image Processing*, 31:4609–4621, 2022.

Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE transactions on visualization and computer graphics*, 16(4):599–608, 2010.

J. Culberson and J.I. Munro. Explaining the behaviour of binary search trees under prolonged updates: A model and simulations. *The Computer Journal*, 32(1):68–75, 1989.

Joseph Culberson and J. Ian Munro. Analysis of the standard deletion algorithms in exact fit domain binary search trees. *Algorithmica*, 5:295–311, 1986.

Rémi Cura, Julien Perret, and Nicolas Paparoditis. Point cloud server (pcs): Point clouds in-base management and processing. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2:531–539, 2015.

Rémi Cura, Julien Perret, and Nicolas Paparoditis. A scalable and multi-purpose point cloud server (pcs) for easier and faster point cloud data management and processing. *ISPRS Journal of Photogrammetry and Remote Sensing*, 127:39–56, 2017.

Raphael A Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4:1–9, 1974.

Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231, 1998.

Irene Gargantini. An effective way to represent quadtrees. *Communications of the ACM*, 25 (12):905–910, 1982.

Xuefeng Guan, Peter Van Oosterom, and Bo Cheng. A parallel n-dimensional space-filling curve library and its application in massive point cloud management. *ISPRS International Journal of Geo-Information*, 7(8):327, 2018.

Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

Martijn Meijers Xuefeng Guan Edward Verbree1 Mike Horhammer Haicheng Liu, Peter van Oosterom. Histsfc: Optimization for nd massive spatial points querying. *International Journal of Database Management Systems*, 12(3):7–28, 2020.

David Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. *Dritter Band: Analysis· Grundlagen der Mathematik· Physik Verschiedenes: Nebst Einer Lebensgeschichte*, pages 1–2, 1935.

Martin Isenburg. Laszip: lossless compression of lidar data. *Photogrammetric engineering and remote sensing*, 79(2):209–217, 2013.

ISPRS. Las 1.4 format specification (tech. rep.). 2019. URL `http://www.asprs.org/wp-content/uploads/2019/07/LAS_1_4_r15.pdf`.

Guohua Jin and John Mellor-Crummey. Sfcgen: A framework for efficient generation of multi-dimensional space-filling curves by recursion. *ACM Transactions on Mathematical Software (TOMS)*, 31(1):120–148, 2005.

Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: An improved r-tree using fractals. Technical report, 1993.

Jonathan K Lawder. *The application of space-filling curves to the storage and retrieval of multi-dimensional data*. PhD thesis, Citeseer, 2000.

Jinglan Li. manage 4d historical ais data by space filling curve. 2020.

H Liu, P Van Oosterom, M Meijers, and E Verbree. An optimized sfc approach for nd window querying on point clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 6:119–128, 2020.

Haicheng Liu. nd-pointcloud data management: continuous levels, adaptive histograms, and diverse query geometries. *A+ BE— Architecture and the Built Environment*, (12):1–206, 2022.

Martijn Meijers. Pcserve–nd-pointclouds retrieval over the web. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 10:193–200, 2022.

Guy M Morton. A computer oriented geodetic data base and a new technique in file sequencing. 1966.

WG Nulty and JJ Barholdi III. Robust multidimensional searching with spacefilling curves. In *Proceedings of the 6th International Symposium on Spatial Data Handling, Edinburgh, Scotland*, pages 805–818, 1994.

Jack A Orenstein. Multidimensional tries used for associative searching. *Information Processing Letters*, 14(4):150–157, 1982.

Styliani Psomadaki. Using a space filling curve for the management of dynamic point cloud data in a relational dbms. 2016.

Paul Ramsey. Lidar in postgresql with pointcloud. *FOSS4G, Nottingham*, 2013.

Rico Richter and Jürgen Döllner. Concepts and techniques for integration, analysis and visualization of massive 3d point clouds. *Computers, Environment and Urban Systems*, 45: 114–124, 2014.

Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

Markus Schütz. Potree: Rendering large point clouds in web browsers. 2016.

Raymond CW Sung, Jonathan R Corney, and Doug ER Clark. Octree based assembly sequence generation. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 120–129, 2001.

Jippe Van der Maaden. Vario-scale visualization of the ahn2 point cloud. 2019.

Peter van Oosterom. Spatial access methods. *Geographical information systems*, 1:385–400, 1999.

Peter Van Oosterom, Oscar Martinez-Rubi, Milena Ivanova, Mike Horhammer, Daniel Geringer, Siva Ravada, Theo Tijssen, Martin Kodde, and Romulo Gonçalves. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. *Computers & Graphics*, 49:92–125, 2015.

Peter van Oosterom, Oscar Martinez-Rubi, Theo Tijssen, and Romulo Gonçalves. Realistic benchmarks for point cloud data management systems. *Advances in 3D Geoinformation*, pages 1–30, 2017.

Peter van Oosterom, Simon van Oosterom, Haicheng Liu, Rod Thompson, Martijn Meijers, and Edward Verbree. Organizing and visualizing point clouds with continuous levels of detail. *ISPRS Journal of Photogrammetry and Remote Sensing*, 194:119–131, 2022.

Jun Wang and Jie Shan. Space filling curve based point clouds index. In *Proceedings of the 8th International Conference on GeoComputation*, pages 551–562, 2005.

Lizhe Wang, Yan Ma, Jining Yan, Victor Chang, and Albert Y Zomaya. pipscloud: High performance cloud computing for remote sensing big data management and processing. *Future Generation Computer Systems*, 78:353–368, 2018.

Peter F Windley. Trees, forests and rearranging. *The Computer Journal*, 3(2):84–88, 1960.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.