nD-PointCloud Data Management

continuous levels, adaptive histograms, and diverse query geometries

Liu, H.

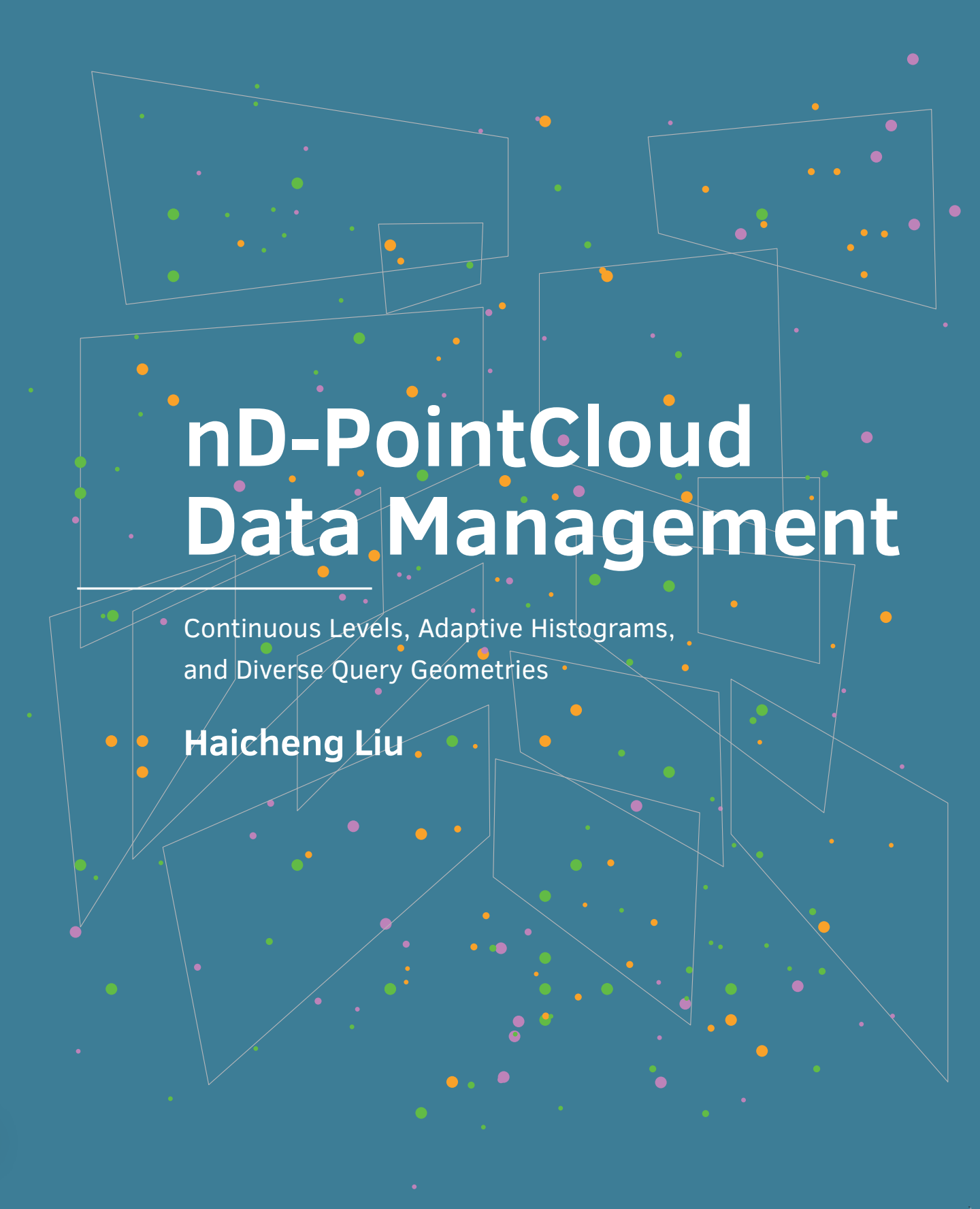
DOI
10.7480/abe.2022.12

Publication date
2022

Document Version
Final published version

Citation (APA)
Liu, H. (2022). *nD-PointCloud Data Management: continuous levels, adaptive histograms, and diverse query geometries*. [Dissertation (TU Delft), Delft University of Technology].
https://doi.org/10.7480/abe.2022.12

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.



nD-PointCloud Data Management

Continuous Levels, Adaptive Histograms, and Diverse Query Geometries

Haicheng Liu

nD-PointCloud Data Management

Continuous Levels, Adaptive Histograms, and Diverse Query Geometries

Haicheng Liu

A+BE | Architecture and the Built Environment | TU Delft BK

22#12

Design | Sirene Ontwerpers, Véro Crickx

Keywords | nD point clouds, space filling curves, continuous level of importance, histograms, skewed distributions, point set coverage, convex polytope query, flood risk mapping, AHN, benchmarks

ISBN 978-94-6366-572-8
ISSN 2212-3202

nD-PointCloud Data Management

Continuous Levels, Adaptive Histograms, and Diverse Query Geometries

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de RectorMagnificus prof.dr.ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
woensdag 22 juni 2022 om 10:00 uur

door

Haicheng LIU
Ingenieur in de Geomatics, Technische Universiteit Delft, Nederland
geboren te Jining, China

Dit proefschrift is goedgekeurd door de promotoren.

Samenstelling promotiecommissie bestaat uit:

RectorMagnificus	voorzitter
Prof. dr. ir. P.J.M. van Oosterom	Technische Universiteit Delft, promotor
Dr. ir. B.M.Meijers	Technische Universiteit Delft, copromotor

Overig lid:

Prof. dr. ir. N.C. van de Giesen	Technische Universiteit Delft
Prof. dr. E. Eisemann	Technische Universiteit Delft
Prof. dr. rer. nat. T.H. Kolbe	Technische Universiteit München
Dr. R.C. Lindenbergh	Technische Universiteit Delft
Prof. dr. ir. arch. I.S. Sariyildiz	Technische Universiteit Delft, reservelid

Onafhankelijke leden:

Dr. R. Thompson	Technische Universiteit Delft

CHINA SCHOLARSHIP COUNCIL
FUGRO
TU Delft Delft University of Technology

Why be afraid of the infinity of truth?
Joy resides in every inch of progress.

Hu Shih

ACKNOWLEDGEMENTS

In 2017, I started this PhD research. At that time, a PhD seemed like a new period of education which should be full of joy and fancy innovation. However, when looking back 5 years later, I missed half of the story — stressfulness, frustration, and anxiety. This is not only due to the challenging topic itself, but also caused by external disturbance including COVID-19 and family accidents. A strong will is needed to overcome all these, but the PhD dream can also not come true without the help of a group of people.

I would like to express deepest gratitude to my supervisors. I thank Prof. Peter van Oosterom for introducing me to the spatial data domain. Starting from the master program, you taught me to manage multidimensional array data which aroused my particular interest. Then, several years later, I got the great PhD topic — nD-PointCloud — from you. When looking back on the whole PhD, not sure if I have screwed up the original idea, but it seems that at least the eScience Center acknowledged our work and awarded us a project! More importantly, the skills and techniques I developed and learnt from you finally become my expertise. Apart from the research, we did have several nice parties that we got together to chat and celebrate. Such occasions are unmemorable, and also significant after reorganization of our department. We have been united to find the way out. I am grateful to Dr. Martijn Meijers. You have played a wonderful role in bridging Prof. Peter and me, and stood several times on my side to prevent me being overloaded. You also reduced my pressure by directly help me solve specific problems in the research. I feel very fortunate to have you as my co-promotor. I thank Ir. Edward Verbree. You influenced me a lot by your optimistic mood, and I always feel relaxed with you. By the way, you imparted “important messages" to me several times concerning my research, speeding up the whole pace.

My special thanks go to Dr. Rodney Thompson who helped me with my research in the last period of the PhD. This is the most important period because all kinds of troubles flooded in. In fact, you can fully enjoy your life after retirement, but you chose to help me and guide me through difficulties when received our request. Such generosity also makes me ponder if I should also do this when retired. I would also like to acknowledge Dr. Xuefeng Guan for the reception of my research visit to Wuhan University. You helped me a lot program the nD-histogram which is the first innovation of my PhD research. I also express my gratefulness to Dr. Dongliang Peng, Dr. Jianping Wang and Drs. Marian de Vries for stimulating ideas and solving practical issues during the research. It is my great pleasure to discuss with you.

Last but not least, I would like to express my sincere acknowledgment to my Dad and Mom. Without your support all the way, I could never realize a PhD dream abroad. The countless calls we made overseas are full of joy, praise and love. You raise me up and push me forward to the success today. Besides, I am very proud to become the first Doctorate from both my Dad’s family and my Mom’s, and the degree is awarded by a world prestigious university. I also thank my girl friend for accompanying me through

the toughest period of PhD. The difficulties and hardships finally made us grow. There are more people providing essential help to me during this journey, and I thank you all.

Haicheng Liu
May 24, 2022, Delft

CONTENTS

ACRONYMS

AIS Automatic Identification System

ALS Airborne Laser Scanning

API Application Programming Interface

AR Augmented Reality

BLOB Binary Large OBject

CD Cumulative Density

CDF Cumulative Distribution Function

CHC Cumulative Hypercubic Coverage

cLoI Continuous Level of Importance

DBMS DataBase Management System

DDL Data Definition Language

DML Data Manipulation Language

DIM Dense Image Matching

dLoI Discrete Level of Importance

FLANN Fast Library for Approximate Nearest Neighbors

FPN False Positive Node

FPS Frames Per Second

GIS Geographical Information System

GNSS Global Navigation Satellite System

GPS Global Positioning System

ICDF Inverse Cumulative Distribution Function

IDG Ideal Data Group

IOT Index Organized Table

kNN k Nearest Neighbour

LiDAR Light Detection And Ranging

LoD Level of Detail

MBB Minimum Bounding Box

MBES MultiBeam EchoSounder

MBR Minimum Bounding Rectangle

PDAL Point Data Abstraction Library

PDF Probability Density Function

RDG Realistic Data Group

RGB Red, Green, and Blue

SFC Space Filling Curve

SRS Spatial Reference System

TPN True Positive Node

VR Virtual Reality

SUMMARY

In the Geomatics domain, a point cloud refers to a data set which records the coordinates and other attributes of a huge number of points. Conceptually, each of these attributes can be regarded as a dimension, representing a specific type of information. Apart from routinely concerned spatio-temporal dimensions for coordinates, other dimensions such as intensity and classification are also widely used in spatial applications. In fact, more dimensions can be involved. For instance, a point in the hydraulic modelling grid also records the flow direction, speed, sediment concentration, and other related attributes. As these point cloud data can be directly collected, computed, stored and analyzed, this thesis proposes the term — nD-PointCloud, as a general spatial data representation to cover them.

At present, drastically increasing production of nD-PointCloud data raises essential demand for smart and highly efficient data management and querying solutions. However, we lack effective tools. Prevalent software for nD-PointCloud processing, analyzing and rendering are built on file-based systems, requiring substantial development of data structures and algorithms. To make things worse, when other data types are involved, multiple formats, libraries and systems need enormous effort to be integrated. Aimed at generic support for diverse applications, DataBase Management Systems (DBMSs) on the other hand avoid these issues to a large extent. However, since they are initially developed to resolve 2D or 3D issues, they do not provide native support for nD data indexing and operations. Yet the 2D and 3D operators cannot be easily extended to nD.

This thesis aims at developing a generic yet efficient solution for managing and querying nD-PointCloud data. The work is based on an existing solution called PlainSFC, which maps nD data into 1D space. PlainSFC is implemented in the DBMS, adopting space filling curve based clustering and B+-tree indexing strategies. Besides, PlainSFC applies an advanced querying mechanism which refines hypercubic nD spaces to 1D ranges recursively to approach the query geometry for primary filtering. This achieves high querying efficiency. However, the solution still has drawbacks, and this research focuses on resolving them by developing and using novel methods:

- A continuous Level of Importance (cLoI) method for data organization to eliminate visual artifacts of density shocks in points' rendering, which is introduced by conventional tree structures such as Quadtree or Octree. The cLoI method computes an importance value for every point according to an ideal distribution generalized from the discrete distributions of those tree structures. This forms an additional cLoI dimension, and each point actually represents a level. By integrating the cLoI dimension into PlainSFC, smooth and efficient rendering is realized.

- An nD-histogram approach to improve querying efficiency on non-uniformly distributed data. PlainSFC decomposes the nD space into sub-spaces recursively to

approach the query geometry without considering point distribution. This is not optimal when the distribution of points is severely skewed. To improve this, an nD-histogram which records the number of points inside each nD sub-space is established as a representation of data distribution. The developed solution called HistSFC decomposes and refines the nD space more smartly, which improves the accuracy and efficiency of primary filtering.

- A convex polytope querying function. Besides orthogonal window queries, the polytope query, which is the extension of the widely adopted polygonal query in 2D, also plays a critical role in many nD spatial applications. To address this type of query, an easy-to-use polytope formulation for querying is firstly proposed. Then, based on PlainSFC and HistSFC, efficient intersection algorithms are developed for convex polytope querying on nD point clouds. These algorithms are tested through experiments with up to 10D point data. Using this newly developed function, applications including perspective view selections and flood risk queries are resolved more efficiently, achieving sub-second performance.

Additionally, other optimization techniques such as parallelization are developed and experimented with, which also bring performance gain. To verify the whole framework, several benchmark tests devised by considering real applications are conducted, and comparisons with different state-of-the-art solutions are performed. The result shows that the newly developed solution outperforms the others, overall. In certain cases, the solution can be applied without further optimizations. However, this will not be the end. Rapidly arising high tech such as cloud computing platforms can boost the solution further to incorporate more data and users. Potential nD-PointCloud based applications still need to be explored, prototyped and tested to serve the society in practice.

SAMENVATTING

In het Geomatics domein verwijst een puntenwolk naar een dataset die de coördinaten en andere attributen van een groot aantal punten vastlegt. Conceptueel kan elk van deze attributen worden beschouwd als een dimensie, die een specifiek type informatie vertegenwoordigt. Naast de routinematig betrokken ruimtelijk-temporele dimensies voor coördinaten, worden ook andere dimensies, zoals intensiteit en classificatie, veel gebruikt in ruimtelijke toepassingen. In de praktijk kunnen er meer dimensies betrokken zijn. Een punt in een hydraulisch modelleringsraster registreert bijvoorbeeld ook de stroomrichting, snelheid, sedimentconcentratie en andere gerelateerde attributen. Aangezien deze puntenwolk gegevens direct kunnen worden verzameld, berekend, opgeslagen en geanalyseerd, stelt dit proefschrift de term nD-PointCloud voor als een generieke representatie van ruimtelijke gegevens.

Op dit moment verhoogt de drastisch toenemende productie van nD-PointCloud-gegevens de essentiële vraag naar slimme en zeer efficiënte oplossingen voor gegevensbeheer en query's. Het ontbreekt echter aan effectief digitaal gereedschap. Veelvoorkomende software voor nD-PointCloud-verwerking, -analyse en -weergave is gebouwd op bestandsgebaseerde systemen, waarvoor een aanzienlijke ontwikkeling van gegevensstructuren en algoritmen nodig is. Om het nog erger te maken, als er andere gegevenstypen bij betrokken zijn, vergen meerdere formaten, bibliotheken en systemen een enorme inspanning om te worden geïntegreerd. Gericht op generieke ondersteuning voor diverse toepassingen, vermijden Database Management Systemen (DBMS'en) daarentegen deze problemen grotendeels. Omdat ze echter in eerste instantie zijn ontwikkeld om 2D- of 3D-problemen op te lossen, bieden ze geen directe ondersteuning voor nD-gegevensindexering en -bewerkingen. Eveneens kunnen de 2D-operatoren en 3D-operatoren niet eenvoudig worden uitgebreid tot nD.

Dit proefschrift heeft tot doel een generieke maar efficiënte oplossing te ontwikkelen voor het beheren en opvragen van nD-PointCloud gegevens. Het werk is gebaseerd op een bestaande oplossing genaamd PlainSFC, die nD gegevens in 1D-ruimte in kaart brengt. PlainSFC is geïmplementeerd in een DBMS, waarbij gebruik wordt gemaakt van op ruimte vullende curve gebaseerde clustering en B+-tree indexeringsstrategieën. Bovendien past PlainSFC een geavanceerd query mechanisme toe dat recursief hypercubische nD-ruimten verfijnt tot 1D-bereiken om de query geometrie voor primaire filtering te benaderen. Hierdoor wordt een hoge query-efficiëntie bereikt. De oplossing heeft echter nog steeds nadelen, en dit onderzoek richt zich op het oplossen ervan door nieuwe methoden te ontwikkelen en te gebruiken:

- Een continue Level of Importance (cLoI)-methode voor gegevensorganisatie om visuele artefacten van dichtheidsschokken in de weergave van punten te elimineren, die wordt geïntroduceerd door conventionele boomstructuren zoals Quadtree of Octree. De cLoI-methode berekent een belangrijkheidswaarde voor elk punt

volgens een ideale verdeling gegeneraliseerd uit de discrete verdelingen van die boomstructuren. Dit vormt een extra cLoI-dimensie en elk punt vertegenwoordigt in feite een niveau. Door de cLoI-dimensie te integreren in PlainSFC, wordt een soepele en efficiënte weergave gerealiseerd.

- Een nD-histogrambenadering om de query-efficiëntie op niet-uniform gedistribueerde gegevens te verbeteren. PlainSFC ontleedt de nD-ruimte recursief in subruimten om de query geometrie te benaderen zonder rekening te houden met puntdistributie. Dit is niet optimaal wanneer de puntenverdeling ernstig scheef is. Om dit te verbeteren, wordt een nD-histogram opgesteld dat het aantal punten binnen elke nD-subruimte registreert als een weergave van de gegevensdistributie. De ontwikkelde oplossing genaamd HistSFC ontleedt en verfijnt de nD-ruimte slimmer, wat de nauwkeurigheid en efficiëntie van primaire filtering verbetert.

- Een convexe polytoop bevragingsfunctie. Naast orthogonale window query's, speelt de polytope-query, die de uitbreiding is van de algemeen aanvaarde polygon query in 2D, ook een cruciale rol in veel nD-ruimtelijke toepassingen. Om dit type zoekopdracht aan te pakken, wordt eerst een gebruiksvriendelijke polytoopformulering voor zoekopdrachten voorgesteld. Vervolgens worden, op basis van PlainSFC en HistSFC, efficiënte intersectie-algoritmen ontwikkeld voor convexe polytope-query's op nD-puntenwolken. Deze algoritmen worden getest door middel van experimenten met maximaal 10D-puntgegevens. Met behulp van deze nieuw ontwikkelde functie worden toepassingen, waaronder selecties van perspectiefweergaven en vragen over overstromingsrisico's, efficiënter opgelost, waardoor prestaties van minder dan een seconde worden bereikt.

Daarnaast worden andere optimalisatietechnieken zoals parallellisatie ontwikkeld en geëxperimenteerd, die ook prestatiewinst opleveren. Om het hele raamwerk te verifiëren, worden verschillende benchmarktests uitgevoerd die zijn opgesteld door rekening te houden met echte toepassingen, en worden vergelijkingen gemaakt met verschillende state-of-the-art oplossingen. Het resultaat laat zien dat de nieuw ontwikkelde oplossing over het algemeen beter presteert dan de andere. In bepaalde gevallen kan de oplossing zonder verdere optimalisaties worden toegepast. Dit zal echter niet het einde zijn. Snel opkomende high-tech oplossingen, zoals cloud computing-platforms, kunnen de oplossing een boost geven om meer gegevens op te nemen en meer gebruikers te bedienen. Potentiële op nD-PointCloud gebaseerde applicaties zullen nog verder moeten worden onderzocht, door prototypes te maken en deze te testen en zo de samenleving in de praktijk te dienen.

1

INTRODUCTION

THIS chapter starts by briefly introducing the motivation of this PhD research in Section 1.1. This is then followed by two representative applications which provide readers with a first impression on nD point clouds. They are described in Section 1.2 and Section 1.3. Based on this, Section 1.4 summarizes the issues of state-of-the-art solutions for nD point data management and querying in general. Then, a potential solution is briefly introduced in Section 1.5. This solution also forms the basis for this PhD research. After this, Section 1.6 presents research questions and methodology, while Section 1.7 shows outline of the thesis.

1.1. MOTIVATION

Point clouds are increasingly used in spatial related domains, to name a few: terrain modelling, forest estimation, trajectory analysis and Virtual Reality (VR) (Figure 1.1). Point cloud data features in huge volume. The most commonly used point clouds are collected by Light Detection And Ranging (LiDAR) sensors, containing up to trillions (10^{12}) of points. Besides, point clouds record multidimensional information. Apart from routinely concerned spatio-temporal dimensions, other dimensions such as intensity and classification also constitute indispensable part of the data. In specific fields, points may carry even more information. For instance, in hydraulic modelling, a point may also record the flow direction and speed, sediment concentration, and other dimensions.

These various dimensions are jointly used to support different applications. Take indoor navigation in a VR environment as an example, it is sufficient to only show important objects along the route to avoid excessive data loading. This can be realized using a customized dimension which represents importance of objects. Besides, people should be able to see things through windows and go through doors. The windows and doors are recognizable in a classified point cloud. Then, a query concerned with XYZ, importance and classification will form the query for navigation. As information continues to grow, more dimensions are expected in queries. Generally, I call them nD queries. In past decades, nD queries have caused significant bottlenecks in practice.

1

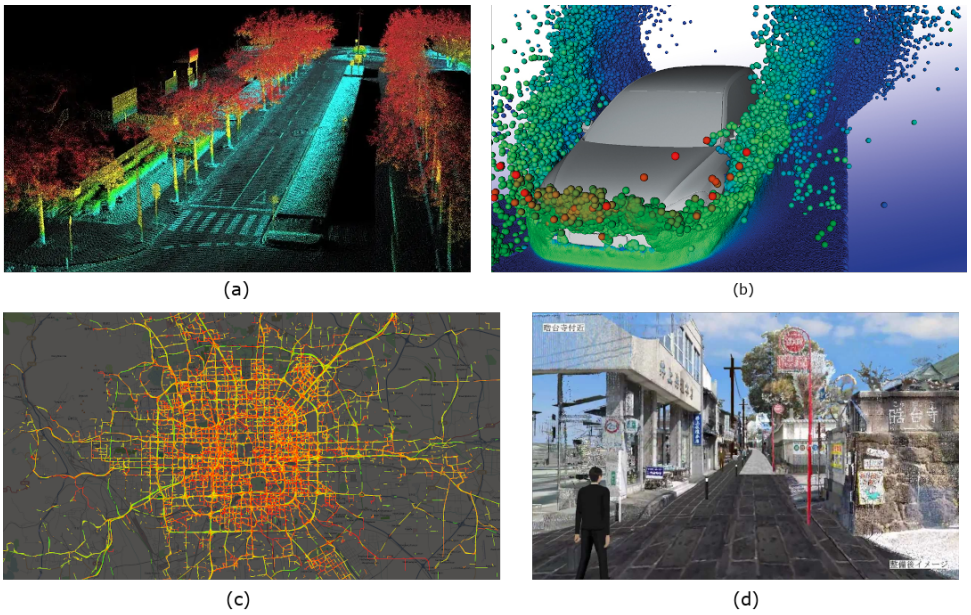


Figure 1.1: nD point clouds for different applications: (a) 3D visualization (b) fluid modelling (c) GPS trajectory analysis (d) VR. (Image sources: (a) OxTS (b) Fraunhofer ITWM (c) Topometries (d) Forum 8)

This PhD research aims to develop an efficient solution to address these nD queries. The core lies in devising efficient data structures and querying algorithms. Existing point data structures normally use routinely concerned spatio-temporal dimensions — XYZT — to cluster and index the data. I call these dimensions the *organizing dimension*. The other type is the *property dimension* which is not frequently queried, such as intensity of reflected laser signals and color expressed by Red, Green, and Blue (RGB) values. They are affiliated to the organizing dimensions, providing additional information. Depending on applications, these two types of dimension are interchangeable. The following Section 1.2 and 1.3 present two representative applications, which provide first impression of different types of dimensions. Both of them show how imperative it is to develop a real nD point cloud solution.

1.2. FLOOD RISK QUERYING

Flood risk mapping projects generate huge amount of modelling data to assess the flood risk. The mapping process mainly includes two parts. The first part concerns running a 1D and 2D coupled hydrodynamic model to compute water depth, flow velocity and direction at different time steps, given a specific breach case. The model stores results in a 2D grid covering the modelling basin (Figure 1.2). The modelling results are then used for making various maps such as the maximum inundation map and inundation duration map, in a following step. Also, engineers compute potential loss tables by combining flood maps with social economic data. The water authorities collect these final products, and use them for decision making. However, they omit a large part of original

modelling results which are cumbersome to manage, analyze and present. This certainly has drawbacks: the products are “static" (Figure 1.3) and no more details can be derived; the maps fail to address new requirements, which significantly confines the use.

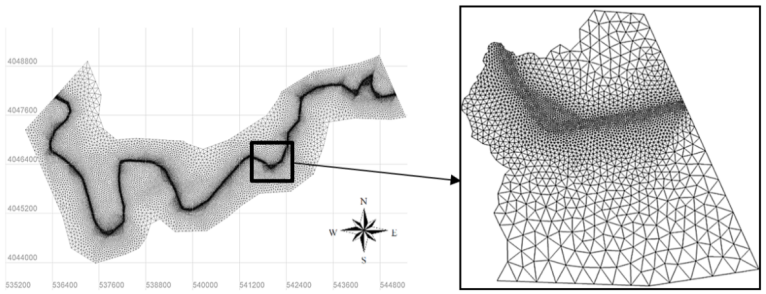


Figure 1.2: A typical flood modelling grid (Gharbi et al., 2016)

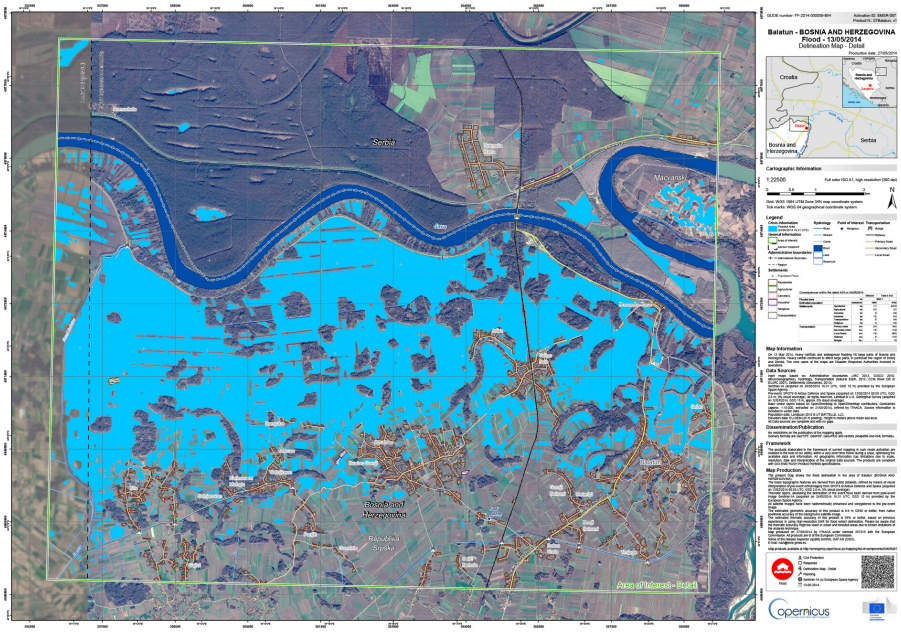


Figure 1.3: A typical flood map. Image source: ESA

In fact, any specific flood map can be expressed and formed by a type of query (Liu, Oosterom, et al., 2021). For example, the inundation extent map can be generated by selecting all the grid cells with water depth greater than 0, while the arrival time map can be generated by selecting cells at different time steps that have been flooded. In addition, new requirements such as flood situation around certain objects can also be resolved by using specific queries. Hence, to address issues mentioned above, developing an ef-

1

ficient database for ad-hoc queries is imperative. Due to the irregular grid (Figure 1.2), data storage and querying in the form of rasters would be cumbersome and inefficient. A possible solution is to extract the centroids of all cells and store the attributes including flow velocity, direction and inundation depth in these centroids. These attributes can either be used as the property dimension or organizing dimension for data management. Flood risk analysis can then be performed by querying this nD point cloud database using all these relevant dimensions besides XYZ.

1.3. TOPOGRAPHIC VISUALIZATION

Both industry and academia have already experienced technical bottlenecks in visualizing large point clouds. For instance, the Fugro company has established an inspiration center to introduce VR services to users. The VR service builds scenes directly on massive points. However, due to limited memory capacity, the VR headset can only store small number of points, which confines the size of the scene. If more points were inserted, the headset would crash. By adopting prevalent indexing structures such as Quadtree (Figure 2.8) or Octree, dynamic buffering can be achieved. When zooming in, which means the eye moves closer to the points, more details can be seen. So, the headset will render points stored at lower levels of Quadtree (Figure 2.8) or Octree. Meanwhile, blocks outside the view will be purged out from the memory. With this data structure, large scenes can be built. However, this structure has a side effect, the “density shock" (van Oosterom et al., 2017): sharp boundaries belonging to blocks at different levels can be seen (Figure 1.5). This is because during the rendering, blocks with different densities are shown in the same scene due to different distance to the view point. Such discrete densities will also cause issues for related analysis.

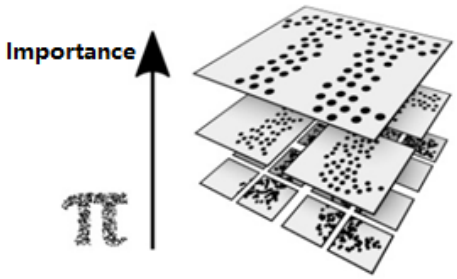


Figure 1.4: A Quadtree structure (Schütz et al., 2020): a block at an upper level indexes four blocks at lower levels, while the number of points inside any block is the same. More important points reside at an upper level, known as level of importance. Octree applies a similar structure, but is used for 3D data, with each block covering eight child blocks.

To solve the problem caused by the discrete Level of Importance (dLoI) implied in the Octree, we should develop “continuous” levels for data organization (van Oosterom, 2019). In a continuous Level of Importance (cLoI) structure, every point is assigned an importance value indicating its ranking among the whole data set. Using it to select points can make the rendering more smooth and natural, e.g., the bird’s-eye view shown in Figure 1.6. The cLoI may thus be treated explicitly as an organizing dimension in the

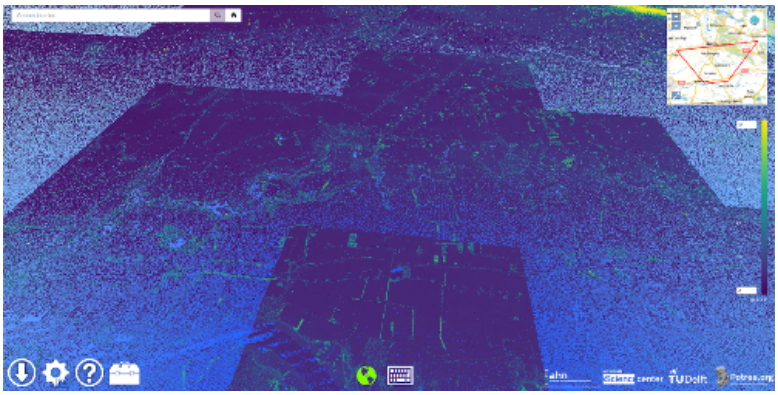


Figure 1.5: The density shock introduced by Octree

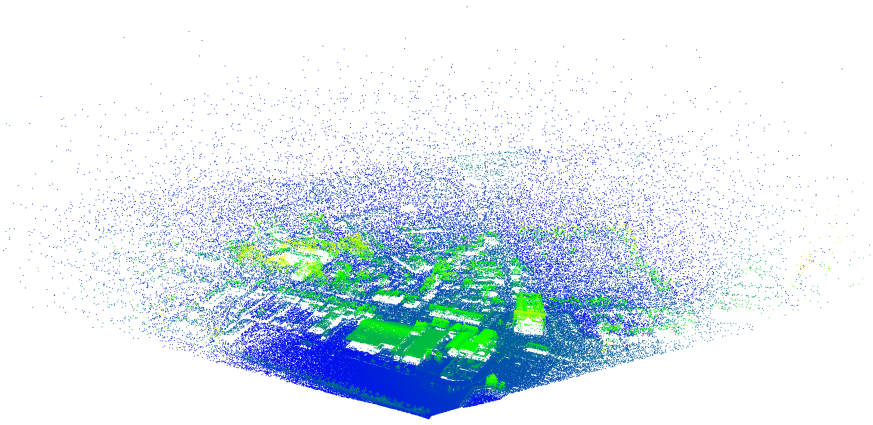


Figure 1.6: Visioning a scene based on cLoI: no density shocks

data storage, to guarantee efficient selections.

The cLoI value should be computed reasonably to support functionalities. Schütz et al. (2019) used a uniform random sampling approach by only considering visual effects, while van Oosterom (2019) proposed to use an exponential distribution to model cLoI. Classification may also get involved in cLoI computation if more advanced requirements are posed such as indoor navigation (Liu et al., 2018a). Different cLoI design will also have an influence on the querying efficiency, which is another essential aspect to consider. To conclude, introducing appropriate organizing dimensions into the data organization is needed to efficiently support nD applications.

1.4. PROBLEMS OF STATE-OF-THE-ART SOLUTIONS

State-of-the-art solutions fail to handle nD point data efficiently (Liu et al., 2018b). Prevalent software for point processing and visualization are based on files. Some are directly

constructed on LAS/LAZ[1] files, HDF[2] files, while others adopt vendor customized formats (Section 2.4). Specific data structures with additional sorting and indexing support have to be developed. With available Application Programming Interfaces (API), developers may program using script languages such as Python to resolve specific tasks in a short time. However, when a new function is required, redevelopment or additional development must be performed. Yet the scalability cannot be guaranteed. Besides, when other data types such as rasters are involved, multiple formats, libraries and systems need enormous amount of effort to be integrated.

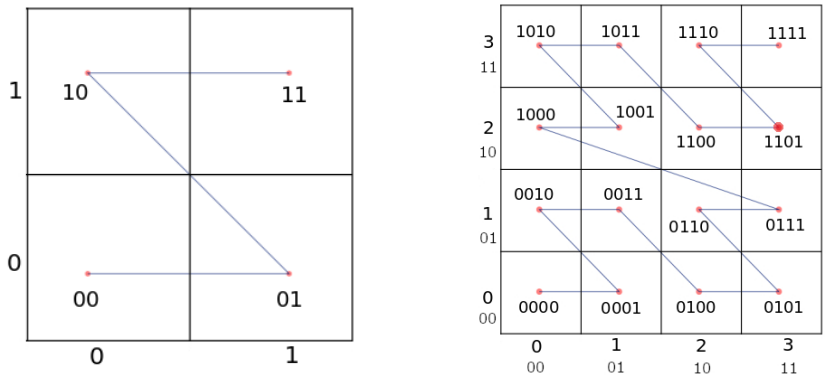
Aimed at generic purposes, Database Management Systems (DBMS) addressed these issues to a large extent. Oracle and PostgreSQL offer state-of-the-art flat table and block based solutions for managing point clouds (Section 2.5). Flat table normally stores each point as a record, and uses one column for each dimension; the block approach groups points into blocks and then indexes these blocks in a base table. It is possible to build a B-tree index on one or several columns of flat tables, but it is only favorable for limited types of queries. If the user selects non-indexed columns, the execution would most likely become much inefficient. The block approach first utilizes spatial indexes such as the R-tree to locate blocks that intersect the query geometry. Then it unpacks the blocks on the query boundary for filtering to achieve accuracy at the individual point level. However, if the blocks do not fit well within the query geometry, the process can be very slow due to the unpacking and filtering processes. To make things worse, there is only support for a limited number of dimensions to be indexed, e.g., at most 3 dimensions for Oracle SDO_PC (Oracle, 2019) and 4 dimensions with PostgreSQL (Strobl, 2008).

1.5. POTENTIAL SOLUTION: FROM nD TO 1D

Some previous studies proposed to organize point data using Space Filling Curves (SFCs) (Section 2.3.1). A SFC is a curve that passes by all basic units in a multidimensional space only once. It preserves spatial relationships of objects it covers and has been widely used to cluster and index spatial data (Lawder, 2000a). Morton curve (also called Z-order or N-order curve) is a representative that has been commonly studied and practiced due to the simplicity of mapping functions (Morton, 1966). It is based on interleaving the bits from the coordinates. For example, given a point with coordinates (3,2), its binary representation is (11, 10). By interleaving these bits, the Morton key 1101 can be derived, which is the 14^{th} node on the curve (Figure 1.7b). Points are then sorted according to the SFC codes to be grouped together, while their spatial relationship still retains.

SFC approaches have been adapted and improved for point data management (Wang & Shan, 2005; Zhang et al., 2014). Specifically, van Oosterom et al. (2015) presented a prospective SFC mapping-based clustering and indexing framework, which I will call PlainSFC, for the sake of convenient referencing. Basically, PlainSFC maps both multidimensional points and query geometries into the one-dimensional SFC space so that a one-dimensional indexing structure such as the B+-tree can be used. In theory, the time complexity for querying is low. PlainSFC has been studied since then (Martinez-Rubi

[1] https://www.asprs.org/divisions-committees/lidar-division/laser-las-file-format-exchange-activities
[2] https://www.hdfgroup.org/



(a) First order

(b) Second order

Figure 1.7: 2D Morton curves

et al., 2015; Psomadaki, 2016; Guan et al., 2018; Meijers & van Oosterom, 2018). All the research demonstrates the superiority of PlainSFC for managing and querying spatial points within 4D, but how it performs in higher dimensional spaces is still unknown. Besides, the evaluation of PlainSFC's performance was limited. How different factors such as data distribution, query type and configuration of PlainSFC affect the performance is not revealed.

1.6. RESEARCH QUESTIONS AND METHODOLOGY

After introducing the problems faced by current solutions including nD point data organizing, indexing and querying, the thesis proposes the main research question and the following sub-questions.

What is a highly efficient nD point cloud data structure supporting different types of applications?

1. To what extent can the transformation between the organizing dimensions and the property dimensions facilitate the management and query of point clouds? How to determine the type of dimensions when managing the data?

2. What is the role of continuous Level of Importance (cLoI) in managing nD point clouds? How to compute the cLoI value for each point?

3. How much does the point distribution influence the performance of query execution?

4. Besides the common orthogonal window queries on different organizing dimensions, what other query geometries are needed and can be efficiently handled by the data structure?

5. What queries and processes should be included in a benchmark to learn the balanced performance of the data structure? How to set up a representative benchmark?

The research aims at developing a highly efficient nD point data structure supporting diverse spatial-related applications. Figure 1.8 presents the overall methodology of the PhD research. By literature study and software experiment, the research identifies and focuses on addressing three issues of PlainSFC and other existing solutions. First, to eliminate density shocks caused by dLoI, the research investigates the mathematical approach to convert dLoI to cLoI representation so that smooth and fast rendering of points can be achieved. Second, the design of PlainSFC does not take account of non-uniform point distribution, while in practice, this can significantly influence the querying performance. Thus, the research explores the possibility of using nD-histogram techniques to improve PlainSFC. The theoretical foundation is laid out to analyze the effectiveness of the nD-histogram under different circumstances, e.g., different dimensionality and distributions. Third, the research resolves point selection with irregular query geometries. Next to the widely used orthogonal nD windows, other nD query geometries which are derived from 2D or 3D primitives (e.g., nD-ball or nD-triangle) can also be frequently used for selecting points. To generally represent and approximate these query geometries, the research develops and uses the nD-polytope model delimited by a set of half-spaces. nD-polytope querying method based on PlainSFC is then proposed and implemented.

By integrating these novel techniques into PlainSFC, the new solution is expected to outperform state-of-the-art solutions significantly, and scale logarithmically with the increase of data size. To verify this, the final phase of the research is to establish a comprehensive benchmark to test the solutions. The benchmark should be devised considering various use cases based on different data sets such as LiDAR data and trajectory data. In this process, it is likely that my solution does not initially perform as is expected. Then, bottlenecks will be identified. Optimizations and improvements should be made in the designing and implementing phase, and verification will be conducted once more. This process iterates until the three main issues are solved and fine scalability is achieved.

On the other hand, the following aspects have been limited or excluded in this research:

- Development of advanced rendering algorithms for enrichment of point clouds
- Algorithms of point registration, segmentation, classification and identification
- Implementation on distributed platforms, instead of a standalone server, for scaling out the core solution

1.7. OUTLINE

The rest of the thesis is organized as follows: Chapter 2 provides essential background knowledge for understanding the thesis. Chapter 3 develops a novel cLoI method for point clouds based on recursive refinement of discrete levels. Essential mathematical

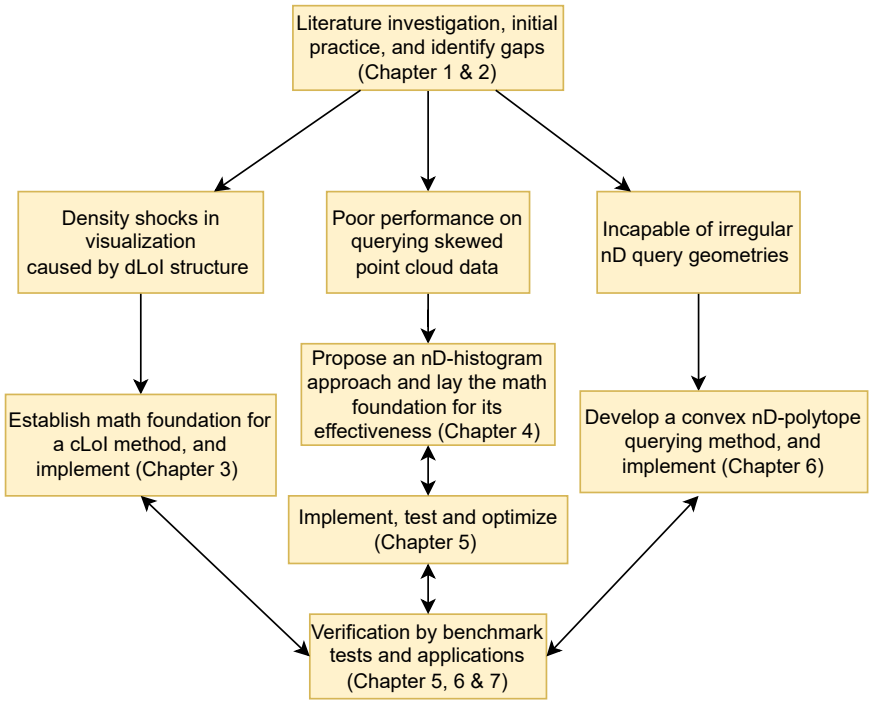


Figure 1.8: Methodology of this PhD research, linking to different chapters

details are provided. The computed cLoI dimension can then be integrated to PlainSFC to realize smooth and fast visualization of points, eliminating density shocks. Chapter 4 devises an nD-histogram technique to address poor performance of PlainSFC on querying non-uniformly distributed data. Statistical concepts and theorems are established to interpret the effectiveness of the nD-histogram with respect to different point distributions. Besides, experiments based on synthetic data are conducted to verify the theorems. Then, Chapter 5 verifies the nD-histogram technique with real use cases. The performance of this improved solution on window queries is evaluated by comparing to PlainSFC and other state-of-the-art solutions. The chapter also makes further optimizations concerning the data organization and querying process to improve efficiency, e.g., parallelization. To address irregular query geometries, Chapter 6 develops the querying algorithm for a different class of query geometry — convex nD-polytope. It serves as a expressive and flexible solution for non-orthogonal query geometries. Benchmark tests are performed using both synthetic data and real world data to verify this new functionality. Chapter 7 further exploits the improved solution for more applications such as indoor visualization, change detection and trajectory extraction. More functionalities are developed in this chapter and convincing preliminary results are acquired. Chapter 8 concludes the whole thesis, and describes future work. Table 1.1 lists the publications resulting from this PhD research, and the relationship to different chapters.

1

Table 1.1: PhD publications, referenced by different chapters

No.	Publications ordered by date	Ch.
1	van Oosterom, P., van Oosterom, S., Liu, H., Thompson, R., Meijers, M., & Verbree, E. (2022). Vario-scale distribution of point clouds over continuous levels of detail. *ISPRS Journal of Photogrammetry and Remote Sensing*, major revision.	3
2	Liu, H., Thompson, R., van Oosterom, P., & Meijers, M. (2021). Executing convex polytope queries on nD point clouds. *International Journal of Applied Earth Observation and Geoinformation*, 105, 102625.	6
3	Liu, H., van Oosterom, P., Mao, B., Meijers, M., & Thompson, R. (2021). An efficient nD-point data structure for querying flood risk. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B4-2021, 367-374.	5, 6
4	Liu, H., van Oosterom, P., Meijers, M., & Verbree, E. (2020). An optimized SFC approach for nD window querying on point clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, VI-4/W1-2020, 119-128.	4
5	Liu, H., van Oosterom, P., Meijers, M., Xuefeng, G., Verbree, E., & Horhammer, M. (2020). HistSFC: Optimization for nD Massive Spatial Points Querying. *International Journal of Database Management Systems*, 12(3), 7-28.	4, 5
6	Zhang, L., van Oosterom, P., & Liu, H. (2020). Visualization of point cloud models in mobile augmented reality using continuous level of detail method. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIV-4-W1-2020, 167-170.	7
7	Liu, H., Xuefeng, G., Meijers, M., & van Oosterom, P. (2019). The design and application of histogram trees for querying massive LiDAR point clouds [in Chinese]. In *Proceedings of 5th China LiDAR Conference*, (p. 8).	4, 5
8	Liu, H., van Oosterom, P., Meijers, M., & Verbree, E. (2018). Management of large indoor point clouds: an initial exploration. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-4, 365-372.	7
9	Liu, H., van Oosterom, P., Meijers, M., & Verbree, E. (2018). Towards 10^{15}-level point clouds management - a nD PointCloud structure. In *Geospatial Technologies for All: Short papers, posters and poster abstracts of the 21th AGILE Conference on Geographic Information Science*, (p. 7).	1, 2

2

BACKGROUND KNOWLEDGE

THIS chapter provides general background knowledge to comprehend the thesis. Section 2.1 introduces the object of this research — nD-PointCloud. Then, Section 2.2 presents previous studies on Level of Importance (LoI), a critical dimension that can be leveraged to manage huge number of points. Sections 2.3 to 2.5 review point data accessing methods, file based solutions and DBMSs developed for point data management. Section 2.6 describes the potential solution — PlainSFC, which is a basis for the following chapters.

2.1. nD-PointCloud

A point cloud generally refers to a set of points in 3D space. We can distinguish each point by its XYZ coordinates. A point may also contain other attribute information, such as color, intensity and classification. Conceptually, attribute and dimension are equivalent terms, both representing a specific type of information. This thesis adopts the dimension term, and uses nD-PointCloud to cover the point cloud data.

I propose nD-PointCloud as the third spatial data representation, besides the vector and the raster. Unlike the point or the multi-point which is a vector geometry, nD-PointCloud can be directly collected, structured, stored, interpreted and analyzed, which forms a new paradigm. That is, many applications can be addressed with only nD-PointCloud. A significant advantage of nD-PointCloud lies in the ultra high accuracy which may be decreased when converting to rasters. Besides, an nD-point is still intuitive to interact with and convenient for computation. In contrast, nD vector representation including nD-polygon or nD-polyline can cause very expensive processing for computations such as intersection.

2.1.1. nD-PointCloud from different sources

This section presents 3 major sources of nD-PointCloud, which are also used in the research. They are Light Detection And Ranging (LiDAR) systems, navigation systems and conversion from other data types.

LIDAR SYSTEMS

A LiDAR scanner emits laser pulses and receives signals reflected by objects. In this way, the 3D coordinates of a point on the object can be derived based on the emitting angle, time duration and location of the scanner. Since the introduction of the LiDAR technology in the 1960s, the volume of point cloud data has seen a rapid increase. This growth mainly results from the developments of all kinds of sensors: mobile, terrestrial, airborne and satellite laser systems. Such systems aim to acquire and build an accurate model of the 3D environment based on points. To keep data updated, repeated scans of the same area are normally conducted at a period of days, months or years depending on the applications. For example, the “sand engine" project (Stive et al., 2013) scans a part of dutch coastline every day to detect changes, to investigate how nature spreads the sand for beach nourishment along the coast as time goes by. Another instance is the AHN data set (AHN, 2014), which is a detailed elevation model of the whole Netherlands measured by Airborne Laser Scanning (ALS). AHN is updated around every 5 years. The second mission collected 640 billion points (Wijga-Hoefsloot, 2012), and the third and fourth missions have also been conducted. In summary, nD-PointCloud acquired this way features in large volume and relatively low update frequency. The dimensions involved, apart from the XYZT, are mostly related to the specific LiDAR system. These include intensity, number of returns and flight angle for example.

Since laser signals can hardly penetrate the water due to absorption, MultiBeam EchoSounders (MBESs) are employed to acquire the topography of the river or sea bed. MBES utilizes sonar instead of laser to detect objects beneath water.

NAVIGATION SYSTEMS

Positioning techniques such as Global Navigation Satellite System (GNSS) (Hofmann-Wellenhof et al., 2007), Automatic Identification System (AIS) (Bhattacharjee, 2017) and indoor positioning system (Xu et al., 2013) collect users' positions in real time. The final result consists of a large number of trajectories composed by the points. These trajectories are then used for analyzing and mining mobility patterns, traffic and social networks (Zheng et al., 2010). As a piece of spatio-temporal information, the trajectory is normally modeled by 3D polylines in XYT space (Wang et al., 2021). However, an nD-PointCloud representation is also possible by adding an additional trajectoryID dimension to each point of the trajectory. Other motion parameters including speed, acceleration and direction can also be incorporated as separate dimensions (Ye & Ai, 2017). Using nD-PointCloud can accelerate several critical operations including kNN, Reverse k-Nearest-Neighbour search (RkNN) and spatial-textual search (Wang et al., 2021). In other cases, a full trajectory may be needed. Then, a mapping table with points indicating the composition of the trajectory can be additionally defined and used. The trajectory point are updated very frequently, within seconds or minutes.

CONVERSION FROM OTHER DATA TYPES

Other than directly collected from sensors, nD-PointCloud data can be acquired by transforming other data types. Dense Image Matching (DIM) is a computer vision technique to extract 3D point clouds from optical images (Figure 2.1). Conventional photogrammetry extracts 3D geometries by matching the same feature from different images, based on

tie points. However, many pixels such as the surface of a road or a green patch of vegetation may not be recognised as specific features, so feature-based algorithms are less favourable for extracting dense point clouds (Remondino et al., 2013; Kodde, 2016). Consequently, DIM algorithms which obtain a corresponding point for almost every pixel are developed. Color information can also be captured for each point from the original images. The volume of DIM nD-PointCloud is also huge.

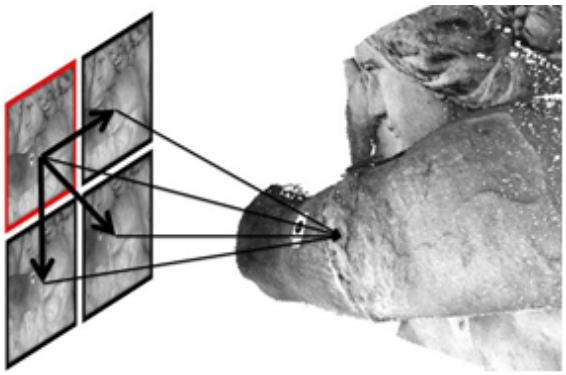


Figure 2.1: nD-PointCloud from dense image matching (Wenzel, 2015): the left four images covering a same region are matched to derive the 3D point cloud model

Modelling results can also be converted. As is indicated in Section 1.2, geophysical models normally run on the irregular mesh or grid. The results including flow or mechanical parameters are stored in each cell, which naturally form an nD-PointCloud data set where each point refers to the centroid of a cell. Representation using rasters is cumbersome: irregular cells have to be averaged and the accuracy can be damaged; the converted rasters also contain lots of empty cells which will make the storage and querying inefficient. In contrast, nD-PointCloud can support very high accurate modelling, incorporating all parameters computed. Generally, nD-PointCloud can be used to manage all results from Finite Element Method (FEM) which is widely adopted in structural analysis, heat transfer, fluid flow and mass transport.

2.1.2. DIMENSIONS IN ND-POINTCLOUD

Various point clouds contain a large number of diverse dimensions. To manage them well, the nature of different dimensions should be identified. This section selects and interprets several representative dimensions which are concerned in this research, from the perspective of data management and querying:

- Spatial dimensions XYZ are most often the fundamental dimensions to organize data, as spatial analysis is still the major task. XYZ values are normally confined in a limited scope with certain redundancy, e.g., several points may share the same X value. Besides, the Z dimension may not be used as often as the X and Y dimension in conventional spatial applications such as cartography and positioning. The Z values of the terrain also follow severely skewed distribution, which differs significantly from the X and Y values obeying approximately normal distribution. Last

but not least, XYZ are not independent dimensions. Objects in the space such as houses and bridges cause certain correlations among them.

- Temporal dimension is another critical dimension as it grasps changes of any object. It is a dynamic dimension. That is, its range can be changed and unlimited, and it is directly related to data updating. As it is also involved in spatio-temporal queries, the priority could be equal or even above spatial dimensions depending on the time granularity of the data and applications' requirements. For example, trajectory analysis needs frequent temporal computation, while change detection based on AHN data can only be performed once in a period of several years.

- LoI is an additionally introduced dimension which is used to express importance. LoI influences computation accuracy and efficiency significantly: at higher levels, only the most representative points get involved in the computation, which can significantly improve the efficiency but with low accuracy; on the other hand, when a large number of less important points get involved, the computation will take much more time but the result becomes more accurate. As one data set may be utilized for multiple applications, there might exist several LoI dimensions. cLoI consists of a series of distinct values. Considering the perception (Jiang, 2013), points having small LoI values should be much less than points with large values. The major access pattern of the cLoI dimension is to select a continuous portion, e.g., retrieving points with cLoI smaller than some value.

- Classification dimension is essential for semantic analytical purposes. It is added after post-processing of raw data such as LiDAR or DIM nD-PointCloud. The dimension is represented by a limited number of classes, where the number of candidates are usually much smaller than that of spatio-temporal dimensions. A sample encoding can be seen in Table 9 from ISPRS (2019). Some classes may consist of much more points than other classes, e.g., the ground versus the houses.

- Identity dimension, also represented by ID, is sometimes established to distinguish different entities within a class. In a number of applications, ID can be the only dimension to query, e.g., trajectory extraction and modelling case analysis.

- Color, intensity and normals are considered as the property dimension in most cases. They are less perceptible for selection, and their selectivity is often lower than the other dimensions. However, they still play important roles in visualization, for instance. Whenever an application frequently retrieves points according to them, they can also be processed as organizing dimensions.

2.2. LEVEL OF IMPORTANCE

LoI is also known as Level of Detail (LoD) in computer graphics. However, as a point is a 0-dimensional object, the concept of detail is inapplicable. Instead, it is more intuitive to say a point is more important than another, and a layer composed by points is more important than another layer. So, the term, importance, is used. LoI can be used to

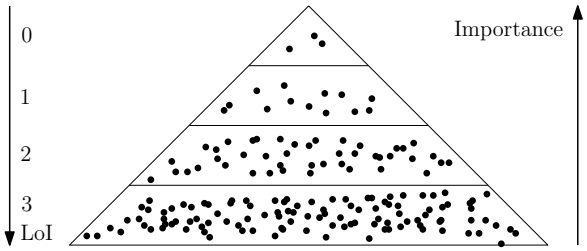


Figure 2.2: Structure of LoI pyramid

generally to solve big data issues. Besides visualization, LoI can also be applied to modelling to balance the accuracy and efficiency. LoI normally forms a pyramid structure (Figure 5.3).

Luebke et al. (2003) categorized three LoD frameworks for vector and raster data — discrete LoD, continuous LoD, and view-dependent LoD, in terms of computer graphics. They also presented metrics which can be used to measure and compare the performance of different LoD schemes. Reddy (1997) specifically discussed the perceptual aspect of LoD structures. This is also critical for visualizing nD-PointCloud, because point based visualization which leaves surfaces with holes is less perceptible. Besides, people are accustomed to meshes.

Conventional LoI for point clouds are implicitly embedded in the Quadtree (Finkel & Bentley, 1974) or the Octree structure (Meagher, 1982). For instance, Xie et al. (2013) implemented a hierarchical LoI structure using a randomly sampled Octree to manage massive point clouds. The random sampling is based on two principles: points at the parent layer should be removed from the child layer for the sake of storage efficiency; the sampling rate of the child layer depends on the number of parent nodes. Based on this structure, favorable performance of interaction, neighborhood searching and dynamic updating are achieved. Apart from these, more recent work can be seen in (Deibe et al., 2019; Virtanen et al., 2020), adopting similar data structures.

Aiming for an explicit discrete LoI (dLoI) framework, Guan et al. (2018) developed a random sampling algorithm to compute the LoI value for each point (Figure 2.3). In the algorithm, the original data sets are treated as the bottom layer, and the points at level $i+1$ are sampled uniformly to derive the upper level i. The sampling rate is $\frac{1}{2^n}$ with n the number of dimensions. That is, a point in every 2^n+1 points of level $i+1$ will be randomly selected and inserted into the level i. Totally 32 levels are defined.

Instead of random sampling, Cura (2016) indicated that a suitable LoI should be homogeneous in space, insensitive to density variations, regular, efficient and deterministic. So, using a virtual Octree, the solution divides points into different layers depending on their distance to the centers of cells. In fact, the study expressed the idea of cLoI where each point can represent a level, but the implementation was still in dLoI. Zheng et al. (2021) proposed a complex cLoI computing method. They first sort all points in the Z-order. Then, by reversing bits, taking a random mask and removing dummy points, a new priority ordering index which is actually the cLoI value can be derived (Figure 2.4). This approach can be more efficient at determining representative points than normal

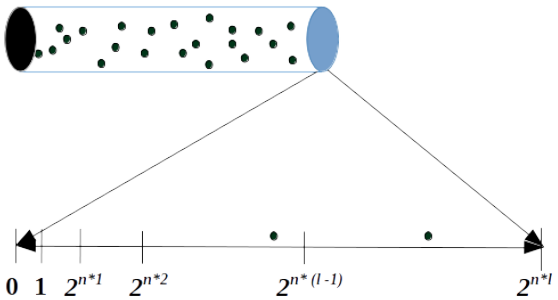


Figure 2.3: LoI value computation using random sampling (Guan et al., 2018), l indicates the level

Input: Z-order index	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**x**
binary representation	000	001	010	011	100	101	110	111
reverse bits	000	100	010	110	001	101	011	111
after random mask $M = 101$	101	001	111	011	100	000	110	010
new binary ordering index	6	2	8	4	5	1	7	3
priority ordering index	5	2	7	3	4	1	6	x

Figure 2.4: An illustration of the priority ordering process (Zheng et al., 2021). The first line describes the input Z-ordering index. There are 7 points and one dummy point designated as x

random sampling approach according to the testing results. To realize gradual rendering of scenes in VR, Schütz et al. (2019) developed a cLoI solution which extracts points from dLoI chunks based on sampling at specified distances. The sampled points are stored and ordered in a vector array which is then processed inside GPU. Points are then filtered according to the view frustum and target spacing in the VR environment, to be visualized.

2.3. DATA ACCESSING METHODS

This section presents some general techniques that are applicable to nD-PointCloud data management. Two major strategies for data accessing are described: one is the spatial clustering method — Space Filling Curve (SFC), while the other is multidimensional indexing.

2.3.1. SPACE FILLING CURVES

As is mentioned, SFC is a curve that passes by all basic units in a multidimensional space only once. It maps an n-dimensional data space to a one-dimensional data space. SFC preserves locality of the data: points that are close to each other in the original nD space are likely to be close to each other in the 1D space, and therefore are also stored physically together on the disk. The most well-known SFCs include Morton, Hilbert, Gray and Sierpinski curve (Lawder, 2000a; van Oosterom, 2005). This section focuses on the Morton curve and the Hilbert curve. Both can be constructed by recursively partitioning the hypercubic space (Bader, 2012), which is an advantageous feature for querying.

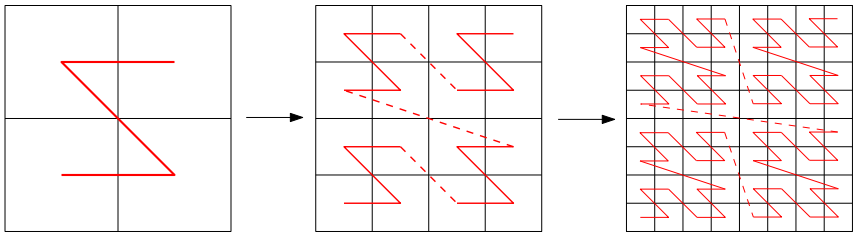


Figure 2.5: First, second and third order Morton curves in 2D space

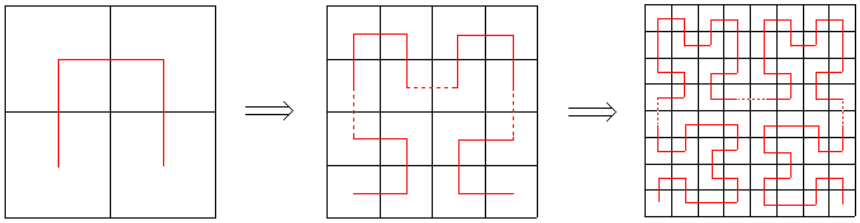


Figure 2.6: First, second and third order Hilbert curves in 2D space

MORTON CURVE

Morton curve, also called Z-order curve is proposed by Morton (1966). By reversing the order of dimensions, we get another form, the N-order curve. Figure 2.5 indicates the subdivision of the 2D space, where different orders of Morton curve can be derived. The order here means the level of subdivision. Basically, to derive the i^{th} order Morton curve, we copy and place the $(i-1)^{th}$ order Morton curve to the four quadrants and connecting their tails and heads.

SFCs are not just defined in 2D, but also in nD. For the Morton curve, the Morton code can be derived by interleaving the bits of the binary values of all dimensions (Section 1.5). An nD point is then mapped to a 1D Morton code.

HILBERT CURVE

Hilbert curve first appears in (Hilbert, 1891). It is also widely employed to manage spatial data due to the superior clustering effect. Figure 2.6 demonstrates the procedure to derive 2D Hilbert curves with different order. That is, we copy and place the $(i-1)^{th}$ order Hilbert curve to the four quadrants of the space. Hilbert curves at the top two quadrants remain the same, while the bottom ones are rotated 90° clockwise and counterclockwise, respectively. After this, different parts are connected.

Using Hilbert curve, a point will most likely get a different code from the Morton curve. However, nD encoding and decoding using Hilbert curve are more complex. Butz (1971) developed a mapping algorithm based on bit operations including shifting and exclusive OR. This method is improved by Lawder (2000b), who also provided executive code for encoding and decoding. Besides, based on the Gray code[1], Skilling (2004) developed another approach to derive Hilbert code which is also widely practiced.

[1] https://en.wikipedia.org/wiki/Gray_code

Table 2.1: Average farthest distance of the neighbors of all the points (Faloutsos & Roseman, 1989)

Order	Grid layout	Morton	Hilbert
2D space			
1	2 × 2	1.50	1.00
2	4 × 4	2.75	2.00
3	8 × 8	4.84	3.28
4	16 × 16	7.91	4.89
3D space			
1	2 × 2 × 2	2.00	1.00
2	4 × 4 × 4	3.31	2.00
3	8 × 8 × 8	5.10	3.23
4	16 × 16 × 16	7.03	4.20

Table 2.2: Average number of clusters for all possible range queries (Faloutsos & Roseman, 1989)

Order	Morton	Hilbert
2D space		
1	1.22	1.11
2	2.16	1.64
3	4.41	2.93
4	9.29	5.60
3D space		
1	1.59	1.33
2	4.49	3.72

Faloutsos and Roseman (1989) compared the clustering performance of Morton curve and Hilbert curve (Table 2.1 and 2.2). Focusing on the number of clusters involved in queries, Moon et al. (2001) conducted more comprehensive tests and analysis. Results indicate that Hilbert curve performs better than Morton curve in terms of clustering, but both can be used to recursively decompose the space.

2.3.2. MULTIDIMENSIONAL INDEXING

Multidimensional indexing mainly consists of hashing and the hierarchical methods (Gaede & Günther, 1998). This section focuses on the hierarchical method — tree based indexing which is the major type dealing with point data. I then categorize these indices into three groups — the R-tree and variants, the 2^n-tree and the B+-tree.

R-TREE AND VARIANTS

R-tree (Guttman, 1984) is the most widely adopted spatial indexing structure. The major database vendors including Oracle and PostgreSQL implement and use it as the de-facto approach. The key idea of the data structure is to group nearby objects and represent them with their Minimum Bounding Rectangle (MBR) in the next higher level of the tree (Figure 2.7). Since all objects lie within this bounding rectangle, a query that does not intersect the bounding rectangle also cannot intersect any of the contained objects. In practice, the R-tree based solutions such as Oracle Spatial normally group points into blocks, and then build an index on these blocks. The blocks can also be clustered first before indexing, such as the Hilbert-R tree structure which can significantly improve data loading performance. Besides, to decrease the overlap ratio between blocks, support frequent updates, and resolve nD queries, variants including the R*-tree (Beckmann et al., 1990), SR-tree (Katayama & Satoh, 1997) and X-tree (Berchtold et al., 1996) have been developed. They outperform the normal R-tree in different aspects, but the basic structure is the same.

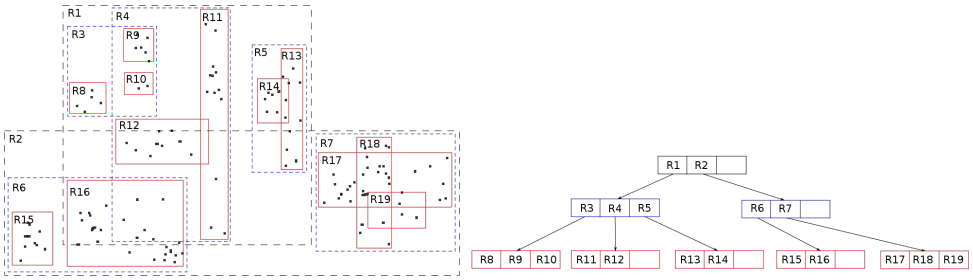


Figure 2.7: A 2D R-tree for point data management, adapted from Wikipedia

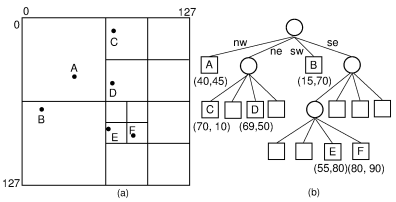


Figure 2.8: Illustration of Quadtree: (a) subdivision of the space (b) the tree structure. Image source: OpenDSA

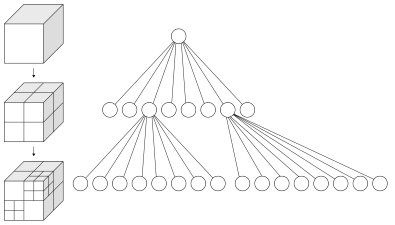


Figure 2.9: Illustration of Octree: recursive partition of the space and the tree structure. Image source: Wikipedia

2^n-TREE

2^n-tree can also be used to index blocks of points. The 2^n-tree represents an indexing category which evenly splits all dimensions in an iteration until the leaf node level. Considering non-uniform data distribution, the leaf node is defined by a node capacity, and will not be split further when the number of points contained is below this capacity (Wang & Shan, 2005). In 2D, 2^n-tree refers to Quadtree (Figure 2.8) (Finkel & Bentley, 1974), while it is Octree (Meagher, 1982) in 3D (Figure 2.9). Prevalent software for 3D point visualization such as Potree and Entwine uses the Octree to organize data. Similar to R-tree, high dimensional 2^n-tree solution is rarely implemented, due to limited applications at present.

B+-TREE

B+-tree (Comer, 1979) is a variant of the B-tree which is the most widely used indexing structure for one-dimensional data. Analogous to B-tree, B+-tree is composed by branch nodes and leaf nodes. Each leaf node corresponds to a disk file block containing the data entries. Each data entry is identified and indexed by a key value, and each leaf node contains an interval of key values. A branch node stores the pointers to a range of leaf nodes identified by the key value. Unlike B-tree, the leaf nodes of the B+-tree are also connected by pointers. That is, besides the top-down traversal, a leaf node can also be visited from its prior leaf node. The indexing key is a one-dimensional value. Hence, to manage nD-PointCloud data with the B+-tree, the data has to be mapped into the one-dimensional space. In addition to the SFC technique, other methods such as Pyramid-Technique or iMinMax have also been developed and implemented to derive a

key value for an nD point (Berchtold et al., 1998; Ooi et al., 2000). Due to the general applicability, B+-tree has been implemented in major DBMSs, e.g., Oracle Index-Organized Tables (Section 2.5.1).

2

2.4. FILE BASED SYSTEMS

The majority of nD-PointCloud data management is based on files. Among them, some formats are vendor specific. For example, the Bentley pointool viewer [2] only supports the native *plt* format so that a converter tool which transforms other formats into *plt* is also provided. The OPALS data manager (Otepka et al., 2012) utilizes a single file to manage ALS data, because it is an efficient light-weight tool tuned for high-speed processing and flexibility compared with DBMSs. The following sections present several formats which are vendor neutral, including LAS/LAZ, HDF and PDAL.

2.4.1. LAS/LAZ

The schema of LAS format (ISPRS, 2019) includes four parts:

1. Public header block: file source ID, project ID, LAS version, generating software, project extents and other metadata.
2. Variable length records: GeoKeyDirectory Tag which mainly refers to the coordinate reference system. For discrete full-waveform LIDAR data, the wave form descriptors are also included.
3. Point data records: X, Y, Z, intensity, return number, classification, channel, scan angle, GPS time, and other possible dimensions. The shift to the first pulse received are recorded for full-waveform data, i.e., X(t), Y(t) and Z(t).
4. Extended variable length record: waveform amplitude values for each packet.

Van Oosterom et al. (2015) built a system to manage point clouds with LAS format. They utilized LASsort[3] to reorder the points of each LAS file in a 2D Morton curve. Then LASindex[4] was adopted to create an index file LAX, based on an adaptive Quadtree over the XY dimensions of all points. The metadata (e.g., bounding box of each block) was stored in a PostgreSQL database. LASmerge[5] and LASclip[6] were employed for querying.

LAZ is a modified LAS format with LASzip compression (Isenburg, 2013). It employs the header and variable length records directly from LAS files. However, the real point cloud part is stored as blocks with scaling and offsetting techniques for compression. LASzip decomposes different attributes into separate atomic items including POINT10, GPSTIME10, RGB12 and WAVEPACKET13 (Figure 2.10). These items are compressed independently based on their specific features which are mainly the data range and interval. Depending on the input, the compression ratio can reach a factor of 10 (e.g., AHN2). Besides, LAX index which adopts a Quadtree structure can be directly built, which makes LAZ a rather flexible and efficient format as the original LAS format.

[2] https://www.bentley.com/en/products/brands/pointools
[3] https://rapidlasso.com/lastools/lassort/
[4] https://rapidlasso.com/2012/12/03/lasindex-spatial-indexing-of-lidar-data/
[5] https://rapidlasso.com/lastools/lasmerge/
[6] https://rapidlasso.com/lastools/lasclip/

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 2.10: Groups of attributes in LAZ (Isenburg, 2013)

2.4.2. HDF

HDF (Koranne, 2011) is a file format designed to store and organize large amounts of scientific data. In the geoscience domain, it is commonly used for storing multidimensional data sets concerned with regular spatial grids, e.g., global precipitation rasters. However, some geophysical models nowadays directly use LiDAR data instead of rasters as the input, for the sake of high accuracy. Due to this, the HDF format has been experimented with for such an adaptation. For example, the National Geospatial-Intelligence Agency (NGA) of USA published a Sensor Independent Point Cloud (SIPC) standard (NGA, 2015). This standard identifies the data and metadata generated by LiDAR systems, and provides a framework for handling the point cloud data processed from a single LiDAR sensor. HDF5 format is implemented, where XYZ values are treated as a data set in the HDF5 file in parallel with other dimensions such as intensity and time. However, such an implementation undermines the advantage of efficient data access through regular spatial grids of HDF5. For each spatial query, XYZ values in the file have to be checked completely. Yet HDF5 has no support for a spatial indexing structure.

2.4.3. PDAL

PDAL (PDAL-Contributors, 2018) is a C++ library for translating and manipulating point cloud data. It also functions in general as an abstraction layer for solutions using LAS files, Oracle and PostgreSQL. Thus, the same operations including reading, writing and filtering are always available, independent from the actual platform implemented. If the schema is not specifically defined, PDAL will utilize its default DBWriter to create blocks inside a DBMS such as Oracle or PostgreSQL. From this perspective, PDAL is also a format. Although PDAL supports a large number of dimensions in its schema [7], it builds blocks only using XY while attach others as the property dimensions. So, PDAL is not a true nD solution. Inside the block, scaling and offset can also be realized. Besides, LAZ-perf compression is available for block storage. The compression ratio is similar to rapidlasso LAZ, around 10× smaller than a LAS file depending on the type of data.

2.5. DATABASE MANAGEMENT SYSTEMS

DBMS is the preferable way for enterprise data management due to properties of Atomicity, Consistency, Isolation and Durability (ACID). In the GIS domain, DBMSs' advantages in scalability, parallelization, efficient caching, interoperability of different data types and multi-user support are highly valued. However, a debate on whether a DBMS should be specific or generic has been existing for a long time. On the one hand, specific optimization on data structure and functionality can improve efficiency for a single type of application. On the other hand, DBMS should be generic to facilitate the management of heterogeneous data which could otherwise be fragmented, and support a wide range of applications. The following sections review the general DBMS solutions — Oracle and PostgreSQL, as well as the specific ones.

2.5.1. ORACLE

Oracle is an object-relational DBMS. Oracle Spatial provides a spatial extension to manage different geometries including the point. Using Oracle Spatial & Graph, we can adopt flat table and block based approaches to manage nD-PointCloud.

FLAT TABLE APPROACH

Flat table stores each point as a single record. There are several possibilities for data management:

1. Normal table. XYZ and other property dimensions are stored as individual columns in the table. B-tree index can be created on one or multiple columns. However, as each column has its favorable order of records, the index cannot function efficiently for different types of queries. Nevertheless, the operations are straightforward and intuitive. In addition, flat table always reserves full precision of all values.

2. Spatially indexed table. Every point is still stored as a record, but with spatial dimensions XYZ stored as a geometry type. The R-tree index is normally built on the

[7] https://pdal.io/dimensions.html

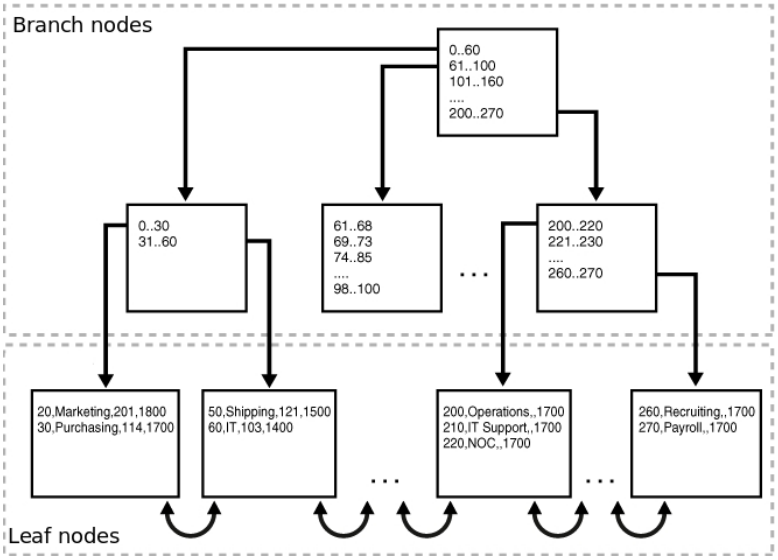


Figure 2.11: The structure of an Index-Organized Table (Oracle, 2013). The leaf node contains the data which is indexed by the key value, while the branch node stores the range of key values and the pointers to the leaf nodes

geometry column. This approach can be more efficient for spatial queries. However, as the geometry type maximally supports 3 dimensions, other dimensions cannot be indexed. Also, the R-tree index can occupy large storage space on the disk, as its branch nodes store the lower-left and upper-right corners of points.

3. Index-Organized Table (IOT) (Figure 2.11). The selected organizing dimensions are encoded into a 1D key value (e.g., SFC code) which is the primary key. Property dimensions are attached to the key, as a complete record. IOT manages all the records using a B+-tree structure, which entails the index and the data are integrated. So, in the execution of a query, the mapping step from a leaf node of the index structure to the actual data is eliminated. This reduces the I/O cost. Due to this, changes to the data also incur less overhead, e.g., updating existing rows only needs to update the B+-tree. However, when querying, the query window has to be converted into the 1D key ranges.

BLOCK APPROACH

The block based approach first groups points into blocks which are then stored using Binary Large OBject (BLOB) types in a table. The approach can build the R-tree on these blocks for indexing. Blocks are the basic unit to interact with. In a query, blocks are first retrieved and then unpacked to extract individual points. Besides, compression is also available inside each block. Oracle spatial provides several possibilities to use blocks:

1. SDO_PC (Figure 2.12) (Oracle, 2019). Two tables will be created in this approach. One is the BASE table recording the SDO_PC object and an identifier indicating the

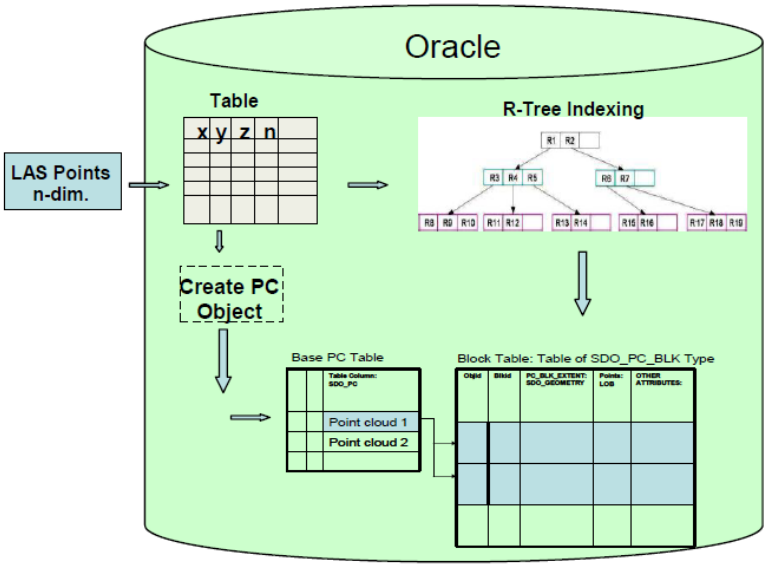


Figure 2.12: Storage Model for SDO_PC type (Ravada et al., 2010). The columns in the SDO_PC_BLK table are OBJ_ID, BLK_ID, PC_BLK_EXTENT (SDO_GEOMETRY), POINTS (LOB), and OTHER ATTRIBUTES

specific point cloud. Point cloud data is blocked and stored in another BLK_TABLE, with additional fields such as block ID and total number of points in the block. SDO_PC also provides the possibility to store dLoD information. Besides, the normal R-tree or Hilbert-R tree indexing is available. However, blocks are based on a 2D XY organization, which means other dimensions have to be stored as property dimensions. Besides, maximally 8 dimensions in total are allowed.

2. SDO_Geometry type. In this type, MULTIPOINT or POINTCLUSTER can be used to group and store a set of points, as a binary object. This data type maximally incorporates 3 dimensions. The structure of POINTCLUSTER is similar to MULTIPOINT, except auxiliary fields recording the offset of data and the number of points. POINTCLUSTER decreases the redundancy of these auxiliary information, compared to MULTIPOINT. Wijga-Hoefsloot (2012) specifically tested POINTCLUSTER and compared it with SDO_PC, indicating POINTCLUSTER achieves better querying efficiency than SDO_PC.

3. PDAL block. This approach creates the same table structures as the SDO_PC method through the API of PDAL. LAZ-perf compression can be realized (Section 2.4.3). However, as the block adopts a different schema from SDO_PC, the operators provided by PDAL are not yet compatible with Oracle's operators.

2.5.2. POSTGRESQL

Unlike Oracle, PostgreSQL is an open-source DBMS. The solutions provided by it are analogous to Oracle, except the following:

```
{"pcid":1,"pts":[
 [-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0],
 [-126.96,45.04,4,0],[-126.95,45.05,5,0],[-126.94,45.06,6,0],
 [-126.93,45.07,7,0],[-126.92,45.08,8,0],[-126.91,45.09,9,0]
]}
```


Figure 2.13: A sample of the PC_PATCH schema with 4 dimensions: X, Y, Z and intensity expressed in plain text

1. IOT does not exist, but it is possible to cluster tables based on an index (Ramsey, 2012). However, this results in two separate structures: one is the table, while the other is the index.

2. The counterpart of SDO_PC is the PostGIS (Strobl, 2008) extension — pgPointcloud (Ramsey, 2020). It stores each block in a row of a table using the PC_PATCH binary data type. A PC_PATCH object records an ID and a list of points containing all dimensions (Figure 2.13). These objects will be stored in a TOAST table, a feature of PostgreSQL that separates and stores large objects. Besides, vast set of functions to manipulate the blocks are provided by the extension. However, as SDO_PC, only XY can be used as the organizing dimension.

3. The MULTIPOINT geometry supports 4D point, which means 4 organizing dimensions can be used to create a MULTIPOINT object. Besides, a 4D R-tree can be built to index the objects.

4. PDAL utilizes the pgPointcloud for reading and writing blocks. So, PDAL's operators are compatible with native operators.

2.5.3. CUSTOMIZED DBMS BASED SOLUTIONS

Dobos et al. (2014) illustrated the shortcomings of relational database model for managing large point clouds. Then based on use cases of astronomy, they proposed a model of a point cloud database including basic requirements and algorithms that should be implemented. Such a database is expected to work with large amounts of disk-resident data and scale out to multiple servers. Cura et al. (2017) presented a state-of-the-art solution based on a PostgreSQL server, to provide services. The solution utilizes patches to group point clouds, with metadata incorporated. Topological framework was proposed to be constructed on patches to take advantage of graph analysis. Besides, several in-database functions such as clustering points using minimum spanning tree, and extracting primitives (planes and cylinders) were prototyped and implemented. Vo (2017) developed an Oracle solution, the UMG_PC with a customized hybrid indexing structure. The top layer is a 2D Hilbert-coded rectangular grid, while the bottom layer consists of separate, in-memory, 3D Octree for each point block. Basic operators were realized in Java classes which were registered with an Oracle DBMS as a package. Three novel functionalities including point clipping, nearest neighbour search, and planar segment selection were further implemented. The later two functions were concerned with inner block computation which was not available using Oracle SDO_PC.

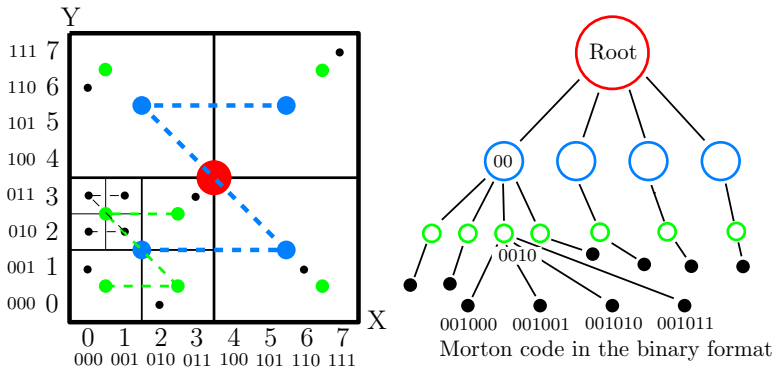


Figure 2.14: Implicit Morton hierarchy: black dots are real points to be managed, while colored dots are Morton branch nodes at different levels

Customized DBMS solutions are also developed to address specific applications. For 3D city modeling, Richter and Döllner (2014) proposed a change detection method to reduce storage in case of incremental updates in the same area. Basically, points that are identified as new surfaces will be inserted into the database, while new points locating on existing surfaces will not be added. In order to analyze surface changes in high mountain environments, Rieg et al. (2014) proposed a database approach with blocking strategies for efficiently managing multi-temporal point clouds. Besides, they also presented an overview of current point clouds management approaches incorporating file-based systems, databases and hybrid systems.

2.6. PLAINSFC

Van Oosterom et al. (2015) first described the design of PlainSFC solution, and provided some benchmark results. PlainSFC is the basis for this research, and is also essential for understanding the following chapters. This section presents the basic structure of PlainSFC, by first introducing the nomenclature.

2.6.1. HYPERCUBE, NODE AND RANGE

In general, a cube refers to a 3D box with equal edge length. This geometric concept extended into nD space becomes the hypercube.

Figure 2.14 illustrates the node and the range in 2D. All points have integer coordinates. By truncating the last n bits of the points' Morton codes recursively, Morton codes at upper levels are derived. That is to say, the Morton codes of points implicitly contain a hierarchy which is equivalent to a Quadtree structure. We can easily extend this scheme to higher dimensional spaces so that a Morton node refers to the corresponding node of a 2^n-tree. A branch node covers the nodes on the level below, and represents the extent of a hypercubic region (e.g., a block in the Quadtree). Thus, the branch node also indicates a range of Morton codes starting from the lower-left corner to the upper-right. A leaf node is not further subdivided.

2.6.2. BASIC SETTINGS

Figure 2.15 presents the workflows of PlainSFC including data loading and querying. PlainSFC first encodes each nD-point to a full resolution Morton key, interleaving the bits of all organizing dimensions. In most cases, values of the organizing dimensions contain decimals. So, these values are first scaled up to integers for encoding. Such a full resolution key can be directly decoded to the original coordinates, without additional storage of dimension values. Besides, due to uniqueness of each full resolution key, they are used as the primary key in tables for indexing. Property dimensions are attached to the key. Based on this organization, PlainSFC utilizes Oracle IOT (Section 2.5.1) to manage the data.

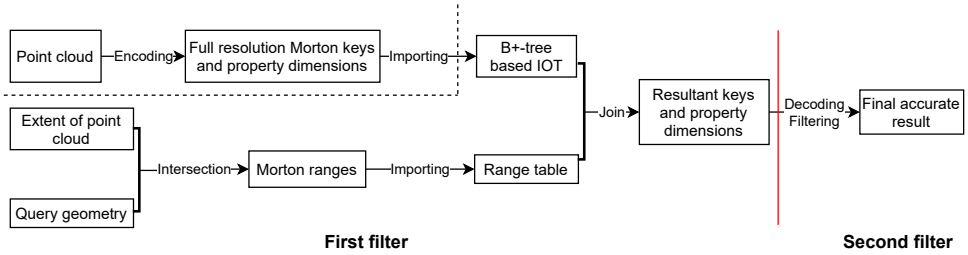


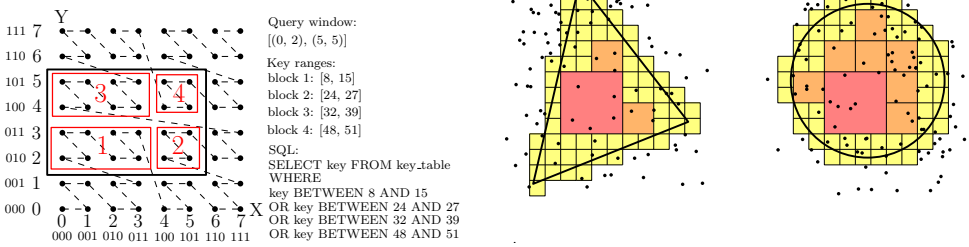
Figure 2.15: The loading and querying procedure of PlainSFC, separated by the dash line

As to querying, PlainSFC adopts two filters. The first filter uses the Morton hierarchy to approximate the query geometry and derive the ranges. Take Figure 2.16a to illustrate: the first filter starts by examining whether the root node (i.e., the overall extent of the data) intersects the query window. If they intersect, the root node will be decomposed into 4 sub-nodes and the spatial relationship between each node and the query window will be assessed again. During the range computing process, if a node is inside the query window, the range will be exported directly without further decomposition. Near the query boundary, the decomposition goes on recursively until a maximum number of ranges (r_{max}) is reached. r_{max} is used as a threshold to balance the performance. A larger number of ranges incurs intensive computation, slowing down the first filter. A smaller number of ranges, however, results in rough query result. This is also unacceptable because the burden is moved to a second filter for an accurate answer. An optimal r_{max} needs to be benchmarked to be derived. After the searching reaches r_{max}, the first filter exports all ranges into a range table, and then joins it with the IOT for selection, where the index automatically functions. A second filtering will be conducted in a following step to complete the query. Other query geometries can also be addressed (Figure 2.16b).

2.6.3. TIME COMPLEXITY

The theoretical querying time of PlainSFC is as follows:

$$T = T_{pre} + T_{io} + T_{post} \tag{2.1}$$



(a) Executing a window query on a uniformly distributed 2D point set based on Morton encoding

(b) Querying with a triangle and a circle: false positive points in boundary cells will be filtered out by a second filter

Figure 2.16: Recursively partitioning the extent of data according to SFC regions to match different query geometries, for selecting data in IOT

where T_{pre} is the time cost of the first filter, and mainly comprises range computation and B+-tree traversal; T_{io} indicates the main I/O cost to retrieve points inside the ranges; T_{post} refers to the final decoding and filtering.

T_{pre} is bounded by $\mathcal{O}(r \log_B N)$, where r is the number of ranges generated and it reaches the threshould r_{max} by default; B is page capacity in the number of points; N represents the input size. This is because searching each range in the B+-tree costs $\mathcal{O}(\log_B N)$ time. T_{io} maximally covers $\mathcal{O}(\frac{k'}{B} + r)$ I/Os, where k' equals the number of points returned by the first filter. This expression is an approximation of $\mathcal{O}(\sum_{i=1}^{r} \lceil \frac{k_i}{B} \rceil)$, where k_i represents the number of points inside a specific range, and $\sum_{i=1}^{r} k_i = k'$. The approximation is adopted because $\sum_{i=1}^{r} \lceil \frac{k_i}{B} \rceil \leq \sum_{i=1}^{r} (\frac{k_i}{B} + 1) = \frac{k'}{B} + r$. T_{post} is bounded by $\mathcal{O}(nk')$. n refers to the dimensionality of the key, assuming the dimensions in the nD-PointCloud data are all used as the organizing dimension. Once parallelism is applied, T_{post} becomes $\mathcal{O}(\frac{nk'}{p})$, given p processors.

An optimal solution should balance the three cost terms. An accurate first filter with large r may cost more time, but it returns a small k' which alleviates I/O and post-processing in the second filter. For this purpose, I introduce False Positive Rate (FPR) to indicate I/O and the performance of second filter (Equation 2.2):

$$FPR = \left| \frac{k' - k}{k} \right| \tag{2.2}$$

Sometimes, FPR can be very large, e.g., in high dimensional spaces. Then, the portion of data selected by the first filter is also indicative (Equation 2.3):

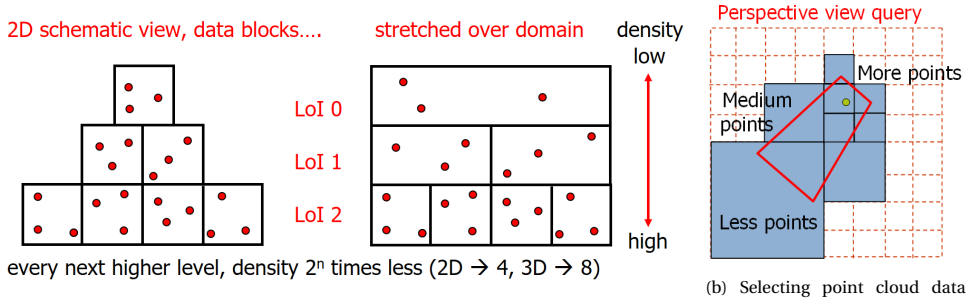
$$selectivity = \frac{k'}{N} \tag{2.3}$$

The memory cost is mainly determined by r and k'.

3

A CONTINUOUS LEVEL OF IMPORTANCE METHOD

BIG geo-data requires efficient spatio-temporal data organization, including levels of detail that allow to zoom in from high-level overviews (complete countries/continents) to the smallest detail (as the curb stones of a sidewalk) and everything in between. For point cloud data, we use the term Level of Importance (LoI) instead, as a point cannot provide different detail. Current solutions only support a limited number of discrete levels for managing nD-PointCloud data. Figure 3.1a shows the current-state-of-the-art solution which organizes large point clouds using data pyramids of discrete levels. Such organization causes density shocks in visualization (Section 1.3). Besides, the abrupt transition between various levels is also very disturbing. Thus, a continuous LoI (cLoI) scheme should take place.



(a) Symbolic 'side view' of 2D point cloud data blocks

(b) Selecting point cloud data blocks for perspective query

Figure 3.1: Discrete levels (scale/importance) data organization in blocks

This is the goal of this chapter which develops a generic method to compute cLoI for massive point clouds to realize smooth and realistic rendering. The chapter is organized

This chapter is a joint work, and the other authors are van Oosterom, P., van Oosterom, S., Thompson, R., Meijers, M., & Verbree, E.

as follows: Section 3.1 reviews previous studies on smooth visualization of point clouds. Section 3.2 answers the question: what is an optimized continuous distribution of point clouds over continuous levels. Then, based on the answer, Section 3.3 derives the distribution function and the cumulative distribution function of continuous levels for 1D to nD point clouds. Section 3.4 explains how to use the cLoI value in order to approximate a certain data density in the output, which may be non-uniform, e.g., in the case of a perspective view. Section 3.5 summarizes the chapter and provides more implications.

3

3.1. RELATED WORK

The earlier study on continuous point clouds were based on post-processing the discrete levels of point clouds. For example, van der Maaden (2019) chose an Octree organized point cloud as the starting point. Based on the viewing position, viewing direction, and opening angle, the required point data blocks are selected at the various levels as explained in the Introduction (Figure 3.1). Next, a post-processing step was conducted to give the impression of continuity by removing some points from the denser data blocks. These points are close to the edge of the sparser data blocks. This is done because the transition zone between denser and sparser blocks is very clear to observe, which is distracting. After this, the change in data distribution between adjacent blocks of different density is sketched, as shown in Figure 3.2.

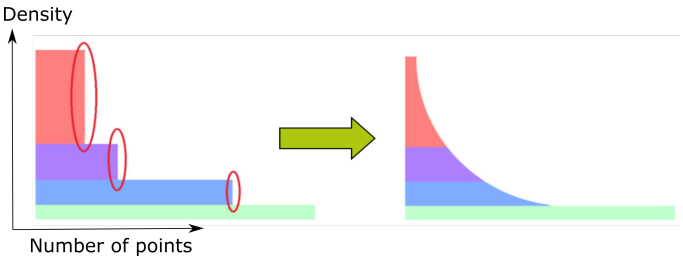


Figure 3.2: Smoothing point cloud data distribution by post-processing (van der Maaden, 2019)

The decision to keep or remove a point is based on an empty sphere condition attached to every point. This starts by assigning a radius to the points based on distance to camera (Figure 3.3). The radius gradually increases from a small value nearby to a large value far away. The points from the selected data blocks are visualized in the current view when their sphere is empty, i.e., intersecting no other spheres in the current view. However, the resultant views can only be smooth when a large portion of points are removed, typically about 70-90% of all points. Another drawback of the method is that it is not stable when panning or zooming in a certain direction — points may be switched on and off repeatedly sometimes. This is unwanted: when zooming or panning, points should appear or disappear cleanly.

This unstable behaviour was solved by Schütz et al. (2019) in a Virtual Reality (VR) environment. However, their solution is not able to address big data organized at the server side. All data, an indoor point cloud containing 86 million points, was processed locally in the main memory of the GPU. The solution applies a different post-processing technique by first adding a uniform random value between 0 and 1 (i.e., cLoI value) to

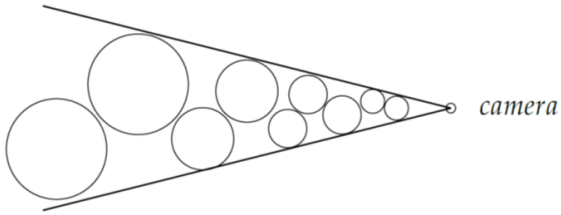


Figure 3.3: Points with empty sphere are selected for the view (van der Maaden, 2019)

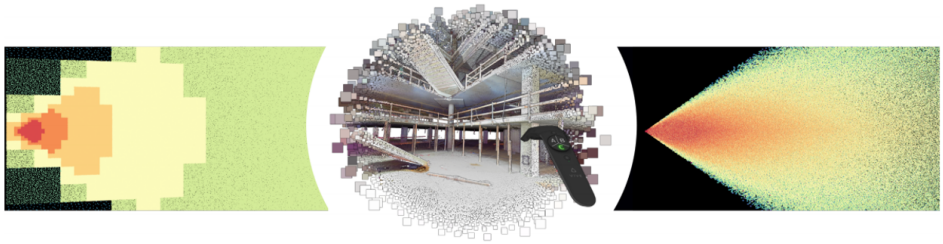


Figure 3.4: Left: discrete LoI, Middle: continuous point cloud view in VR, Right: continuous LoI (image from Schütz et al., 2019)

every point. Then, based on viewing position and viewing direction, they select 5 million points every 5 to 6 frames. The rendering speed is 90 Frames Per Second (FPS) for both left eye and right eye, which means a total of 180 FPS.

In the selection they use the cLoI value to select a sufficient ratio of the points for the needed data density, which depends on distance and eccentricity (Figure 3.4). Because for every point the cLoI value stays the same during an interactive VR session, the solution is stable, and has no issues with flickering points (i.e., points switched on and off repeatedly). As is mentioned, the solution is not intended for addressing big data visualization. To resolve it, spatio-temporal and LoI dimensions may be used together to optimize the multi-dimensional data clustering and indexing for fast retrieval.

3.2. CONTINUOUS LEVELS

This section lays the foundation of our cLoI scheme for nD-PointCloud data. Subsection 3.2.1 presents initial ideas to dispense with discrete levels by pre-processing, and explains why this is preferred over post-processing approaches. Next, Subsection 3.2.2 reviews the well-know discrete levels commonly used to manage large scale vector and raster data. Then, by refining the original discrete integer levels, Subsection 3.2.3 derives a finer grained distribution over more levels, while keeping the general good characteristics of the original distribution. By infinitely refining the discrete levels, Subsection 3.2.4 builds the continuous levels. Finally, Subsection 3.2.5 discusses how many levels are needed for a specific data set.

3.2.1. RESOLVING DISCRETE LEVELS BY PRE-PROCESSING

Inspired by the promising post-processing results in the above described attempts to arrive at continuous point clouds, we designed a solution based on pre-processing of the large point clouds. The solution intends to resolve the unstable rendering process and avoid sending too much data from the server to the clients. Hence, the clients do not have to store all points to serve functionalities. Our cLoI solution is also inspired by the research on vario-scale vector maps (van Oosterom & Meijers, 2014; Meijers et al., 2020): add one continuous dimension to the geometry to represent scale to realize smooth zooming in/out visualization. Instead of the 2D vario-scale vector map, we apply the continuous methodology to the 3D geometries. Assuming it is possible to assign proper cLoI values for points, then, we can apply the following strategy for rendering massive point clouds:

1. Compute the cLoI value;
2. Add cLoI as an organizing dimension, to be clustered and indexed;
3. Define perspective view selection by a view frustum formed by cLoI and the spatial extent. The shape of the view frustum is determined by, for example, viewer distance and displacement from the centre of the view; see Figure 3.5.

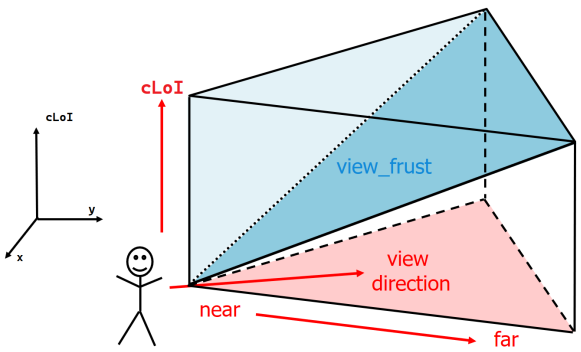


Figure 3.5: Integrated space-cLoI selection via the upper blue tetrahedron (view_frust) from the point cloud. The darker blue bottom plane is not normal geometry, but a combination of spatial and cLoI dimensions. Please note that the vertical axis represents the cLoI dimension (and the z dimension is not shown)

3.2.2. LEARNING FROM DISCRETE LEVELS

How should cLoI be computed? Should we just use random values, like Schütz et al. (2019), or should the level values have more meaning? From raster maps or tiled vector maps being served over the web, there is always a fixed ratio in scale (data density) between two discrete levels: that is, a factor of 2 for every dimension. For example, every next lower level in the pixel data pyramid contains 2 times more data per dimension, which is 4 times overall, as pixel is 2D. Similarly, for 3D voxels, the data ratio between subsequent levels would be 8. Table 3.1 presents the typical 15 Dutch zoom levels in

the 2D data pyramid and their relationships to appropriate map scales. This Dutch 'profile' for tiling can be used by Web Map Tile Services (WMTS), with the Spatial Reference System (SRS) EPSG:28992 (Amersfoort / RD New), where each tile is defined by 256 × 256 pixels (Geonovum, 2012). Global tile sets use more levels, for both raster and vector tiling schemes: the well-known scale set GlobalCRS84Pixel adopts 18 levels (Masó et al., 2010), but sometimes even more are defined.

For point clouds, there are similar approaches using discrete levels based on the Quadtree or Octree structure. For instance, the 3D web-viewer for the Dutch AHN2 data set applies 12 levels (van Oosterom et al., 2017). It is important to be aware that the discrete levels have a relationship to the scales for which they can be used.

Table 3.1: Zoom levels in the Dutch well-known scale set, based on Geonovum (2012)

Zoom level	Resolution (m/pixel)	Scale denominator
0	3440.64	12,288,000
1	1720.32	6,144,000
2	860.16	3,072,000
3	430.08	1,536,000
4	215.04	768,000
5	107.52	384,000
6	53.76	192,000
7	26.88	96,000
8	13.44	48,000
9	6.72	24,000
10	3.36	12,000
11	1.68	6,000
12	0.84	3,000
13	0.42	1,500
14	0.21	750

3.2.3. REFINED DISCRETE LEVELS

The cLoI value should support the level thinking that has been developed in raster and vector mapping over the past decades, as users have got accustomed to it. So, how can we obtain a similar distribution for cLoI values compared to the existing discrete schemes? What if we do not try to solve this question directly, but first solve the question: how can we refine the existing discrete level distributions? This question was raised and answered during the keynote presentation at the ISPRS Geospatial Week 2019 (van Oosterom, 2019).

Figure 3.6 first reviews the traditional distribution of data over discrete integer levels. The target number of points in level l is given by $N_l = 2^l$, for integer l ranging from 0 (most important, few points) to maximum level L (least important, most points). Note that this is for the 1D case the optimized distribution over the levels. The multi-dimensional case is quite similar: $N_l = 2^{n \times l}$ with n the number of dimensions. In the following, we first continue with 1D data, and later on extend the formulas to the nD case. As no refinements have been made on the discrete integer levels, we use refinement $r = 0$ to indicate. The probability that a point belongs to level l is defined by di-

viding the number of blocks at this level by the total number of blocks in all levels: the mathematical expression is provided in the next section (Equation 3.2).

The next step is to refine the discrete integer levels into discrete half levels, so doubling the number of levels in this first refinement $r = 1$ (Figure 3.7). The formulas for computing the number of blocks per (half) level, and the distribution over the levels remain the same. In the formulas, only the total number of blocks at refinement $r = 1$ summed over all levels is different (Equation 3.4). It is important to note that summing the probabilities of two half levels that originated from the same integral level results in exactly the same probability as before. Of course, this procedure can be repeated and after 2 refinements we end up with 16 quarter levels (Figure 3.8). Again, the formulas remain the same (only now with $r = 2$), and summing 4 quarter levels that used to form one level results in exactly the same probability as before.

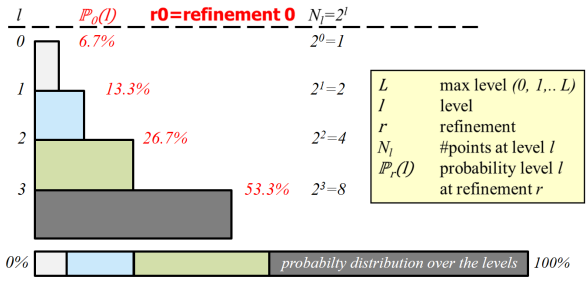


Figure 3.6: Distribution over 4 discrete integer levels ($l = 0, 1, 2, 3$) in case of 1D data

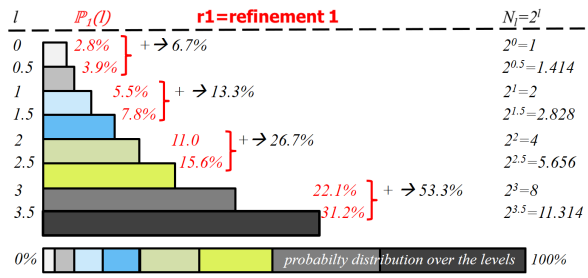


Figure 3.7: Distribution over 8 discrete half levels ($l = 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5$) in case of 1D data after one refinement $r = 1$

3.2.4. INFINITELY REFINED DISCRETE LEVELS

We can continue to refine the levels. When $r \to \infty$, then, the actual result is a continuous distribution function $f(l)$ over the levels l from 0 to L (Equation 3.6). In addition, we define the cumulative version of the distribution function, which is $F(l)$ (Equation 3.7).

The inverse of the Cumulative Distribution Function (CDF) is then used to compute the cLoI value for each input point. Section 3.3 provides more details (see Equation 3.11). The resultant point cloud data has one more cLoI dimension, and the distribution of

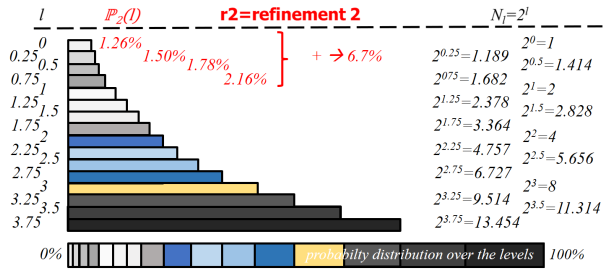


Figure 3.8: Distribution over 16 discrete quarter levels ($l = 0, 0.25, .., 3.75$) in case of 1D data after two refinements $r = 2$

this dimension is described by the optimized continuous distribution function $f(l)$. The cLoI dimension can be used to organize the data next to spatial dimensions, facilitating perspective view selection (Figure 3.5).

3.2.5. HOW MANY LEVELS ARE NEEDED?

Given a certain data set with N points, how many levels would be appropriate, i.e., what is the optimal value for L? We first analyse the discrete integer levels, and the outcome is also valid for refined discrete levels and continuous levels.

In the 2D pyramid, about 80% of the data is at lowest discrete level, i.e., with highest level number (and in 3D, the ratio is about 90%). So, for the sake of convenience, we first assume all data are at the lowest level. In case of a data set like AHN2 with about 640,000,000,000 points, if each block contains 10,000 points, then, 64,000,000 blocks are needed in total. In 2D, with $L = 13$, we can host $4^{13} = 67,108,864$ blocks at level 13 (i.e., the lowest one out of the 14 levels starting from level 0). This is enough for AHN2, and also the actual depth of the AHN2 Octree storage implemented in our earlier research (van Oosterom et al., 2015). Although it is Octree, due to the nature of terrain, the AHN2 data actually captures a 2.5D surface in 3D space.

Then, we derive the expression of L for an nD point cloud with N points organized in blocks with capacity of C:

$$L = \left\lceil \frac{1}{n} \log_2 \frac{N}{C} \right\rceil \tag{3.1}$$

Until the time of writing, the USGS 3D Elevation Program (3DEP) has 38,671,823,167,523 points [1]. The coverage is not yet complete, and the final data set is expected to contain about 10^{14} points. As the data is also a 2.5D surface in 3D space, $n = 2$ is adopted. Assuming the same capacity for the blocks $C = 10,000$, then, we would need $L = 17$ to store the complete USGS LiDAR point cloud. As the continuous levels have the same general distribution characteristics as the discrete integer levels, the same number of levels $L+1$ is sufficient.

[1] https://usgs.entwine.io/

3.3. THE MATHEMATICS

After presenting in the previous section the idea how to arrive at continuous levels with an optimized distribution, this section presents the related mathematics. Subsection 3.3.1 provides the formulas for refining discrete levels, which is then followed by formulas for the infinite refinement to continuous levels in Subsection 3.3.2, both for the 1D case. Next, Subsection 3.3.3 generalizes the formulas from 1D to nD.

3.3.1. REFINED DISCRETE LEVELS

The formulas for the distribution over discrete levels are first introduced for the 1D case. Suppose we have a point p, and want to decide the probability of p being put in some level. In 1D with L levels, the probability of a point to be located in level l is proportional to 2^l, with $l \in \{0,1,\ldots,L\}$:

$$\mathbb{P}_0[p \to l] = \frac{2^l}{\sum_{l=0}^{L} 2^l} = \frac{2^l}{\mathrm{Tot}_0} \tag{3.2}$$

Now, we are going to refine the number of levels by splitting each one in half. The refinement times are indicated by r, meaning the times we split the levels. The refined levels become

$$k \in \{0, 2^{-r}, 2\times 2^{-r}, \ldots, ((L+1)2^r - 1)\times 2^{-r}\}$$

where $k = 0$ is the top level, and $k = L + 1 - 2^{-r}$ is the most detailed/important one. So, after refining the levels r times, the probability is given by:

$$\mathbb{P}_r[p \to k] = \frac{2^k}{\mathrm{Tot}_r} \tag{3.3}$$

We have found a recursive formula for Tot_r, based on Tot_0:

$$\begin{aligned} \mathrm{Tot}_r &= \sum_{k=0}^{(L+1)2^r-1} 2^{k\times 2^{-r}} = \sum_{k=even} \left[2^{k2^{-r}} + 2^{(k+1)2^{-r}}\right] = \sum_{k=even} 2^{k2^{-r}}\left[1 + 2^{2^{-r}}\right] \\ &= (1+2^{1/2^r})\mathrm{Tot}_{r-1} = \mathrm{Tot}_0 \prod_{i=1}^{r}(1+2^{1/2^i}) \end{aligned} \tag{3.4}$$

With this and Equation 3.3, we derive a probability density for refinement r:

$$f_r(l) = 2^r \mathbb{P}_r[p \to l] = 2^l \frac{\prod_{i=1}^{r} \frac{2}{1+2^{1/2^i}}}{2^{L+1}-1} \tag{3.5}$$

Figure 3.9 shows the change of distribution by refining discrete levels, where the max level $L = 2$.

3.3.2. INFINITE REFINEMENT

If we want the continuous distribution, we let $r \to \infty$, then

$$f_\infty(l) = K2^l$$

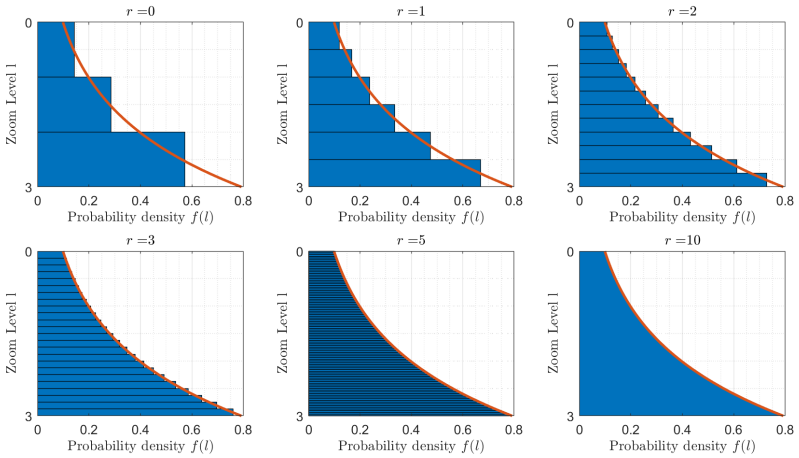


Figure 3.9: Distribution over the refined levels with dimension n=1, max levels L=2, refinements r=0, 1, 2, 3, 5, and 10: the blue bars depict the probability of discrete levels, while the red curve depicts the exact, continuous probability function

where K is a normalization constant:

$$\int_0^{L+1} K2^l dl = K[2^l/\ln 2]_0^{L+1} = K\frac{2^{L+1}-1}{\ln 2} = 1$$

Thus

$$f(l) = \frac{2^l \ln 2}{2^{L+1}-1} \tag{3.6}$$

for $l \in [0, L+1]$. The corresponding CDF $F(l)$ is obtained by integration $f(x)$ over x from $x = 0$ to l:

$$F(l) = \int_0^l f(x)dx = \frac{2^l - 1}{2^{L+1}-1} \tag{3.7}$$

With this, a 1D cLoI level l can be randomly generated by:

$$l = F^{-1}(U) = \log_2(U(2^{L+1}-1)+1) \tag{3.8}$$

where U is uniformly distributed between 0 and 1.

3.3.3. HIGHER DIMENSIONS

For higher dimensions, we can perform similar computations and get:

$$f_n(l) = \frac{2^{nl} n \ln 2}{2^{n(L+1)}-1} \tag{3.9}$$

Where n is the number of dimensions. This function has CDF:

$$F_n(l) = \frac{2^{nl} - 1}{2^{n(L+1)} - 1} \tag{3.10}$$

An nD cLoI level value can be generated by:

$$l = \frac{1}{n} \log_2(U(2^{n(L+1)} - 1) + 1) \tag{3.11}$$

3.4. USING THE CLOI VALUES

After having added the cLoI dimension to point data and having computed its values for individual points, the next challenge is how to use the cLoI value. Subsection 3.4.1 first explores the relation between discrete levels and expected data density. Then, Subsection 3.4.2 describes the use of continuous levels for producing visualizations with uniform data density. Finally, Subsection 3.4.3 shows how to use cLoI values to realize smooth perspective view, with mixed data density.

3.4.1. EXPECTED DATA DENSITY AT DISCRETE LEVELS

Assume that we have in total N points in an nD domain in which every dimension has the extent E. That is, the whole domain is a hypercube, with equal size of all dimensions. Then, the total expected data density is given by $D = N/E^n$, without any levels being involved. As the discrete nD probability function at refinement r is

$$\mathbb{P}_{r,n}[p \to l] = \frac{2^{l \times n}}{\mathrm{Tot}_{r,n}} \tag{3.12}$$

Then, the expected data density at discrete level l at refinement r is given by

$$D_{r,n}(l) = \frac{N}{E^n} \mathbb{P}_{r,n}[l] \tag{3.13}$$

3.4.2. VISUALIZATION USING CONTINUOUS LEVELS, UNIFORM DATA DENSITY

The continuous level also corresponds to data density. If we take a slice of the cLoI values that belong to one discrete integer level, which is a semi-open range $[l, l+1)$, then we get exactly the same expected density as that of the discrete level l. However, the 'thickness' of single value of cLoI is 0, which gives near 0 probability of having any point with exactly that value (and thus 0 expected density at that level). Therefore, it is very convenient to take the range from the top (cLoI= 0) to a specific level (cLoI= l) to achieve a required cumulative data density. This corresponds well to the CDF for nD case, i.e., $F_n(l)$ (Equation 3.10). So, the expected Cumulative Density CD_n at continuous level l for nD case is

$$CD_n(l) = \frac{N}{E^n} F_n(l) \tag{3.14}$$

We can then use this for visualizing points. To illustrate with a simple example: suppose a 2D space ($n = 2$) with extent 5×5 m^2 ($E = 5^2$), 10^7 points ($N = 10^7$), $L = 6$, with

continuous levels. The maximum total density = $10^7/25 = 400{,}000$ points/m^2. By substituting these numbers in Equation 3.14, we get

$$CD_2(l) = 3149.6 \times (2^l - 1) \tag{3.15}$$

With a target cumulative density TCD_2, the cLoI value l can be obtained by rewriting Equation 3.15:

$$TCD_2 = 3149.6 \times (2^l - 1) \Rightarrow \tag{3.16}$$

$$l = \log_2(1 + (TCD_2/3149.6)) \tag{3.17}$$

Assume we have a rendering budget $B = 100{,}000$ points. The following three queries mimic a zoom out action with a square, by selecting to different continuous levels (Figure 3.10):

- 1×1 m^2 box $\Rightarrow TCD_2 = 100{,}000$ points/m^2 $\Rightarrow l = 5.03$
- 2×2 m^2 box $\Rightarrow TCD_2 = 25{,}000$ points/m^2 $\Rightarrow l = 3.16$
- 4×4 m^2 box $\Rightarrow TCD_2 = 6{,}250$ points/m^2 $\Rightarrow l = 1.58$

The case of 0.5×0.5 m^2 is not presented because we would arrive at the maximum density $TCD_2 = 400{,}000$ points/m^2 $\Rightarrow l = 7.0$. That is, further zooming in will not result in higher density as we are already at the maximum continuous level.

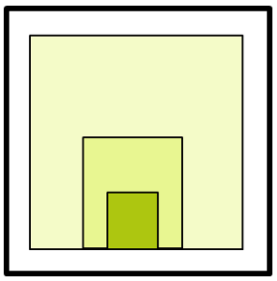


Figure 3.10: Three queries of different density, in a 5×5 m^2 domain

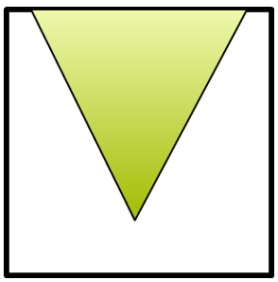


Figure 3.11: Smooth perspective view query, in the 5×5 m^2 domain

3.4.3. VISUALIZATION USING CONTINUOUS LEVELS, MIXED DATA DENSITY

To avoid density shocks (Figure 1.5), the density should not be constant in a perspective view, but depend on the distance to the viewer. For example, when at a distance of 1 meter, the density could match the situation of 1×1 m^2 box with cLoI equal to 5.03. When at 4 meter, the density could match the situation of 4×4 m^2 box with cLoI equal to 1.58. To make it continuous, we can use Equation 3.17 with required cumulative density $TCD_2 = \frac{B}{d^2}$ (with d for distance) to obtain the cLoI values anywhere in the domain. By integrating the wanted density over all distances in the query region (i.e., the triangular shaped view frustum), we can obtain the result (Figure 3.11).

Figure 3.12 demonstrates what happens when we zoom out or change perspective (width of viewing angle). When zooming out, the shape of view frustum remains the same, just getting bigger. Then, with the same budget B, all cLoI values should get smaller to get lower density, bu the cLoI values are still distance dependent. The same reasoning applies when we change perspective (Figure 3.12 Bottom). In Chapter 6, we realized such a perspective view functionality based on PlainSFC, for visualizing AHN2 data. Convincing results have been achieved.

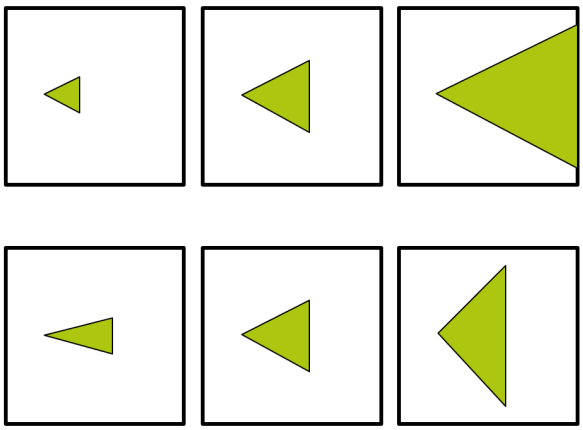


Figure 3.12: Viewer is left side vertex, looking to the right, Top: zooming out, Bottom: changing perspective

3.5. DISCUSSION

This chapter has described a novel cLoI approach to realize smooth visualization of point clouds, without data density shocks. This has been the bottleneck of state-of-the-art solutions. We achieved an optimized continuous distribution of point clouds by iteratively refining discrete levels. We developed both the continuous distribution function and the cumulative distribution function for 1D and nD point clouds. The continuous distribution function is used together with a uniform random number generator to compute the cLoI dimension. Together with the spatial-temporal dimensions, the cLoI dimension can be used to cluster and index the point cloud data. This results in a stable solution for visualizing points: points are not flickering during zooming in and out operations. Besides, the presented cLoI scheme is optimized in the sense that it uses the same factor 2 per dimension as the well known raster and vector data pyramids. This keeps in line with user's habits established from previous spatial applications. Moreover, although only the visualization application is demonstrated, the cLoI developed is also useful to improve different kinds of computation and analysis, including solar energy potential, viewshed or line-of-sight , 3D routing (e.g.,for a drone), change detection, volume analysis, flow computation, vegetation analysis, etc. In the future, we will investigate these applications, e.g., show some directions of how they can be achieved, and develop prototypes.

4

PRINCIPLES OF nD-HISTOGRAM AND EFFECTIVENESS VERIFICATION

BASED on the Morton hierarchy, PlainSFC decomposes the multidimensional space recursively to drive 1D ranges that intersect the query geometry. The query can then be accomplished by selecting points that are indexed by a B+-tree. This highly efficient mechanism makes PlainSFC a potential solution for nD-PointCloud data querying. However, a bottleneck of querying with PlainSFC lies in the cases where points are inhomogeneously distributed in the space. This happens frequently when points are in 3D or higher dimensional spaces. When the skewed dimensions are involved in the query, PlainSFC generates a large amount of ranges without any points inside. This contributes nothing to the final result, but significantly increases the time cost of the first filter. Therefore, the overall querying efficiency declines. Consequently, a distribution-aware method is badly needed for computing ranges.

This chapter addresses the query problem caused by the non-uniform distributions of data. In the following, Section 4.1 first reviews previous studies about querying non-uniformly distributed data, where the nD-histogram technique shows the potential to be used. Then, Section 4.2 focuses on developing a specific nD-histogram technique based on PlainSFC, including building and querying algorithms. To investigate the effectiveness of the nD-histogram given different data distributions, Section 4.3 proposes a statistical metric — Cumulative Hypercubic Coverage (CHC) — for quantifying the uniformity of a point cloud. Based on CHC, Section 4.4 then interprets how the effectiveness of nD-histogram changes with different distributions, theoretically. For verification, Section 4.5 conducts experiments with synthetic data sets incorporating artificially designed uniform distributions as well as distributions simulated based on real data. Section 4.6 provides further discussion on the techniques and results presented in this chapter. More benchmarking results using real world data are provided in Chapter 5.

4.1. RELATED WORK

Former studies have addressed the issue of performance degradation caused by the inhomogeneous data distribution. The techniques mainly include uniforming point distributions, adaptive blocking, and using histograms. Figure 4.1 presents the overview of these techniques with representative studies.

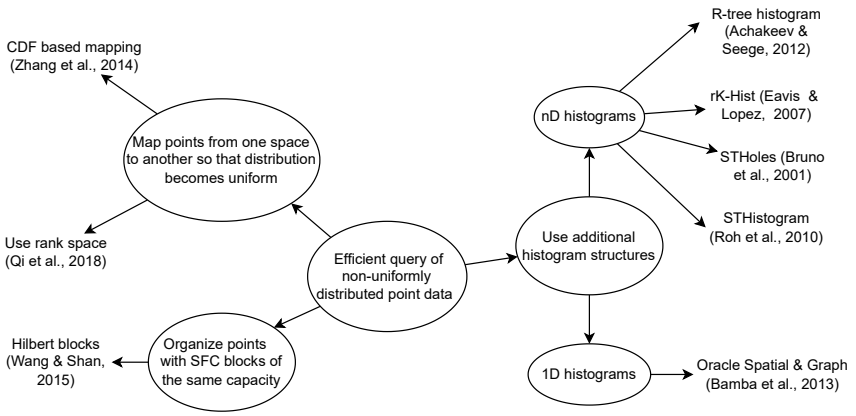


Figure 4.1: Related studies on querying non-uniformly distributed point clouds

Focusing on spatial objects with non-zero extents, Zhang et al. (2014) developed a solution to handle them based on their centroids. The solution utilizes piece-wise Cumulative Distribution Functions (CDFs) to transform each skewed dimension into a uniform dimension independently. Then, the solution uses a similar approach to PlainSFC for managing the transformed data. When querying, the query window has also to be transformed to select data which will be restored in the end. Based on this, although skewed data distribution is resolved, the data transformation is an expensive operation and inapplicable to massive points. Besides, this solution may lead to inaccurate selection due to the decrease of data accuracy after transformation. Qi et al. (2018) took another strategy which maps original points into a rank space such that their coordinates are mapped to their ranks in each dimension (Figure 4.2). After the mapping, points are more uniformly distributed. Then, the solution uses Morton curve to organize and block the points according to their transformed coordinates. Then, R-tree is built to index the blocks. As to querying, window queries should also be firstly mapped to the rank space for execution. According to the testing results, the performance of worst-case window querying is largely improved. However, the drawback of the solution is as before that the transforming process costs excessive time.

Wang and Shan (2005) developed an adaptive block solution which decomposes the 3D point cloud into blocks according to Hilbert curve. The blocks are stored as BLOB objects, with the same capacity. Each block corresponds to a Hilbert node in a certain level. In this way, blocks created are adaptive to the point density. When querying, the solution utilizes an Octree index to retrieve the blocks that overlap with the 3D query window, and then unpacks the blocks to filter point-wisely. So, this solution implicitly handles point distribution when building blocks, and the querying efficiency is directly

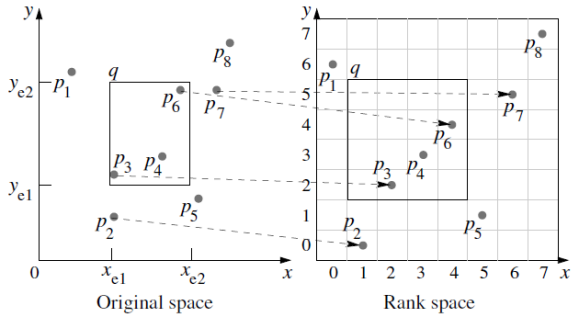


Figure 4.2: Mapping 2D points from original space to the rank space (Qi et al., 2018)

related to the fixed block capacity. This is less flexible than PlainSFC's adaptive range approach.

As a common technique to deal with non-uniform data distribution, histograms are largely used in major DBMSs for querying (i.e., statistics collection module). This is because histograms incur little run-time overhead and produce low-error estimates with compact storage, compared to other techniques such as sampling and wavelet transformation (Liu, 2009). In particular, Oracle Spatial & Graph has developed state-of-the-art solutions to build spatial histograms for query optimization purposes (Bamba et al., 2013). However, these histograms are based on individual columns, instead of a joint representation. Hence, the querying performance on nD data cannot be optimal.

As a possible solution, nD-histograms have been investigated, mostly used as a synopsis technique for selectivity estimation to optimize query execution plan (Liu, Shen, et al., 2021). For instance, Achakeev and Seeger (2012) built a convenient spatial histogram based on R-tree, mainly for managing rectangular or point objects. The multi-dimensional histogram buckets are built by merging Minimum Bounding Boxes (MBBs) of adjacent R-tree leaf nodes until the required parameters of bucket capacity are met. This histogram achieves higher accuracy for selectivity estimation of 2D/3D spatial data queries than alternative solutions, but nD data is not tested. rK-Hist (Eavis & Lopez, 2007) also derives an nD-histogram structure from R-tree. Basically, data are firstly sorted according to the Hilbert curve, to be blocked. Then, R-tree is built for indexing the blocks. The histogram buckets are built by aggregating MBBs of R-tree nodes at a certain level sequentially so that each bucket covers the same number of nodes. R-tree nodes at upper levels are directly transformed to histogram buckets. That is, the hierarchical structure of R-tree is reserved to organize the whole nD-histogram structure. Additionally, optimizations on partitioning histogram buckets are also made to 1) avoid extremely stretched bucket due to Hilbert sorting; 2) guarantee a uniform distribution of points inside each bucket. Experimental tests demonstrate consistent and superior performance of this solution in terms of estimation quality. Besides these R-tree based approaches, nD-histograms can also be constructed differently, such as STHoles (Bruno et al., 2001) and STHistogram (Roh et al., 2010). All these studies provide promising results by using nD-histograms.

4.2. ND-HISTOGRAM METHOD — HISTSFC

We can resolve the skewed distribution problem of PlainSFC by either modifying the data or the solution itself. Modifying the data means to uniform the points, e.g., using the ranking space or CDFs for mapping (Section 4.1). However, concerned with huge volume, which is common for nD-PointCloud, this strategy costs significant time in preprocessing. Yet the conversion may cause data loss. The other method is to optimize the range computation module of PlainSFC to become distribution-aware. Then, to approach the query geometry, PlainSFC decomposes the multidimensional space into finer ranges where the point density is high, while generates coarser ranges where points are sparsely distributed.

This section focuses on developing an adaptive nD-histogram technique that improves the performance of PlainSFC. We assume all dimensions in the nD-PointCloud data are used as the organizing dimension, to ease expression and description. Subsection 4.2.1 first presents the conceptual idea of the nD-histogram design based on PlainSFC. Then, Subsection 4.2.2 and 4.2.3 describe the building and querying algorithms of the nD-histogram, respectively.

4.2.1. CONCEPTUAL DESIGN

I implemented the nD-histogram by using a tree structure — HistogramTree (Liu, van Oosterom, Meijers, Guan, et al., 2020). It records the count of points inside an nD node (Section 2.6.1). If the count exceeds a threshold, i.e., the capacity of a leaf node, then the node is decomposed into 2^n children with corresponding counts. Using HistogramTree, it is expected that the number of vacant ranges generated by PlainSFC can be greatly diminished (Figure 4.3). I call the new solution HistSFC, that uses HistogramTree for range computation.

Figure 4.4 presents the C++ data structure of a node in HistogramTree. A height field is recorded to distinguish different nodes, because branch nodes at different levels may possess identical keys. Alternative design exists. For instance, $\mathscr{MD}$-HBase (Nishimura et al., 2013) employs the longest prefix of the binary SFC key to represent a region. For example, a node with key value 1100** covers child nodes from 110000 to 110011 in 2D space. It is convenient to retrieve the parent node from a child node by truncating the key, e.g., the parent of 1100** is 11****. However, CHAR type should be used to support such a design, which is inefficient for computation and storage compared to the Oracle NUMBER type adopted by HistogramTree.

It should be noted that a HistogramTree node contains neither points nor pointers to points. So, HistogramTree is not an indexing structure. It is an additional structure used by the first filter when computing ranges for a query geometry. It is also compact and can be stored in a flat table.

4.2.2. BUILDING ALGORITHMS

This section describes two algorithms to build HistogramTree — HistML and HistSTREAM.

HISTML

To build the HistogramTree, a conventional top-down approach is to first create a root node to cover the whole point cloud. Then it builds nodes in lower levels by reading all

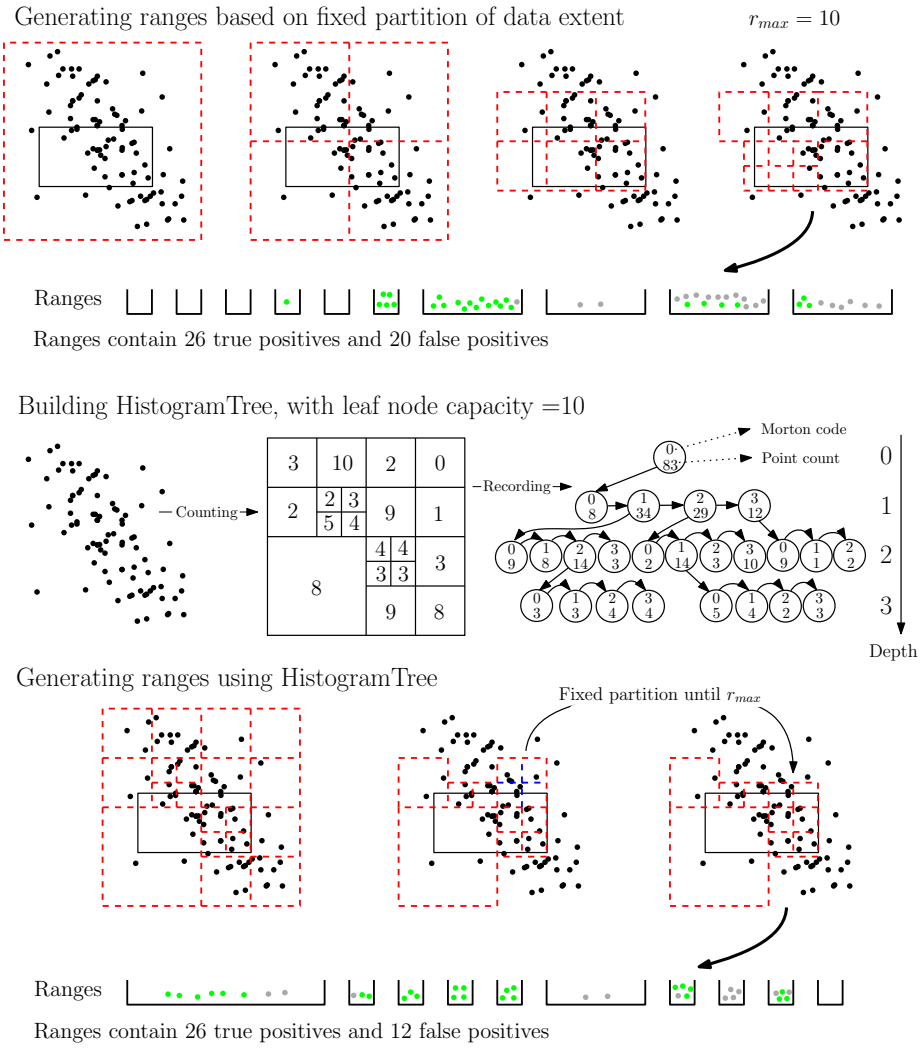


Figure 4.3: Range computation using nD-histograms, where r_{max} refers to the number of ranges

points once per level. This can incur $\mathcal{O}(N \log N)$ I/Os. By contrast, The bottom-up approach starts creating nodes from raw points, and incrementally builds nodes at upper levels. Although this can decrease the I/O cost to $\mathcal{O}(N)$, it leads to significant memory consumption. Consequently, an improved approach is to construct nodes from a middle level (ML), which is the HistML approach (Figure 4.5).

HistML first determines an appropriate ML where most nodes satisfy the capacity threshold. HistogramTree nodes are then built in this level. This is followed by a fixing step in which the approach picks up those overloaded ML nodes, and then reads the point list once more to build nodes below. After it, the top part of HistogramTree is gen-

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

Figure 4.4: The data structure of a node in HistogramTree

erated from ML. In such a way, the memory and time cost can be drastically reduced. However, the challenge of this approach lies in devising a robust algorithm to compute the height of ML. Considering variety and complexity of different data sets, manual interventions may still be needed to determine the ML, e.g., experimenting with a small sample taken from original data.

HISTSTREAM

The streaming approach HistSTREAM, as is shown in Figure 4.6, utilizes the IOT to build HistogramTree. Basically, HistSTREAM reads sorted Morton keys sequentially and meanwhile computes a node N_0 that could be the parent of all traversed keys. This stops until reading a key that belongs to the sibling or parent of N_0, and the number of keys traversed exceeds the leaf node's capacity. Then, HistSTREAM returns to the beginning of this traversal and creates child nodes of N_0. Afterwards, HistSTREAM continues and starts a new traversal. When the scanning of IOT is completed, nodes are aggregated till the root node. Unlike HistML, HistSTREAM has to be implemented after data loading, as the order of the keys is critical. The I/O cost is $\mathcal{O}(N)$, as IOT has already been built. Besides, no matter how large the input is, the memory usage is a constant which depends on the leaf node capacity. Moreover, without the ML determination, HistSTREAM can be a fully streaming process. The actual performance of both algorithms for building HistogramTree is assessed in Section 5.1.3.

4.2.3. QUERYING

HistSFC employs HistogramTree to compute more effective ranges than PlainSFC. Figure 4.7 presents the querying procedure. Starting from the root node, by performing intersection between the HistogramTree and the query window, the function retrieves all relevant nodes to build the range table. Non-overlapping nodes are abandoned. Nodes which are inside the query window are immediately added to the range table with no further processing needed. The nodes on the boundary with few points inside are also exported immediately, e.g., nodes that contain less than 2^n points. The remaining nodes intersecting the boundary of the query window are temporarily held in a refinement pool. These can be further refined based on fixed recursive decomposition. The process stops when the refinement pool is empty or the number of ranges reaches the threshold. The rest of querying remains the same as PlainSFC.

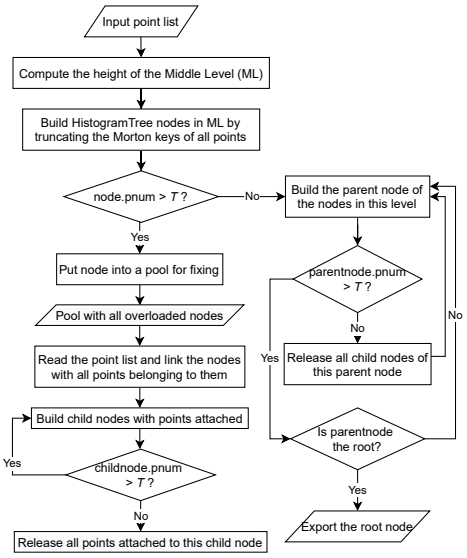


Figure 4.5: HistML, where T refers to the capacity threshold of HistogramTree

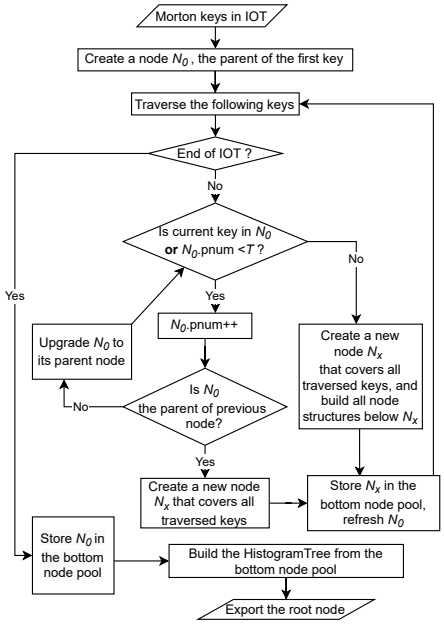


Figure 4.6: HistSTREAM

4.3. CUMULATIVE HYPERCUBIC COVERAGE

In principle, HistogramTree helps improve the performance of querying when data distribution is skewed. However, to what extent can HistogramTree improve the performance? How is this related to specific data distributions? This section proposes a metric — Cumulative Hypercubic Coverage (CHC) — to quantify the uniformity of points. Then, CHC is used to investigate the theoretical effectiveness of HistogramTree in querying nD data, in Section 4.4.

Gunzburger and Burkardt (2004) summarized 8 metrics for measuring the uniformity of points in the nD space. Some metrics are point-to-point measures, such as the variance of the nearest neighbor distance, where the nearest neighbor distance of each point is measured first and the variance of all these distance values is then computed. A small variance indicates a high uniformity. Metrics belonging to this type may describe the regional uniformity well, but fail to grasp the global pattern, e.g., a point set with several clusters with points distributed uniformly inside. The other metrics are volumetric measures where a Voronoi tessellation of the point set is firstly generated and then various quantities associated with the points and the Voronoi regions are determined. However, as these Voronoi regions can deviate significantly from the hypercubic ranges used by PlainSFC or HistSFC in the nD space, volumetric metrics are also not appropriate to describe the uniformity for measuring HistogramTree's effectiveness. Ong et al. (2012) categorized three types of uniformity measures, namely, discrepancy, point-to-point measures, and volumetric measures. Then, they proposed a new measure based on the physical analogy of potential energy, which is advantageous in several aspects

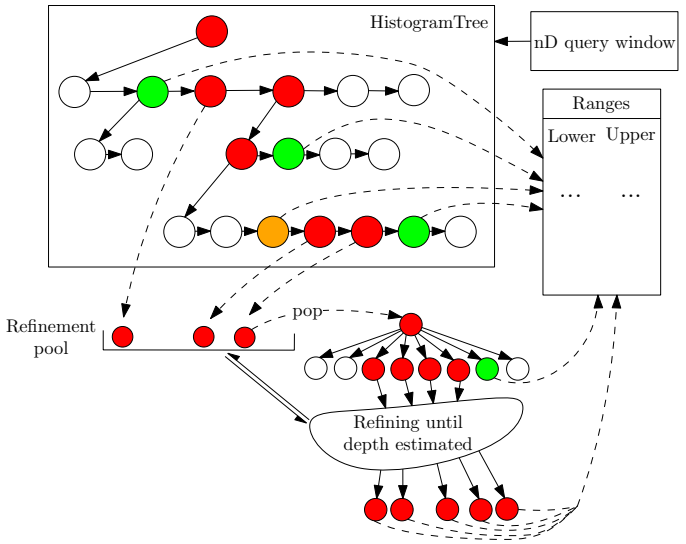


Figure 4.7: Range computation using an nD HistogramTree: with respect to the query window, green nodes are inside; red nodes are on the boundary; orange nodes are on the boundary but with few points; white nodes are outside

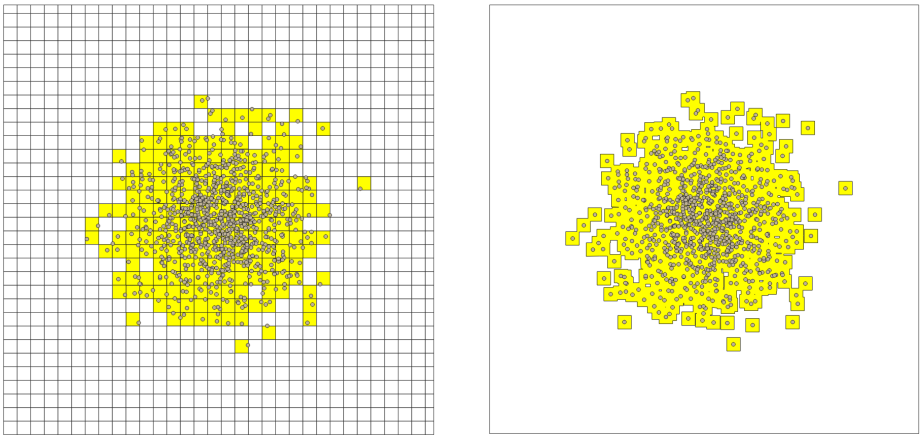
such as consistency and computation complexity. However, they only focused on 2D data sets. Considering the particular problem that hypercubic ranges are used to select nD data, I thus define a new uniformity metric — CHC. The metric is generally applied to nD space, as will be shown in the following formulations.

Given a point set consisting of N points within an nD domain which is defined by the range of all dimensions, suppose the hypervolume of the domain is V. I provide two formulations of CHC, and later prove that they approach to the same expectation when N becomes arbitrarily large:

Formulation 1 (Grid occupancy): We divide the domain into N nD-cells with the same size (Figure 4.8a). Suppose the extent of the ith dimension Di equals E_{Di}, the edge length of the cell at that dimension then equals $\frac{E_{Di}}{\sqrt[n]{N}}$. If at least one point falls into a cell, then the cell is counted. As the hypervolume of each cell is $\frac{V}{N}$, then, $CHC = \frac{1}{V}\frac{\text{count of occupied cells}\cdot V}{N} = \frac{\text{count of occupied cells}}{N}$.

Formulation 2 (Entity union): For each point p, we use it as the center to build an nD-cell c (Figure 4.8b) of which the edge length at Di equals $\frac{E_{Di}}{\sqrt[n]{N}}$. Then, $CHC = \frac{hypervolume(\bigcup c)}{V}$.

In Formulation 1, occupancy grid is used. In fact, it has been widely used for classifying point clouds (Kuhn et al., 2016) and building 3D maps (Saarinen et al., 2013). In ecology, the area of occupancy is proposed and used (IUCN-Committee, 2019). However, due to specific purposes, they use different expressions instead of CHC. Compared with Formulation 2, Formulation 1 is more convenient and efficient to compute. For example, given a DBMS flat table PC storing N points with dimensions $D1$,$D2$, ... Dn, the SQL command "SELECT COUNT(ct)/N from (SELECT COUNT(*) AS ct from PC group by



(a) Grid occupancy (b) Entity union

Figure 4.8: Two forms defining the 2D cumulative hypercubic coverage (yellow)

TRUNC(D1/el_{D1}), TRUNC(D2/el_{D2}), ... TRUNC(Dn/el_{Dn}) HAVING ct>0)" can be used to derive CHC: el_{Di} refers to the edge length of a cell in Di, which equals $\frac{E_{Di}}{\sqrt[n]{N}}$. In the following, I provide theorems that show CHC's property of being a constant when N becomes arbitrarily large.

Theorem 1: According to Formulation 1, given a point set following a specific distribution, when N becomes arbitrarily large, then, the expectation of CHC is a constant.

Proof: Assuming the point set is distributed in a **2D unit domain**, for a specific cell c defined by its center (x, y), the probability that it is occupied can be computed:

$$P(c(x,y)) = 1 - (1 - F_{c(x,y)})^N = 1 - (1 - \frac{\overline{f_{c(x,y)}}}{N})^N \tag{4.1}$$

where $F_{c(x,y)}$ is the cumulative probability in c, and $\overline{f_{c(x,y)}}$ refers to the average probability density. When $N \to \infty$, the size of c becomes arbitrarily small, we have

$$\lim_{N\to\infty} P(c(x,y)) = \lim_{N\to\infty} \left(1 - (1 - \frac{\overline{f_{c(x,y)}}}{N})^N\right) = 1 - e^{-f(x,y)} \tag{4.2}$$

where e is the Euler's number, approximately equal to 2.71828. The specific derivation is based on $\lim_{N\to\infty}(1 - \frac{1}{N})^N = \frac{1}{e}$. We just need to change it to $\lim_{N\to\infty}(1 - \frac{\overline{f_{c(x,y)}}}{N})^N$, to derive the limitation which equals $e^{-\overline{f_{c(x,y)}}}$.

Since

$$E(CHC) = \sum_{i=1}^{N} \frac{P(c_i)}{N}$$

when $N \to \infty$, we derive

$$E(CHC) = \iint_\Omega P(\sigma)\, d\sigma = \iint_\Omega \left(1 - e^{-f(x,y)}\right) dx\, dy \tag{4.3}$$

where Ω refers to the unit domain, $[0, 1] \times [0, 1]$ in this case.

It is easily to extend current theorem to the nD unit hypercubic domain, where f_n represents the joint Probability Density Function (PDF):

$$E(CHC) = \int \cdots \int_{\Omega} \left(1 - e^{-f_n}\right) dv \tag{4.4}$$

In reality, it is very likely that a point cloud can spread over a much larger space than a unit domain. In such cases, the PDF can firstly be scaled to the unit domain, and the expectation can then be derived. Such scaling does not change CHC because a cell in the original space corresponds to a distinctive cell in the unit domain. So, the count of occupied cells remains the same. □

Theorem 2: In Formulation 2, the expectation of CHC converges to the same constant as in Formulation 1, when N becomes arbitrarily large.

Proof: Based on the 2D unit domain assumption, the difficulty to compute CHC using entity union lies in computing the overlapping area of different cells. So, instead of the cell, we focus on a small Δs region ($\frac{1}{N} \gg \Delta s$) in the domain that will be covered by the union of cells. We use two bounds to derive the expectation of the area of which Δs is covered, denoted by $E(s)$. As Figure 4.9 shows, when a point falls into the red box, Δs is considered to be totally covered. Apparently, the lower bound omits the region where an entity intersects Δs, while the upper bound regards all partial intersection as full coverage. This yields

$$\Delta s F_{\sigma_L} \leq E(s) \leq \Delta s F_{\sigma_U} \tag{4.5}$$

where σ_L refers to the effective region that a point falls into of the lower bound, while σ_U refers to that of the upper bound. Besides,

$$E(s) = \iint_{\sigma_U} s(x, y) f(x, y)\, dx\, dy = \Delta s F_{\sigma} \tag{4.6}$$

where σ represents the average effective area that once a point falls into it, Δs is covered. σ is a probabilistic concept. Then, according to Equation 4.5 and 4.6, when $\Delta s \to 0$, the size of σ approaches to a cell. Suppose $P(\sigma)$ is the actual probability that Δs is covered by the entity union of the point cloud. Then, we could derive a same expression of $P(\sigma)$ as Equation 4.1. Then, we adopt the size of Δs so that the whole domain can be decomposed to m Δs pieces, where $m \in \mathbb{Z}$ and $m \gg N$. Based on this,

$$E(CHC) = \sum_{i=1}^{m} (P(\sigma_i)\Delta s) \tag{4.7}$$

When $\Delta s \to 0$,

$$E(CHC) = \iint_{\Omega} \left(1 - e^{-f(x,y)}\right) dx\, dy \tag{4.8}$$

This is the same as Equation 4.3. Analogously, we can extend the derivation of CHC's expectation to the nD domain, and the expression is the same as Equation 4.4. □

I also performed a numerical simulation to illustrate the theorems. In Figure 4.10, the CHC of a 2D point cloud with a joint Gaussian distribution is computed. X and Y are two

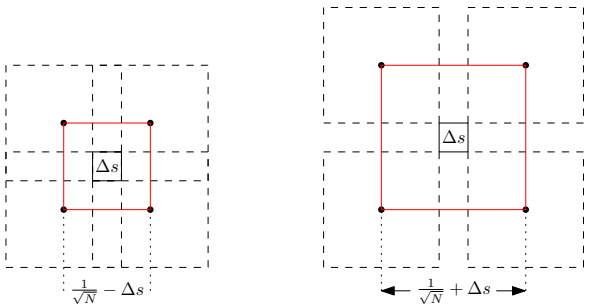


Figure 4.9: Area of Δs being covered, lower bound on the left, upper bound on the right

independent dimensions, both following $\mathcal{N}(0.5, 0.1)$ (shown in Figure 4.8). Figure 4.10 clearly shows that CHC computed by the two methods gradually converge to the true value which is given by Equation 4.4.

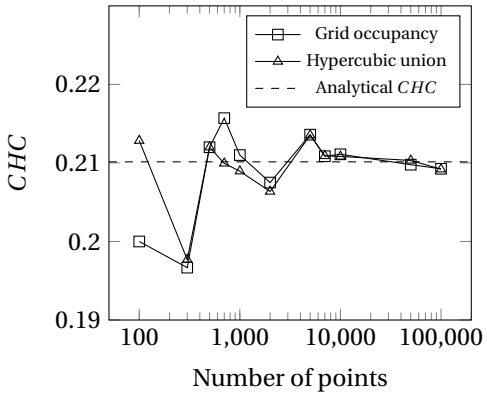


Figure 4.10: Numerical simulation of CHC using both methods

Considering the stable nature of dimensions, the point distribution cannot be changed significantly by adding more data. In other words, CHC can be seen as an intrinsic property of a point cloud. Besides, CHC is computed based on accumulating the hypercubes, which keeps consistent with the querying strategy of PlainSFC and HistSFC. Therefore, CHC is used to interpret how the effectiveness of HistogramTree changes with different distributions.

4.4. EFFECTIVENESS OF ND-HISTOGRAM

As is mentioned, when points are distributed non-uniformly, part of the ranges computed by PlainSFC will contain no points. To evaluate this quantitatively, I define the Empty Ratio (ER) of ranges (Equation 4.9), where a **vacant range** has no point inside. ER can be used to evaluate both PlainSFC and HistSFC.

ER cannot directly indicate the effectiveness of HistogramTree, and another metric is

needed. First, I define an **effective range** as a range exported by PlainSFC that contains at least one point. Assume the capacity threshold of HistogramTree is 1 (i.e., the highest precision), so that HistSFC only returns effective ranges. Then, I use Equation 4.10 to measure the effectiveness of HistogramTree with respect to a query.

$$ER = \frac{\text{Number of vacant ranges exported}}{\text{Total number of ranges exported}} \tag{4.9}$$

$$E_{hist} = \frac{\text{Total number of ranges exported by PlainSFC}}{\text{Number of effective ranges exported by PlainSFC}} \tag{4.10}$$

Based on the formulations above, $E_{hist} = (1 - ER_{PlainSFC})^{-1}$. The following subsections explores how E_{hist} changes with point distribution which is represented by CHC. Subsection 4.4.1 provides a theorem indicating the overall relationship between CHC and E_{hist}. Then, Subsection 4.4.2 specifically investigates the influence of dimensionality on E_{hist}, while Subsection 4.4.3 focuses on the effect of correlation.

4.4.1. HOW EFFECTIVENESS CHANGES WITH CUMULATIVE HYPERCUBIC COVERAGE

Intuitively, the more severely skewed the data distribution is, the more effective should the HistogramTree be. I prove this in the following.

Theorem 3: Given point sets with different distributions in nD space, each point set contains N points. For a window query, the expected E_{hist} is a monotonically decreasing function of CHC.

Proof: We first build the occupancy grid (Formulation 1) for each point set (Figure 4.11). Assuming l is the search depth, as N may not equal 2^{nl} which is the number of subspaces, we set l to $\left\lceil \frac{\log_2 N}{n} \right\rceil$. With current setting of HistogramTree, such a depth can be reached. Based on this, we assume the query window totally matches the boundaries of SFC cells at l.

In Figure 4.11, the incremental process of CHC reveals how effective range and E_{hist} change. Starting from $CHC \to 0$, i.e., all points reside in a grid cell, and there is no effective range. So, $E_{hist} \to +\infty$. In practice, this means HistSFC can immediately report an empty set as the result, instead of PlainSFC's strategy that first exports a set of ranges and then retrieves no points. When a point moves from the central grid cell to another, which means CHC increases, E_{hist} decreases due to a higher probability to encounter a point inside (middle sub-figure). When more points move out from the original grid cell, E_{hist} is most likely to decrease again or remain the same (right sub-figure). Sometimes, it is likely that two points belong to different cells of the occupancy gird, but resides in the same SFC cell, due to the mismatch of these two cell size. In this case, E_{hist} also remains the same, and will not increase. Consequently, for a large number of random window queries, $\overline{E_{hist}}$ is monotonically decreasing function of CHC. □

Theorem 3 indicates a general pattern: given two nD point clouds A and B where $CHC_A > CHC_B$, for a large number of window queries, $\overline{E_{Ahist}} < \overline{E_{Bhist}}$. The smallest $\overline{E_{hist}}$ equals 1, which happens when points follow a chessboard distribution (i.e., CHC = 1) which is a correlated uniform distribution. When this happens, HistogramTree is not needed. In all other cases, $\overline{E_{hist}} > 1$.

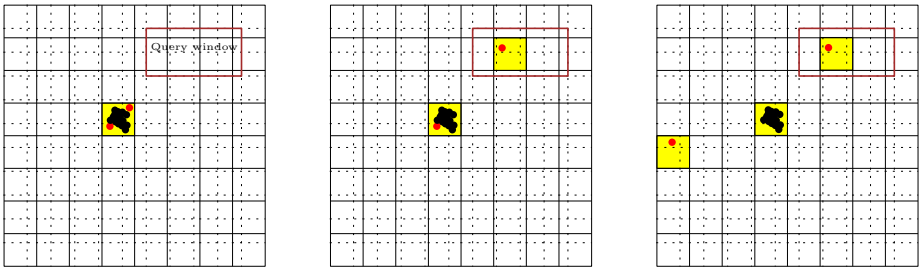


Figure 4.11: The change of effective ranges with respect to CHC, where solid lines constitutes the occupancy grid, while dash lines indicate the subdivision of space

4.4.2. INFLUENCE OF DIMENSIONALITY

CHC is not necessarily related to dimensionality. For instance, when all dimensions follow a uniform distribution independently, according to Equation 4.4, CHC always equals a constant value which is about 0.63212. In other cases, dimensionality may influence CHC, and therefore E_{hist}. To investigate the influence of dimensionality on E_{hist}, I categorize distributions into the uniform and the non-uniform ones considering common properties of dimensions (Section 2.1.2). Besides, I use the term m-nD query to express among n organizing dimensions, the query window only specifies m out of them. This implies that the other $n-m$ dimensions are fully selected. Based on this, I establish the following theorem.

Theorem 4: Given a point cloud consisting of N points, the number of ranges used for a m-nD query is r. When an independent uniform dimension is added as an organizing dimension, E_{hist} remains the same for the new m-$(n+1)$D query. In contrast, when the added organizing dimension follows non-uniform distribution, E_{hist} increases.

Proof: We first focus on a 2-nD query. Starting from a 2D point set in the unit domain, suppose the whole coverage of the query window is C and the coverage of effective ranges is C^E_{2D}. Then, by adding an organizing dimension with independent uniform distribution, the coverage of the 3D query window (with full range of the added dimension) is still C. Hence, the coverage of each 2D range and each 3D range are both equal to $\frac{C}{r}$. As a result, the number of effective ranges generated equals $\frac{rC^E_{2D}}{C}$ for the 2-2D query, and $\frac{rC^E_{3D}}{C}$ for the 2-3D query. As $C^E_{2D} = \iint_C \left(1-e^{-f(x,y)}\right) dx\,dy$ and the third dimension $f(z)=1$, then, $C^E_{3D} = \int_0^1 \iint_C \left(1-e^{-f(x,y)\cdot f(z)}\right) dx\,dy\,dz = C^E_{2D}$. So, the number of effective 2D ranges and 3D ranges both equal $\frac{rC^E_{2D}}{C}$. With this, we derive that for the 2-nD query,

$$E_{hist3D} = E_{hist2D} = \frac{C}{C^E_{2D}} \tag{4.11}$$

We could extend this to n organizing dimensions, with the assumption that the query window totally matches the boundary of all nD Morton nodes and N is large enough. Based on this proof, we could derive that for a m-nD query, Equation 4.11 also holds. On the other hand, when the added dimension follows non-uniform distribution such as the extreme distribution, then, $C^E_{3D} < C^E_{2D}$. This is because instead of spreading over

the third dimension, points will be absent in some parts. As a result, E_{hist3D} increases. This can also be extended to nD obviously. □

An implication of Theorem 4 is that for a specific m-nD query, E_{hist} will not change by converting other uniform property dimensions into organizing dimensions. Referring to Equation 2.1, such conversion has insignificant influence on the time cost. However, when the converted dimensions follow skewed distributions, the gap between time cost of PlainSFC and that of HistSFC should become more significant.

4.4.3. INFLUENCE OF CORRELATION

Dimensions may not always be independent from each other. Take the flood computation as an example (Section 1.2), water depth correlates with elevation: normally, the lower the elevation is, the deeper the inundation will be. Also, the GPS trajectory points are normally distributed along paths. So, X and Y are correlated (non-)linearly, and $f(x,y) \neq f(x)f(y)$. This influences CHC as well. Intuitively, correlation decreases CHC, compared to independent distributions of dimensions. This section investigates how data correlation affects E_{hist}. The following theorem derives a general pattern of E_{hist} based on the linear correlation.

Theorem 5: Given independent variable X and Y, The corresponding PDFs are $f(x)$ and $f(y)$. When X and Y become linearly correlated, but with $f(x)$ and $f(y)$ unchanged, then, $0 < P(CHC_{cor} \leq CHC_{ind}) < 1$ holds (where P stands for probability; CHC_{cor} and CHC_{ind} refer to CHC in the correlated case and independent case respectively). In other words, correlation can also increase CHC. Consequently, the change of E_{hist} cannot be generally derived.

Proof: A generic explanation may be tough to reach due to the implicit primitive functions of the PDFs. Thus, I prove the theorem by using rationale examples. I first performed numerical simulation to test a bivariate normal distribution. Figure 4.12 indicates that $CHC_{cor} \leq CHC_{ind}$ when correlation happens (i.e., $\sigma_{xy} > 0$). Furthermore, it also shows CHC keeps decreasing as $COV(X,Y)$ increases.

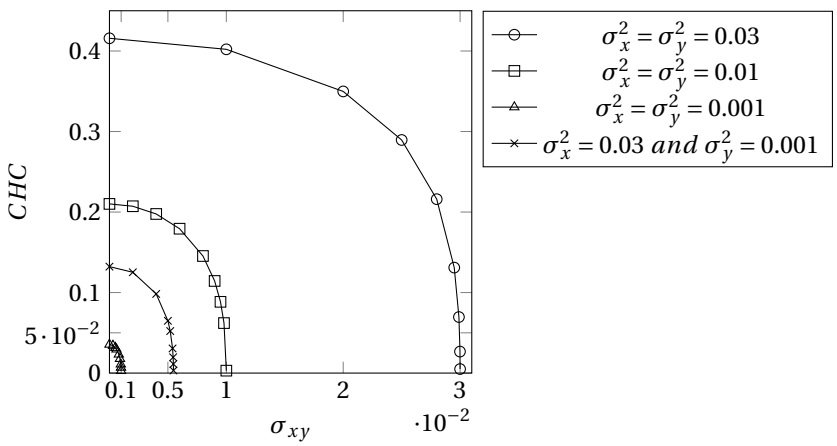


Figure 4.12: Numerical simulation of bivariate normal distribution and CHC, $\mu_x = \mu_y = 0.5$

On the other hand, I provide a counter example, as is shown in Figure 4.13. Given a distribution where X and Y are independent, we focus on the number of points in 4 cells shown in Figure 4.13 which are noted as Num_1 to Num_4: $Num_1 = m$, $Num_2 = 0$, $Num_3 = m + k$ and $Num_4 = k$, where $m \geq 2$ and $k \geq 2$. This case is rationale as $Num_1 - Num_2 = Num_3 - Num_4 = m$ and $Num_3 - Num_1 = Num_4 - Num_2 = k$, i.e., Y has no influence on the distribution of X since they are independent variables, and vice versa. Now we move one point from cell 1 to cell 2, and another point from cell 4 to cell 3. In this way, $f(x)$ and $f(y)$ are unchanged. Then, the original coordinate pairs change from (x_1, y_1) and (x_2, y_2) to (x_2, y_1) and (x_1, y_2). Considering $COV(X, Y) = E(X - \mu_X)(Y - \mu_Y) = E(XY) - \mu_X\mu_Y$, we could derive $\Delta COV = (x_2 - x_1)(y_1 - y_2) > 0$. In other words, the moving operation has added correlation to the data. However, with respect to the grid occupancy, cell 2 will be filled in, which causes the increase of CHC. Therefore, $CHC_{cor} > CHC_{ind}$ in this case. Besides, the chessboard distribution where $CHC = 1$ is also a typical case of non-linear correlation. This means any other independent distributions will only cause the reduction of CHC. Based on the analysis above, we cannot draw a general conclusion on how E_{hist} changes. □

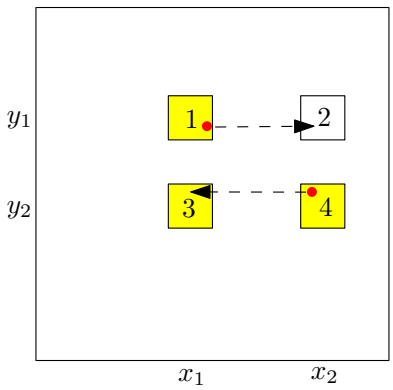


Figure 4.13: The change of CHC with correlation, by moving points

It should be noted that whatever correlation it is, CHC can always be used to quantify the uniformity of points and evaluate how skewed a point distribution is. Besides, HistogramTree deals with correlations implicitly, as it focuses on where the points are located instead of the specific correlation.

4.5. EXPERIMENTAL VERIFICATION

Experiments are conducted to learn how distribution influences the effectiveness of HistogramTree practically. This also aims to verify the theorems proposed above. The first ideal test is based on uniform and partial uniform distributions, where the analytical CHC values can be computed (Equation 4.4). The second test simulates the distribution of dimensions according to real nD-PointCloud data. So, the result derived should be more close to reality.

4.5.1. INDICATORS FOR EXPERIMENTS

Previously, the ideal E_{hist} (Equation 4.10) is developed for proving theorems, where HistogramTree is supposed to have a leaf node capacity of 1. In practice, this can hardly be achieved given a large input. Thus, this experimental section uses two variants of E_{hist} to measure the effectiveness of HistogramTree (Equation 4.12 and 4.13). They are suitable for different scenarios.

$$E'_{hist} = \frac{\text{Number of ranges exported by PlainSFC}}{\text{Number of ranges exported by HistSFC}} \tag{4.12}$$

$$E''_{hist} = \frac{FPR_{PlainSFC}}{FPR_{HistSFC}} \tag{4.13}$$

where $FPR_{PlainSFC}$ and $FPR_{HistSFC}$ stand for False Positive Rate (FPR) of PlainSFC and HistSFC, respectively.

Given a large point set, we build HistogramTree with a proper capacity threshold. To fully benefit from HistogramTree, HistSFC searches to the level of leaf nodes. For a fair comparison, PlainSFC should also search to the same depth. In this case, E'_{hist} can be used to evaluate the effectiveness of HistogramTree. The larger E'_{hist} is, the more effective HistogramTree functions. However, this is not the default way of using PlainSFC and HistSFC where a maximum number of ranges (r_{max}) is set in advance for query execution, as r_{max} cannot guarantee the search reaches the bottom level of HistogramTree.

On the other hand, E''_{hist} is more practical. As has been mentioned in Section 2.6.3, FPR of the first filter is a key metric to evaluate the efficiency in practice. With the default use of PlainSFC and HistSFC, E''_{hist} can be adopted to indicate the effectiveness of HistogramTree. This is because given a fixed r_{max}, more vacant ranges implies less accuracy of the effective ranges. This corresponds to a large FPR, and vice versa. Hence, a larger E''_{hist} also indicates higher effectiveness of HistogramTree.

4.5.2. IDEAL TEST

In this experiment, I tested PlainSFC and HistSFC with 2D to 10D points. The coordinate value for each dimension uses 12 bits so that the 10D Morton key requires 120 bits which fits within Oracle NUMBER type (128 bits). I only used two types of distributions: one is uniform distribution in the whole domain; the other is partial uniform distribution which spans half the domain but with random boundaries. I established 6 Ideal Data Groups (IDGs), from IDG1 to IDG6, with each consisting of $(2i)D$ ($i \in [1,5]$) data sets. Table 4.1 shows the distributions of the dimensions in each data set. For instance, with respect to the IDG4 8D data set, the first 6 dimensions follow partial uniform distributions with different boundaries while the last 2 dimensions follow the uniform distribution. For all data sets, dimensions are independent from each other.

Each 2D, 4D, 6D, 8D and 10D data set contains 10^4, 10^6, 10^7, 10^8 and 10^{10} points, respectively. This is because as the domain range can maximally be 4095, too many points in low dimensional spaces can cause redundancy, which undermines the distribution. For instance, the 2D data set can maximally contain 16,777,216 unique points. To avoid redundant points generated by the uniform distribution, the total points modelled should even be less. On the other hand, insufficient number of points for testing

Table 4.1: Distributions adopted by different data sets: U stands for the uniform distribution, and PU represents the partial uniform distribution

	2D	4D	6D	8D	10D
IDG1	2 U	4 U	6 U	8 U	10 U
IDG2	2 PU	2 PU + 2 U	2 PU + 4 U	2 PU + 6 U	2 PU + 8 U
IDG3	2 PU	4 PU	4 PU + 2 U	4 PU + 4 U	4 PU + 6 U
IDG4	2 PU	4 PU	6 PU	6 PU + 2 U	6 PU + 4 U
IDG5	2 PU	4 PU	6 PU	8 PU	8 PU + 2 U
IDG6	2 PU	4 PU	6 PU	8 PU	10 PU

will cause inaccurate computation of CHC (Figure 4.10). Then, Formulation 1 is used to practically compute CHC. Table 4.2 lists the tested CHC values and theoretical values.

Table 4.2: CHC (tested/theoretical) of different data sets: different color represents different category divided by the order of magnitude

	2D	4D	6D	8D	10D
IDG1	0.635/0.632	0.641/0.632	0.658/0.632	0.632/0.632	0.632/0.632
IDG2	0.250/0.245	0.267/0.245	0.297/0.245	0.297/0.245	0.297/0.245
IDG3	0.249/0.245	0.081/0.063	0.107/0.063	0.110/0.063	0.108/0.063
IDG4	0.252/0.245	0.070/0.063	0.032/0.016	0.041/0.016	0.039/0.016
IDG5	0.248/0.245	0.077/0.063	0.027/0.016	0.011/0.004	0.015/0.004
IDG6	0.248/0.245	0.082/0.063	0.029/0.016	0.014/0.004	0.005/0.001

Table 4.2 clearly shows the gradually decreasing pattern of CHC from IDG1 2D to IDG6 10D. However, due to insufficient number of points generated, the tested CHC values deviate from the theoretical values. Nonetheless, the tested CHC values are categorized by 3 different orders of magnitude, which is significant enough to verify how HistogramTree's effectiveness changes with CHC.

Then, I built HistogramTree by decomposing the space to a specific depth, instead of using the default approach based on node capacity (Section 4.2.2). In this way, all leaf nodes have the same size in space, which keeps in line with the setting in the theorems. To determine the depth for space decomposition, for one thing, the size of leaf node should approach the cell size of the occupancy grid as much as possible; for another, the HistogramTree should not cost too much memory. Take IDG6 10D data set as an example, the partition depth is expected to be 3 so that every dimension can be decomposed into 8 segments which approach the 10 segments of the occupancy grid. However, in that case, the HistogramTree would comprise more than 1 billion nodes, which can bloat the memory. So, I built the HistogramTree with only 3 levels including the root. Table 4.3 shows the settings of HistogramTree. The maximum size of HistogramTree always occurs in IDG1, as its points spread over the whole space.

When querying, I randomly generated 500 nD hypercubes with the same size but different positions. They were used as query windows. To guarantee enough points selected, for 2D, 4D and 6D tests, I used 410 as the edge length of each hypercube, which accounts for 10% of the length of each dimension. For 8D and 10D tests, I used 1000 as

Table 4.3: Settings of the HistogramTree for all data groups

	2D	4D	6D	8D	10D
Depth of leaf nodes	7	5	4	3	2
Edge length of a leaf node	32	128	256	512	1,024
Maximum number of nodes ($\times 10^3$)	13	714	7,800	16,800	1,050
Maximum storage size (MB)	0.6	30	331	626.5	40

the edge length, accounting for 25% of each dimension. A query window has first to be examined if it intersects the data region, to assure points can be selected. For a query, HistSFC searched to the bottom level of the HistogramTree, and PlainSFC searches to the same level. Based on this setting, I adopted E'_{hist} (Equation 4.12) to measure the effectiveness of HistogramTree.

Table 4.4 lists E'_{hist} derived. In 2D, as all data groups adopt the uniform distribution, E'_{hist} fluctuates gently. Unlike the chessboard distribution, uniform distribution can generate holes in the data space. Hence, HistogramTree also works positively. In 4D and 6D, we observe a clear gap among E'_{hist} belonging to different category of CHC. It shows that with the decrease of CHC, E'_{hist} increases. In 8D and 10D, the influence of CHC becomes more significant where E'_{hist} keeps increasing as CHC declines from IDG1 to IDG6. An odd pattern occurs that the increment of E'_{hist} in 10D is smaller than 8D. This is most likely caused by the different searching depth using HistogramTree. With a lower depth of leaf nodes, the 10D HistogramTree provides less information about distribution, and is thus less effective.

Table 4.4: Average E'_{hist} of different data sets, colored according to the value category of the tested CHC

	2D	4D	6D	8D	10D
IDG1	1.86	1.64	2.21	1	1
IDG2	1.96	1.72	2.00	1.65	1.46
IDG3	1.82	3.17	2.08	3.04	1.84
IDG4	2.10	2.70	2.76	4.73	2.95
IDG5	1.70	2.45	3.39	9.02	3.37
IDG6	2.18	2.70	2.52	9.89	5.04

In fact, this ideal test is also devised to verify Theorem 4 which asserts that by adding uniform dimensions into the organization of data storage will not influence the effectiveness of HistogramTree. However, as the settings of HistogramTree is confined by the hardware and software (e.g., memory size and number of bits to use), the pattern is not clearly shown. Nonetheless, from Table 4.4, we do not observe an evident decreasing trend of E'_{hist} in each data group, as dimensionality rises. Instead, IDG5 and IDG6 show an increasing trend to some extent.

4.5.3. REALISTIC SIMULATION AND TEST

To further test the effectiveness of HistogramTree, I devised another experiment with realistically simulated data. I first collected point clouds from different sources such as

indoor laser scanning or ALS, and then plotted distributions of the dimensions involved. In addition to the uniform distribution of X and Y, normal distribution and Gamma distribution also commonly exist, e.g., classification and intensity. So, I simulated 6 dimensions (from D_1 to D_6) with different distributions. D_1 and D_2 follow the uniform distribution of which the normalized PDF is $\mathscr{U}(0,1)$. Other dimensions follow skewed distributions where different parameters are adopted to incorporate different skewness. Table 4.5 and Figure 4.14 present the distributions of D_3 to D_6. The value of each dimension is based on 20 bits, ranging from 0 to 2^{20}. This allows more distinct values in each dimension, which is more realistic than the previous experiment.

Table 4.5: Distributions of different dimensions simulated based on real data

	D_3:Gamma1	D_4:Normal	D_5:Gamma2	D_6:Gamma3
Gentle	$\Gamma(1,2)\times 2^{17}$	$\mathscr{N}(2^{19},2^{18})$	$\Gamma(2,3)\times 2^{15}$	$\Gamma(10,2)\times 2^{16}$
Sharp	$\Gamma(0.05,1)\times 2^{17}$	$\mathscr{N}(2^{19},2^{17})$	$\Gamma(10,0.1)\times 2^{15}$	$\Gamma(820,0.02)\times 2^{16}$

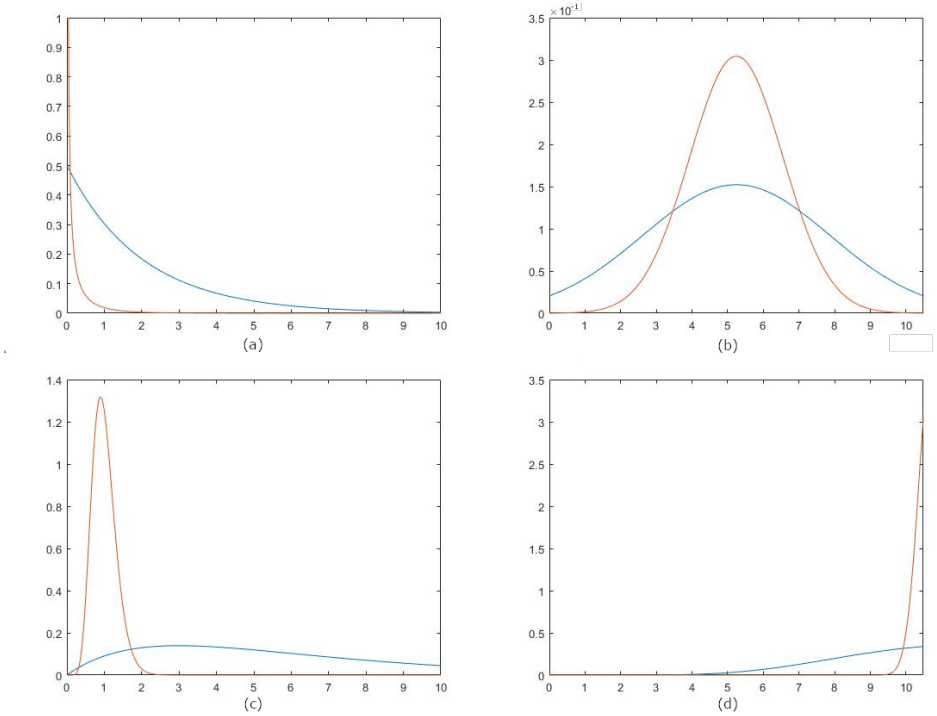


Figure 4.14: Normalized probability density functions: (a) D_3, (b) D_4, (c) D_5, (d) D_6, where red lines are the sharp distributions while blue lines are the gentle ones

Then, I built 5 “Realistic" Data Groups (RDG) from 3D to 6D, with different dimensions (Table 4.6). Each data set contains 10,000,000 points. Table 4.7 lists the CHC values of different data sets. Some CHC values are omitted because their data sets repeat the

dimensions included by other data sets. In this experiment, I adopted the regular procedure to build HistogramTree based on capacity threshold: 3D and 4D solutions used 100 as the capacity, while 5D and 6D adopted 1000, instead. When querying, I randomly generated 500 nD query windows with varying edge lengths at each dimension, but guaranteed at least 100 points to be selected. The maximum number of ranges for 3D and 4D querying was 1000; while that for 5D and 6D querying was 10,000. Based on these settings, E''_{hist} (Equation 4.13) is more appropriate to be used as the effectiveness metric.

Table 4.6: The additional dimension added to each data set. In each data group, D_1, D_2 are always used. Besides, a higher dimensional data set includes all dimensions of the lower dimensional data set, with the additional dimension added

	3D	4D	5D	6D
RDG1	gentle D_3	gentle D_4	gentle D_5	gentle D_6
RDG2	sharp D_3	gentle D_4	gentle D_5	gentle D_6
RDG3	sharp D_3	sharp D_4	gentle D_5	gentle D_6
RDG4	sharp D_3	sharp D_4	sharp D_5	gentle D_6
RDG5	sharp D_3	sharp D_4	sharp D_5	sharp D_6

Table 4.7: CHC values of different data sets, colored by different orders of magnitude

	3D	4D	5D	6D
RDG1	0.4790	0.3329	0.2210	0.1571
RDG2	0.0704	0.0573	0.0484	0.0410
RDG3	-	0.0102	0.0082	0.0090
RDG4	-	-	0.0017	0.0019
RDG5	-	-	-	0.0005

Table 4.8 presents the average E''_{hist} tested. The result once again verifies that HistogramTree works more effectively when querying a point cloud having a larger CHC, with randomly distributed query windows. Given a certain dimensionality, the gap between different data sets becomes more significant when CHC values are of different orders of magnitude. When the point cloud's CHC is extremely low, such as the RDG4 5D, RDG4 6D and RDG5 6D data set, HistogramTree must be used. Otherwise, the querying process can be orders of magnitude slower due to large FPR (reflected by E''_{hist}). Considering that skewed dimensions commonly exist in point clouds (Chapter 5), HistogramTree is therefore an essential technique to improve the querying performance.

4.6. DISCUSSION

To improve PlainSFC's performance on querying non-uniformly distributed data, this chapter developed an nD-histogram method — HistogramTree. It records the information of point distribution and can thus be used to guide range computation in the first filter. The intention is to keep HistogramTree in the memory after the first query, so that it will not be repeatedly loaded for subsequent queries. Hence, HistogramTree should also be compactly built, e.g., with sufficient capacity threshold. Besides, HistogramTree

Table 4.8: Average E''_{hist} of different data sets, colored according to categories of CHC values

	3D	4D	5D	6D
RDG1	1.01	1.05	1.06	1.35
RDG2	3.70	5.08	2.92	4.24
RDG3	-	6.44	6.68	8.00
RDG4	-	-	40.83	10.07
RDG5	-	-	-	62.17

can always be used whatever the distribution is, as it does not bring additional time cost for querying uniformly distributed data.

To evaluate how data distribution relates to the effectiveness of HistogramTree (E_{hist}), I first proposed a uniformity metric, CHC, to quantify the coverage/volume of a point cloud based on adaptive hypercubes. Then, I established theory that indicates E_{hist} increases monotonically as CHC decreases, i.e., Theorem 3. The influence of the number of organizing dimensions and correlation between dimensions on E_{hist} was also discussed afterwards. The tests with ideal data (Section 4.5.2) and realistically simulated data have verified Theorem 3 that HistogramTree functions more effectively for point clouds with smaller CHC. More specifically, the tests indicated that when the CHC value is smaller than 0.1, HistogramTree becomes essential to use.

In practice, as CHC is convenient to compute based on the occupancy grid, developers are suggested to first measure the CHC before determining the final solution. If a point cloud is too large, it can firstly be sampled to derive a subset with sufficient number of points for computing CHC. This is because CHC is only related to the point distribution, regardless of the total number of points (Equation 4.4). Moreover, CHC provides an intuitive indication of how data is skewed in nD space. This is significant because human beings can hardly imagine and comprehend data distributions in nD. From this point of view, CHC has the potential to facilitate more studies on point clouds, besides the investigation into nD-histograms.

5

BENCHMARKING AND OPTIMIZING HISTSFC IN PRACTICE

AFTER acquiring the convincing results of HistSFC by theoretical verification and simulation, it is crucial to evaluate the performance in practice. This chapter elaborates benchmark results on real data including a 4D Airborne Laser Scanning (ALS) point cloud in Section 5.1 and an 8D flood modelling result set in Section 5.2. The effect of HistogramTree size and the number of ranges on querying efficiency is investigated. Besides PlainSFC and HistSFC, state-of-the-art solutions including Pyramid-Technique, PostGIS MultiPoint solution and Oracle SDO_PC are also benchmarked for comparison. After this, Section 5.3 explores different techniques to optimize HistSFC further. In the end, Section 5.4 further discusses the benchmark tests and provides more implications.

All benchmark tests are performed on a HP DL380p Gen8 server with 2 × 8-core Intel Xeon processors, E5-2690 at 2.9 GHz, 128 GB of main memory, a RHEL6 operating system. The disk storage is a 41 TB SATA 7200 rpm in RAID6 configuration.

5.1. AHN2 TEST

AHN2 is an ALS point cloud recording the terrain elevation of the whole Netherlands (AHN, 2014). It contains totally 640 billion points, with a density of 6-10 points/m^2. I cropped a sample which locates at the south-western part of the Netherlands (Figure 5.1), containing 10 billion points. The range of the bounding box is [13427.6, 359007.3, -8.8; 38000, 415990.9, 119.7] in spatial reference system Amersfoort/RD New, EPSG:28992.

This section conducts benchmark tests using this sample. Subsection 5.1.1 describes state-of-the-art solutions implemented for comparison. Then, the data organization is established in Subsection 5.1.2. Subsection 5.1.3 tests the two algorithms for building HistogramTree (Section 4.2.2). After this, Subsection 5.1.4 executes orthogonal window queries, and analyzes results.

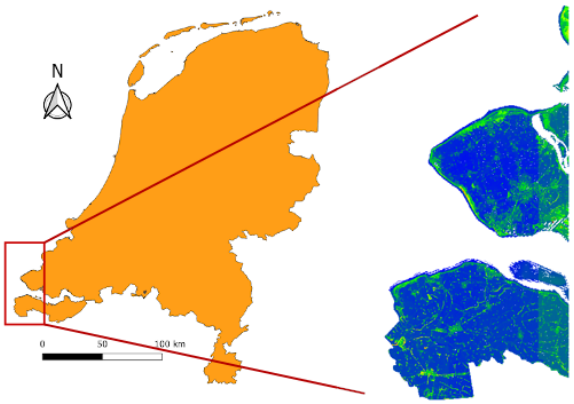
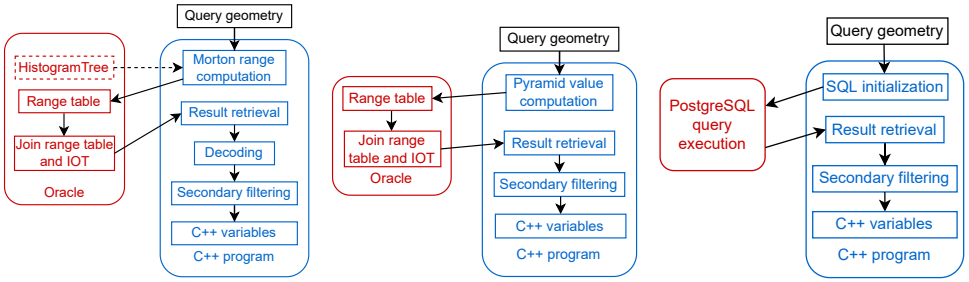


Figure 5.1: The AHN2 sample used for benchmarking

5

5.1.1. STATE-OF-THE-ART SOLUTIONS

In addition to PlainSFC and HistSFC, Pyramid-Technique, Extended Pyramid-Technique and PostGIS are implemented. Figure 5.2 presents the querying process of different solutions. The final result is stored as C++ in-memory objects, before being exported to disks. Listings 5.1 - 5.4 provide details about the implementation, i.e., the SQL Data Definition Language (DDL) and Data Manipulation Language (DML).



(a) PlainSFC and HistSFC (b) (Extended) Pyramid-Technique (c) PostGIS

Figure 5.2: Querying process of different solutions

PYRAMID-TECHNIQUE

In high dimensional spaces, the Pyramid-Technique (Berchtold et al., 1998) avoids excessive access to data pages by partitioning data into pieces which cater to the shape of hypercubes (Figure 5.3). In Figure 5.3, Pyramid-Technique splits the 2D domain into 4 pyramids of equal size, and assigns an ID from 0 to 3 to each of the pyramids. The approach then maps each nD point into a one-dimensional pyramid value according to the pyramid the point belongs to and its height in the pyramid. All the resultant one-dimensional keys are then managed by a B+-tree structure to be indexed. This approach can be generalized to nD where totally 2^n pyramids are defined. When querying (Fig-

Listing 5.1: PlainSFC implementation

```
-- suppose Morton codes and dimension values are stored in a CSV file,
    AHNsample.csv.

-- first create a staging table
CREATE TABLE AHNflat (morton number, x number, y number, z number, cloi
    number)

-- then, load data into AHNflat using SQL*Loader:
sqlldr control = control.ctl username/password direct=true

-- control.ctl defines the specifics of data loading, for example:
LOAD DATA
INFILE "/HOME/AHNsample.csv"
truncate
INTO TABLE AHNflat
FIELDS TERMINATED BY ','
TRAILING nullcols
(morton, x, y, z, cloi)

-- create the IOT which only stores the full-resolution Morton key
CREATE TABLE AHNiot (morton, constraint ahn_iot_idx primary key (morton))
    ORGANIZATION INDEX AS SELECT morton FROM AHNflat

-- when querying, first transform the query window into 1D Morton ranges and
     store them in a range table
CREATE TABLE ranges (lower number, upper number)

-- to perform a query in the first filter. This also works for HistSFC and
    Pyramid-Technique. /*+ use_nl*/ is a hint for query execution in Oracle,
     forcing the use of B+-tree index of IOT while iterating all ranges
SELECT /*+ use_nl (t r)*/ t.morton FROM AHNiot t, ranges r WHERE t.morton
    BETWEEN r.lower AND r.upper
```

Listing 5.2: HistSFC implementation

```
-- HistSFC uses the same IOT as PlainSFC, i.e, AHNiot. Additionally, HistSFC
     builds a flat table storing the HistogramTree.

CREATE TABLE AHNhist (id number, morton number, pointcount number,
    childcount number, height number, childptr number, neighborptr number)
```

ure 5.4), the vertices of the query window are used to derive which pyramids are affected. Then, inside each pyramid, the specific range of height values are determined. In this way, the query window is converted to ranges of pyramid values. The number of ranges can maximally equal the number of pyramids. I implemented the Pyramid-Technique based on the code described by Shi and Nickerson (2006).

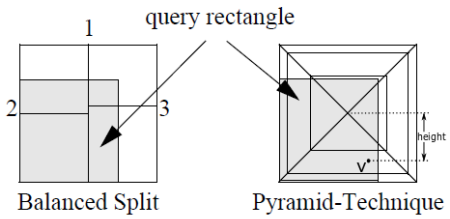


Figure 5.3: Data partition of Pyramid-Technique, adapted from (Berchtold et al., 1998)

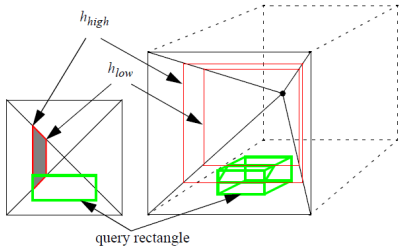


Figure 5.4: 3D window querying using Pyramid-Technique (Berchtold et al., 1998)

5

EXTENDED PYRAMID-TECHNIQUE

This solution optimizes the Pyramid-Technique to handle skewed data, where the partitioning center of pyramids is moved to the center of data (Figure 5.5). To achieve this, for each dimension, a histogram is maintained to keep track of the median of a dimension. The nD median is then approximated by the combination of the one-dimensional medians, to define the partitioning center, which is the top of all pyramids. After this, the pyramid value of each point is shifted to the new data space, and the index is rebuilt. Besides, the query window should also be converted in the new data space. As can be seen, the extended Pyramid-Technique is cumbersome to use: when adding more data, the whole data set has to be fully scanned to rebuild the storage.

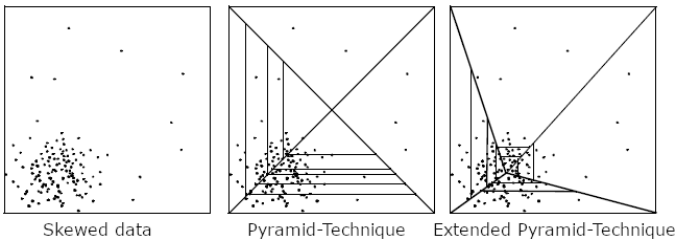


Figure 5.5: Partitioning skewed data using Pyramid-Technique and extended Pyramid-Technique (Berchtold et al., 1998)

POSTGIS

pgPointcloud can maximally support 2 organizing dimensions, which will perform inefficiently in 4D queries. Thus, I implemented a 4D solutions based on the 4D MultiPoint geometry. The original “M" dimension is replaced by cLoI. To keep in line with HistSFC,

Listing 5.3: Implementation of Pyramid-Technique and extended Pyramid-Technique

```
-- I first compute the pyramid values and the shifted pyramid values. They
    are stored in a CSV file, together with all dimension values.

-- create a staging table and load data into it
CREATE TABLE AHNpyramid (pv number, pvshift number, x number, y number, z
    number, cloi number)

-- create IOT based on Pyramid-Technique and extended Pyramid-Technique
CREATE TABLE pyramid_iot (pv, x, y, z, cloi, constraint pt_idx primary key (
    pt,cloi)) ORGANIZATION INDEX AS SELECT pv, x, y, z, cloi FROM AHNpyramid
    ;

CREATE TABLE expyramid_iot (pvshift, x, y, z, cloi, constraint ptex_idx
    primary key (pvshift,cloi)) ORGANIZATION INDEX AS SELECT pvshift, x, y,
    z, cloi FROM AHNpyramid;
```

Listing 5.4: PostGIS implementation

```
-- scan the original AHN file. For each point, attach the nodeid indicating
    the leaf node of the HistogramTree when building the AHNhist table.
    Export all dimension values and nodeid into a new CSV file as input, i.e
    ., pg_data.csv.

-- create a staging table in PostgreSQL, and load pg_data.csv
CREATE TABLE AHNpg (x double precision, y double precision, z double
    precision, cloi double precision, nodeid bigint);

COPY AHNpg FROM '/HOME/pg_data.csv' DELIMITERS ',' CSV

-- then, build the MULTIPOINT geometry table
CREATE TABLE AHNgeom AS SELECT st_collect(st_makepoint(x,y,z,cloi)) as
    mpoint FROM AHNpg GROUP BY nodeid

-- create a 4D R-tree index
CREATE INDEX ahn_mp_gix on AHNgeom USING GIST (mpoint gist_geometry_ops_nd);

-- cluster the storage according to the R-tree index
CLUSTER AHNgeom USING ahn_mp_gix;

-- when querying with a 4D window, e.g., [0, 0, 0, 0; 1, 1, 1, 1]. After
    this, the string is parsed and filtered by the second filter in the C++
    program
SELECT st_asewkt(mpoint) FROM AHNgeom WHERE mpoint &&& 'LINESTRING(0␣0␣0␣0,␣
    1␣1␣1␣1)';
```

an MultiPoint object which corresponds to a leaf node of HistogramTree is created to store point data. In fact, such an object can be regarded as a 4D block. Then, a 4D R-tree is built on all MultiPoint objects for indexing.

5.1.2. DATA ORGANIZATION

cLoI is an effective technique to alleviate intensive computation and can be used to facilitate visualization. So, I added a cLoI organizing dimension using the method proposed in Chapter 3, which ranges from 0 to 12,000 after stretching (Figure 5.6). In order to learn the scalability of different solutions, I split the data into 5 vertical slices from west to east. Starting from the first piece, by adding one more slice each time, 5 different data sets are built. Table 5.1 presents the storage size of different solutions. Raw TEXT refers to point records with 4 fields stored in TEXT files. Pyramid and PyramidEx refer to Pyramid-Technique and extended Pyramid-Technique, respectively. From the table, the size of SFC IOT is the smallest. This is mainly because during Morton encoding, all points are shifted to the origin and lots of bits are thus saved.

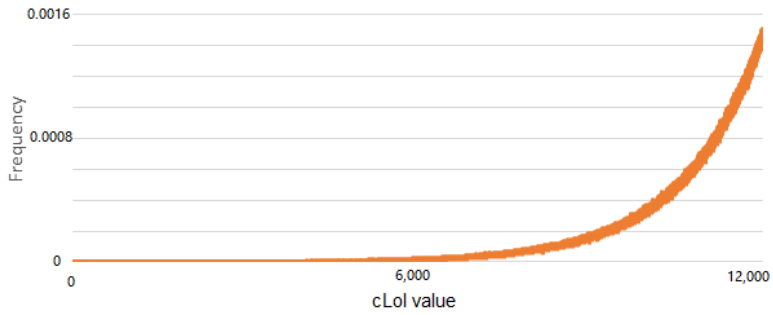


Figure 5.6: Distribution of the cLoI dimension

Table 5.1: Storage size of AHN2 data sets on the disk (in GB)

Data set	Number of points	Raw TEXT	SFC IOT	Pyramid	PyramidEx	PostGIS
1	5×10^8	16.49	10	18.24	18.52	7.21
2	10^9	32.98	19.95	36.39	36.95	14.17
3	2×10^9	64.42	38.97	71.06	72.01	28.1
4	6×10^9	193.9	118.3	213.6	216.2	82.32
5	10^{10}	323.4	199.7	356.9	360.7	138.0

5.1.3. BUILDING HISTOGRAMTREE

Before benchmarking different solutions, I first tested the two algorithms to build HistogramTree (Section 4.2.2). To briefly recap, HistML first locates a middle level, and then builds the HistogramTree in a bottom-up manner. On the other hand, HistSTREAM reads Morton keys in the IOT sequentially to build HistogramTree nodes and then the

whole structure. In this test, both approaches utilize 10,000 as the capacity threshold for the 4D HistogramTree, and export exactly the same HistogramTrees for the 5 data sets. The size of HistogramTrees ranges from 9.5 MB to 203 MB on the disk. Figure 5.7 presents the time cost and maximum memory consumption. The time measurement starts from reading Morton keys until exporting the HistogramTree to a flat table in Oracle.

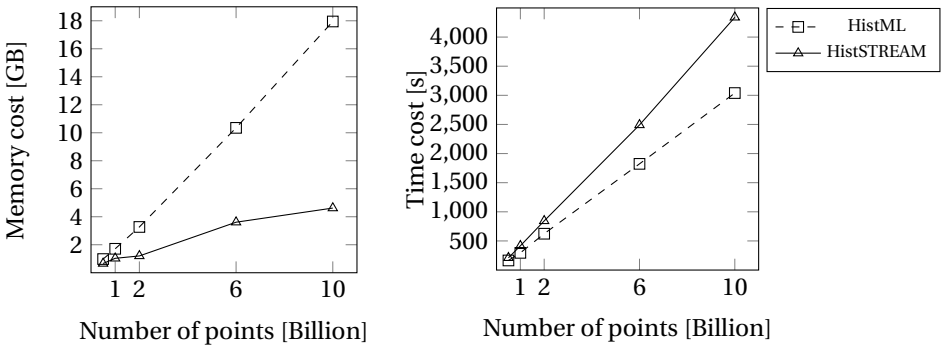


Figure 5.7: Time and memory cost to build HistogramTree for the AHN2 sample

Figure 5.7 clearly shows the superiority of HistSTREAM in memory cost: HistML increases linearly with input data size where it reaches 20 GB for the largest data set, while HistSTREAM gradually rises and converges to a constant in the end. The memory consumption of HistSTREAM depends on the longest consecutive data segment it processes. Given irregular data tiles shown in Figure 5.1, the longest consecutive data segment can increase when more irregular data tiles are added. This is the reason for the rise of HistSTREAM at certain stages. In terms of time cost, both approaches increase linearly as input goes up, but HistSTREAM grows faster. Currently, both HistSTREAM and HistML are implemented as independent C++ programs. HistSTREAM reads data from Oracle where intensive data conversions happen, e.g., from Oracle binary number to C++ objects. Consequently, the time cost can be larger than HistML which reads data directly from TEXT files. However, as is mentioned, HistML builds nodes from a middle level which has to be decided first. In this test, I first generated the HistogramTree using HistSTREAM. Based on it, I then determined the middle level to run HistML. So, practically, HistSTREAM is preferable.

5.1.4. BENCHMARKING WITH WINDOW QUERIES

To obtain a convincing benchmark result, random yet realistic orthogonal query windows should be used. I devised the query window size and location by considering query logs and testing requirements (e.g., diverse selectiveness of queries defined by Equation 5.1).

$$selectiveness = \frac{\text{Number of points within the query range}}{\text{Total number of points}} \tag{5.1}$$

I first used AHN2 query logs from AHN2-viewer[1] to explore query window locations in XY domain. As the AHN2-viewer is based on the Octree data structure, the query log records which Octree blocks are retrieved. I extracted Minimum Bounding Boxes (MBBs) of blocks in level 8, as a block in this level corresponds to the size of a normal city in the Netherlands. So, the result should reasonably reflect the distribution of query windows. Figure 5.8 plots the locations of these query boxes. It clearly shows that users are interested in both urban and rural areas. Besides, coastal areas are also frequently explored by different users, apart from inland. Hence, In the benchmark test, the query locations should be randomly distributed in the XY domain. The locations in Z dimension are not involved in the query log which focuses on 2D spatial queries. So, I randomly generated the Z range with devised selectiveness, mostly close to the earth surface. cLoI is related to the Octree levels. However, as the test data is a sample from the whole AHN2 data, the Octree levels extracted from the log cannot be directly used. Thus, I devised the cLoI range by choosing different selectiveness.

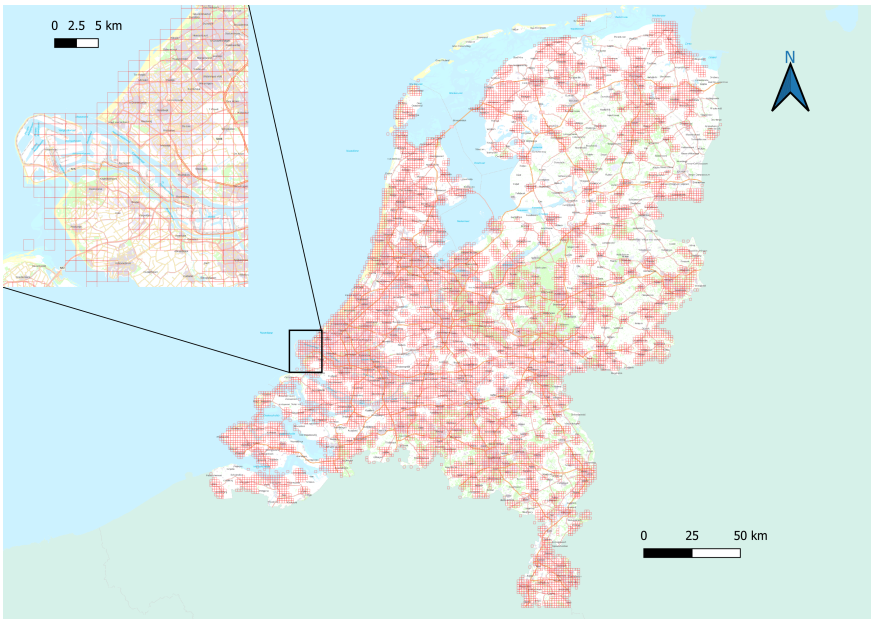
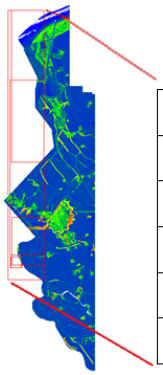


Figure 5.8: AHN2-viewer query boxes in red

Based on the consideration above, I randomly generated 30 query windows of which I picked up 5 representatives to interpret. These query windows are within the range of Data set 1 (Figure 5.9). Table 5.2 presents the selectiveness of these query windows. With this, I first investigated how HistSFC improves the performance of PlainSFC. I tested different size of HistogramTree, as well as different number of ranges for querying, to learn how they influence the efficiency. Then, I evaluated the performance of HistSFC by comparing it with other solutions.

[1] http://ahn2.pointclouds.nl/



Query window	4D MBR	Size (X × Y × Z × cLoD)	Number of points
SmallA	[13691.53, 368858.59, -0.97, 0.00; 14124.7, 369291.76, 61.74, 11957.71]	433.17 × 433.17 × 62.71 × 11957.71	230,739
SmallB	[13582.83, 368380.82, 8.04, 0.00; 15000.00, 379000.00, 10.24, 10954.84]	1417.17 × 10619.18 × 2.2 × 10954.84	275,910
SmallC	[13742.03, 368961.59, -2.86, 0.00; 15000.00, 370843.63, 13.87, 8700.00]	1257.97 × 1882.04 × 16.73 × 8700	239,818
Medium	[13722.4, 373013.86, -0.97, 0.00; 15000.00, 379000.00, 14.13, 10422.78]	1277.6 × 5986.14 × 15.1 × 10422.78	3,374,880
Large	[13665.62, 369354.04, -2.86, 0.00; 15000.00, 376247.77, 10.56, 11339.54]	1334.38 × 6893.73 × 13.42 × 11339.54	22,663,207

Figure 5.9: Locations of representative query windows

Table 5.2: Selectiveness of different dimensions of the query windows, with respect to Data set 1

	X	Y	Z	cLoI	Overall
SmallA	1.73%	4.37%	99.46%	94.31%	0.05%
SmallB	20.5%	79.11%	1.01%	23.48%	0.05%
SmallC	20.25%	17.58%	98.8%	1.03%	0.05%
Medium	20.29%	35.02%	98.29%	11.23%	0.67%
Large	20.39%	55.3%	98%	40.03%	4.53%

Table 5.3 and 5.4 present the size of HistogramTrees with different capacity threshold. To verify the reduction of vacant ranges after using HistogramTree, the Empty Ratio of ranges (ER) is measured, recalling that $ER = \frac{\text{Number of vacant ranges exported}}{\text{Total number of ranges exported}}$ (Section 4.4). Table 5.5 presents ER of different solutions, where HistSFC_1K refers to HistogramTree with a capacity threshold of 1,000 (i.e., leaf node capacity). The total number of ranges for querying is 1 million for all solutions. Table 5.5 clearly shows that more than 95% ranges generated by PlainSFC are empty, while the ratio decreases when using HistSFC, especially HistSFC_1K. In the Large window query, HistSFC_1K halves the number of vacant ranges compared to PlainSFC. However, ER decreases less significantly when using HistSFC_10K or HistSFC_100K (around 10% decrease on average).

Table 5.3: Total number of nodes in HistogramTree

	Capacity threshold		
Data set	1,000	10,000	100,000
1	2,689,357	231,342	23,675
2	5,297,346	463,259	47,510
3	10,610,241	936,491	95,351
4	28,158,323	2,483,944	237,396
5	52,292,745	4,836,880	482,827

Table 5.4: Storage size of HistogramTree (MB)

	Capacity threshold		
Data set	1,000	10,000	100,000
1	109.5	9.5	1.5
2	215	19	2
3	430	38	4
4	1159	101.5	10
5	2140	196.5	20

Table 5.5: Empty ratio of ranges using different HistogramTrees in AHN2 test, with 1 million ranges used

Query window	HistSFC_1K	HistSFC_10K	HistSFC_100K	PlainSFC
SmallA	94.66%	95.39%	98.32%	99.35%
SmallB	78.27%	82.35%	89.07%	98.04%
SmallC	85.35%	89.11%	91.32%	95.19%
Medium	65%	85.94%	92.03%	95.83%
Large	42.28%	80.40%	80.05%	95.73%

On the other hand, ER cannot directly indicate the querying performance, while False Postive Rate (FPR) can: $FPR = \frac{\text{Number of points from first filter}}{\text{Number of points in accurate answer}} - 1$ (Section 2.6.3). Table 5.6 presents FPR of different solutions. Over all, using HistogramTree significantly decreases FPR of PlainSFC, and HistSFC_1K performs the best. Besides, HistSFC_10K and HistSFC_100K also improve the performance evidently.

Table 5.6: False positive rate using different HistogramTrees in AHN2 test, with 1 million ranges used

Query window	HistSFC_1K	HistSFC_10K	HistSFC_100K	PlainSFC
SmallA	0.084	0.121	0.248	1.031
SmallB	0.778	1.143	1.257	2.339
SmallC	0.022	0.052	0.059	0.068
Medium	0.079	0.1	0.124	0.285
Large	0.028	0.033	0.063	0.136

Figures 5.10 – 5.14 show the querying time cost of different solutions. The time cost corresponds to T in Equation 2.1, and is composed by the first filter time (T_{pre}) and second filter time ($T_{io} + T_{post}$). Appendix A provides exact time measurements. On the whole, HistSFC_10K and HistSFC_100K perform the best. PlainSFC follows behind, while HistSFC_1K ranks last. In SmallA, SmallC and Medium query, PlainSFC takes nearly the same time or slightly more, compared to HistSFC_10K and HistSFC_100K. The main reason lies in the high uniformity of the data. For example, CHC of Data set 1 is 0.01. Hence, the effectiveness of HistogramTree is limited, which causes insignificant improvement in FPR. For SmallB query, the ranges generated by either PlainSFC or HistSFC spread over the 1D SFC space, which significantly increases database fetching time of the first filter. With higher FPR, PlainSFC spends more time on database fetching. The Large query involves large output which needs more time to decode. Hence, even with small FPR, PlainSFC still takes significantly more time to finish. In contrast to the favourable performance in empty ratio and FPR (Tables 5.5 and 5.6), HistSFC_1K degrades remarkably in time cost as data size increases. This is mainly caused by traversing the huge HistogramTree which takes enormous amount of time. Besides, HistogramTree has first to be loaded into memory to use. The loading process of HistSFC_1K of Data set 5 can take as long as 160 seconds, which is unacceptable. With current hardware and software settings, the size of HistogramTree should be limited to less than 1 GB.

With respect to HistSFC_10K and HistSFC_100K, the gap between them is small. However, the size of HistogramTree of HistSFC_100K is much smaller (Table 5.4). If we con-

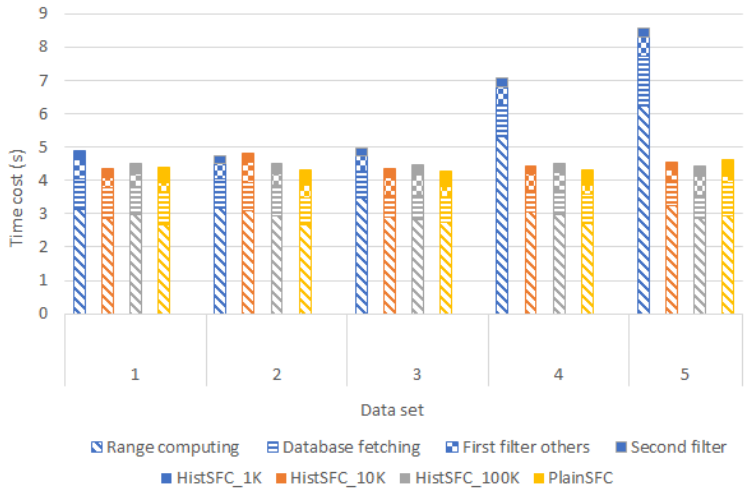


Figure 5.10: Time cost of SmallA query using different HistogramTrees with 1 million ranges

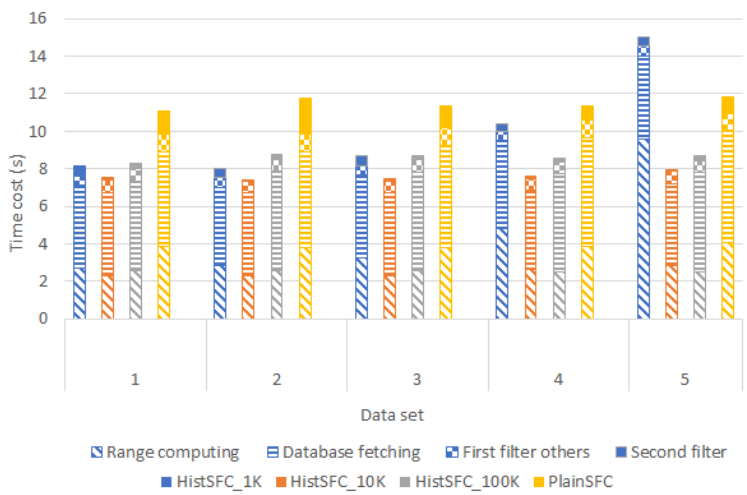


Figure 5.11: Time cost of SmallB query using different HistogramTrees

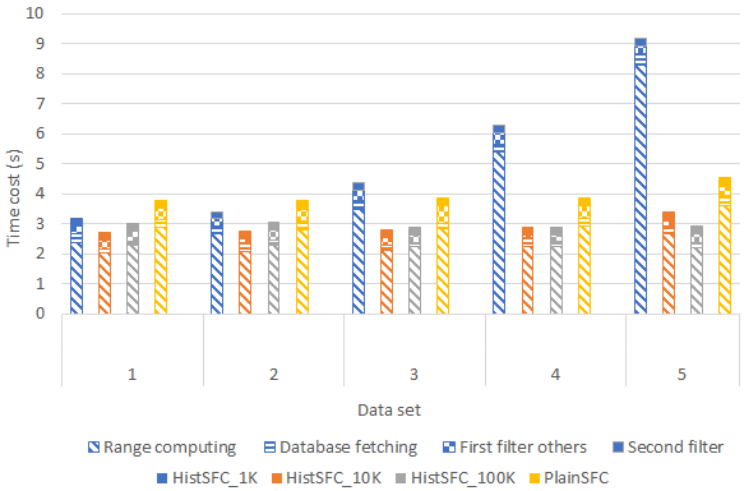


Figure 5.12: Time cost of SmallC query using different HistogramTrees with 1 million ranges

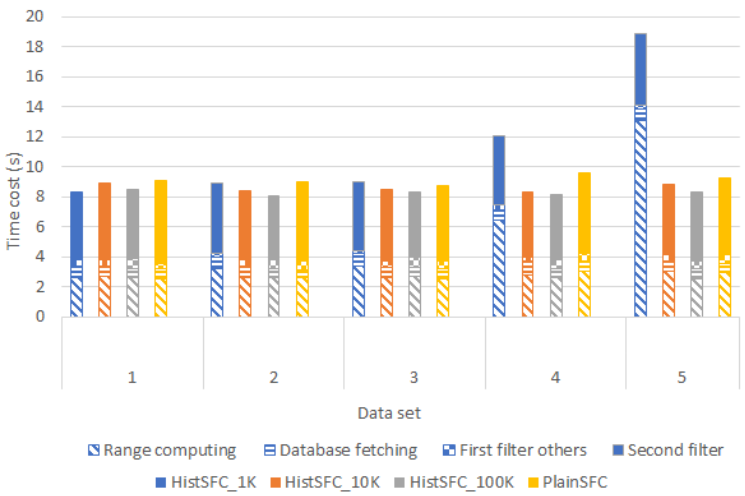


Figure 5.13: Time cost of Medium query using different HistogramTrees with 1 million ranges

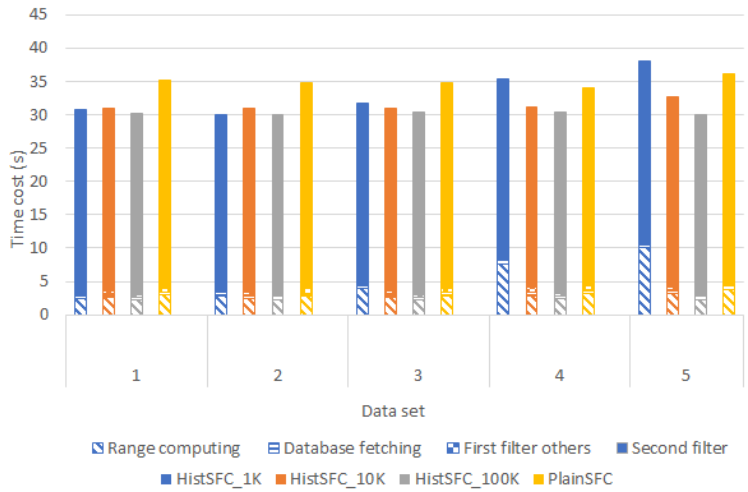


Figure 5.14: Time cost of Large query using different HistogramTrees with 1 million ranges

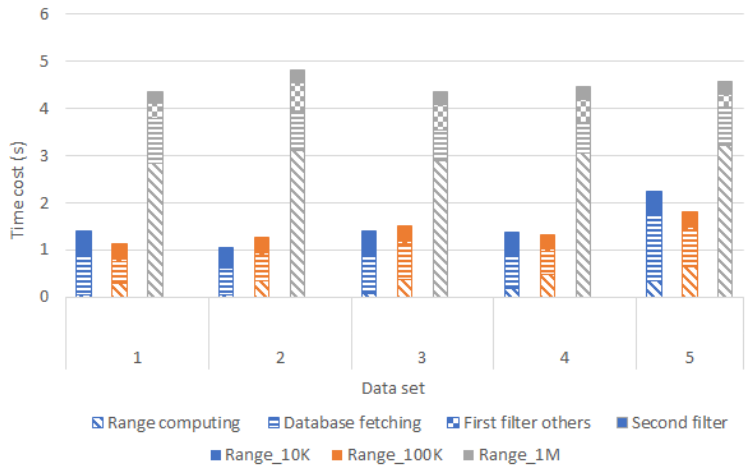


Figure 5.15: Time cost of SmallA query with different number of ranges, using HistSFC_10K

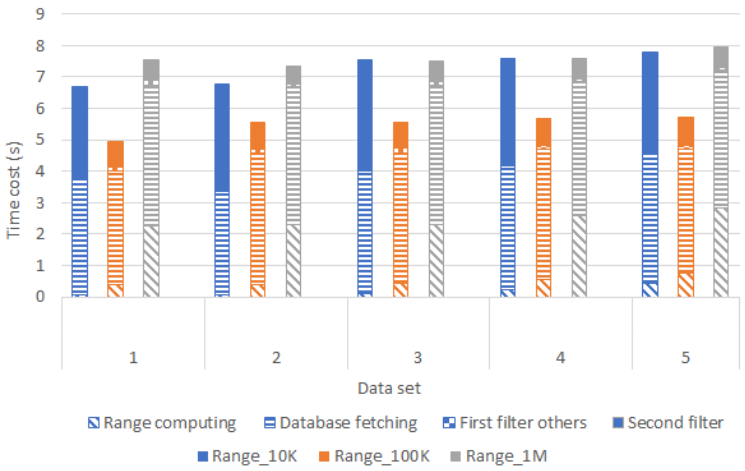


Figure 5.16: Time cost of SmallB query using different number of ranges, using HistSFC_10K

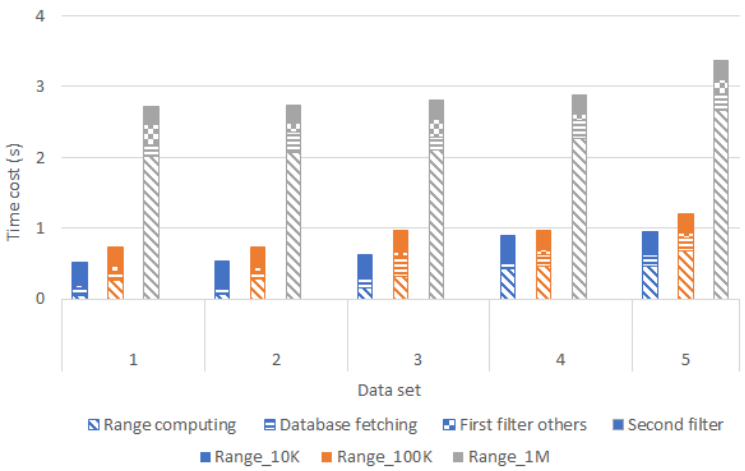


Figure 5.17: Time cost of SmallC query using different number of ranges, using HistSFC_10K

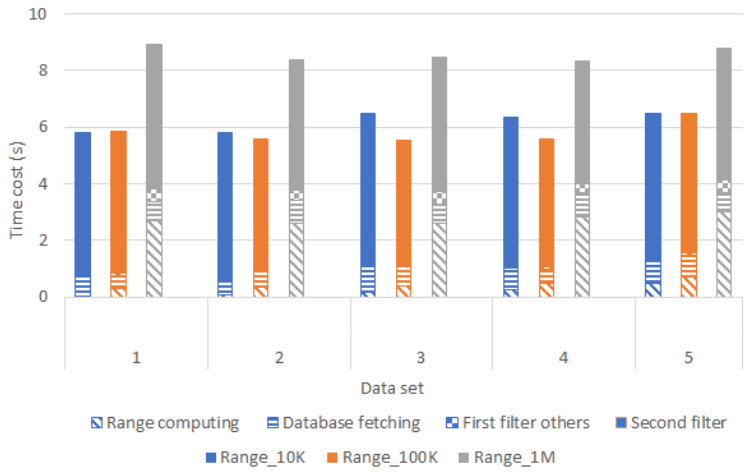


Figure 5.18: Time cost of Medium query using different number of ranges, using HistSFC_10K

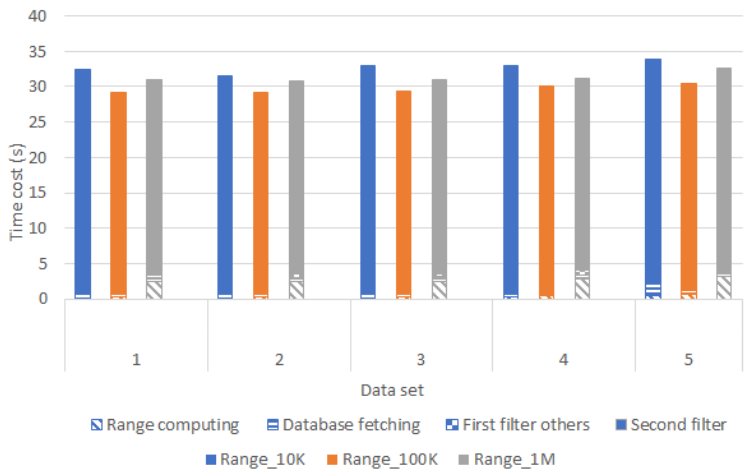


Figure 5.19: Time cost of Large query using different number of ranges, using HistSFC_10K

sider the whole AHN2 data set which is 64× larger than Data set 5, HistogramTree of HistSFC_100K can occupy several GB in the memory. So, larger capacity threshold may be used then. As a stress test, I chose HistSFC_10K for further benchmarking.

Figures 5.15 – 5.19 present the querying performance of HistSFC_10K with different number of ranges, where Range_10K means total number of ranges generated for querying is 10,000. Table 5.7 presents FPRs of different solutions. On the whole, Range_100K performs the best, balancing the time cost of the first filter and second filter. Although Range_10K performs better in certain cases, when it comes to larger query windows, the higher FPR slows down its second filter significantly.

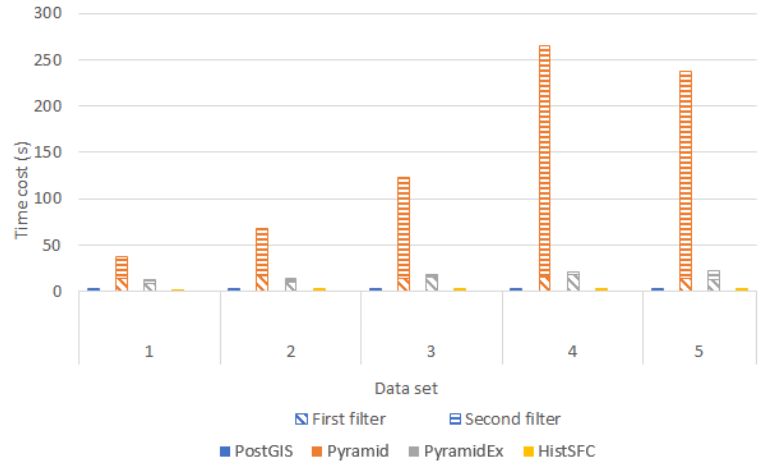
Table 5.7: False positive rate using different number of ranges for querying

Query window	Range_10K	Range_100K	Range_1M
SmallA	0.848	0.33	0.121
SmallB	6.359	1.92	1.143
SmallC	0.297	0.156	0.052
Medium	0.318	0.178	0.1
Large	0.158	0.082	0.033

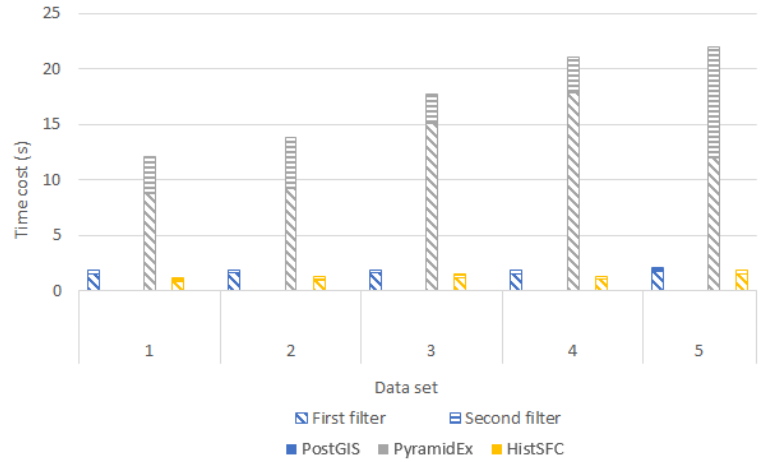
After tuning the settings of HistSFC, I compared it with other state-of-the-art solutions including Pyramid-Technique, extended Pyramid-Technique and PostGIS (Section 5.1.1). HistSFC_10K was used. The number of ranges for querying is 100,000. For PostGIS, the capacity of MultiPoint object is 10,000. Figures 5.20 – 5.24 present time cost of different solutions. As has been tested, if the output size is very large, the second filter can cost huge amount of time to accomplish. So, for all solutions, I implemented parallel post-processing with 32 processors. To illustrate with Figure 5.2, this means parallel decoding and secondary filtering for HistSFC, parallel secondary filtering for (extended) Pyramid-technique and PostGIS. The parallelization distributes workload equally among different processors to achieve the best performance (Section 5.3.3).

From Figures 5.20 – 5.24, HistSFC always takes the least time to execute. PostGIS ranks behind, while Pyramid and PyramidEx are the slowest solutions. More specifically, in SmallA and SmallC, PostGIS spends slightly more time than HistSFC. However, SmallB is different, where HistSFC outperforms PostGIS significantly. This is because the specific shape of SmallB results in more blocks of the PostGIS solution intersecting the query window. Thus, PostGIS spends much more time on unpacking blocks. For the same reason, HistSFC outperforms PostGIS in the Medium and Large query. This, however, does not happen for SmallA and SmallC, where only a small number of blocks intersects the query windows. Nonetheless, the R-tree index functions very efficiently to retrieve blocks for all queries. Both HistSFC and PostGIS scale constantly with respect to the input data size, over all.

Pyramid and PyramidEx spend much more time on querying, and the performance fluctuates significantly. This is mainly attributed to the large and changing FPR (Table 5.8). Due to the specific pyramid decomposition of the space, when the query window reaches the bottom of a pyramid, all points residing in the bottom level of the pyramid will be selected (Figure 5.5). This brings large number of false positive points, increasing FPR. On the other hand, the FPR may also decrease when input data size in-



(a) Overall



(b) Omitting Pyramid

Figure 5.20: Time cost of SmallA query using state-of-the-art solutions

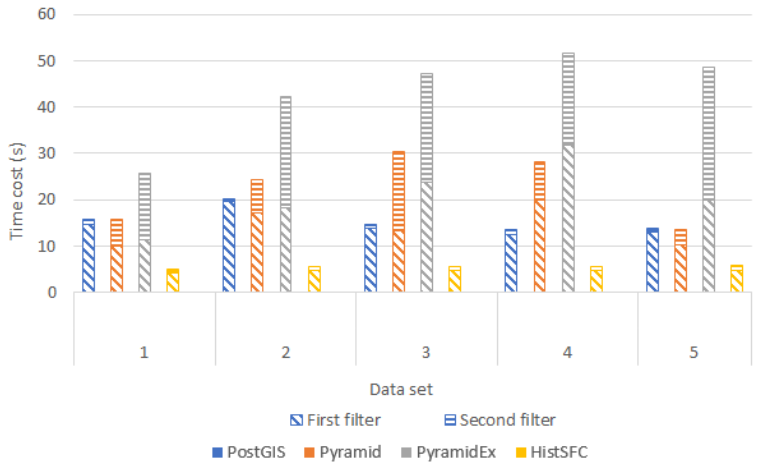


Figure 5.21: Time cost of SmallB query using state-of-the-art solutions

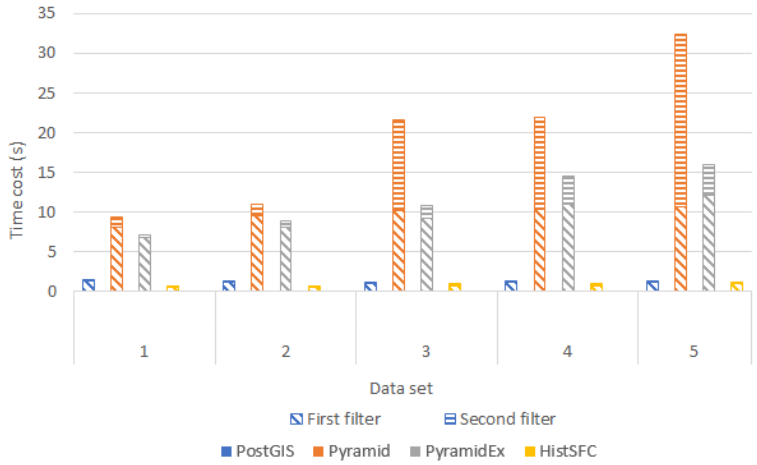


Figure 5.22: Time cost of SmallC query using state-of-the-art solutions

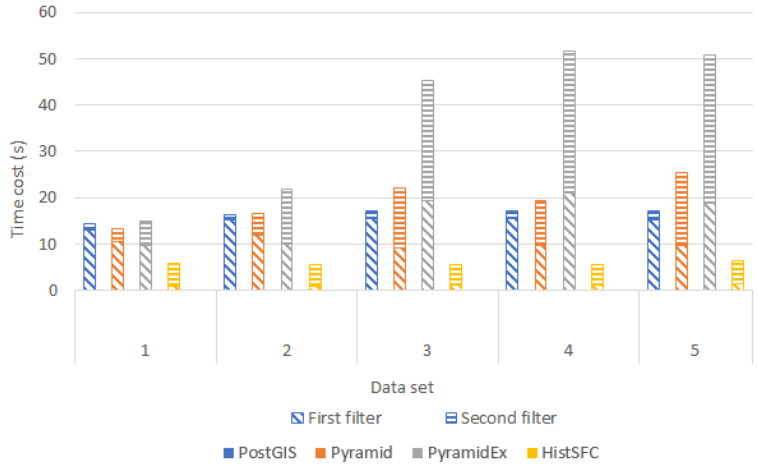


Figure 5.23: Time cost of Medium query using state-of-the-art solutions

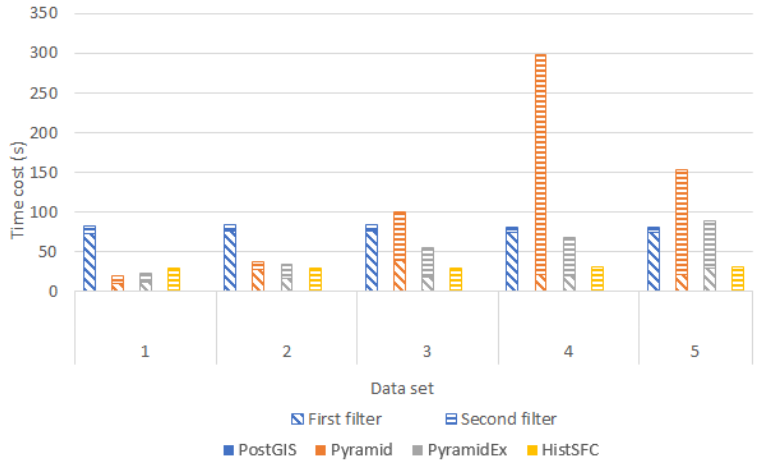


Figure 5.24: Time cost of Large query using state-of-the-art solutions

creases. This is because for each data set, the boundary of data and the medians for computing the pyramid value are changed. So, for the same query window, either Pyramid or PyramidEx may select different portions of data, which then influences FPR. Besides, the fluctuation of these two solutions' performance is also caused by the unstable parallelization. The amount of points to be decoded is so large that it is difficult to guarantee a stable performance of all processors. Another odd pattern occurs in SmallB that Pyramid performs faster than PyramidEx. This happens sometimes, as PyramidEx can return higher FPR for certain query windows depending on their positions. For the large selection, PyramidEx runs even faster than PostGIS which spends large amount of time on unpacking blocks. However, PostGIS performs more stable, and thus possesses a better scalability than PyramidEx.

Table 5.8: False positive rate of state-of-the-art solutions

	SmallA	SmallB	SmallC	Medium	Large
PostGIS	1.57	7.14	0.66	0.36	0.16
	1.57	7.15	0.66	0.36	0.16
	1.57	7.15	0.66	0.36	0.16
	1.57	7.15	0.66	0.36	0.16
	1.57	7.15	0.66	0.36	0.16
Pyramid	1739.32	336.69	32.85	14.77	5.73
	3328.40	677.63	84.20	29.58	13.34
	5739.03	862.65	708.46	54.45	30.07
	11345.91	612.88	585.33	33.65	50.16
	13568.89	170.72	1405.94	67.47	35.60
PyramidEx	255.80	689.05	20.43	23.87	6.40
	338.27	1148.89	41.91	44.66	9.36
	408.23	1083.39	84.84	86.67	12.12
	465.24	1266.48	263.35	101.90	14.34
	757.63	1733.64	432.85	139.94	20.01
HistSFC	0.33	1.92	0.16	0.18	0.08
	0.33	1.92	0.16	0.18	0.08
	0.33	1.92	0.16	0.18	0.08
	0.33	1.92	0.16	0.18	0.08
	0.33	1.92	0.16	0.18	0.08

Besides the time cost, it should also be noted that the final output of Pyramid and PyramidEx may be inaccurate, because the pyramid value is not so precise to avoid errors at the query boundaries.

To conclude, HistSFC is the most favourable solution for the 4D window query. PostGIS performs very efficiently in retrieving blocks (i.e., MultiPoint objects), and thus functions efficiently for queries with small outputs. However, with more blocks selected, PostGIS spends significantly more time on unpacking them. This CPU-intensive process is a drawback for all block based approaches. HistSFC organizes each point as a record, which naturally avoids the unpacking problem. Pyramid and PyramidEx are originally devised for very high dimensional hypercubic window queries. So, they perform less efficiently in this experiment, as they return very large FPRs.

5.2. FLOOD DATA TEST

As is mentioned in Section 1.2, flood modelling results can be converted to point clouds for further analysis. This section focuses on a research area, Niansi Levee, located at Jiangxi province, China (Figure 5.25). To its northeast is the Poyang Lake. The total area is about 183 km^2. Niansi Levee is one of the key areas that are modelled in the national flood mapping project of China.

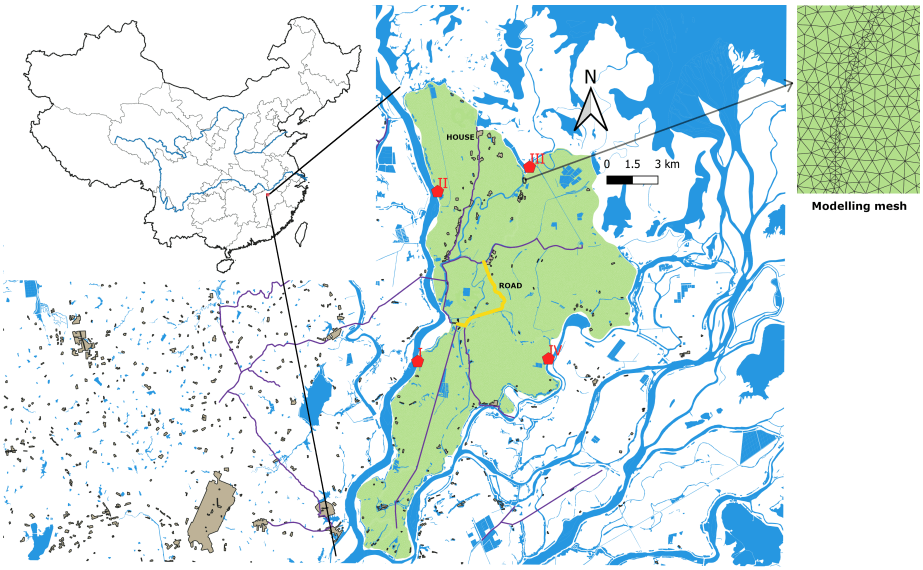


Figure 5.25: Niansi Levee (green zone), with locations of simulated breach in red. A small piece of mesh used for modelling is also shown on the right

The hydrodynamic model consists of a 1D channel model, and a coupled 2D flood routing model in the basin. The channel model provides extreme flow for the flood simulation in the 2D mesh which totally contains 59,680 triangular cells (Figure 5.25). The model computes water depth, velocity and flow direction. The project modelled 8 cases: 4 locations of breach, combined with extreme rainfall with a return period of 20 and 50 years. Each case simulates 720 steps (corresponding to a 30-min resolution). So in total, I get 59,680 × 720 × 8 = 343,756,800 points, in an 8D space composed by case ID, X, Y, Z, time, depth, velocity and direction.

5.2.1. SOLUTIONS FOR TESTING

HistSFC, PlainSFC and Oralce SDO_PC solutions were tested. In fact, the Pyramid and PyramidEx were also tested, but their performance was much worse than the others. So, they were excluded. PostGIS solution can maximally support 4D data organization, while the flood data involves 8 dimensions. Hence, I chose SDO_PC which is in the same Oracle environment as PlainSFC and HistSFC.

According to practical experience of flood maps, the flow direction is seldomly used for ad-hoc analysis, compared with other dimensions. So, I set it as the property dimen-

sion when building PlainSFC and HistSFC, while encode the other 7 dimensions into the Morton key. The scripts to build HistSFC and PlainSFC are similar to AHN2 (Listings 5.1 and 5.2).

The SDO_PC solution uses X and Y dimension to organize data and create blocks. The other dimensions are regarded as the property dimension to be attached. The block capacity is 5,000. Figure 5.26 shows the query execution process. Listing 5.5 provides scripts for building and querying of SDO_PC.

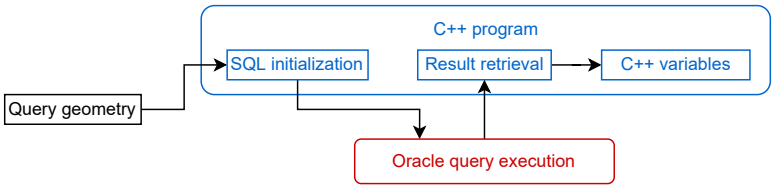


Figure 5.26: Querying process of SDO_PC

5

5.2.2. DATA ORGANIZATION

The project modeled 8 cases, and each case produces a result set. To explore the scalability, I divided the whole result set into 4 benchmark data sets according to the case ID. Data set 1 consists of the result of case 1. Data set 2 consists of case 1 and 2. Data set 3 includes the first 4 cases. Data set 4 refers to the whole data set. Table 5.9 lists the storage size of different solutions. The large size of SDO_PC is mainly due to additional information stored. For example, each point gets an additional point ID and block ID, while each block records extra metadata such as number of points, resolution and so on (Oracle, 2019).

Table 5.9: Storage size of flood data sets on the disk (in GB)

Data set	Number of points	Raw TEXT	SFC IOT	SDO_PC
1	42,969,600	2.69	1.67	3.63
2	85,939,200	5.41	3.34	7.26
3	171,878,400	10.8	6.67	14.1
4	343,756,800	20.7	12.9	33.3

5.2.3. QUERIES FOR RISK ANALYSIS

According to the experience acquired during the project and potential needs, I devised 5 queries for testing (Table 5.10). As these queries all concern case 1, the execution using different data sets will return the same results. Locations of HOUSErisk and ROADrisk are depicted in Figure 5.25.

The result of DEPTH3m corresponds to a typical flood depth map. ARRIVAL24h generates a typical arrival time map, while EXTENTmax results in a maximal inundation map. They are conventional products of flood risk mapping, while HOUSErisk and ROADrisk are new, providing more insights. HOUSErisk selects all points around 4 houses

Listing 5.5: SDO_PC implementation

```
-- suppose all dimension values are stored in a CSV file, nsdata.csv
-- first create a staging table, where VAL_D1 refers to X dimension and
    VAL_D2 refers to Y dimension
CREATE TABLE ns_sdo_flat (RID varchar2(24), VAL_D1 number, VAL_D2 number,
    VAL_D3 number, VAL_D4 number, VAL_D5 number, VAL_D6 number, VAL_D7
    number, VAL_D8 number)

-- then, load data into ns_sdo_flat using SQL*Loader:
sqlldr control = control.ctl username/password direct=true

-- control.ctl defines the specifics of data loading, for example:
LOAD DATA INFILE "HOME/nsdata.csv" truncate INTO TABLE ns_sdo_flat FIELDS
    TERMINATED BY ',' TRAILING nullcols (RID sequence (MAX,1), VAL_D1,
    VAL_D2, VAL_D3, VAL_D4, VAL_D5, VAL_D6, VAL_D7, VAL_D8)

-- create base table of SDO_PC
CREATE TABLE ns_base (ID number, PC sdo_pc)

-- create block table of SDO_PC
CREATE TABLE ns_block AS SELECT * FROM MDSYS.SDO_PC_BLK_TABLE WHERE 0 = 1

-- populating data from the staging table
declare
  pc sdo_pc;
begin
  pc := sdo_pc_pkg.init('ns_base', 'PC', 'ns_block', 'blk_capacity=5000',
      sdo_geometry(2003, 4527, null, sdo_elem_info_array(1,1003,3),
      sdo_ordinate_array(Xmin, Ymin, Xmax, Ymax)), 0.0001, 8, null, null,
      null);
  sdo_pc_pkg.create_pc(pc, 'ns_sdo_flat', null);
  insert into ns_base values (1,pc);
  commit;
end;

-- register in the metadata table
INSERT INTO USER_SDO_GEOM_METADATA VALUES ('ns_block','BLK_EXTENT',
    sdo_dim_array(sdo_dim_element('X', [Xmin], [Xmax], 0.0001),
sdo_dim_element('Y', [Ymin], [Ymax], 0.0001)), 4527);

-- create R-tree index on blocks
CREATE INDEX ns_sdo_idx ON ns_block (BLK_EXTENT) INDEXTYPE IS MDSYS.
    SPATIAL_INDEX

-- when querying with an 8D window, e.g., [0, 0, 0, 0, 0, 0, 0, 0; 1, 1, 1,
    1, 1, 1, 1, 1]
SELECT x, y, z, w, v5, v6, v7, v8 from table(sdo_pc_pkg.clip_pc((SELECT pc
    FROM ns_base WHERE ID = 1), sdo_geometry(2003,4527, NULL,
    sdo_elem_info_array(1,1003,3), sdo_ordinate_array(0, 0, 1, 1)),sdo_mbr(
    sdo_vpoint_type(0,0,0,0,0,0), sdo_vpoint_type(1,1,1,1,1,1)),null,null))
    query_blocks, table(sdo_util.getvertices(sdo_pc_pkg.to_geometry(
    query_blocks.points, query_blocks.num_points,8,4527))) query_points
```

Table 5.10: Queries used for benchmarking

	Description	Organizing dimensions involved	Number of points
DEPTH3m	Select the area that is flooded with depth greater than 3 m, in case 1	caseID, depth	26,484,215
ARRIVAL24h	Select the area that is flooded (depth > 0) in 24 hours, in case 1	caseID, depth, time	925,691
EXTENTmax	Select the maximum inundation area (depth > 0), in case 1	caseID, depth	32,183,314
HOUSErisk	Select the area that is flooded (depth > 0) around several houses (a rectangle area), in case 1	caseID, depth, X, Y	170,417
ROADrisk	Select the dangerous points along a country road (velocity ≥ 0.5), in case 1	caseID, velocity, X, Y	83

that have been flooded (Figure 5.27). The rectangular area is about 1.5 km^2. ROADrisk uses an irregular geometry (i.e., the road) for querying. The result of ROADrisk is 83 8D points, presented as 6 distinct spatial points at different time steps (Figure 5.28). These points indicate the vulnerable parts of the road which need enhancement. During flood evacuation, people should also avoid these locations.

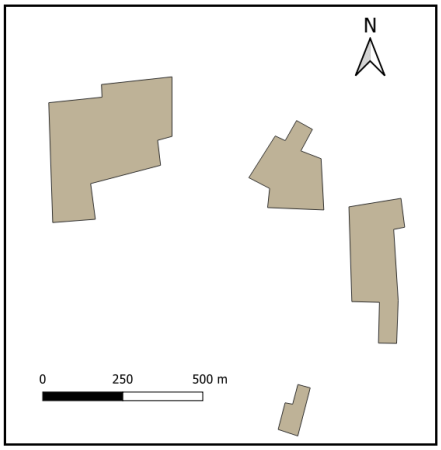


Figure 5.27: Range of HOUSErisk, covering 4 houses

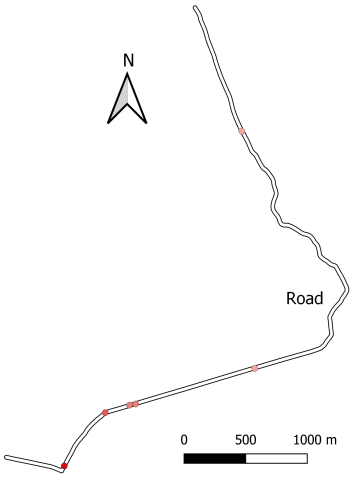


Figure 5.28: Result of ROADrisk: the red points on the road. The color becomes stronger when the maximum velocity is larger

5.2.4. RESULTS AND ANALYSIS

Similar to the AHN2 test, I first explored the influence of the size of HistogramTree. Then, different number of ranges were tested to derive the optimal setting. In the end, SDO_PC

was tested and compared with HistSFC. For all tests, a single thread was used.

Table 5.11 and Table 5.12 present the size of HistogramTrees with different capacity threshold. As the data set is smaller than AHN2, smaller node capacities are tested. As the tables show, HistogramTrees' sizes are much smaller here than in the AHN2 experiment. They are all below 100 MB.

Table 5.11: Total number of nodes in HistogramTree

	Capacity threshold		
Data set	1,000	5,000	10,000
1	276,577	53,900	34,795
2	547,166	107,562	72,400
3	1,094,669	215,380	142,536
4	2,100,498	410,444	271,052

Table 5.12: Storage size of HistogramTree (MB)

	Capacity threshold		
Data set	1,000	5,000	10,000
1	11.5	2.5	2
2	22.5	4.5	3.5
3	45	9	7
4	86	17	11

Table 5.13 presents the empty ratio (Equation 4.9) of different solutions for the querying. The improvement of using HistSFC is much more remarkable than the AHN2 test, e.g., the empty ratio can decrease from 99.72% to 0 in EXTENTmax. However, in ROADrisk, even HistSFC_1K's empty ratio is exceptionally large. This is most likely due to the long and narrow shape of ROADrisk of which the area is very small, containing a few points which have to be allocated to a large number of ranges.

Table 5.13: Empty ratio of ranges using different HistogramTrees on flood Data set 4

Query window	HistSFC_1K	HistSFC_5K	HistSFC_10K	PlainSFC
DEPTH3m	0	5.61%	15.46%	99.49%
ARRIVAL24h	8.13%	23.68%	27.09%	98.69%
EXTENTmax	0	0.15%	0.27%	99.72%
HOUSErisk	15.41%	65%	67.51%	97.72%
ROADrisk	98.51%	98.61%	98.09%	99.72%

I still use FPR and time cost to evaluate the performance of different solutions. Table 5.14 presents FPRs for queries on Data set 4. Other data sets slightly vary for each solution, except PlainSFC. This is mainly due to the large size of each range generated by PlainSFC: on average, the resolution of caseID of each range can span from 1 to 4. So, with more data of different caseID added, the same set of ranges incorporate increasing false positive points. This causes the dramatic increase of FPR.

Table 5.14: False positive rate using different HistogramTrees on flood Data set 4

Query window	HistSFC_1K	HistSFC_5K	HistSFC_10K	PlainSFC
DEPTH3m	0.048	0.084	0.106	4.739
ARRIVAL24h	4.958	5.571	5.884	25.433
EXTENTmax	0.339	0.357	0.39	4.327
HOUSErisk	1.922	1.876	2.048	13.457
ROADrisk	745.1	4,565.2	8,379.2	373,651.0

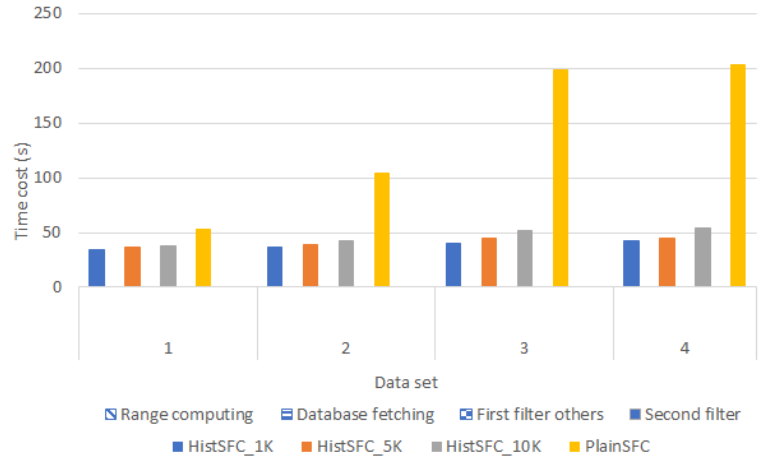
Figures 5.29 – 5.33 show the querying time cost using different HistogramTrees. Appendix B provides exact time measurements. The number of ranges used for querying is 100,000. As the figures show, in DEPTH3m, ARRIVAL24h and EXTENTmax, HistSFC_1K takes the least time to accomplish the queries, while HistSFC_5K and HistSFC_10K follow after. This is due to higher FPRs (Table 5.14) of the latter two solutions, which causes the increase of I/O and decoding time cost. In HOUSErisk and ROADrisk, HistSFC_5K outperforms HistSFC_1K. HistSFC_5K's high efficiency in HOUSErisk is due to the smaller FPR compared to HistSFC_1K. In ROADrisk, HistSFC_1K spends much time to traverse the large HistogramTree, which slows down the whole process. For all queries, PlainSFC significantly lags behind, due to large FPR. Generally speaking, Compared to the AHN2 test, the improvement by using HistogramTree is more evident in this test. This is majorly caused by the small CHC of the flood data which equals 0.0007. HistogramTree works more effectively for such non-uniformly distributed data. In the following, capacity threshold of 5,000 is adopted as it balances all type of queries.

The number of ranges for querying also influences the performance. Figures 5.34 – 5.38 present the querying performance with different number of ranges. Table 5.15 presents corresponding FPRs for queries on Data set 4. The capacity threshold of HistogramTree is 5,000. From DEPTH3m to EXTENTmax, the three solutions are close to each other, with subtle difference. In HOUSErisk and ROADrisk, Range_500K becomes significantly slower than the other two. HOUSErisk concerns a small query window, the I/O and decoding cost of different solutions do not differ much. However, as Range_500K spends more time on range computation, it takes longer to finish the query. In ROADrisk, the gap between Range_500K and the other two becomes more significant. This is because Range_500K spends more time on computing intersection between nodes and the road polygon. It should be noted that with 50,000 ranges, the HistogramTree is not fully visited in HOUSErisk. The search stops at a certain depth of the HistogramTree. Hence, considering the performance and potential needs, 100,000 ranges for querying is an optimal setting.

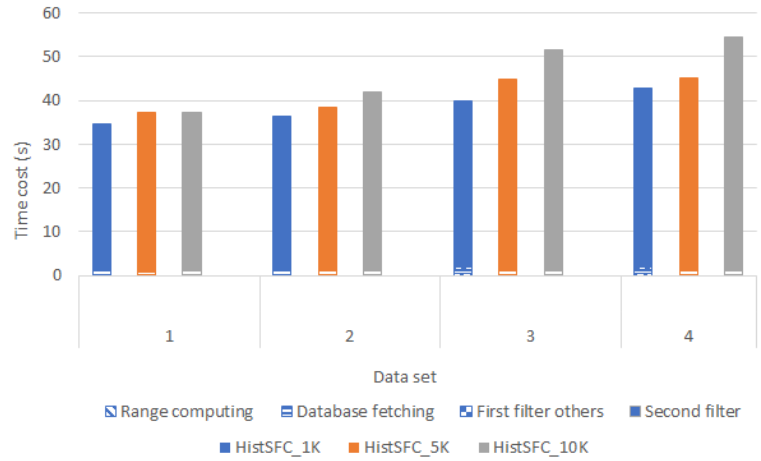
Table 5.15: False positive rate of HistSFC_5K using different number of ranges for querying

Query window	Range_50K	Range_100K	Range_500K
DEPTH3m	0.085	0.084	0.073
ARRIVAL24h	5.688	5.571	4.793
EXTENTmax	0.358	0.357	0.356
HOUSErisk	1.876	1.876	1.868
ROADrisk	6949.3	4565.2	4231.1

Figures 5.39 - 5.43 show the time cost of HistSF and SDO_PC. HistSFC_5K is used, with 100,000 ranges for querying. SDO_PC adopts 5,000 as the block capacity. From DEPTH3m to EXTENTmax, HistSFC responds faster than SDO_PC. Besides, HistSFC also scales better, while SDO_PC takes more time as input data becomes larger. This is mainly because SDO_PC organizes and indexes data using XY only. So, queries on other dimensions instead of XY (e.g., the temporal or the depth dimension) need to unpack and scan all blocks, which is costly. In HOUSErisk and ROADrisk, SDO_PC shows superior performance. This is because both queries use XY range, which caters to the strength of

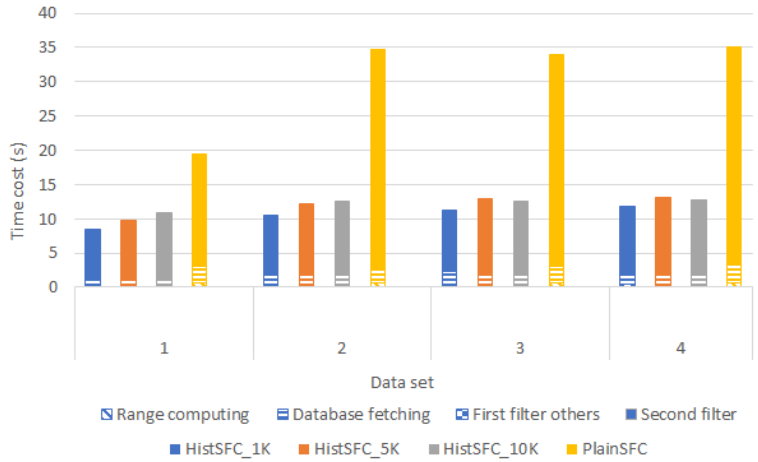


(a) Overall

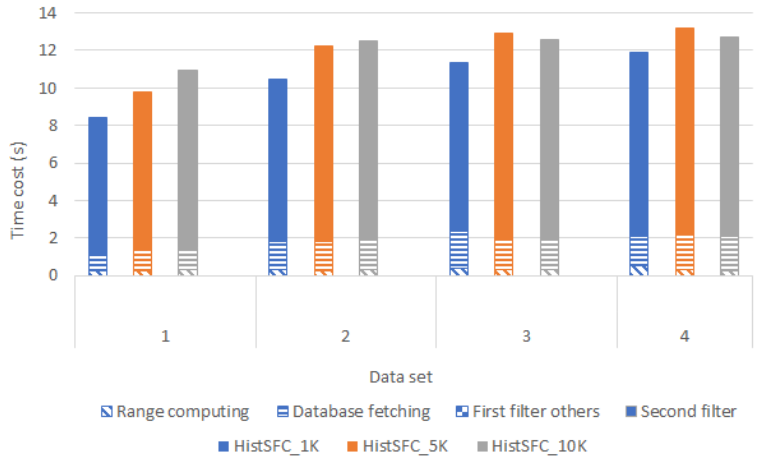


(b) Omitting PlainSFC

Figure 5.29: Time cost of DEPTH3m using different HistogramTrees, with 100,000 ranges used

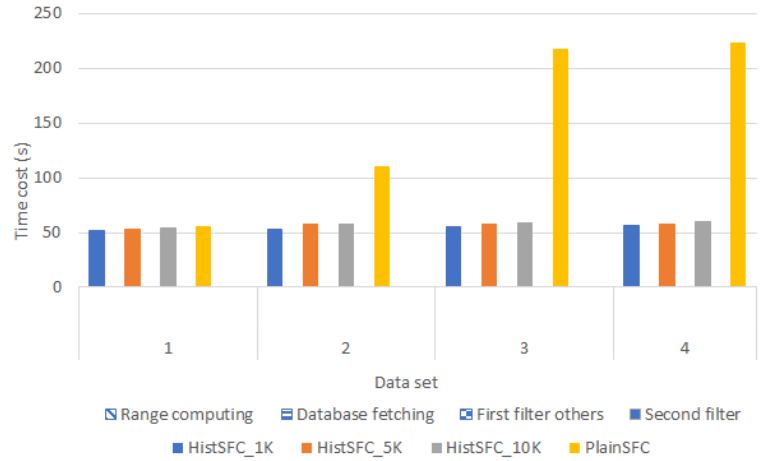


(a) Overall

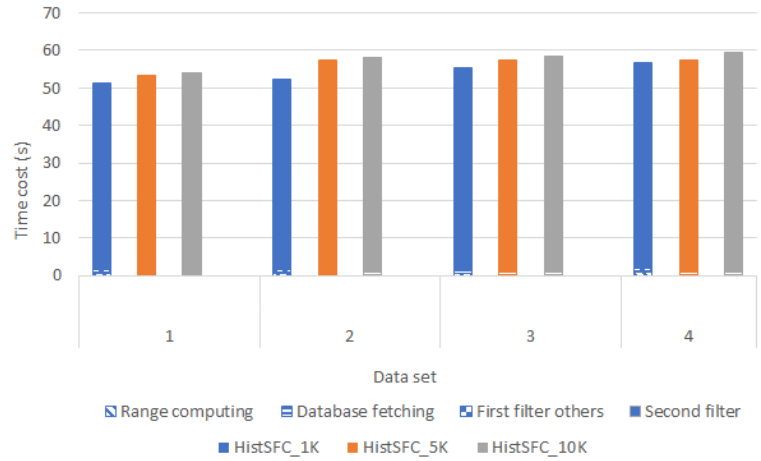


(b) Omitting PlainSFC

Figure 5.30: Time cost of ARRIVAL24h using different HistogramTrees, with 100,000 ranges used

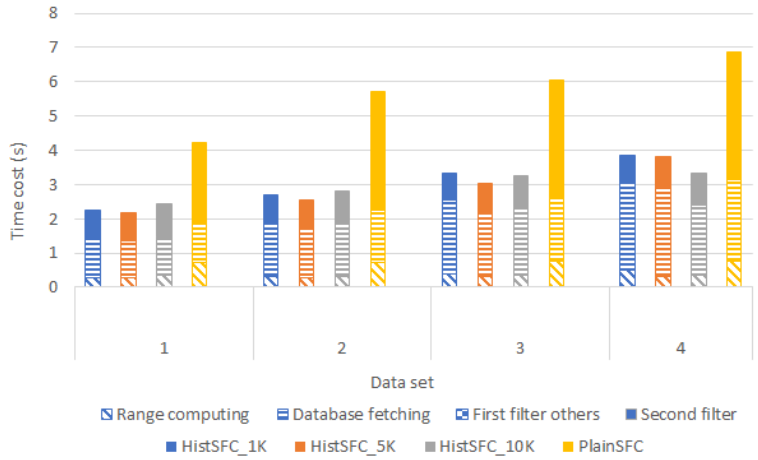


(a) Overall

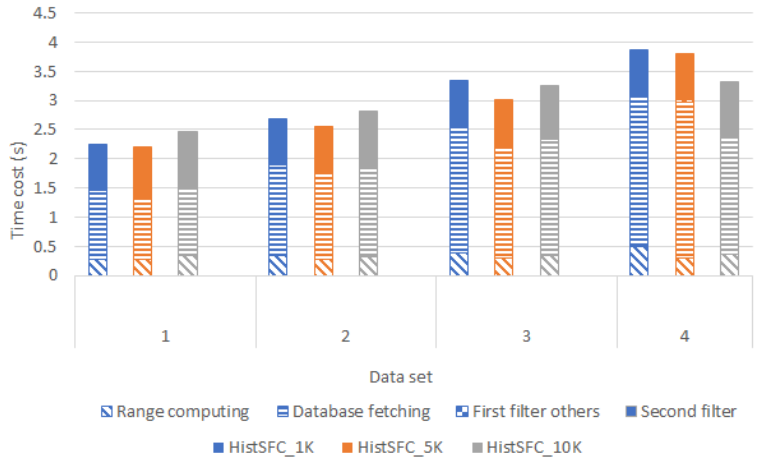


(b) Omitting PlainSFC

Figure 5.31: Time cost of EXTENTmax using different HistogramTrees, with 100,000 ranges used

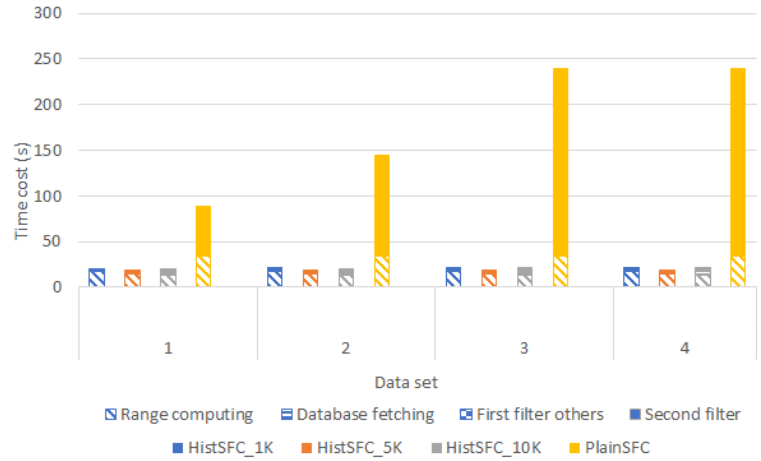


(a) Overall

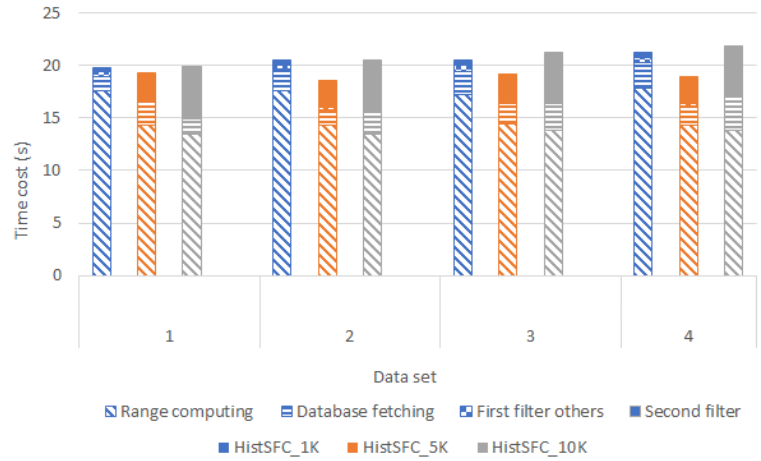


(b) Omitting PlainSFC

Figure 5.32: Time cost of HOUSErisk using different HistogramTrees, with 100,000 ranges used



(a) Overall



(b) Omitting PlainSFC

Figure 5.33: Time cost of ROADrisk using different HistogramTrees, with 100,000 ranges used

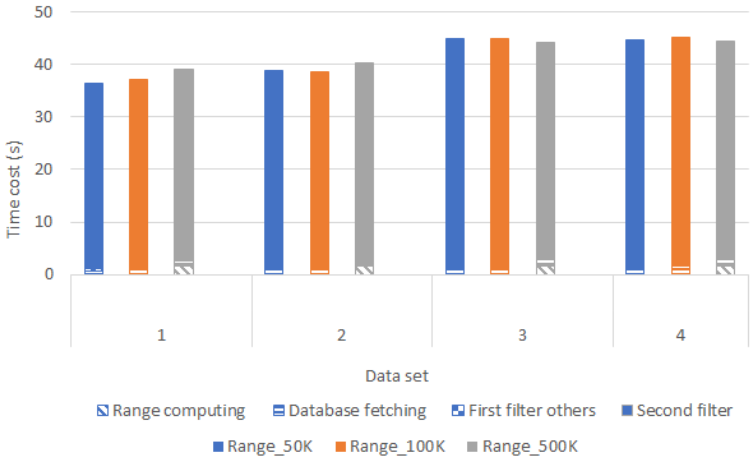


Figure 5.34: Time cost of DEPTH3m with different number of ranges, using HistSFC_5K

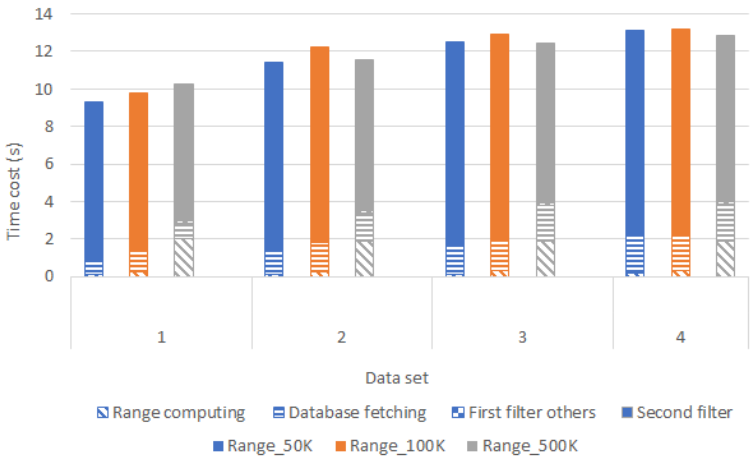


Figure 5.35: Time cost of ARRIVAL24h using different number of ranges, using HistSFC_5K

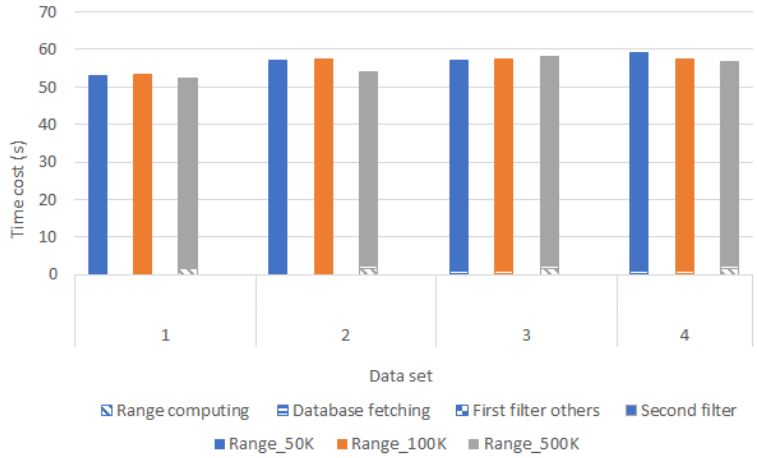


Figure 5.36: Time cost of EXTENTmax using different number of ranges, using HistSFC_5K

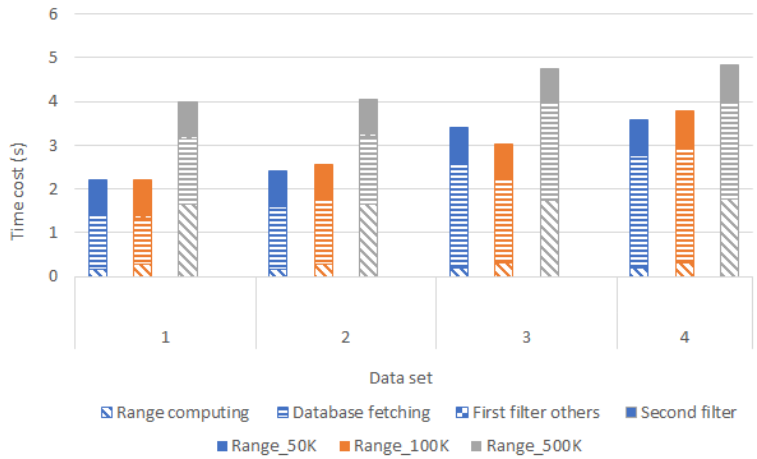


Figure 5.37: Time cost of HOUSErisk using different number of ranges, using HistSFC_5K

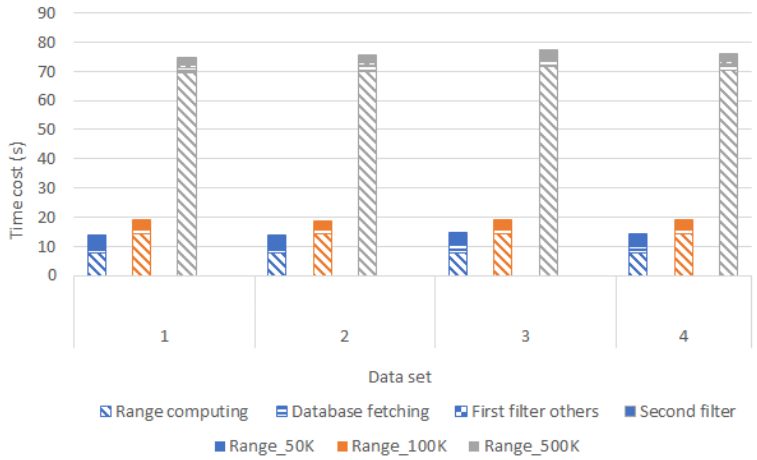


Figure 5.38: Time cost of ROADrisk using different number of ranges, using HistSFC_5K

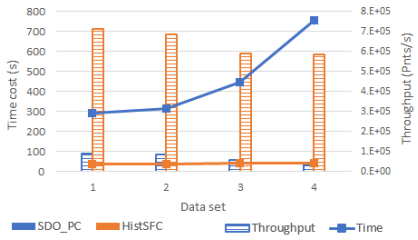


Figure 5.39: Time cost of DEPTH3m using state-of-the-art solutions

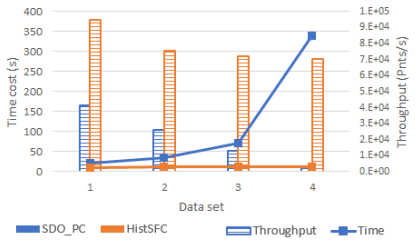


Figure 5.40: Time cost of ARRIVAL24h using state-of-the-art solutions

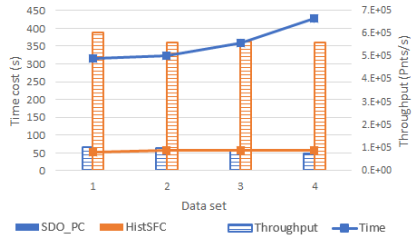


Figure 5.41: Time cost of EXTENTmax using state-of-the-art solutions

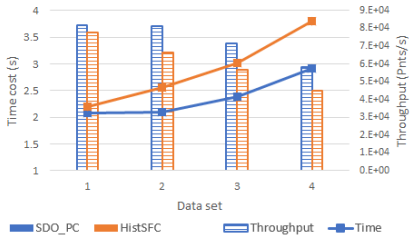


Figure 5.42: Time cost of HOUSErisk using state-of-the-art solutions

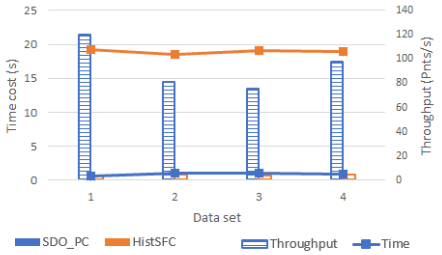


Figure 5.43: Time cost of ROADrisk using state-of-the-art solutions

SDO_PC, especially ROADrisk: SDO_PC provides efficient intersection computation for 2D polygonal geometries (i.e, the road). Besides, only 4 blocks are retrieved in ROADrisk, which needs little I/O. HistSFC adopts 7D data organization, which means more nodes are checked for intersection. This is less efficient. In addition, the ranges returned by HistSFC include large amount of false positive points with velocity below 0.5 or outside the road. This significantly increases FPR. To conclude, this test indicates that HistSFC is advantageous in processing queries on different combinations of dimensions, while SDO_PC is preferable for conventional 2D spatial queries.

5.3. OPTIMIZATION OF HISTSFC

Although HistSFC has shown superior performance in above tests, there is still plenty room for improvement. For certain queries, HistSFC can still generate a large number of vacant ranges, with large FPR. This section investigates 3 possibilities to further optimize HistSFC — refining ranges after HistogramTree, uniforming skewed dimensions and decoding in parallel. In fact, parallel decoding has been implemented for benchmarking HistSFC which is compared with state-of-the-art solutions for AHN2 queries. This section describes this technique in detail.

5.3.1. REFINING RANGES AFTER HISTOGRAMTREE

When the searching process reaches the leaf nodes of HistogramTree, a large number of false positive points near the boundary may not be filtered out. Besides, the maximum number of ranges normally has not been reached. In current settings, selected leaf nodes will be decomposed in a breadth-first way to intersect the query window, to filter out more false positives. This strategy assumes all leaf nodes are equally important for decomposition, which may not be appropriate.

In Figure 5.44, some nodes intersect with the query window by a large proportion, like N_1. Thus, most points inside these nodes are within the query window as well. In contrast, other boundary nodes (e.g., N_2 and N_3) which intersect the query window by a small portion mainly contain false positives. These nodes should get priority for the refinement so that the decomposition becomes more effective. Consequently, I optimize the nodes' partitioning process, by considering the intersection ratio which equals the volume of intersection divided by the volume of node. After this, I compute the es-

timated number of false positive points based on Equation 5.2, to rank the nodes for partitioning. After one decomposition, I assume points are allocated evenly to different child nodes. Then, these child nodes are added into the original pool of leaf nodes, and are ranked again for further decomposition. To verify the applicability of this technique, AHN2 and the flood data are tested in the following.

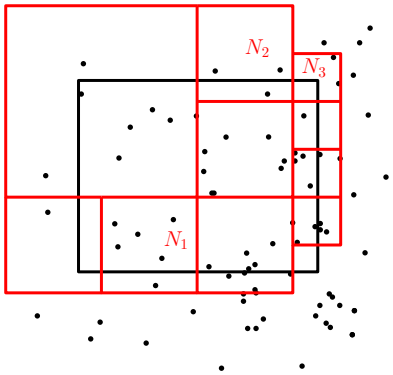


Figure 5.44: A 2D query sample: leaf nodes selected (red) in the HistTree to match the query window

$$\tilde{fpp} = (1 - R) \times c \tag{5.2}$$

where $\tilde{fpp}$ = estimated number of false positive points in a boundary node
R = intersection ratio
c = total number of points in the boundary node

AHN2 TEST

I generated 500 4D query windows based on query logs (Section 5.1.4). I used 10,000 as the capacity threshold to build HistogramTree, and used 100,000 ranges for querying. By averaging the results, I found that the optimized range computing approach decreased the FPR by 28%, compared to the original breadth-first approach. When the output was more than 10 million points, the optimization can reduce the total time cost by 20%, e.g., the Large query (Table 5.2).

FLOOD DATA TEST

Analogous to HOUSErisk (Table 5.10), the testing query window consists of 4 dimensions: X and Y should be within a rectangular region, with depth > 0 and caseID a random number. I randomized the 2D spatial region of which the area ranges from 100 m^2 to 25 km^2. In total, 500 queries were tested. The capacity threshold to build HistogramTree was 5,000. The maximum number for querying was 100,000. The result showed that on average, the FPR was reduced by 4% by using the optimized range computing approach. The insignificant improvement is because as dimensionality goes high, the number of children of a node becomes very large (128 in this case). So, by partitioning a node once, each child gets much less points. Consequently, nodes at a higher level will most likely to be partitioned first, and then the level below. This returns to the breadth-first strategy.

5.3.2. UNIFORMING SKEWED DIMENSIONS

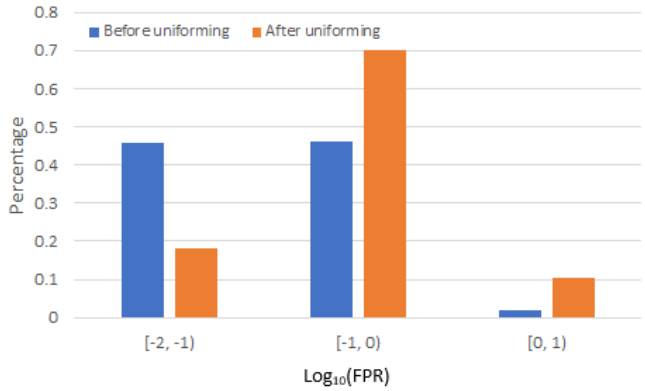
When a dimension is skewed, we may end up with large FPR if the boundary of the query window crosses the dense point area. For example, a query selecting all objects above the ground from an ALS point cloud will hardly reach a clean result in the first filter. The massive points near the ground differ so little in the Z dimension that only nodes refined sufficiently can remove the false positives. However, refining to a very deep level is inapplicable given a huge input. Thus, a straightforward solution is to transform the skewed dimension to a more uniform one so that worst cases can be avoided. From another perspective, uniforming skewed dimensions increases the CHC of the data, which increases the number of effective ranges (Section 4.4).

To uniform a skewed dimension, a generic way is to determine the CDF and Inverse CDF (ICDF) of the skewed dimension, and transform the values according to them (Zhang et al., 2014). Specifically, we divide the dimensions' domain into m pieces, and assume that each piece follows a uniform distribution. Then, after building the 1D histogram of that dimension, we derive the linear piecewise-defined CDFs (i.e., $F_i(x) = a_i x + b_i, i = 0, 1, \ldots, m$) as an approximation of the actual CDF. Using these CDFs, we transform the original values of that dimension into the corresponding CDF values. Then, we encode the SFC keys using the transformed values, while decode the keys using the ICDFs to retrieve original records after a query. The query window should also be transformed using the CDFs before execution. If the ICDF is simple enough, we add little overhead in post-processing of querying results, but effectively decrease FPR. It should be noted that this method is based on the assumption that the distribution of a dimension does not change when adding more data. This assumption is true in most cases, considering possible nD-PointCloud data.

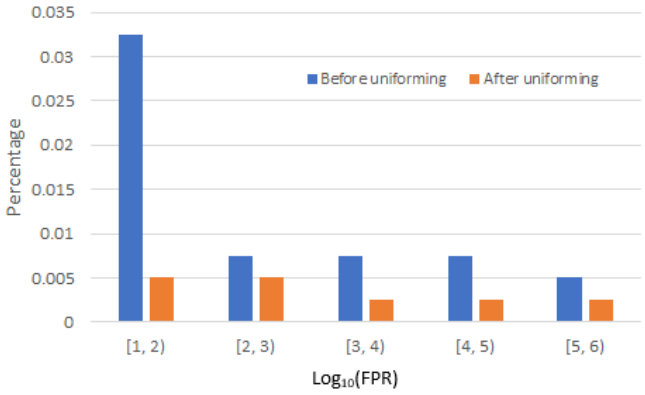
AHN2 TEST

Using piecewise-defined CDFs, I transformed all Z coordinates. The transformed values range from 0 to 1, following a uniform distribution. As the cLoI values were computed using a random and uniform generator (Section 3.3), I restored the values back to the random numbers. Then, two HistSFC solutions were built: one is based on skewed dimensions, while the other is based on the transformed dimensions. The setting of HistogramTree and querying was the same as the above test. In total, I tested 500 randomly generated queries. It turns out that before uniforming, the average FPR is 2,442, but decreases to 401 after uniforming. This is a significant improvement. The large average FPRs are caused by several extremely large values: 6% of FPR values are above 10 before uniforming, while this percentage becomes 1.75% after uniforming. Figure 5.45 specifically shows the histogram of the logarithm of FPR values in base 10.

Figure 5.45b clearly indicates that uniforming skewed dimensions can greatly diminish worst cases. However, the side effect is the increase of low FPR values (Figure 5.45a). This is because uniforming the data on the one hand stretches the dense point region, while on the other hand, compresses the sparse point region. So, when the query window falls in the sparse data region, uniforming can bring negative effect. Consequently, in practice, the distribution of query windows has also to be considered when applying the uniforming strategy.



(a) $-2 \leq \log_{10} \text{FPR} < 1$



(b) $1 \leq \log_{10} \text{FPR} < 6$

Figure 5.45: Histogram of $\log_{10}$FPR

FLOOD DATA TEST

The query for testing is the same as the range refinement test, i.e., confined by caseID, X, Y and depth. CaseID, X and Y follow the uniform distribution approximately. So, I only uniformed the depth dimension. Figure 5.46 presents the original distribution of depth. The 0 value occupies 28.4% out of all values.

I executed 500 queries, and compared the original HistSFC and the one built after uniforming the depth dimension. The result indicates that the average FPR changes from 1.78 to 0.92 after the uniforming. Unlike the AHN2 test, all queries benefit from the uniforming. Besides, originally, 6 FPR values are above 10 with the largest value 129; after uniforming, 4 FPR values are above 10 with the largest value 79. Such significant improvement is mainly due to the specific query type which only selects points with positive depth. So, HistSFC nodes have to distinguish 0 and 0.001 on the depth dimension to return accurate result. This is hard to achieve with the original data organization, as the gap in the SFC key is too small. However, after uniforming, 0.001 is mapped to 0.284 in

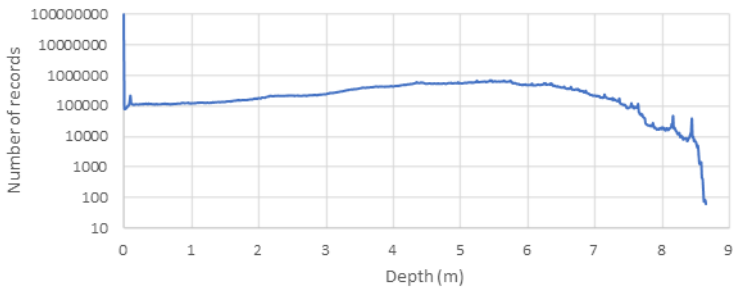


Figure 5.46: Distribution of depth values in the flood data, with resolution 1 mm

the unit domain. Then, selecting non-zero values in the first filter becomes much simpler. However, if other depth values are used in the query (e.g., larger than 3 m), then, uniforming depth may not be necessary. Nonetheless, selecting flooded area (depth > 0) is a very basic operation and has been used much more frequently.

5.3.3. DECODING IN PARALLEL

The previous optimizing techniques are aimed at decreasing the false positive points returned by the first filter. However, the decoding process can still be time consuming if large quantities of keys have to be processed anyway. To address this issue, I adopt the parallel technique for decoding. The approach is to evenly distribute the ranges to different processors so that each processor executes a part of the query and decode the result. However, as each range contains different number of points due to skewed data distribution, the actual workload can be unbalanced among processors. To resolve this issue, I first rank the ranges according to their lengths. This is based on the assumption that the length represents the number of points insides, since the result is from nD-histogram nodes. Then, an ID between 0 and the number of processors is randomly generated and attached to each range. In the first filtering, each processor gets its corresponding ranges and then conducts fetching from IOT.

As decoding can be an independent module without considering specific query windows, I chose large query windows in both AHN2 and the flood data for testing. The test was conducted with 32 processors.

The query window [14890.98, 363438.40, -2.33, 0.00 ; 54392.47, 402939.89, 23.76, 10722.64] is used for AHN2 test. This results in 350,269,367 points for decoding. Table 5.16 presents the time cost of decoding, where $speedup = \frac{T_{serial}}{T_p}$ and T_p refers to the parallel execution time. It indicates that the decoding efficiency can be improved by an order of magnitude when using 16 processors or more.

Table 5.16: Time cost of AHN2 decoding in parallel

Number of processors	Serialized	2	4	8	16	32
Time cost (s)	393	216.8	112.9	62.3	36.9	32.2
Speedup	1	1.81	3.48	6.31	10.64	12.2

After AHN2 test, I used the whole flood data for decoding test. Totally 343,756,800 points are decoded. Table 5.17 shows the time cost. Unlike the AHN2 test, the highest speedup appears using 16 processors instead of 32. The additional overhead introduced by the SQL execution and communication during the parallelization is more significant in the 7D flood data test. Nonetheless, parallel decoding is still very crucial for the flood queries, as large output may always be needed, e.g., statistical analysis.

Table 5.17: Time cost of flood data decoding in parallel

Number of processors	Serialized	2	4	8	16	32
Time cost (s)	336.3	218.3	118.8	66.9	43.7	44.4
Speedup	1	1.54	2.83	5.03	7.7	7.57

5.4. DISCUSSION

This chapter has conducted benchmark tests to evaluate HistSFC, using 4D AHN2 and 8D flood data. In AHN2 test, HistSFC presents similar performance as PlainSFC, while in the flood data test, HistSFC performs much faster than PlainSFC. The major reason, according to my analysis, lies in the different uniformity of these two nD-PointCloud data sets, where AHN2 tends to be more uniformly distributed. To process such uniformly distributed data, HistogramTree may not able to improve further. However, we can still improve PlainSFC's performance practically by changing the range computing method. Previously, I utilize the 2^n-tree embedded in the Morton hierarchy, to iteratively intersect the query window to derive the intersecting nodes. This is unnecessary as we actually know exactly where all nodes are located. The 2^n-tree is strictly formed by splitting each dimension by half in each iteration. So, without the 2^n-tree, we can explicitly derive the nodes at depth l, using lower-left corner of the domain and lower-left corner of the query window. In this way, we avoid the recursive intersection computation and achieve higher efficiency. This method remains as a future work.

Different size of HistogramTree and the number of ranges for querying (r_{max}) are varied to gain knowledge about the optimal settings. The optimal size of HistogramTree depends on the data and implementing environment. For example, considering prevalent settings of hardware and typical applications of AHN2, HistogramTree is suggested to be kept under 1 GB. Otherwise, loading and using will become very cumbersome. The influence of r_{max} is also significant, and a balance has to be found. A general solution is to set r_{max} of different orders of magnitude and compare them by benchmarking, to derive the optimum. This may end up with a suboptimal result as the optimum may not be covered by the test. However, the suboptimal performance may be satisfactory (Meijers & van Oosterom, 2018).

The empty ratios (Table 5.5 and 5.13) show that even using HistogramTree, considerable amount of empty ranges can still be generated. So, I developed a range optimization technique considering intersection ratio between nodes and the query window (Section 5.3.1). However, the improvement is limited, especially when the dimensionality of the data is high. An alternative to decrease the empty ratio is performing a gluing operation after ranges have been generated (Figure 5.47). Basically, some of the ranges ex-

ported may still be neighbors on the SFC curve which are divided in the refining process. They can be directly concatenated. Some ranges are separated, but are close to each other with small in-between gaps. Then, by adding an additional short SFC segment, two separate ranges can be glued to form a single range. As the SFC segments inserted are short, they most likely contain few points or no points at all. Based on this operation, we may derive more effective ranges in the end. With current settings, HistSFC can first generate more ranges than r_{max}, and rank the range pairs according to the gap between them. Range pairs with small in-between gaps get priority in gluing. The gluing stops when r_{max} is reached. I will further verify this technique in the future.

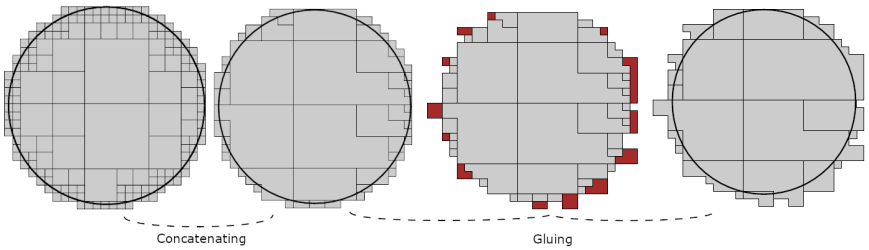


Figure 5.47: Gluing technique to reduce the number of ranges (Psomadaki, 2016)

To efficiently use PlainSFC and HistSFC, it is crucial to determine the organizing dimensions. This is because high dimensional nodes generate a large number of child nodes by partitioning once, but r_{max} is confined by memory size. Hence, the nodes cannot be refined sufficiently to reduce false positives. As has been mentioned, the determination depends on applications, where dimensions queried frequently should be used as organizing dimensions. However, sometimes, it is difficult to predict users' needs, e.g., a new application which needs to conduct queries on property dimensions. Then, rebuilding the whole storage has to be performed. Another practical detail in implementation is to scale values of organizing dimensions for Morton key encoding. As the bits taken from different organizing dimensions are interleaved to derive the key, the extent of these dimensions should first be scaled to an equal size for computing the key. In this way, when searching to a lower depth, the organizing dimensions can all get involved to be fairly filtered.

HistSFC is also compared with state-of-the art solutions. The block based solutions including PostGIS MULTIPOINT and Oracle SDO_PC present superior performance in conventional spatial queries, thanks to efficient indexing and I/O operation. However, their performance degrades significantly if large amount of blocks are selected, e.g., in high dimensional queries. Unpacking blocks causes intensive computation in CPU, which costs even more time than the I/O operation. Hence, to still use block based solutions which are advantageous in I/O and network transmission, the block unpacking task can be moved to GPU and parallelized, or sent to the users' side to process.

The Pyramid-Technique which is proposed to query high dimensional point data performs poorly. The reason concerns 2 aspects: one is the number of dimensions in the query, and the other is the query shape (Liu, van Oosterom, Meijers, & Verbree, 2020). Unlike general nD vectors used in multimedia or data mining (Böhm et al., 2001), every

dimension of nD-PointCloud has a physical or semantic meaning. Thus, confined by the cognition, it is unlikely that users initialize queries concerning more than 10 dimensions. In addition, the majority of current spatial applications still focus on several typical dimensions, such as spatio-temporal dimensions, LoI and classification (Section 2.1.2). With respect to the query shape, it is not always hypercubic which caters to the design of Pyramid-Technique. The Pyramid-Technique clusters data according to the pyramids instead of spatio-temporal coherence. So, when the query window is of a long and narrow shape which crosses all the levels of the pyramid, all points in the pyramid will be selected. This can cause huge number of false positive points selected. I list some typical non-hypercubic querying windows below. In contrast to Pyramid-technique, these queries can be handled properly by HistSFC which approaches different query geometries by recursively refined SFC nodes:

- Buffer of a curve, e.g., a river or a road in ALS point clouds
- Diagonal rectangle for an inclined feature, e.g., escalator inside a mall in indoor point clouds
- Arbitrary geometry, e.g., footprint of a municipality in ALS point clouds
- View frustum for perspective view selection in LiDAR point clouds
- Trajectory of an object in GPS point clouds, i.e., a 4D window query on XYT and identity, where identity is a specific value while XYT can be the entire extent
- Spatio-temporal rectangle, e.g., one day out of a year in the whole spatial region repetitively scanned by laser scanners (Schreijer, 2021).

6

EXECUTING CONVEX POLYTOPE QUERIES ON ND-POINTCLOUD

BESIDES window queries, other query geometries can also be resolved by being converted to 1D Space Filling Curve (SFC) ranges. This can also be done by recursively decomposing the nD space to approximate the query geometry. The convex polytope query, which includes the widely adopted convex polygonal query in 2D, also plays a critical role in many nD spatial applications such as the perspective view selection. However, there are few nD solutions addressing this type of query. Aiming for an nD solution to resolve the polytope query, this chapter integrates three approximate geometric algorithms — SWEEP, SPHERE, VERTEX, and a linear programming method CPLEX, developing a solution based on PlainSFC (Liu, Thompson, et al., 2021). Additionally, the solution can adopt nD HistogramTree to handle non-uniformly distributed data efficiently.

The chapter first reviews related work on polytope querying algorithms in Section 6.1. This is then followed by the description of newly designed algorithms in Section 6.2. Section 6.3 evaluates the performance of different algorithms using synthetic data. Section 6.4 evaluates the performance in real use cases, including AHN2 perspective view selection and flood risk queries. Based on these results, Section 6.5 concludes the chapter with a comprehensive discussion.

6.1. RELATED WORK

Studies on polytope querying originates from geometric algorithms, where researchers mainly propose and analyze different algorithms theoretically (Chazelle, 1989; Matoušek, 1992, 1994) based on in-memory data structures. Agarwal et al. (2000) proposed a solution based on a partition tree structure managing data on disks. They specifically focused on analyzing the worst-case querying performance, but no practical experiments were conducted. These theoretical approaches are difficult to implement and may not be applicable to address big point data (Khan et al., 2014).

The development of spatial indexing has facilitated the design and implementation

of polytope querying. Based on the R-tree, Goldstein et al. (1997) developed two algorithms: one is the “simple" method which computes a scalar product indicating the minimum distance between a block and a half-space to examine whether an intersection happens; while the other method iteratively uses half-spaces to clip the R-tree block to detect intersection (Figure 6.1). However, they discussed little about the performance in nD space, beyond testing a uniformly distributed 5D data set from the business domain. None of the algorithms proposed distinguishes the type of intersection (inside or touching), leading to redundant intersection computation for a branch block totally inside the polytope. Kollios et al. (1999) developed an approximation algorithm to resolve trajectory searching problems. They also adapted and implemented algorithms of Goldstein et al. (1997) using the hB^{Π}-tree (Evangelidis et al., 1997) as a comparison. The result is that for 2D and 3D issues, their approximation algorithm works more efficiently.

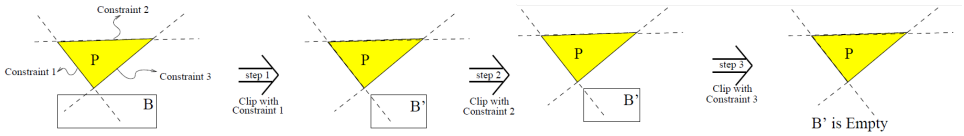


Figure 6.1: The clipping operation where P refers to the polytope and B is an R-tree block, from Goldstein et al. (1997)

More recently, Wang and Ravishankar (2013) developed an encrypted R-tree structure for polytope querying on the cloud computing platform. However, the solution determines whether a tree node intersects the polytope only based on the lower-left corner or the upper-right of the node, which is not rigorous and may omit possible intersections. Besides, they only tested queries on 2D point data. Khan et al. (2014) developed a novel Planar index composed by multiple set of hyperplanes to solve scalar product queries which covers the polytope query. However, given non-parallel half-spaces, the method needs several Planar indices to function, which is extremely expensive when the number of half-spaces for querying is large.

Compared with those approaches, my approach provides a true nD operator for executing convex polytope queries on nD points. It can be directly implemented and used in any DBMSs that support the B+-tree structure. Besides, the solution always returns the correct result. It is also very efficient thanks to the advanced clustering and indexing mechanisms of PlainSFC. Besides, nD-histogram can also be adopted to improve the performance of querying on inhomogeneously distributed point data.

6.2. POLYTOPE QUERYING

A convex polytope is defined as an nD geometry for which, given any 2 points within the region, every point along a straight line joining the points is also within the region. To use the polytope practically, this section first provides the mathematical formulation in 6.2.1. Then, Subsection 6.2.2 develops novel intersection algorithms, which extends the range computing module of PlainSFC to allow the transformation from an nD-polytope query to 1D ranges. The symbology used in the following is listed in Table 6.1.

Table 6.1: Notations

Notation	Description
n	Number of dimensions, assuming the dimensions in the nD-PointCloud data are all used as the organizing dimension
m	Number of half-spaces in the polytope
N	Input size in the number of points
k'	Output size from the first filter
k	Size of the accurate answer
r	Number of ranges generated by the first filter
r_{max}	Maximum number of ranges for querying (threshold)
B	Page capacity of storage

6.2.1. MATHEMATICAL FORMULATION

A half-space is a division of space along a hyperplane (Figure 6.2). Here it is defined as the set of points $\mathbf{x}$ such that $\boldsymbol{\omega} \cdot \mathbf{x} + \beta \leq 0$, where $\boldsymbol{\omega}$ is a unit vector ($\boldsymbol{\omega} \cdot \boldsymbol{\omega} = 1$) and β is a scalar. Note that the inequality is used here for compatibility with the conventions of computer representation software (3D) that the normal vector $\boldsymbol{\omega}$ is oriented so that it points to the outside of the solid object. A half-space can be denoted by the tuple ($\boldsymbol{\omega}$, β).

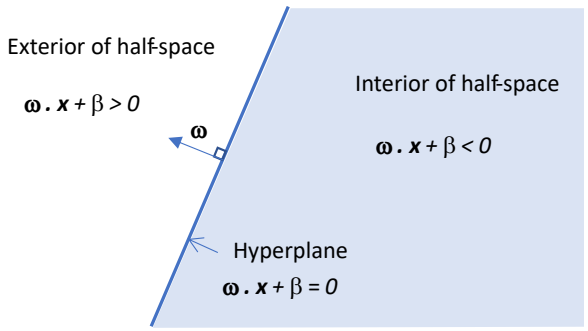


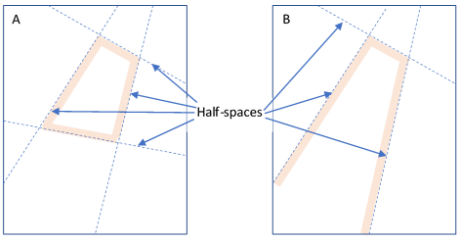
Figure 6.2: The definition of a 2D half-space which can be generalised to nD

In 2D the term half-plane is sometimes used, defined by an infinite straight line — its only boundary. In 3D space, the half-space is defined and bounded by an infinite plane, while in higher dimensions the half-space represents all points on a particular side of an $(n-1)$D hyperplane. In all cases, the dividing $(n-1)$D hyperplane has the definition $\boldsymbol{\omega} \cdot \mathbf{x} + \beta = 0$. A convex polytope is defined as the intersection of a finite set of half-spaces, where the boundaries may not be complete (Figure 6.3):

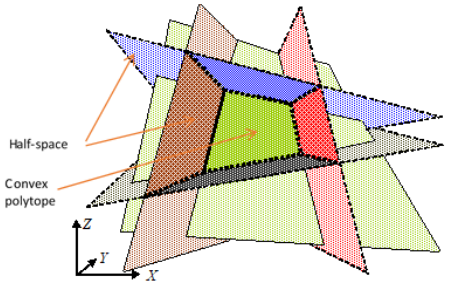
$$C = \cap_{i=1}^{m} H_i$$

where H_i is a set of m half-spaces.

Based on this formulation, we can test whether a point is within a half-space by evaluating $\boldsymbol{\omega} \cdot \mathbf{p} + \beta$: if the value is non-positive, the point $\mathbf{p}$ is within the half space. Since $\boldsymbol{\omega}$ and $\mathbf{p}$ are vectors of length n, the operation per point costs $\mathcal{O}(n)$ time. Then, computing



(a) 2D convex polytopes defined by half-spaces. A: a completely bounded convex polytope, B: a convex polytope bounded on three sides only



(b) A 3D convex polytope, from Thompson (2007)

Figure 6.3: Convex polytopes in 2D and 3D spaces

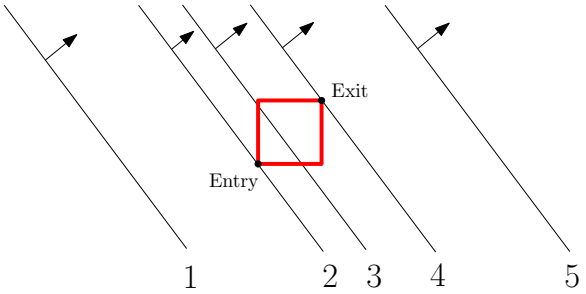


Figure 6.4: The “sweeping" process: in (1), a half-space starts sweeping with the node outside; (2), (3), (4), the half-space sweeps over the node, where intersection happens; (5), the sweeping ends with the node inside the half-space

the relationship between the point and the convex polytope, can cost $\mathcal{O}(mn)$ time per point. However, globally traversing all the points for selection is too costly and scales badly with the size of input. Consequently, I adopt PlainSFC to speed up the search.

6.2.2. INTERSECTION ALGORITHMS

Given a set of ω and β, we can then use PlainSFC to retrieve the result. The core of querying lies in the intersection computation between nodes and the convex polytope to generate ranges in the first filter. As is mentioned in Section 2.6.1, a node in the Morton hierarchy represents a specific region in nD space, indicating a range of Morton codes. Besides, the principle can also be applied to a HistogramTree node, for handling non-uniformly distributed point data. This section develops 3 geometric algorithms, and an additional linear programming solution provided by CPLEX. These algorithms return ranges which are then joined with the IOT, with a final filter performing the aforementioned point-in-polytope test.

SWEEP

SWEEP first identifies the “entry” and “exit” of a node with respect to a half-space (Figure 6.4): imagine if the half-space were to be moved from a great distance away, towards and across the node so that ultimately the node is within the half-space; the entry and

exit are the first and last vertices to cross the boundary. The categorization as entry / exit is only true in relation to a single half-space, and must be re-appraised for others. Then, based on the distance between the half-space's boundary and the entry or the exit, SWEEP determines if an intersection happens. Figure 6.5 presents the whole workflow of SWEEP for one half-space.

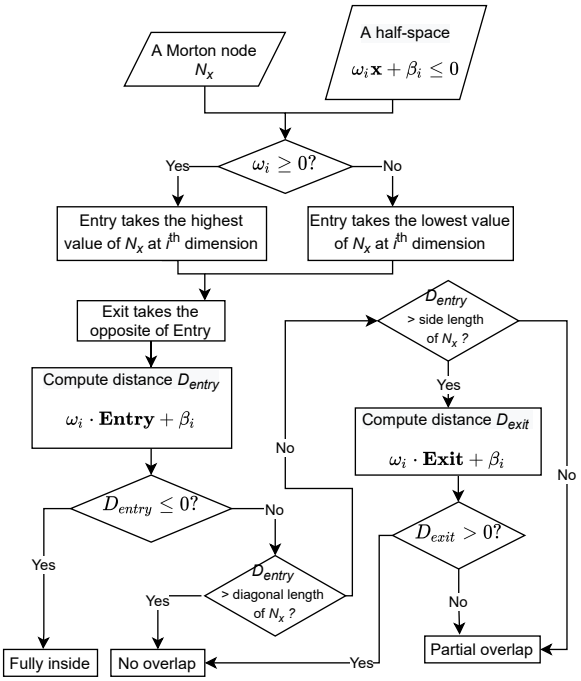
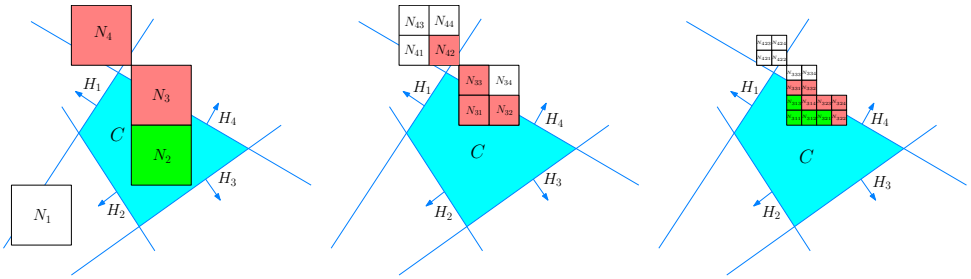


Figure 6.5: The workflow of SWEEP

Figure 6.6 shows how SWEEP works with different nodes after several iterations of decomposition. Node N_1 is not within the half-space H_2, therefore it is external to polytope C, and can be dropped. N_2 is within all of H_1 to H_4, and its range can be exported. In the case of N_3, since it fulfils neither of these cases, it must be placed in a refinement pool before being accepted or rejected. The case of N_4 is significant, because it partially overlaps or falls within each half-space, but in fact it does not intersect C. We refer to this case as a False Positive Node (FPN) to be discussed later. In the process of searching the nodes from the refinement pool, sub-nodes at the next lower levels are processed (2^n of them). These are then applied to the same tests against C. Some sub-nodes are found to be internal, some to be on the boundary, and the rest external.

FPNs exist at crossings of half-spaces (e.g., N_4 in Figure 6.6a and N_{42} in Figure 6.6b). It is a practical proposition to ignore the problem, and allow FPNs to be processed as if they are true positive nodes. In Figure 6.6, N_4, after first refinement, has three of its sub-nodes eliminated, leaving only N_{42} whose sub-nodes are eliminated at the next level. This appears to be a common event — that as a FPN is decomposed into sub-nodes at one level below, those sub-nodes are largely eliminated. With the second filter, points in



(a) Representative nodes (b) Refining once (c) Refining twice

Figure 6.6: SWEEP selection based on PlainSFC, with white nodes outside, green nodes inside and red nodes on the boundary

any remaining FPNs will all be eliminated.

SPHERE AND VERTEX

SPHERE and VERTEX are alternatives. They also detect intersection by examining the relationship between a node and all half-spaces.

The SPHERE algorithm first computes the centre of a node. If the centre is in the half-space, or the Euclidean distance between the centre and the half-space is within half of the diagonal length of the node, the node will be selected. "inside" or "partial overlap" can be decided depending on the distance. SPHERE only needs a central point for intersection detection, which is favorable. However, as the distance computed is an upper bound, FPNs will be selected (N_4 in Figure 6.7).

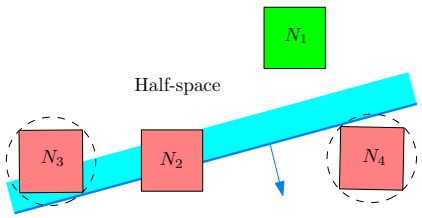


Figure 6.7: Intersection detection using SPHERE: N_1 is inside; N_2 partially overlaps the half-space; N_3 and N_4 are falsely detected as partial overlap

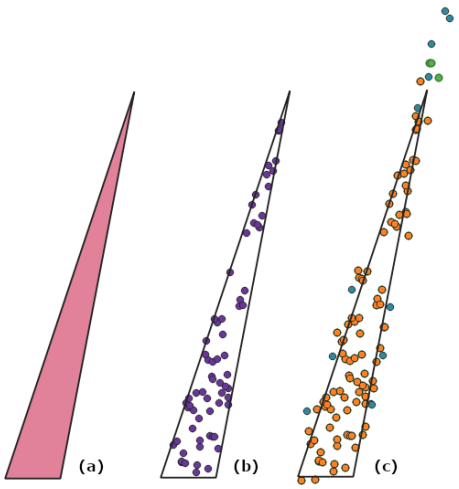


Figure 6.8: Querying results from different algorithms: (a) querying geometry, (b) accurate result, (c) Overlapping results of CPLEX (orange), SWEEP (green) and SPHERE (blue), without a second filter

The VERTEX algorithm is more straightforward, as it examines every vertex of a node to determine whether the node intersects a half-space. If all vertices are outside, then no intersection happens. If all the vertices are in the half-space, the node is inside. For all other cases, a partial overlap is returned. The implementation is simple, but the algorithm degrades in high dimensional spaces because the number of vertices of a node grows exponentially with dimensionality. False detection will also arise at crossings of half-spaces (e.g., Figure 6.6a), as with SWEEP.

CPLEX

The rigorous linear programming method detects intersection by finding solutions for a set of equations defined by the $(n-1)$D hyperplanes of the polytope and a node. I realize this by using CPLEX which is a tool developed by IBM to solve linear optimization problems (Lima, 2010). It provides optimal solutions to an objective function confined by a set of constraints. Using CPLEX, I can create a variable array x (x_1, x_2, x_3,... x_n), and set their range according to the bounds of a node (i.e., $L_1 \le x_1 \le U_1$, $L_2 \le x_2 \le U_2$, ... $L_n \le x_n \le U_n$). Then, I convert all half-spaces to constraints in the form of $\boldsymbol{\omega} \cdot \mathbf{x} + \beta \le 0$. I set the objective function of the linear model to 0, meaning that once a solution found, the program will stop. In this way, CPLEX detects whether an intersection happens.

Figure 6.8 presents the results of a 2D triangle query on a uniformly distributed point set, with a proper setting of PlainSFC. SPHERE contains all points selected by SWEEP which again contains points selected by CPLEX. VERTEX returns the same result as SWEEP. The result indicates a general pattern of k', which is SPHERE $\ge$ SWEEP (VERTEX) $\ge$ CPLEX. The false positive points are distributed along the boundaries and around the acute corners. Several factors influence the occurrence of these points, including the dimensionality, relative positions of the half-spaces to nodes, and r_{max} for querying. The actual performance of these algorithms depends on the specific settings and implementation, which is evaluated by experimenting in the following sections.

6.2.3. THEORETICAL COMPLEXITY

Since these algorithms are all based on PlainSFC, Equation 2.1 still applies. However, in SWEEP and SPHERE, T_{pre} is bounded by $\mathcal{O}(mnr \log_B N)$. In VERTEX, every vertex of a node has to be examined. Thus, its T_{pre} is bounded by $\mathcal{O}(2^n mnr \log_B N)$. T_{io} maximally covers $\mathcal{O}(\frac{k'}{B} + r)$ I/Os, while T_{post} is bounded by $\mathcal{O}(mnk')$. Once parallelism is applied, T_{post} becomes $\mathcal{O}(\frac{mnk'}{p})$, given p processors. Besides, all intersection algorithms introduce FPNs except CPLEX. So, k' can be varied. An optimal solution should balance the three cost terms, as has been mentioned in Section 2.6.3.

6.3. nD-SIMPLEX AND nD-PRISM TESTS

This section presents ideal tests to evaluate the performance of different algorithms described above. A regular nD-simplex model and an nD-prism model are built for testing. The nD-simplex is the simplest nD-polytope, while the nD-prism is devised to investigate how the number of half-spaces influences the querying efficiency. Both tests use a single thread, recording 3 indicators for evaluating the performance:

1. Selectivity of the first filter (Equation 2.3).

2. Number of iterating cycles (i.e. for-loops) to generate ranges. A cycle of CPLEX means resolving the optimal problem once (Section 6.2.2). A cycle of SWEEP, SPHERE and VERTEX means computing the distance from a half-space to a point (or vertex) in the node.

3. Time cost of the first filter and the second filter. The first filter time corresponds to T_{pre}, while the second filter takes $T_{io} + T_{post}$ to accomplish.

Selectivity shows the accuracy of intersection computation. The number of iterating cycles is used to explain the scalability and efficiency of range computation of algorithms. Time cost indicates the overall performance.

6.3.1. SYNTHETIC DATA SETS

I apply an independent uniform distribution in all dimensions. The value for each dimension uses 12 bits, so the 10D Morton key requires 120 bits which fits within Oracle NUMBER type (128 bits). Thus, the value of each dimension is between 0 and 4095. This limits the unique points possible in 2D. So, I generate 10^4 2D points, and 10^6, 10^7, 10^8 and 10^{10} points for 4D, 6D, 8D and 10D data sets respectively. With these data sets, it is possible to investigate the querying scalability with respect to dimensionality.

6.3.2. THE REGULAR ND-SIMPLEX QUERY

In the following, I first build the simplex model for querying at different dimensionality. Then, I perform the test and discuss the results.

QUERY GEOMETRY CONSTRUCTION

An nD-simplex has $n+1$ vertices, from v_0 to v_n, where $v_i = (v_{i0}, v_{i1}, \ldots, v_{i(n-1)})$. I use a unit vector along each axis to represent a vertex to get the first n vertices of the simplex (Figure 6.9), i.e., $v_0 = (1,0,0,\ldots)$, $v_1 = (0,1,0,\ldots), \ldots, v_{n-1} = (0,0,\ldots,1)$. The last vertex v_n must have the same value for every dimension to make a regular nD-simplex. Assuming $v_n = (\varepsilon, \varepsilon, \ldots)$, I calculate it by solving a quadratic equation on the length of (v_i, v_n), $(i < n)$. So, there are two possible solutions, $\varepsilon = \frac{1 \pm \sqrt{n+1}}{n}$. I take $\varepsilon = \frac{1+\sqrt{n+1}}{n}$. Then, the mean of all vertices is derived: $M = (\mu, \mu, \ldots)$, where $\mu = \frac{1+\varepsilon}{1+n}$. Then, I shift the whole simplex by moving M to the origin O. In this way, the simplex is centralized at O, while the vertex $v_i = (v_{i0}, v_{i1}, \ldots, v_{i(n-1)})$, where $v_{ii} = 1 - \mu$ for $i < n$; $v_{ij} = -\mu$ for $i \neq j$ *and* $i < n$; $v_{nj} = \varepsilon - \mu$ for $j = 0, 1, \ldots, n-1$.

Afterwards, I normalize the vertices to build the normal vectors (i.e., $\vec{a}$, $\vec{b}$ and $\vec{c}$ in Figure 6.9) for a new set of faces. These faces constitute another nD-simplex (with dashed green boundaries). As the simulated data (Section 6.3.1) are positive numbers, I shift the new simplex to the positive zone: by shifting the first n faces (H_0 to H_{n-1}) to pass through O (except H_n which is opposite to O), I could build such a "positive" regular nD-simplex for querying. The size of the simplex depends on the position of H_n which is adjustable (Figure 6.9). Hence, half-spaces constituting this regular nD-simplex are derived:

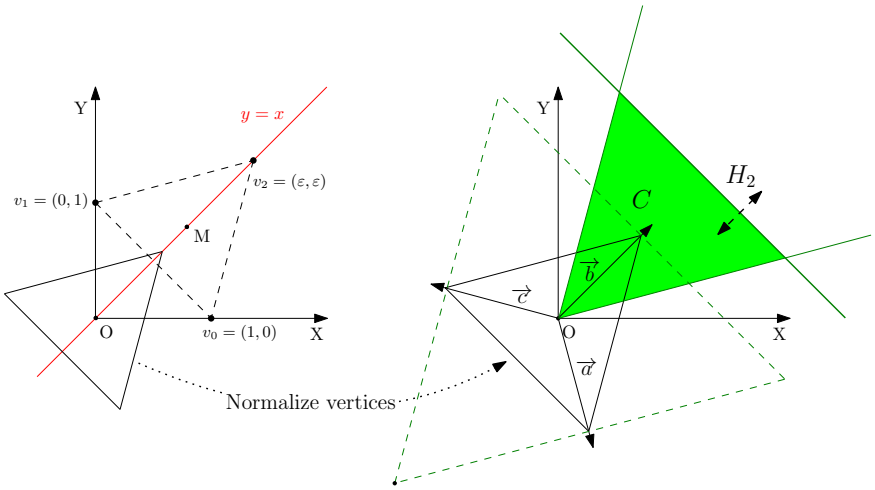


Figure 6.9: The workflow to build a regular simplex testing model in 2D

$$
\begin{aligned}
H_0 &: \frac{1-\mu}{A}x_0 + \frac{-\mu}{A}x_1 + \cdots + \frac{-\mu}{A}x_{n-1} \le 0 \\
H_1 &: \frac{-\mu}{A}x_0 + \frac{1-\mu}{A}x_1 + \cdots + \frac{-\mu}{A}x_{n-1} \le 0 \\
&\vdots \\
H_n &: \frac{1}{\sqrt{n}}x_0 + \frac{1}{\sqrt{n}}x_1 + \cdots + \frac{1}{\sqrt{n}}x_{n-1} \le -\beta
\end{aligned}
$$

where $A = \sqrt{n\mu^2 - 2\mu + 1}$, and $-\beta$ is the distance from O to H_n. To maximize the simplex within the data region, $-\beta$ should be taken as large as possible. Suppose the domain is a unit hypercube with every dimension ranging from 0 to 1. Then, the maximum simplex formulated above leads to an intersection of H_n and H_i $(i < n)$ on the face $x_i = 1$. In other words, when $x_i = 1$, we will get the same expression based on the equation of either H_i or H_n. I just use H_0 and H_n to derive

$$\beta = -\frac{1}{\mu\sqrt{n}} = -\frac{\sqrt{n}}{1 + \frac{1}{\sqrt{n+1}}}$$

However, with such a formulation, all vertices of the simplex are on the boundary faces, and there is no room for SWEEP making false detection at the corners (Figure 6.6a). So, I raise β to move H_n towards O, which creates some gap between the first n vertices and the boundaries, except the last vertex which coincides with O. Mathematically, this means the following formulation is adopted:

$$\beta = R\sqrt{n} - \frac{1}{\mu\sqrt{n}}, \left(0 \le r \le \frac{\sqrt{n+1}}{1+\sqrt{n+1}}\right) \tag{6.1}$$

where R is a ratio indicating the residual gap created. I call it the residual ratio. A larger R corresponds to larger room for false detection. I multiply R by $\sqrt{n}$ to keep the product a similar decreasing speed as the original β, when n grows.

Based on the formulation above, I can compute the volume of the nD-simplex which indicates the selectivity. Suppose the edge length of the simplex is s, while the height is h, i.e., $-\beta$ in this case, we could derive

$$-\beta = h = s\sqrt{\frac{n+1}{2n}}$$

$$V = \frac{s^n}{n!}\sqrt{\frac{n+1}{2^n}} = \left(-\beta\sqrt{\frac{2n}{n+1}}\right)^n \frac{1}{n!}\sqrt{\frac{n+1}{2^n}} \tag{6.2}$$

As the test is up to 10D, I substitute $n = 10$ and $R = 0.1$ to Equation 6.1, and then substitute the resultant β to Equation 6.2 to compute V: $V_{10} = 0.0010091$. This results in a 1‰ selectivity. Then, I compute R using Equation 6.3 for other data sets, to achieve the same selectivity:

$$R = \frac{\sqrt{n+1}}{n\sqrt{2}}\left(\frac{n\sqrt{2}}{1+\sqrt{n+1}} - \left(n!V_{10}\sqrt{\frac{2^n}{n+1}}\right)^{\frac{1}{n}}\right) \tag{6.3}$$

where $n = 2, 4, 6, 8$. Table 6.2 shows the computed R. Table 6.3 presents the ratio between the distance from the origin O to H_n and the diagonal length, i.e., $-\frac{\beta}{\sqrt{n}}$. This ratio indicates how the simplex stretches over the data region as dimensionality rises. Figure 6.10 provides some illustrations of the simplexes created.

Table 6.2: Residual ratio for different dimensionality

	2D	4D	6D	8D	10D
R	0.6044127	0.5106438	0.3701990	0.230515	0.1

Table 6.3: The ratio between the distance from O to H_n and the diagonal length of the simplex

	2D	4D	6D	8D	10D
$-\frac{\beta}{\sqrt{n}}$	0.029562	0.180339	0.355509	0.519485	0.668337

RESULTS AND ANALYSIS

The experiment sets r_{max} to 10^6. This guarantees a low FPR in high dimensional spaces, without bloating the memory. CPLEX (two versions), SWEEP, SPHERE and VERTEX are tested. CPLEX#1 distinguishes between two types of intersection: inside or partial overlap, while CPLEX#2 does not, meaning nodes inside the simplex are refined unnecessarily.

Tables 6.4 – 6.6 show the results. Over all, CPLEX#1 owns the lowest FPR, while SWEEP responses the fastest below 10D with CPLEX#2 fastest in 10D. The low FPR of

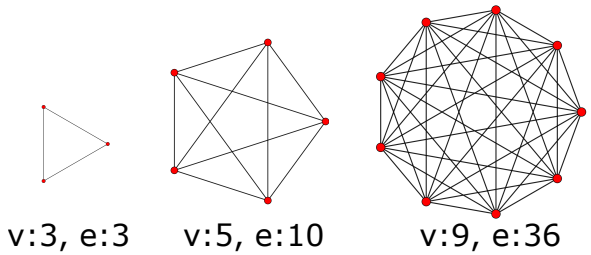


Figure 6.10: Orthogonal projections of a 2D-simplex, 4D-simplex and 8D-simplex built, with the number of vertices (v) and edges (e). Image source: Wikipedia

CPLEX#1 leads to the smallest k' for post-processing. However, as CPLEX#1 spends significantly more time for each iteration for computing ranges, such advantage is insignificant until 10D (Table 6.6). CPLEX#2 presents analogous selectivity as CPLEX#1, but is faster. This is because CPLEX#1 decomposes the simplex to individual half-spaces to further compute inside or partial overlap, while ignoring this reduces about 50% computation (Table 6.5). Besides, it becomes less necessary to distinguish these two intersection types when n increases, as all nodes returned by the first filter tend to fall on the boundary (Table 6.7). This is because the number of nodes at each level of the Morton hierarchy increases exponentially with n, the final nodes selected mainly reside in higher levels with larger sizes. So, these nodes are more likely to partially intersect the simplex.

Table 6.4: Selectivity of the first filter

	2D	4D	6D	8D	10D
CPLEX#1	0.1%	0.1345%	0.4805%	2.503%	40.01%
CPLEX#2	0.1%	0.1387%	0.4815%	2.503%	40.01%
SWEEP	0.1%	0.1364%	0.9244%	16.45%	60.50%
SPHERE	0.1%	0.1386%	1.193%	46.71%	92.95%
VERTEX	0.1%	0.1364%	0.9244%	16.45%	60.50%

Table 6.5: Number of iterating cycles for computing ranges

	2D	4D	6D	8D	10D
CPLEX#1	3,653	4,412,563	4,262,738	4,650,728	6,299,304
CPLEX#2	23,696	1,302,544	1,566,912	2,489,856	3,550,208
SWEEP	2,259	2,223,188	6,451,764	33,842,261	16,336,597
SPHERE	2,243	1,829,719	5,676,139	23,967,213	11,245,251
VERTEX	8,260	28,574,320	316,524,032	1,711,305,472	3,732,959,232

False positive points selected by CPLEX are caused by boundary nodes. On the other hand, in addition, SWEEP, SPHERE and VERTEX also select FPNs as they apply approximate intersection computation (Table 6.8). So, larger k' are returned, which cause significant performance degradation in higher dimensions. SPHERE processes similar number of iterations as SWEEP, and is even faster in generating ranges. However, the ranges

Table 6.6: Time cost (seconds) (first filter/second filter)

	2D	4D	6D	8D	10D
CPLEX#1	0.52/0.001	633.7/0.001	616.8/0.041	660/5.51	845.7/5,502
CPLEX#2	2.078/0.001	120.4/0.001	147.2/0.041	250.2/5.51	360.8/5,502
SWEEP	0.001/0.001	2.129/0.001	3.078/0.082	11.11/31.69	3.388/8,691
SPHERE	0.001/0.001	2.35/0.001	3.331/0.103	9.518/67.32	2.729/11,213
VERTEX	0.001/0.001	2.685/0.001	7.624/0.082	133.8/31.69	434.9/8,691

Table 6.7: Number of nodes selected by the first filter (inside/boundary)

	2D	4D	6D	8D	10D
CPLEX#1	323/191	335,290/664,711	78,778/921,227	137/999,941	1/1,000,199
CPLEX#2	0/4,357	0/1,000,003	0/1,000,020	0/1,000,078	0/1,000,200
SWEEP	323/191	333,082/666,930	75,139/924,880	9/1,000,003	1/1,000,074
SPHERE	341/209	367,127/632,884	53,789/946,230	0/1,000,016	0/1,000,210
VERTEX	323/191	333,082/666,930	75,139/924,880	9/1,000,003	1/1,000,074

Table 6.8: Number of true positive nodes (TPNs) and false positive nodes (FPNs) selected (TPN; FPN)

	2D	4D	6D	8D	10D
SWEEP	514/0	660,077/6,853	836,021/88,859	370,333/629,670	680,585/319,489
SPHERE	550/0	614,512/18,372	779,259/166,971	230,862/769,154	446,376/553,834

contain more false positive points, which undermines SPHERE's overall performance. Especially in 10D, SPHERE selects nearly the whole data set. VERTEX returns the same ranges as SWEEP, but takes much more iterations due to the multiple vertex-based distance computation. An odd pattern occurs that the iterations of SWEEP and SPHERE decline from 8D to 10D. This is because the simplex's boundary is very close to the boundaries of the data region in 10D. So, the false detection related to this half-space is constrained by the data region.

In general, the FPRs of all approaches increase drastically with increasing dimensionality. For one thing, this is because to keep the 0.1% selectivity, the simplex increasingly covers the data region as n grows, so that it intersects an increasing portion of nodes at each level. For another, r_{max} is set to a constant for all data sets, while each node is decomposed into 2^n children, thus limiting selected nodes to the larger ranges, and introducing more false positive points.

6.3.3. ND-PRISM QUERY

In theory, the number of half-spaces constituting the nD-polytope affects the time cost of range generation linearly (Section 6.2.3). This section uses a simple nD-prism model to verify this. Inscribed regular polygons of a circle are used as the base (Figure 6.11). To create the prism with $2f$ vertical faces, I apply a rotation formulation: suppose $\theta = \frac{\pi j}{f}$, where $j = -f+1, \ldots, f$. Then, I create each half-space with

$$\boldsymbol{\omega} = (\cos\theta, \sin\theta, 0, 0, \ldots, 0)$$

$$\beta = -\sqrt{\frac{selectivity}{\pi}} \cdot scale - \frac{scale}{2}(cos\theta + \sin\theta)$$

where scale determines the size of the prism — 4096 in this case. I do not create half-spaces at the ends of the prism, as they are implicitly defined by the data region. Based on this formulation, I generate $8i$-gonal ($i \in [1,8]$) prisms from 2D to 10D. The hyper-volumes of these $8i$-gonal prisms are close to each other, and they approximately equal the product of selectivity 0.1% and the hyper-volume of the data region.

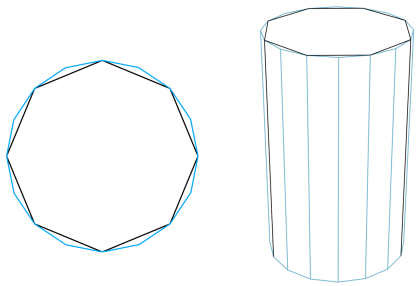


Figure 6.11: The nD 8-gonal and 16-gonal prisms projected to 2D and 3D spaces

The test uses the same uniform data sets from 2D to 10D. r_{max} is still 10^6. The approaches are the same as before. Table 6.9 presents the selectivity of different algorithms. Figures 6.12 – 6.16 show the number of iterations of different algorithms, while Figures 6.18 – 6.21 show the time cost.

Table 6.9: Selectivity of the first filters in the nD-prism test

	2D	4D	6D	8D	10D
CPLEX#1	1‰	1.857‰	13.39‰	71.29‰	247.9‰
CPLEX#2	1‰	1.932‰	13.39‰	71.29‰	247.9‰
SWEEP	1‰	1.857‰	13.39‰	71.29‰	247.9‰
SPHERE	1‰	1.938‰	15.33‰	71.29‰	749.4‰
VERTEX	1‰	1.857‰	13.39‰	71.29‰	247.9‰

Table 6.9 indicates that all algorithms share almost the same selectivity, except SPHERE deviate significantly from others in 10D. SWEEP returns no FPNs in this test, most likely due to the wide angle between two adjacent faces of the $8i$-gonal prisms. The figures indicate that for all solutions, the number of half-spaces influences the time cost of range computation linearly. CPLEX#2 holds the best scalability. It takes a constant number of iterations to compute ranges, and the number of half-spaces influences insignificantly on the time cost in each iteration. SWEEP and SPHERE hold the superiority over others

in time cost of range computation. However, because of more FPNs, SPHERE takes more iterations than SWEEP. This gap becomes larger with larger dimensionality. This gap becomes larger when the dimensionality goes high, as Figure 6.15 shows. In 10D, although iterations do not vary much, the FPNs from SPHERE is much larger than that of SWEEP (Table 6.9). This will greatly slow down the second filter.

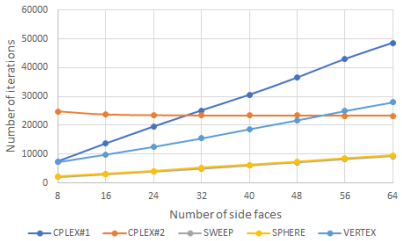


Figure 6.12: Number of iterations of range computation in the 2D-prism query

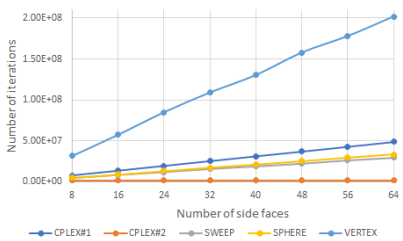
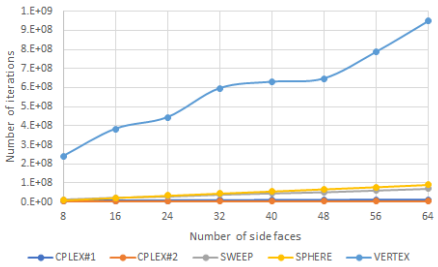
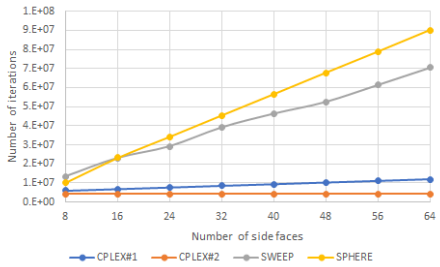


Figure 6.13: Number of iterations of range computation in the 4D-prism query

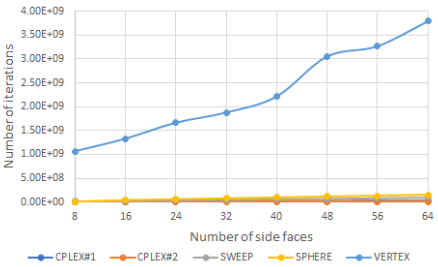


(a) Overall

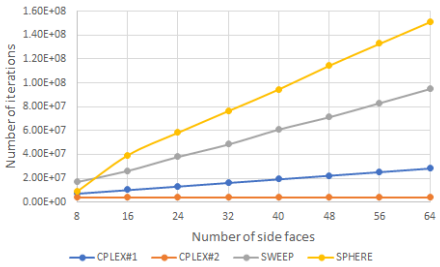


(b) Without VERTEX

Figure 6.14: Number of iterations of range computation in the 6D-prism query

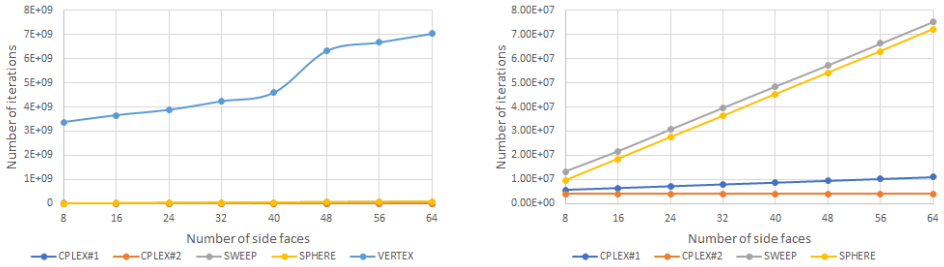


(a) Overall



(b) Without VERTEX

Figure 6.15: Number of iterations of range computation in the 8D-prism query



(a) Overall (b) Without VERTEX

Figure 6.16: Number of iterations of range computation in the 10D-prism query

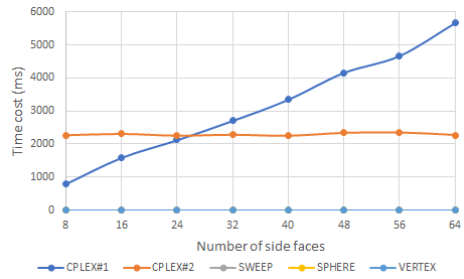
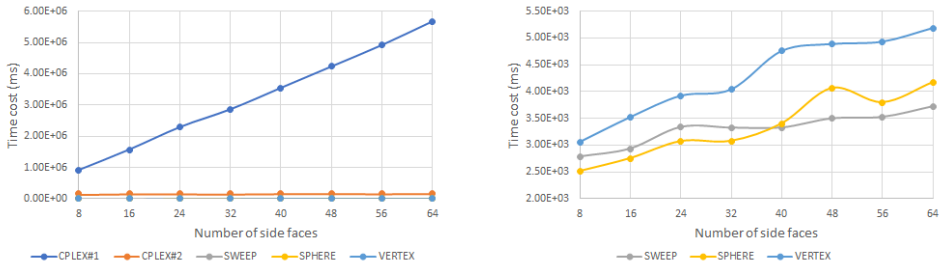
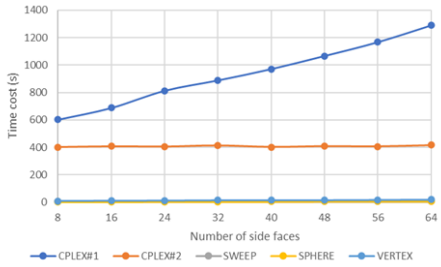


Figure 6.17: Time cost of range computation in the 2D-prism query

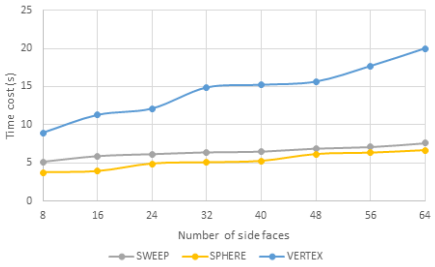


(a) Overall (b) Without CPLEX#1 and CPLEX#2

Figure 6.18: Time cost of range computation in the 4D-prism query

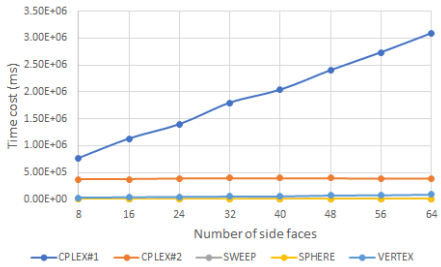


(a) Overall

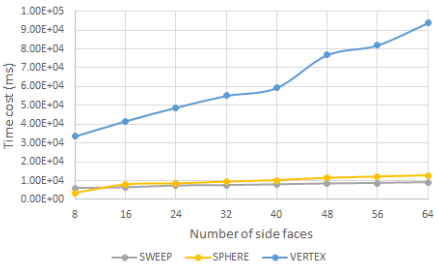


(b) Without CPLEX#1 and CPLEX#2

Figure 6.19: Time cost of range computation in the 6D-prism query

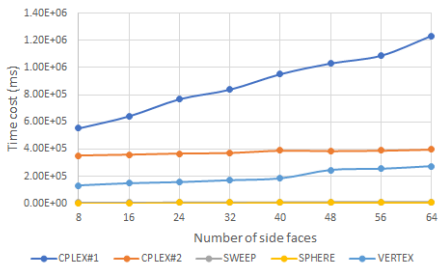


(a) Overall

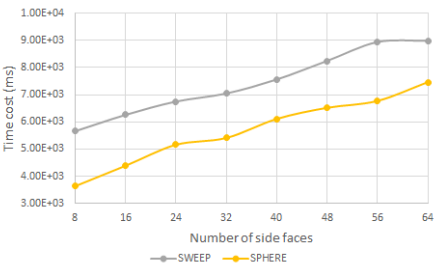


(b) Without CPLEX#1 and CPLEX#2

Figure 6.20: Time cost of range computation in the 8D-prism query



(a) Overall



(b) Without CPLEX#1, CPLEX#2 and VERTEX

Figure 6.21: Time cost of range computation in the 10D-prism query

6.4. REAL CASE STUDIES

After the ideal tests, I explored the applicability of the algorithms to real data, focusing on the time cost. One is the perspective view query, which is a basic operation to realize point clouds' visualization; the other is a flood risk query, based on modelling results.

6.4.1. PERSPECTIVE VIEW SELECTION

This section investigates perspective views on AHN2, by restating the query region as a convex polytope. Two kinds of perspective view, the close-up view and distant view, are used to verify the performance of different algorithms.

MODELLING 4D PERSPECTIVE VIEW

The data used is the 4D AHN2 sample, as has been described in Section 5.1. The computed cLoI values follow the exponential distribution: the smaller the cLoI value, the more important the point is. In the 4D perspective view selection, all points should be within a 3D view frustum in XYZ. Also, to imitate a realistic scene, a hyperplane in the cLoI dimension ensures nearby points are all be rendered, while fewer faraway points are selected. This corresponds to our cognition. So, I build the 4D view model as follows:

$$a_i x + b_i y + c_i z \leq d_i (i = 0, 1, \ldots, 4) \tag{6.4}$$

$$\sqrt{(x-u)^2 + (y-v)^2 + (z-w)^2} \leq D - \frac{D \cdot cLoI}{cLoI_{max}} \tag{6.5}$$

where Equation 6.4 describes the 3D view frustum, while Equation 6.5 defines the cLoI range. a_i, b_i, c_i and d_i are parameters based on the view point, direction and maximum view distance D. (u, v, w) represents the coordinates of the view point.

Based on this formulation, perspective views were selected using GEOM, SWEEP and CPLEX. GEOM detects intersection based on rigorous geometric computation. For example, Equation 6.5 defines a 4D cone. To determine whether a 4D node intersects it, GEOM first computes the boundaries of the two geometries: the boundaries of the 4D cone are actually 2 3D balls and 6 3D cones, while that of the 4D node are 8 3D cubes. Then, GEOM detects intersection of these boundaries at the 3D space. Using GEOM, no FPNs will be detected, but the computation is non-trivial.

SWEEP uses a polytope for selection. So, Equation 6.5 is approximated by linear equations. The left term in Equation 6.5 represents a sphere, and can be approximated by 15 half-spaces surrounding the sphere (Figure 6.22). Practically, the horizontal and vertical span of the vision could exceed 120° but cannot reach 180°. So, I rotate the central half-space by an interval of 30° both horizontally and vertically, to build the discrete approximation of the sphere.

CPLEX supports quadratic constraints and can be used without linearization. However, in practice, CPLEX collapses due to the high complexity of the quadratic problem (Equation 6.5). Consequently, CPLEX uses the 4D polytope model established in the SWEEP approach. Note that the CPLEX approach implemented here distinguishes inside nodes as does CPLEX#1 in Section 6.3.

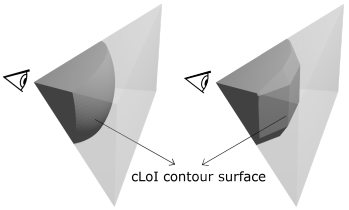


Figure 6.22: Approximating cLoI range by half-spaces: all points on the contour surface have the same maximum cLoI value

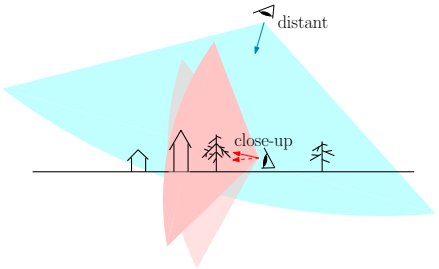
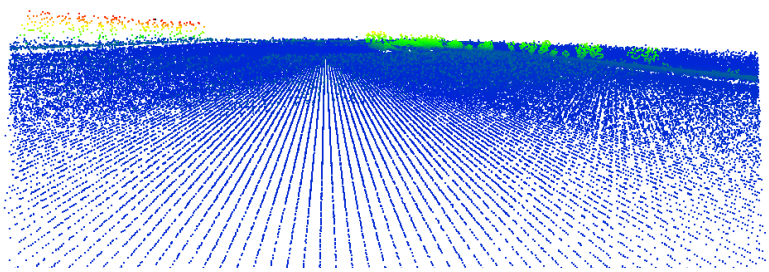


Figure 6.23: Illustration of a distant view and two close-up views

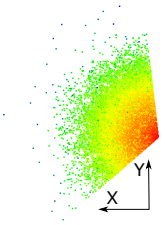
BENCHMARK TESTS

The test sample contains 2 billion points (data set 3 in Table 5.1). GEOM, SWEEP and CPLEX all set r_{max} to 10^5 for querying. The tests include two view modes (Figure 6.23): the close-up view simulates the situation where a user is standing looking around; the distant view imitates the bird-eye view located 800m to 1km high, looking at the ground. Each view test randomly generates 100 queries for benchmarking.

Figure 6.24 shows the visuals of a typical close-up query. Figure 6.25 presents the perspective view from another distant query covering the same region as Figure 6.24. Figure 6.26 presents the output size and FPR of both types of view selection based on all queries tested. Table 6.10 shows the average time cost of different approaches.



(a) Perspective view, colored by elevation



(b) Orthogonal projection, colored by maximum cLoI value

Figure 6.24: Results from a close-up query

As indicated in Figure 6.26b, SWEEP returns the most false positive points from the first filter. Although SWEEP selects FPNs, the second filter times are not greatly affected - the FPNs' negative effect is limited for this application, which is consistent with the nD-simplex experiment (Table 6.4). GEOM holds the lowest FPR thanks to the accurate intersection between the original geometry (Equation 6.5) and nodes. However, implementing is cumbersome, and can be impossible for other query geometries in high dimensional spaces.

Table 6.10 shows the superior performance of SWEEP, where the total time cost is below 1 s. In the first filter, range computation costs the most, while the others take constant time (< 0.2 s). Although CPLEX possesses a more accurate first filter than SWEEP,

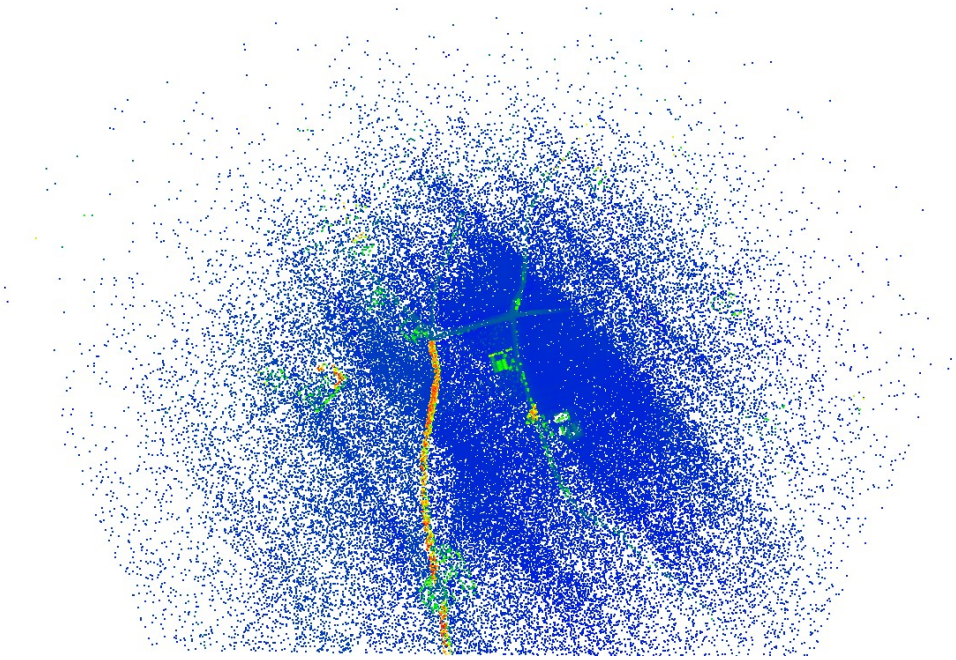


Figure 6.25: Perspective view resulting from a distant query

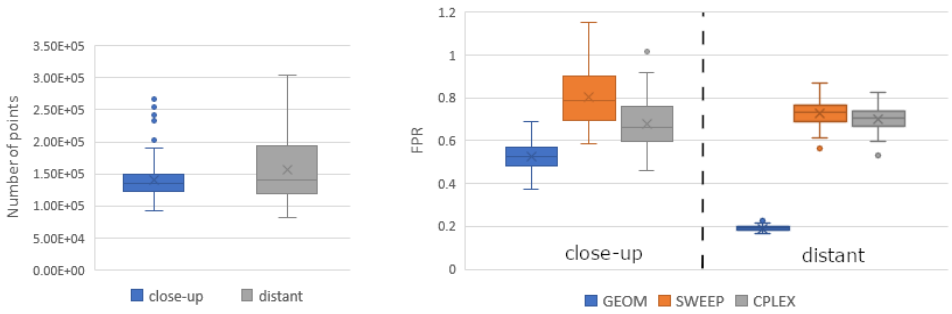
its range computation is 200× slower than SWEEP. This also agrees with previous results.

6.4.2. FLOOD RISK QUERY

This section presents another use case based on the flood data (Section 5.2). The data set totally contains 343,756,800 points, in an 8D space composed by case ID, X, Y, Z, time, depth, velocity and flow direction. The query is to select dangerous locations evaluated by human instability (i.e., depth × velocity ≥ 2) (Jonkman & Penning-Rowsell, 2008) in case 1.

Among the 8 dimensions, the flow direction is seldomly used for ad-hoc analysis. So, I set it as the property dimension when building PlainSFC, while used the other 7 dimensions for Morton key encoding. To employ SWEEP and CPLEX, I first converted the query into the polytope representation. It consists of a half-space indicating the case ID, and the other 7 half-spaces of which the points of tangency spread over the query boundary in the 2D projection (Figure 6.27). As a comparison, I also built a customized GEOM approach. GEOM reports an intersection if the upper-right corner of a node is in the original geometry. An inside is returned if the lower-left corner is inside. r_{max} is set to 10^5, the same as the AHN2 test. I also changed the case ID to 5, which leads to a different result. The exact answer of case 1 contains 28,351 points, while that of case 5 contains 175,758 points. Table 6.11 presents the accuracy of different first filters, and Table 6.12 presents the time cost.

The large FPRs in both queries (Table 6.11) result from the data set's high dimension-



(a) Size of accurate answer

(b) False positive rate of different approaches

Figure 6.26: Statistics of executing the 100 close-up and the other 100 distant queries

Table 6.10: Average time cost of the perspective view queries (seconds)

	GEOM	SWEEP	CPLEX
Close-up view query			
First filter	1.196	0.564	127.0
Second filter	0.239	0.275	0.259
Total	1.435	0.839	127.3
Distant view query			
First filter	0.99	0.571	165.1
Second filter	0.211	0.298	0.294
Total	1.201	0.869	165.4

ality and the skewed distribution of the depth and velocity dimension. Both dimensions contain large amount of zero values which are selected by these algorithms. In case 5, SWEEP and CPLEX select less points than GEOM, applying a different path for node decomposition. So, using the polytope for approximation may not always lead to negative result. Thanks to similar k', SWEEP costs nearly the same time as GEOM (Table 6.12), but CPLEX still takes more time by an order of magnitude. Additionally, as GEOM needs hard-coded programming, it is still less applicable than the generic SWEEP.

6.5. DISCUSSION

The advantage of nD-simplex model for query tests lies in the pseudo-randomness in terms of faces' directions. This results in diverse intersection angles between axis-parallel nodes and the simplex, which makes the result more generic and convincing. I achieved constant selectivity for both nD-simplex and nD-prism test, facilitating analysis of querying efficiency dependency on dimensionality and the number of half-spaces.

As shown in Tables 6.4 and 6.6, SWEEP becomes less competitive after 8D. FPNs occur at the acute corners where boundaries meet, and this occurs increasingly frequently in higher-dimensional simplexes. The nD-simplex is effectively a worst-case for the generation of FPNs, as SWEEP does not return any FPNs in the nD-prism test.

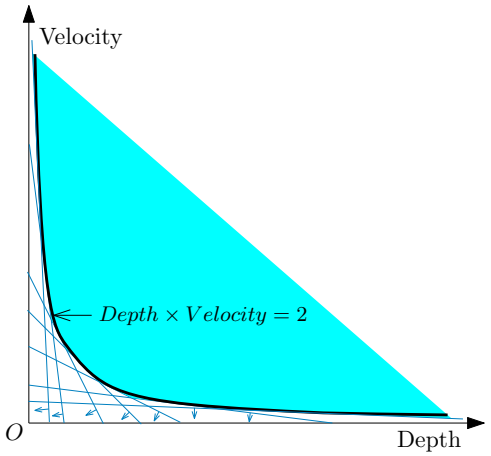


Figure 6.27: Converting the constraint on human instability into a polytope model

Table 6.11: Accuracy of the first filters in the flood query

	GEOM	SWEEP, CPLEX
	Case 1	
k'	11,734,557	12,031,206
FPR	412.9	423.4
selectivity	3.414%	3.5%
	Case 5	
k'	10,182,544	10,171,486
FPR	56.9	56.9
selectivity	2.962%	2.959%

Table 6.12: Time cost of the flood risk query (seconds)

	GEOM	SWEEP	CPLEX
	Case 1		
First filter	1.273	1.297	87.86
Second filter	14.99	15.08	15.08
Total	16.25	16.37	102.9
	Case 5		
First filter	1.336	1.474	98.46
Second filter	15.29	15.01	15.01
Total	16.62	16.48	113.5

The clipping method developed by Goldstein et al. (1997) on the other hand, clips the nodes intersecting each half-space. In this way, the intersection detection becomes a joint determination from all half-spaces, and FPNs are expected to be reduced. However, the clipping position should be computed optimally in high dimensional spaces. Otherwise, using the original method, several iterations of clipping has to be performed to detect accurately whether a node intersects the polytope, which costs significant amount of time.

The rigorous method CPLEX takes more time in each iteration, but it holds a more constant performance over dimensionality thanks to accurate intersection computation. So, CPLEX remains to be a competitive solution when dimensionality is high.

In general, all approaches suffers from “the curse of dimensionality" that the FPR becomes very large in high dimensional spaces. As different types of geometry (e.g., triangle, cube and sphere) develop differently as dimensionality increases (Figure 6.28), using hypercubes to approximate the polytope in high dimensional spaces may not be easy and can cause significant error. In fact, such approximating process is also a discretizing process, as the de-facto way to discretize in 2D is to use pixels. Consequently,

developing new methodologies for discretizing geometries in high dimensional spaces is imperative.

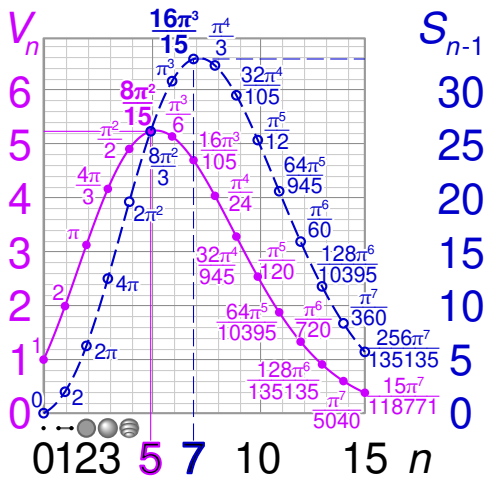


Figure 6.28: Volumes (V) and surface areas (S) of nD-balls of radius 1. Image source: Wikipedia

6

Additionally, the querying framework can solve more abstract queries whose constraints on combinations of different dimensions are expressible as a polytope model. The method for polytope modeling adopted in applications above can be generalized: given an equation of a convex curved face, $f(x_0, x_1, \ldots, x_{n-1}) = 0$, a generic method is to generate a set of tangent planes which together form a superset of the geometry for approximation. More specifically, we first randomly generate m points on f, from p_0 to p_{m-1}. Then, we compute the gradients at these points:

$$\nabla f(p_i) = \left(\frac{\partial f}{\partial x_0}(p_i), \frac{\partial f}{\partial x_1}(p_i), \ldots, \frac{\partial f}{\partial x_{n-1}}(p_i) \right)$$

where p_i is one of the random points. These gradients serve as normal vectors of a set of hyperplanes which are what we need. In practice, for a specific query geometry, the user can pick a small set of points of tangency, P, which spread over the geometry to build the polytope. By iteratively performing benchmark tests and densify P, optimal polytope models can be acquired according to Equation 2.1. Alternatively, users can apply a customized method if the query geometry can be modelled more efficiently.

7

EXTENDED APPLICATIONS AND PRELIMINARY RESULTS

In previous chapters, I mainly discussed the use of PlainSFC and HistSFC for two applications — AHN2 visualization and flood risk querying. In fact, the general principle of PlainSFC, nD-histogram and cLoI can be applied to address more applications. This chapter applies my nD-PointCloud framework to more data, platforms and applications, developing specific solutions. As the main goal is to demonstrate the applicability of the framework and explore the possibilities, the results presented are mainly from preliminary investigation and experiments. Nonetheless, these initial solutions can still be developed and optimized further.

In the following, 5 applications are presented. Section 7.1 realizes a bird eye shot function on AHN2. It differs from perspective view in the sense that the viewer adopts a straight downward viewing angle, and more efficient 3D geometric computation can be performed. Besides, visualization can be run on different platforms other than computers. Section 7.2 explores the possibility to render point clouds in an Augmented Reality (AR) environment on mobile devices. Also based on the mobile devices, Section 7.3 describes the idea to use the cLoI method to efficiently render point clouds for the purpose of indoor navigation. Section 7.4 explores the k-nearest-neighbor (kNN) search and change detection. This is another type of query besides the orthogonal window and polytope query. Section 7.5 applies PlainSFC to address queries on trajectory point clouds, which have very different nature from AHN2 and the flood data.

Despite tests on mobile devices, all experiments in this chapter are performed on a personal laptop — HP ZBOOK STUDIO G5. The processor is Intel(R) Core(TM) i7-8750H @ 2.2GHz. The RAM is 16GB, with a 500GB SSD.

7.1. BIRD EYE SHOT OF AHN2

Unlike the perspective view which looks ahead with a specific angle, the eye shot here refers to the straight downward view from above. That is, given a 2D region for viewing,

the bird eye shot imitates looking at the center of the region from above. This means that in the result, the point density reaches the highest in the center while gradually reduces towards the boundaries. I realize this by using the cLoI dimension, where all points are selected near the view point, while at distant, only important points are selected. The principle is similar to perspective view selection, as has been formulated in Equation 6.5. However, due to the viewing mode, the Z dimension of point data is not involved in the computation. This means 3D geometric operators can be directly used in XYcLoI space.

From the above analysis, two approaches can be used to acquire the bird eye shot: one is the APPROX which directly adopts the nD-polytope solution omitting the Z dimension; the other approach called EXACT is to conduct the XYcLoI query based on geometric computation. As Figure 7.1 shows, EXACT uses a 3D cone as the query geometry to realize bird eye shot: in the view center of the XY plane, all cLoI values should be selected, while the cLoI decreases further from the center, until the boundary of the XY plane which is defined by the 2D region for viewing. For computation, the first filter of EXACT examines whether nodes intersect the 3D cone to derive the corresponding Morton ranges. As each node can be casted to a 3D XYcLoI cube consisting of 6 faces parallel to axis, whether the intersection happens depends on the relationship between each face and the cone. In total, 5 relationships exist (Figure 7.2):

1. The face intersects the cone and at least one edge of the face intersects the cone
2. The face cuts the cone and crosses the bottom of the cone
3. The face cuts the cone and crosses the axis of the cone
4. The face is inside the cone
5. The face is outside the cone

In practice, not every face has to be checked. For example, if one face of the cube intersects the cone, the cube must intersect the cone. However, to determine whether a cube is fully inside or outside the cone, all faces of the cube have to be checked.

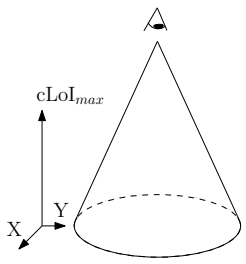


Figure 7.1: The 3D view cone to realize bird eye shot

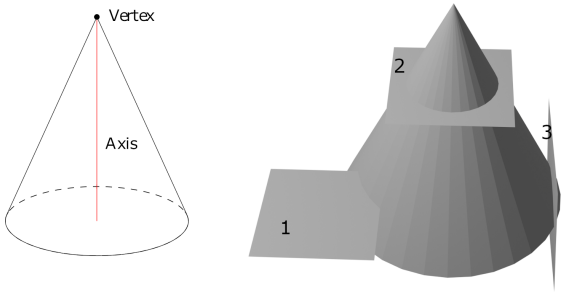


Figure 7.2: The first 3 relationships between a face and a cone

Compared to EXACT, APPROX needs an additional step to convert the 3D cone into a polytope model. This can return more false positive points due to the approximation. As the 3D geometric intersection functions used in EXACT is also convenient to develop, I

adopt EXACT, while still employ the 4D organization to allow other possible 4D queries. For each relationship shown above, I developed the mathematical functions [1] and integrated them into the first filter.

To demonstrate the functionality practically, I performed a test on an AHN2 sample containing 26,196,244 points. The spatial reference system is Amersfoort / RD New, EPSG:28992. By adding the cLoI dimension, the minimum bounding box is [19000, 369000, -0.83, 0; 19999.99, 369999.99, 25.68, 12000]. The 2D region for viewing is [19694.56, 369005.05; 19995.31, 369521.53] (Figure 7.3). Figure 7.4 shows the final result, where white points are those removed after the filtering. The density of points is determined by the distance to the view point, which keeps in line with our expectation. With proper settings (e.g., 100,000 ranges), the selection can be finished within 1 second using PlainSFC, on the laptop.

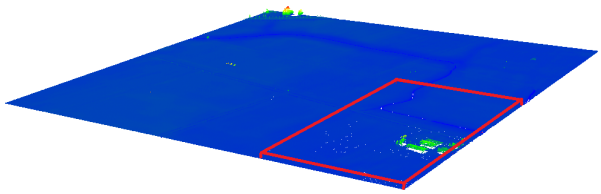


Figure 7.3: The AHN2 sample and the query region in red for testing

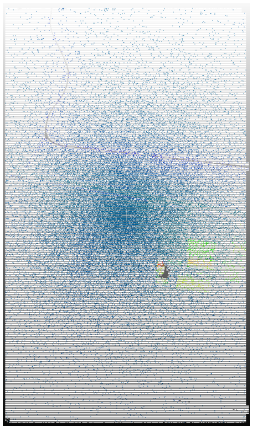


Figure 7.4: The result of the tested bird eye shot query

7.2. AHN2 IN AR

Previous applications presented in this thesis is realized on the computer platform. In fact, nD-PointCloud data can also be used to build scenes in the AR environment on mobile devices. However, because of the conflict between the large amount of data and the limited resources of mobile devices, most mobile AR applications based on point clouds lack the ability to handle large data sets and fail to interact with users fluently. To address this problem, Zhang (2020) conducted a master thesis to explore the use of cLoI to significantly reduce the number of points for rendering while preserve the visual quality. This was a collaboration with this PhD research.

To achieve the goal, a real-time cLoI method is firstly considered. It computes the distance between each point and the camera as well as space between points. However, such computation is very expensive, and can cause an extremely low frame rate and even the crash of the software. Consequently, the static cLoI method developed in

[1] https://github.com/rencailhc/SFCLib-New/blob/master/Histgen/Geom.h

Chapter 3 is used, where the cLoI value for each point is computed in prior. Then, we determine the level of cLoI to filter points according to a target data density for rendering (Section 3.4.3). For this, the Cumulative Density (CD) at a certain level is computed (Equation 7.1).

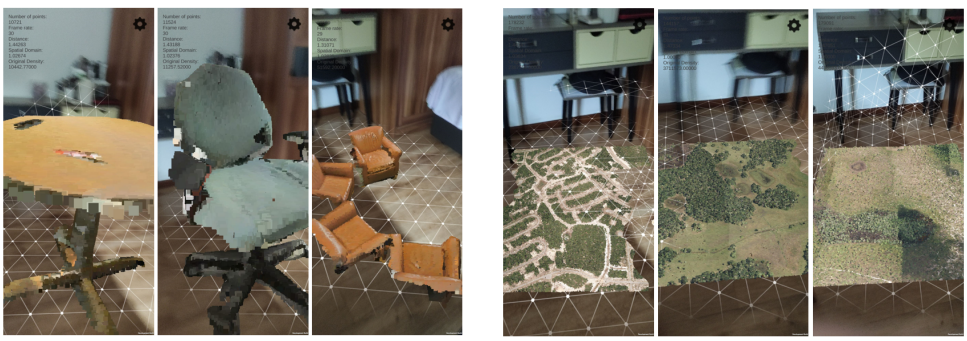
$$CD(l,n) = \frac{(2^{nl}-1)N}{(2^{n(L+1)}-1)\Omega} \tag{7.1}$$

where l refers to the continuous level that is needed; N is the total number of points; L is the maximal integer level that can hold such amount of data; Ω refers to the size of whole domain (if every dimension has an extent E, then $\Omega = E^n$); n is the dimensionality where it equals 2 for visualizing surfaces such as terrain, while 3 for true 3D objects.

After testing, we got an ideal density D, between 100,000 to 200,000 points/m^3. With this, we derive another expression of CD, Equation 7.2, which is established based on the distance from a point to the viewer. In the equation, x, y and z represents the center of a point cluster in the world space, while u, v and w refers to the position of camera in the world space. By joining Equation 7.1 and Equation 7.2, we can derive the particular level l (i.e., cLoI value) for filtering points. All selected points will be stored in a vertex buffer for rendering.

$$CD(l) = \frac{D}{\ln\sqrt{(x-u)^2+(y-v)^2+(z-w)^2}+1} \tag{7.2}$$

To practically realize this system, the hardware Redmi K20 Pro mobile phone running Android 10 is used, with configuration Qualcomm Snapdragon 855 processor at 2.84 GHz, 8 GB of Random Access Memory (RAM), 128 GB of main memory. Software used includes ARCore (version 1.17.0) together with the Unity game engine (version 2018.4.21). Basic interactions are developed including placement of a point cloud, rotation, scaling, zooming in and out. Besides, user interfaces with setting options are also added to the system: if the users are not satisfied with the automatic estimation, they can change the value of parameters, such as update frequency, desired density, and point size. Figure 7.5 shows the rendering results of different point clouds.



(a) Furniture point clouds

(b) Terrain point clouds

Figure 7.5: Rendering results by using the cLoI method (Zhang, 2020)

In addition, we also measured the performance by using the cLoI method. Figure 7.6 shows the proportion of reduced points against different input size. Since the scenes are scaled to a relatively small area, when the input contains more than 1 million points, this proportion can be very high (i.e., greater than 70% and can rise to 95%). In addition, Figure 7.7 indicates that the utilized memory increases when the input size grows. When visualizing large point clouds with more than 5 million points, the memory cost is more than 400 MB. It reaches 1,200 MB which is close to the boundary of the rendering system with the largest input. This is still too high for a mobile phone to run smoothly. Nonetheless, we still observed significant improvement using cLoI.

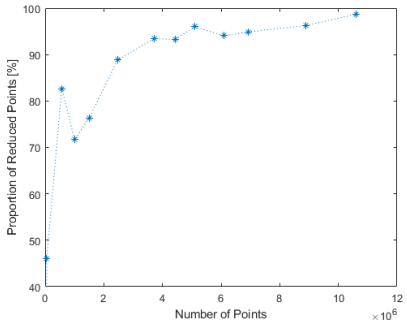


Figure 7.6: The proportion of points reduced (Zhang, 2020)

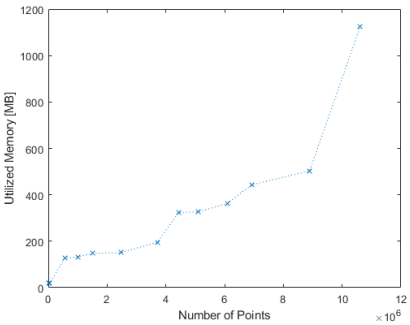


Figure 7.7: Memory consumption (Zhang, 2020)

7

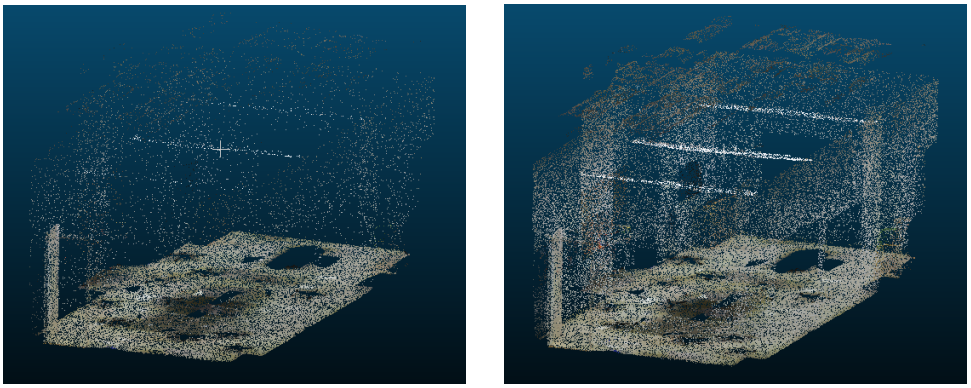
7.3. INDOOR VISUALIZATION FOR NAVIGATION PURPOSES

Indoor navigation becomes increasingly used nowadays. For one thing, people spend most of the time indoors. For another, indoor environments can be very complicated such as shopping malls and museums. The indoor navigation system is established in a 3D virtual environment, which can be directly built using the nD-PointCloud representation without being converted to mesh. The point cloud data can be conveniently collected by various mobile platforms (Thomson et al., 2013) and rendered by mobile devices. However, the cLoI method has to be adopted to guarantee a smooth rendering process, similar to the AR application. This section develops a simple cLoI method by considering semantic information that can facilitate the navigation.

As Brown et al. (2013) and Isikdag et al. (2013) indicate, classification of indoor objects (e.g, window, wall, door and stair) is essential for route planning in an intelligent indoor navigation system. Given a classified indoor point cloud where each object is also identified within each class, I build three categories to represent the importance level. The first category (i.e., the most important) includes stairs, floor, window and door. They are essential for navigation. The second class contains objects such as wall, ceiling, clutter which constitute the other part of indoor environment. The last category refers to sofa, bookcase, board, table and chair. They are movable objects which are less significant for navigation.

To integrate such information into the cLoI computation, I apply a simple weighting

technique. Basically, after each point is assigned a random cLoI value (Section 3.3), the cLoI values of the objects which belongs to the second category will be multiplied by 0.9, while for the third category a factor 0.8 is applied. These two weighting factors can be modified to achieve the best visual quality. In this way, large objects but less important objects such as wall and floor would not be represented by much more points than windows and doors, by choosing a same range of cLoI. Users can focus more on the key objects along the route. Figure 7.8 demonstrates an office by utilizing the semantic cLoI method. It clearly shows the different densities of objects resulting from the cLoI.



(a) 10% of all points

(b) 20% of all points

Figure 7.8: Visualizing an office by choosing different cLoI range for selection

Besides visualization, nD-PointCloud also facilitate other types of computation involved in navigation. As Liu, Li, et al. (2021) indicate, the range query which returns all objects within a certain distance of a query point concerns XYZ and ObjectID dimensions. Additionally, if a specific class of objects is needed, the classification dimension gets involved in the query as well. Using PlainSFC and HistSFC, these queries are expected to be accomplished efficiently. However, more comprehensive functionalities still need to be developed and evaluated thoroughly in the future.

7.4. KNN AND CHANGE DETECTION

Besides the orthogonal window and polytope query, my nD-PointCloud solution can address more types of queies. kNN, as a crucial algorithm in GIS, has also been widely used in point cloud applications. Key use cases include noise removal (Sankaranarayanan et al., 2006), registration (Elseberg et al., 2012) and change detection (Girardeau-Montaut et al., n.d.), for example. This section presents how to use PlainSFC or HistSFC to solve kNN, which is then used for change detection.

The formal definition of KNN search is: given a point p and a point cloud S, return a set of k points from S whose distance to p are the smallest. In fact, the nodes of HistogramTree can be used directly to facilitate the kNN query. Figure 7.9 shows the workflow of kNN query using HistSFC, where NNL is a key-value container using a binary tree structure. The key of NNL is distance d, while the value is the coordinates of the point.

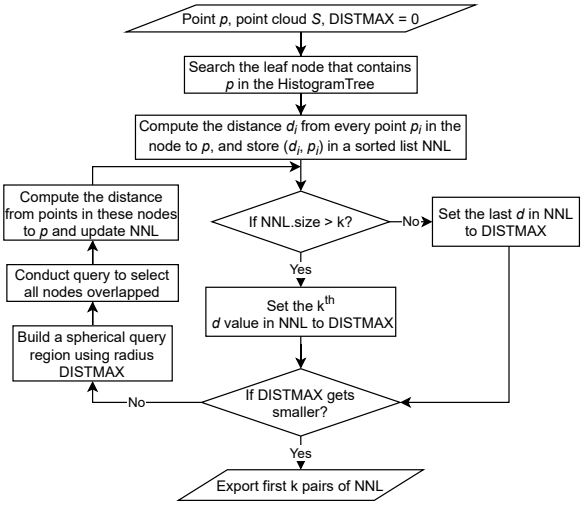


Figure 7.9: Execution path of kNN using HistSFC

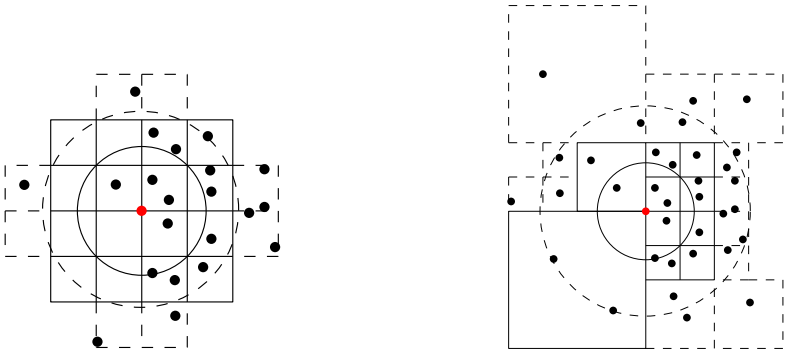
In the implementation, points examined in a new iteration exclude those that have been examined before, to decrease the cost. In the worst case, k HistogramTree nodes get involved in the searching. So, the time complexity is $\mathcal{O}(k \log N + kT)$, where N is the input number of points, and T is the capacity threshold of HistogramTree. Besides, the HistogramTree leaf nodes can still be refined to improve the performance, e.g., based on intersection ratio between a node and the searching hypersphere (Section 5.3.1). However, this is less convenient than PlainSFC which generates nodes of the same size at each level. For PlainSFC, kNN is realized by decomposing the nodes to a predefined depth. At that depth, intersection between nodes and the searching hypersphere is computed. In fact, using HistogramTree may not always be suggested. Figure 7.10, for instance, indicates clearly that the HistogramTree nodes cause drastic increase of the searching radius, which returns more false positive points. In practice, this also happens. For some 3D kNN query on the AHN2 sample, the False Positive Rate (FPR) of HistSFC can be several times larger than PlainSFC.

Figure 7.11 shows a kNN example computed using PlainSFC, where the AHN2 data sample is the one used in Section 7.1, a rectangle region containing 26,196,244 points. The query point is above the houses.

An crucial application of kNN is change detection between two point clouds. Various scenarios of change detection exist, which can be implemented in different ways to achieve the optimal performance. This is caused by different data and use cases. I list three common scenarios below:

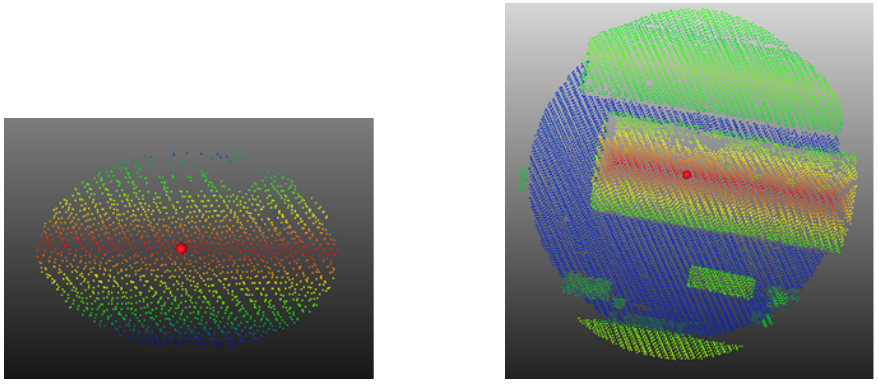
1. Large scale point clouds acquired at different times with nearly the same density are compared to derive changes, e.g., AHN2 and AHN3, which are different versions of dutch national Airborne Laser Scanning (ALS) data.

2. Two large scale point clouds collected by different sensors are compared. The den-



(a) PlainSFC (b) HistSFC, with node capacity equal to 2

Figure 7.10: 10-NN searching of a 2D point set. Solid circle and cells are involved in the first iteration, while dashed ones are involved in the second iteration.



(a) Nearest 2,000 points (b) Nearest 20,000 points

Figure 7.11: kNN query on a AHN2 sample (top-down view), colored by elevation

sities of the two point clouds are different. For instance, the ALS data is updated by Radar data to decrease the financial cost.

3. A large scale point cloud is used as the “base map”, while another regional point cloud is dynamically collected. The densities of these two point clouds can be different. For example, autonomous driving uses a High Definition (HD) map to detect local environment dynamically to make decisions.

Scenarios 1 and 2 do not need the support of a database solution, as they are only executed once. In contrast, Scenario 3 concerns dynamic queries, which caters to this research background. To realize this scenario, kNN is applied. Instead of a single point, here I focus on changes at the scale of patches. I perform change detection at the node level using PlainSFC. Figure 7.12 shows the workflow of 3D XYZ change detection based on 4D PlainSFC, i.e., XYZcLoI. In Figure 7.12, the cLoI dimension can be flipped to 0 by adopting “&" operation between 4D keys and a binary mask, e.g., 111011101110 . . . given

that the last bit of every 4 bits is used for encoding the cLoI value. Then, the processed keys are truncated to a certain level, depending on the threshold for change detection. Nodes represented by the truncated keys are compared to derive the changes.

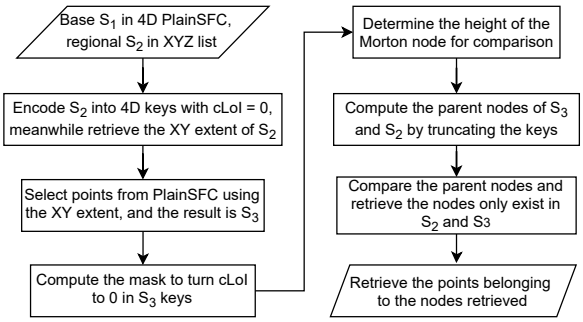
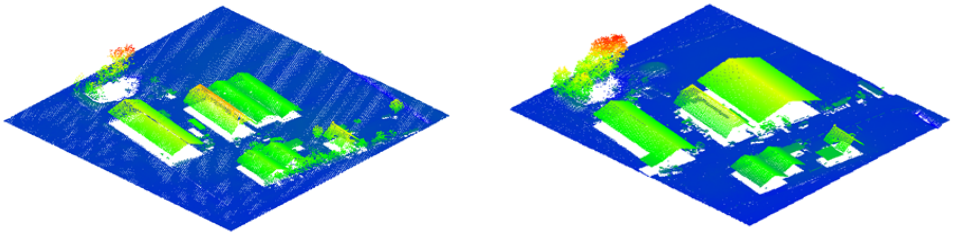


Figure 7.12: Execution path for change detection using XYZcLoI organization

As an experiment, I use AHN2 as the base map, while another piece of AHN3 as the dynamic point cloud. Figures 7.13 and 7.14 present the results of change detection using PlainSFC and Fast Library for Approximate Nearest Neighbors (FLANN). The latter is a widely used framework for change detection integrating different kNN algorithms and can automatically determine the optimal one to use depending on the input (Muja & Lowe, 2009).

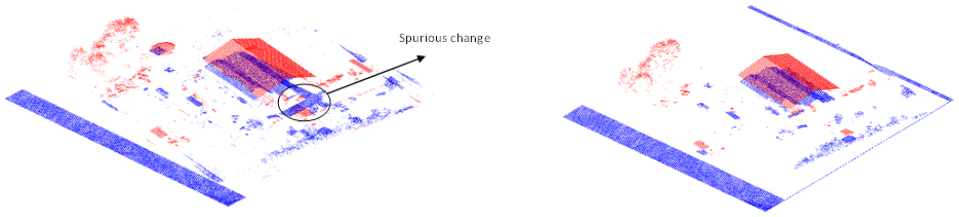


(a) AHN2 data

(b) AHN3 data

Figure 7.13: AHN2 and AHN3 samples used for change detection experiment

For PlainSFC, I set the level for node comparison using a tolerance of 5m for XY dimensions, while 0.5m for Z dimension. This can be achieved by scaling Z with factor 10 when encoding the key. In Figure 7.14a, the points in the circle belongs to the ground. So, they should not change in reality. However, due to slight shift of Z dimension in AHN3 (e.g., growth of grass), this part of points falls into different nodes in AHN2 Morton hierarchy and AHN3 Morton hierarchy (i.e., with different head part of Morton keys). So, the change detected is a spurious change, which is an error due to the shortcomings of the algorithm (Xu, 2015). This error does not occur using FLANN with a default setting (Figure 7.14b). Nonetheless, both solutions take similar amount of time to accomplish, i.e., around 2 seconds on the laptop.



(a) PlainSFC (b) FLANN

Figure 7.14: Changes detected using different solutions: blue points only exist in AHN2 while red points only exist in AHN3

To diminish the error of the spurious change in Figure 7.14a, I add another point-wise filter in PlainSFC. Such a filter specifically searches the nearest neighbor for single points in the previous result. The distance between the nearest neighbor and the point is then compared to the threshold, to determine whether an actual change happens. Figure 7.15 presents the updated result. It indicates that the spuriously detected points are largely removed. However, As can be deduced, such additional filtering process can be very expensive. In this experiment, the time cost of PlainSFC steps from 2 seconds to 15 seconds.

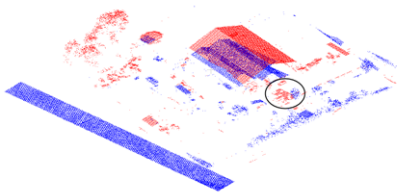


Figure 7.15: Change detection using PlainSFC with an additional nearest neighbor filter

In current algorithms, cLoI is omitted, but it can be utilized to decrease the computation. Basically, given a specific cLoI level which also indicates a sampling rate, any large objects that contain sufficient points will be sampled at that level. Then, the changes happening to these objects are most likely to be detected by comparing the point clouds that have been filtered by the cLoI. In this way, computation can be greatly reduced, but changes of smaller objects may be omitted. So, the use of cLoI also depends on the requirements on resolution of change detection. This indicates a new research direction on how the level of cLoI influences the accuracy and efficiency of change detection. For this, novel cLoI methods may be devised in the future.

7.5. AIS TRACKING

The Automatic Identification System (AIS) is developed and used worldwide to avoid the frequent occurrence of maritime accidents. Basically, the vessels are equipped with

Global Navigation Satellite System (GNSS) receiver and AIS transponder. Depending on the cruising speed of the vessel, such a transponder broadcasts its identity and position to the base station in intervals ranging from 2 seconds to 3 minutes. This results in another type of nD-PointCloud data which are the trajectories of moving vessels. This section summarizes the results of a master thesis research of Li (2020) which applies PlainSFC to address AIS queries. This is another collaboration within the PhD research.

The AIS data contains many fields (Figure 7.16), including static part such as vessel name and type, and dynamic part such as location, speed and voyage related information. Because of the massive and useful information, a large number of applications have been developed based on AIS data. For example, a critical application for maritime safety is to analyze the trajectories of specific vessels by using the longitude, latitude, time, and the Maritime Mobile Service Identify (MMSI) which identifies a vessel. Anomalies of motions can thus be detected. In the following experiment, we focus on two types of query which are frequently used. They are spatio-temporal window query which reports all vessels' locations within an area and time period, and the trajectory query which retrieves the positions of a specific vessel in a period of time. As can be seen, this experiment only concerns MMSI, longitude, latitude and time dimensions.

mmsi	ts	imo	callsign	shipname	shiptype	bow	stern	port	starboard
244270394	2016-12-10 00:00:00.148 UTC	0	PE7278	STERN	69	4	7	1	1
244670835	2016-12-10 00:00:00.861 UTC	0	PD3343	MARCHIENA	89	85	0	4	4
305648000	2016-12-10 00:00:01.184 UTC	9504047	V2FE2	PHOENIX J	71	136	15	11	12
235192000	2016-12-10 00:00:01.216 UTC	9143506	ZNBJ6	HAPPY BEE	80	94	20	12	4
244700938	2016-12-10 00:00:01.443 UTC	0	PD7425	PL	99	13	15	3	4
235076272	2016-12-10 00:00:01.000 UTC	9398723	2CWA7	PARAMOUNT HANOVER	80	207	43	28	16
229630000	2016-12-09 23:59:59.194 UTC	9365960	9HA3465	X-PRESS MULHACEN	74	133	9	1	9

(a) Static part

month	day	hour	minute	draught	destination
0	0	24	60	0.7	HEEN EN TERUG
1	8	21	25	1	-
12	10	0	0	7.1	ROTTERDAM
12	7	20	20	4.7	ZEEBRUGEE
0	0	24	60	1.8	SCHIEDAM
12	8	4	0	11.3	TEXAS CITY
12	9	20	30	7.7	CARTAGENA

(b) Dynamic part corresponding to the same vessels

Figure 7.16: A sample of decoded AIS data

Unlike the Oracle version of PlainSFC implemented before, in this experiment, PostgreSQL and Python are used. This is because PostgreSQL's PL/PYTHON extension can directly run the Python script inside the database. As a result, additional communication between the database and PlainSFC's components are avoided. The code in Listing 7.1 builds PlainSFC solution in PostgreSQL. For querying, we create PL functions in a similar way so that the first filter and second filter of PlainSFC are all accomplished inside the database (Li, 2020).

Listing 7.1: PlainSFC implementation for AIS in PostgreSQL

```
---first create the staging table, and load data into it
CREATE TABLE ais_flat (mmsi INT, ts TIMESTAMP WITHOUT TIME ZONE, long REAL,
    lat REAL)

COPY ais_flat FROM '/HOME/ais_data.csv' DELIMITERS ',' CSV

---create the Monton encoding function, mortonencode, by calling the Python
    script test_sfc.py
CREATE FUNCTION mortonencode (ARR BIGINT []) AS $$
FROM test_sfc IMPORT mortonencode
RETURN mortonencode(ARR)
$$ LANGUAGE PLPYTHON3U

---create the table to store 4D Morton keys composed by mmsi, time,
    longitude, latitude
CREATE TABLE ais_iot as SELECT mortonencode(mmsi, extract(epoch from ts),
    long*1000, lat*1000) as sfc FROM ais_flat;

---create B-tree index on the Morton keys
CREATE INDEX ais_idx ON ais_iot (sfc)

---cluster the ais_iot table to improve querying efficiency
CLUSTER ais_iot USING ais_idx
```

The experimental data set contains in total 11,389,217 points. Figure 7.17 shows the point data and the 4 spatio-temporal query windows for testing, namely, TinyRec, SmallRec, MediumRec and LargeRec. The time interval is 2 minutes for all windows. The number of points inside them are 39, 90, 1787 and 2846, respectively. As to the trajectory query, it retrieves the historical positions of a vessel whose MMSI is 244670079 in 1 hour. The final answer contains 45 points. All tests are performed on the laptop. Tables 7.1 and 7.2, and Figure 7.18 show the querying performance. Instead of setting a maximum number of ranges for querying, the experiment adopts different search depth of the Morton hierarchy for querying.

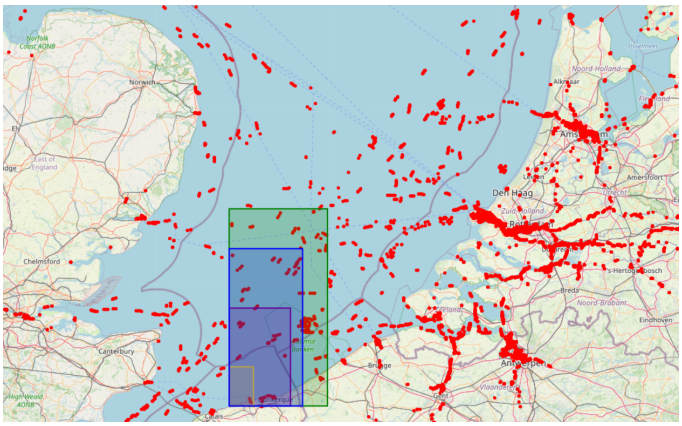


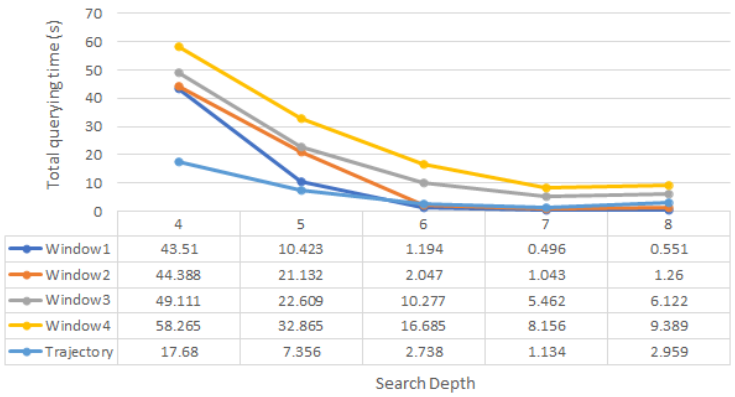
Figure 7.17: AIS point data and spatio-temporal query windows (Li, 2020)

Table 7.1: Selectiveness of the 4 spatio-temporal window queries and the trajectory query, recalling $selectiveness = \frac{\text{Number of points within the query range}}{\text{Total number of points of input}}$

Depth	TinyRec	SmallRec	MediumRec	LargeRec	Trajectory
4	0.1009	0.0494	0.5747	0.1696	0.2391
5	0.0259	0.0047	0.0256	0.0787	0.0960
6	0.0026	0.0017	0.0122	0.0379	0.0463
7	0.0007	0.0004	0.0057	0.0184	0.0221
8	0.0002	0.0002	0.0028	0.0089	0.0109

Table 7.2: False positive rate of the 4 spatio-temporal window queries and the trajectory query

Depth	TinyRec	SmallRec	MediumRec	LargeRec	Trajectory
4	29,460.03	6,249.00	3,662.00	677.56	60,515.42
5	7,576.21	589.32	162.45	313.94	24,293.93
6	744.54	214.52	76.80	150.78	11,718.16
7	196.41	43.80	35.55	72.46	5,588.98
8	58.85	20.32	16.88	34.74	2,758.87



	4	5	6	7	8
Window1	43.51	10.423	1.194	0.496	0.551
Window2	44.388	21.132	2.047	1.043	1.26
Window3	49.111	22.609	10.277	5.462	6.122
Window4	58.265	32.865	16.685	8.156	9.389
Trajectory	17.68	7.356	2.738	1.134	2.959

Figure 7.18: Querying time of AIS queries with different search depth

Tables 7.2 shows that with the increase of search depth, the false positive rate decreases monotonically. This has been expected, as a large search depth corresponds to finer refined nodes to approximate the query window. However, as Figure 7.18 shows, from depth 7 to 8, time cost increases. This is because before level 7, the second filter dominates the time cost as not too many ranges are generated. However, after this, the first filter spends so much time to compute ranges that the increased time cost exceeds the reduced time cost of the second filter. As a result, we see the turning point. Nonetheless, in most cases, the total time cost at depth 8 is not significantly greater than that of depth 7. That is, the optimal search depth may not always be needed, and a range of optimal depths can be accepted. This keeps in line with the prior result that the number of ranges should be controlled within a certain range.

This experiment verifies the fact that the nD-PointCloud solution can indeed be applied to process the trajectory data, although the query types tested are limited. Compared to the common vector representation, the latter stores full trajectories, and provides more possibilities for spatial computation such as intersection of two trajectories. nD-PointCloud solutions may hardly achieve this currently, due to temporal resolution of point data. Nonetheless, we may develop certain interpolation techniques which add essential fake in-between points to facilitate such intersection computation, for instance. Although it is still possible that the nD-PointCloud solution may be less optimal, it can still be considered as a significant supplement on the whole.

8

CONCLUSIONS AND FUTURE WORK

AFTER describing the core technology and applications of nD-PointCloud which I developed, this chapter concludes the thesis. Section 8.1 summarizes the main findings and knowledge acquired by answering the research questions. Section 8.2 lists the main scientific contributions. As this PhD research still has limitations and new thoughts frequently arise during this process, Section 8.3 describes potential research directions in the future, to continue paving the way to the new nD-PointCloud field.

8.1. CONCLUSIONS

With the development of remote sensing technology and advanced applications such as Virtual Reality (VR) and catastrophe modelling, nD-PointCloud has become the third spatial data representation in addition to vector and raster. nD-PointCloud can be directly collected, managed and analyzed without converting to other representations. This thesis focuses on nD-PointCloud data management and querying, by considering a great variety of applications such as LiDAR data visualization and flood risk querying. The solution proposed is based on the nD-PointCloud data structure —— PlainSFC. The basic idea of PlainSFC is to treat every type of information of a point cloud as a dimension. For data storage, PlainSFC only uses the organizing dimensions to sort and index the data, with property dimensions attached. PlainSFC is convenient to implement, and can be built in any DataBase Management System (DBMS) that supports the B+-tree structure. Based on PlainSFC, this thesis develops innovations in conceptual design, query optimization and nD functionalities for point clouds. The majority of the code used by the research is provided in https://github.com/rencailhc/HistSFC.

In the following, I summarize and discuss the research results by answering sub research questions and the main research question.

1. To what extent can the transformation between the organizing dimensions and the property dimensions facilitate the management and query of point clouds? How to determine the type of dimensions when managing the data?

If a property dimension D is involved in a query, PlainSFC cannot process it effectively in the first filter which only concerns organizing dimensions. Thus, by default, the

first filter selects the whole range of D and compute the corresponding Morton ranges. The second filter conducts the actual filtering of D. If the query is not selective on D, then, the first filter will not make significant errors. To this end, treating D as a property dimension is acceptable. However, if the query is very selective on D, then, the errors made by the first filter will be greatly attributed to D. In this case, converting D to an organizing dimension can facilitate the querying. However, such a conversion will also influence other queries which do not concern D. This is because with one more dimension involved in the key, the fanout of the Morton hierarchy will become 2× more. To achieve the same accuracy on these queries, the number of ranges for querying should also be raised. This will in turn increase the memory cost and slow down the querying. Hence, it is also crucial to consider the frequency of D appeared in queries to determine the optimal data organization, apart from the selectivity.

Also note that the description is qualitative here, i.e., selectivity and querying frequency, while the actual performance depends on the real use cases. A rule of thumb is to first benchmark solutions using a sample of the data set to derive initial performance patterns. Then, by iteratively experimenting and benchmarking with more data, the final decision can be made. Considering GIS applications, in most cases, XY should be organizing dimensions. Z, time, continuous Level of Importance (cLoI) and classification can be essential to certain applications. So, they should be considered seriously to be used as the organizing dimensions. In general, the number of organizing dimensions should be controlled as few as possible to guarantee efficient range computation. Also note that the knowledge described above also applies to block based solutions.

2. What is the role of cLoI in managing nD point clouds? How to compute the cLoI value for each point?

LoI is a critical technique that can be used to balance computation accuracy and efficiency: at higher levels, only the most representative points get involved in the computation, which can significantly improve the efficiency but with low accuracy; on the other hand, when a large number of less important points get involved, the computation will take much more time but the result becomes more accurate. This principle applies to both discrete LoI (dLoI) and cLoI. However, conventional dLoI method such as Quadtree or Octree causes density shocks in visualization. So, a novel cLoI method was devised in Chapter 3 to eliminate this effect where each point represents a level and can be rendered in a gradual manner. Additionally, although I mainly demonstrate the benefit of cLoI in visualization, there must exist other cases that suffer from the discrete levels due to the continuous nature of the world. To use cLoI in a same manner as dLoI, cLoI has to be explicitly computed for each point and used as an organizing dimension. In this way, the representative points can be efficiently retrieved, since the first filter can function effectively.

Computing cLoI in real time is costly when the application is based on large input data. So, cLoI should be computed in advance and used in data organization. Chapter 3 proposed a method to convert the distribution of the number of points at different levels of the 2^n-tree to a continuous form. In this way, the final cLoI values follow an exponential distribution. Then, the method assigns these values randomly to each point. The total time complexity is $\mathcal{O}(N)$, where N is input number of points. The cLoI can be further customized. In certain applications such as indoor navigation (Section 7.3), the

classification can be considered when computing the cLoI. This is because objects such as stairs, doors and windows are more crucial than ceilings and chairs for navigation.

3. How much does the point distribution influence the performance of query execution?

PlainSFC generates Morton ranges for the query geometry without considering point distribution. When data is non-uniformly distributed, the ranges generated will most likely contain a large number of empty ranges. These empty ranges take time to compute and query but contributes nothing to the final result. Consequently, to improve the efficiency, the nD-histogram technique, HistogramTree, is developed to guide the computation of ranges (Chapter 4). HistogramTree records the number of points inside each Morton node in the hierarchy, as a representation of data distribution. To quantify the effectiveness of HistogramTree, a metric, Cumulative Hypercubic Coverage (CHC), is proposed. CHC measures the uniformity of a point cloud. By definition, the smaller the CHC is, the more skewed the distribution is. Then, I established theory which indicates that the effectiveness of HistogramTree increases monotonically as CHC decreases, given randomly distributed queries.

The influence of point distribution on querying is also evaluated in practice (Chapter 5). In AHN2 test, PlainSFC presents high efficiency, slightly lagging behind HistSFC which utilizes HistogramTree to compute ranges. This is mainly caused by the relatively high uniformity of the data. Although Z and cLoI follows non-uniform distributions, the CHC of the 4D data set is still around 0.01. However, in the flood data test where CHC of the 8D data store equals 0.0007, the performance of HistSFC is much better than PlainSFC. Hence, depending on the data, point distribution can significantly influence the querying efficiency. In addition, it should be realized that the data distribution is not the only factor causing empty ranges. Converting skewed dimensions to more uniform distributions may not improve efficiency as the query windows are also transformed. In other words, distribution of the data and queries should be considered as a whole, to determine the optimal settings of the solution.

4. Besides the common orthogonal window queries on different organizing dimensions, what other query geometries are needed and can be efficiently handled by the data structure?

In principle, PlainSFC and HistSFC can resolve any query geometry that can be mathematically expressed (e.g., in boundary representation or equations). This is because the first filter can always derive the ranges by intersecting Morton nodes with the query geometry by using specific intersection functions. The drawback is that for every distinct type of geometry, the intersection function should be developed. As a general solution to this issue, Chapter 6 developed a convex polytope querying method. Basically, the mathematical definition of a polytope consisting of a set of half-spaces is used. Then, a generic intersection function which examines whether a node intersects these half-spaces is developed. In this way, the first filter can process a polytope query very efficiently. In fact, all query geometries can be approximated by the polytope model to be solved, e.g., using random hyperplanes of tangency (Section 6.5). However, this may aggravate the filtering process of the second filter due to the extended selection resulting

from the approximation. So, the equilibrium between the number of half-spaces to approximate the query geometry and the efficiency of the second filter has to be decided by experimenting.

5. What queries and processes should be included in a benchmark to learn the balanced performance of the data structure? How to set up a representative benchmark?

Chapter 5 conducted benchmark tests based on two representative use cases, AHN2 exploration and flood risk querying. The data sets involved have distinct characters: one is a 4D ALS data set, while the other is a point cloud converted from an 8D modeling result. The AHN2 queries not only include conventional spatio-temporal window queries with cLoI involved, but also perspective view queries which are commonly used in visualization. In the query design, real query logs are used to derive realistic settings for randomizing query sizes and locations. As to the flood data benchmark, it integrates typical queries that support flood risk mapping, e.g., maximum inundation depth map. Specific queries for risk analysis, such as risky point retrieval along a road and human instability computation, are also included. To further evaluate the performance of PlainSFC and HistSFC, other state-of-the-art solutions including PostGIS, Oracle SDO_PC and Pyramid-technique are implemented and compared with in the benchmark test. Moreover, I optimized HistSFC by improving range refinement, uniforming skewed dimensions and decoding in parallel. The effect of these optimizations were also tested.

Querying time cost is the major indicator of efficiency, and specific time cost on different querying processes is also recorded such as range computing and database fetching. The memory cost is mainly influenced by the number of ranges used for querying, which can be specified. On the whole, the benchmark test performed confirmed the higher efficiency of HistSFC than other solutions. By considering extended applications, Chapter 7 reveals more comprehensive benchmarks including trajectory data and indoor point cloud data, augmented reality platform, and more functionalities such as k-Nearest-Neighbor (kNN) search.

To summarize, the crucial aspects for devising a comprehensive benchmark include data, queries (functionalities), different solutions, settings, optimizations, and platforms. In fact, I also noticed that the other database processes such as data loading and index building are also important to study. However, these processes are excluded, as the research focuses on the performance of querying which is directly related to applications.

What is a highly efficient nD point cloud data structure supporting different types of applications?

Different algorithms need specific data structures to achieve an optimal performance. This is also true for nD-PointCloud applications. However, to avoid excessive development and integration, this thesis devises a generic data structure that can be implemented in most DBMSs for different applications. This provides a convenient and efficient tool for users to explore nD-PointCloud data. The basis of the novel data structure is PlainSFC, which adopts succinct and efficient Space Filling Curve (SFC) clustering and B+-tree indexing strategies. The main querying mechanism is to iteratively examine intersection between Morton nodes and the query geometry to derive ranges for selection. Besides, I improved the solution from the following aspects:

- I developed HistogramTree for inhomogeneously distributed data, and used the corresponding HistSFC solution to generate ranges with higher accuracy;
- I optimized HistSFC further by smartly refining ranges below the bottom level of HistogramTree, uniforming skewed dimensions and decoding in parallel;
- I devised and used the cLoI dimension to facilitate massive point clouds visualization, with density shocks eliminated.
- I developed the querying algorithms for convex polytopes, which can also be used to address other irregular query geometries.

The performance of the data structure is evaluated by comprehensive benchmark tests with different use cases. By comparing PlainSFC and HistSFC to the state-of-the-art solutions, I assert that HistSFC is the most efficient solution.

8.2. SCIENTIFIC CONTRIBUTIONS

The scientific contributions of the PhD research are summarized as follows:

1. Developed a novel cLoI method for point clouds, which is integrated with a new data structure to facilitate related applications.
2. Developed and adopted HistogramTree to address queries on inhomogeneously distributed data.
3. Proposed a novel metric CHC to quantify the uniformity of a point cloud and also to indicate the effectiveness of HistogramTree. The metric is also convenient to compute in practice.
4. Developed different optimization techniques, which function positively and significantly improve the querying performance in certain cases. These include refining ranges below the bottom level of HistogramTree, uniforming skewed dimensions and decoding in parallel.
5. Converted flood modelling results to nD-PointCloud representation and used novel HistSFC for querying, which achieved high efficiency in flood risk queries.
6. Conducted comprehensive benchmark tests using different queries, and compared the newly developed solution to state-of-the-art solutions.
7. Proposed an easy-to-use convex polytope formulation for querying.
8. Developed efficient intersection algorithms for convex polytope querying on nD point clouds.
9. Efficiently solved perspective view selection for smooth rendering and flood risk queries on 8D modelling results using the polytope method.
10. Extended and verified the whole framework against different data, platforms and applications.

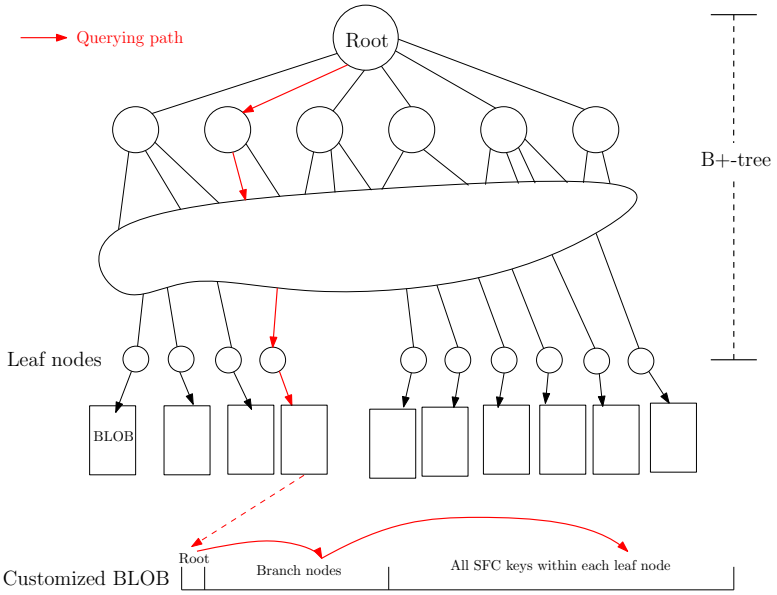


Figure 8.1: Block based storage: each BLOB utilizes a B+-tree to organize data

8.3. FUTURE WORK

Developing nD-PointCloud is not my own task, but should be a joint effort of all related researchers and communities. This section presents potential research topics in the future: Subsections 8.3.1 – 8.3.3 concern the further development of core technology of the nD-PointCloud solution, covering data structures, applications and models. Subsection 8.3.4 and 8.3.5 propose to establish related standards and protocols to promote the use of nD-PointCloud.

8

8.3.1. BLOCK BASED DATA STRUCTURE

In practice, the input data set can be much larger than the ones that have been tested, e.g., the whole AHN2 data with 640 billion points. Besides, the point cloud may also contain more organizing dimensions. These will result in very large SFC keys which exceed the limit of Oralce's NUMBER type. Using strings is an option, but this can take much more space and the computation is less efficient than the number based representations. Another alternative is to use two NUMBER fields for each SFC key, one for the head part of the key while the other for the tail. However, this approach can still take huge amount of storage space. A major reason is that the first NUMBER field would be redundantly stored for many points that share a same head part of the key. To avoid this, I propose a customized block based approach using Binary Large OBject (BLOB), as illustrated in Figure 8.1.

Basically, a B+-tree is used to organize and store the first NUMBER field, i.e., head part of the key. In additional to the first NUMBER field, the leaf node also stores a pointer to a customized BLOB containing all its children whose keys are actually the

second NUMBER fields. In this way, redundant storage can be avoided. The customized BLOB is a B+-tree organized object so that an internal indexing mechanism can be realized. Therefore, unlike conventional block based solutions, this approach will not need to unpack the whole BLOB to filter the data. This significantly decreases the I/O cost. As a consequence, for one thing, the advantage of blocks such as transmission via network can be gained; for another, the highly efficient clustering and indexing mechanisms are reserved.

8.3.2. FUNDAMENTAL TECHNOLOGY FOR APPLICATIONS

Chapter 7 described some fundamental techniques used in various applications, such as kNN for change detection and semantic cLoI for indoor navigation. This section indicates two additional examples – point data caching for VR and mixed representation for flood risk analysis.

For VR applications based on point clouds, since the memory of the headset is limited and real-time rendering is required, fast data selecting algorithms should be specifically developed. Besides the crucial cLoI technique, advanced caching algorithm which facilitates the selection and rendering of point data is also indispensable. To this end, an algorithm which incorporates motion prediction based on current viewpoint, motion direction and pace and caches surrounding data needs to be devised.

The thesis presented the use case of flood risk querying, while a sound decision support system still needs to be developed to really serve the citizens. This concerns a frontend module which allows users to interact with the modelling results. Representing flood modelling results by nD-PointCloud improves the efficiency for querying. However, the nD-PointCloud representation may not be suitable for interaction: for visualizing the water body, meshes may achieve a better result. Then, a mixed representation may be adopted, and strategies to realize this should be investigated. For example, implement a dual storage or construct meshes based on points in real time.

8.3.3. INTEGRATION WITH CONCEPTUAL MODELS

Conceptual models provide a standard way to support vast range of applications. For instance, WaterML is devised for exchange of in-situ hydrological observations (Almoradie et al., 2013), while indoorGML is established for indoor localization and navigation purposes (Kang & Li, 2017). In fact, Poux et al. (2017) have proposed a Smart Point Cloud (SPC) model for managing, processing and using point cloud data. It is a 3-layer model (Figure 8.2) consisting of a bottom data description layer (level-0), a middle connection layer (level-1) and an upper domain adaption layer for applications (level-2).

A potential research topic is to integrate PlainSFC/HistSFC with SmartPC model, where possible changes have to be made. For example, as point clouds may not always be classified, the *semanticPatch* class on level-0 may be upgraded to a more generic *spatiotemporalPatch* class. Further, as the point cloud can also be trajectories, a *timeSeries* can be added besides the *Space* class in the level-1 layer. Correspondingly, the level-2 should also be extended. The final goal is to incorporate more nD applications in the model, which needs more relationships between data and functions being defined.

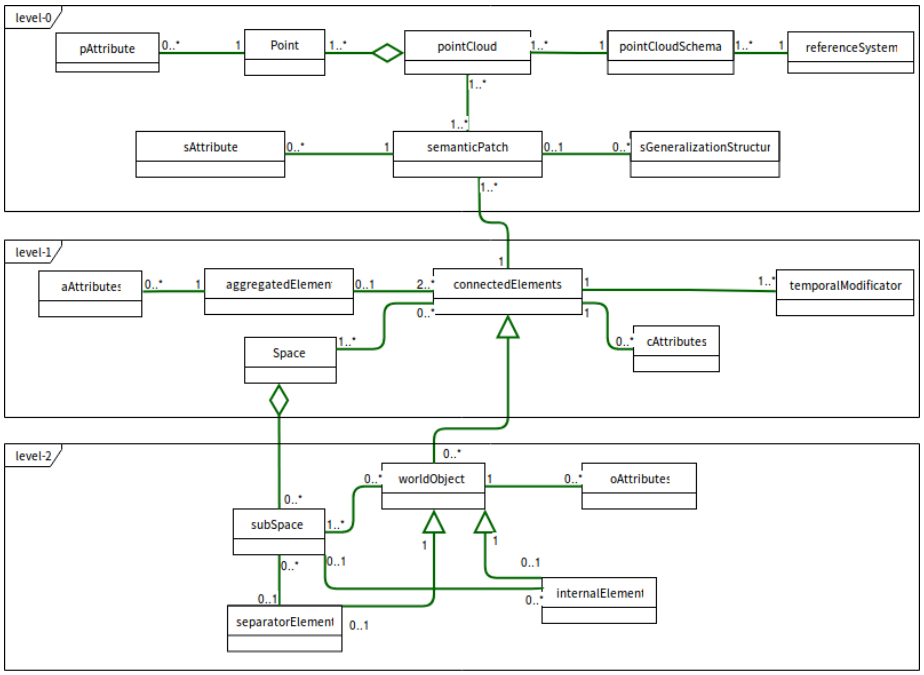


Figure 8.2: Overall Smart Point Cloud model, adapted from (Poux et al., 2017)

8.3.4. NEW PROTOCOLS FOR WEB SERVICES

As point cloud data can be collected and analyzed by different users worldwide, it is convenient to provide all kinds of services online. This needs efficient protocols to facilitate the communication between the server and the user. However, at present, there is no specific protocols for web services defining encoding/decoding, streaming, structuring (e.g., blocking), and compression which are essential techniques for handling nD-PointCloud data. Besides, the dimension concept should be redefined. Public communities (Kodde, 2010) have proposed to establish a new OGC standard, the Web Point Cloud Service (WPCS) for this. It intends to cover all special aspects of nD-PointCloud data to facilitate related services. Before publishing WPCS, fundamental research has to be conducted to investigate the generic techniques for standardization. For instance, the streaming process can be based on cLoI to provide users with better rendering and analytical results.

8.3.5. IMPLEMENTATION ON CLOUD COMPUTING PLATFORMS

Cloud computing platforms have been widely deployed to provide scalable resources and services to users with relatively low cost. The online nD-PointCloud services can also be realized on this platform. By referencing previous work (Richter & Döllner, 2014; Wang et al., 2018), and considering possible use cases, I propose to develop a server-protocol-clients architecture on the cloud computing platform (Figure 8.3).

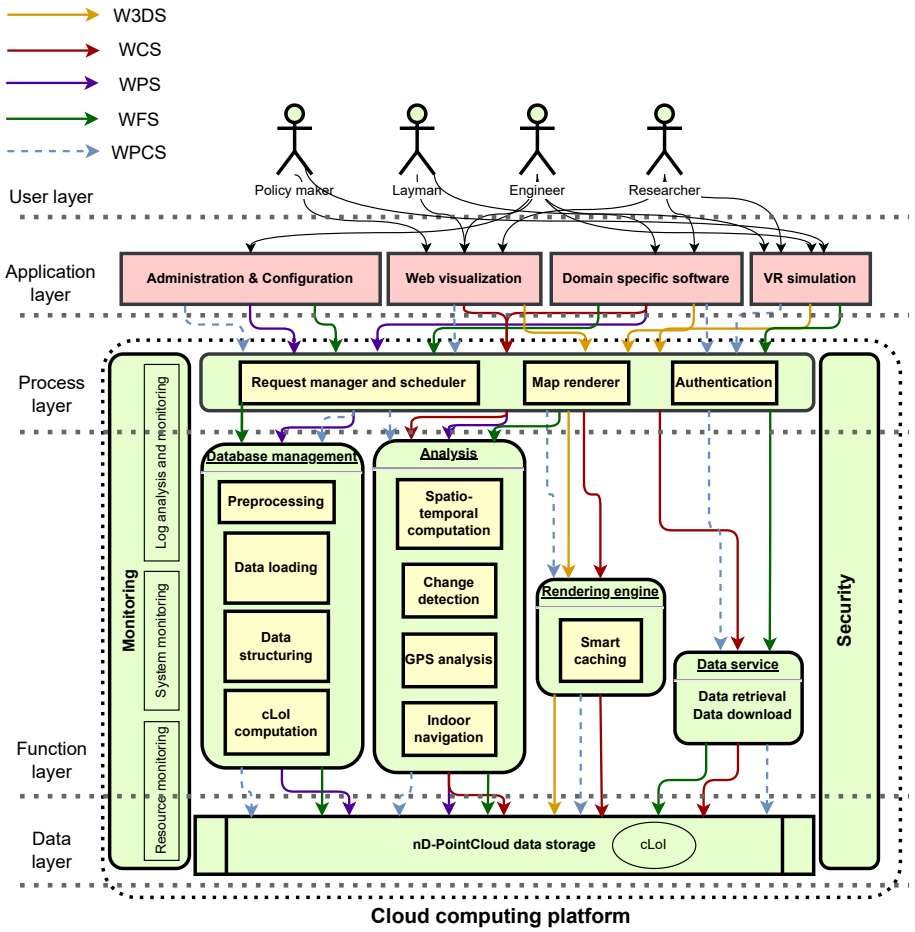


Figure 8.3: Implementation on the cloud computing platform: the 5-layer design is inspired by Richter and Döllner (2014)

DATA LAYER

Data stored are various point clouds that have been collected for different purposes. As current nD-PointCloud solution is developed on a standalone platform, adaptions have to be made for migration. In fact, the prevalent key-value data organization adopted by most distributed DBMSs can be used, where the key is the SFC code, while the value refers to the property dimensions. To facilitate querying, the B+-tree index can also be implemented, e.g., the HBase-BDB solution (Saloustros & Magoutis, 2015). Alternatively, distributed implementation of the Oracle solution can be explored.

FUNCTION LAYER

In the function layer, the data management module is mainly used by administrators and developers to load, store, maintain and benchmark data stores. The analysis module contains the aforementioned core functionalities and techniques supporting different applications. Visualization is separated, as it poses higher requirements than other analytical tasks: limited network bandwidth as well as large data volume confines the massive data rendering in real time. As is mentioned, cLoI and smart caching strategies will be applied to guarantee a smooth and fast rendering process. To realize this, the viewer on the user's side should also be developed to support cLoI. Besides, to facilitate data and knowledge sharing on this cloud computing platform, data uploading functions for the public will be incorporated.

PROTOCOLS

Protocols are used for communication among different layers. In the beginning, protocols supporting vectors or rasters can be adopted temporarily. These will include Web Processing Service (WPS), Web Coverage Service (WCS), Web Feature Service (WFS), and Web 3D Service (W3DS). However, they may present limitations on handling the nD-PointCloud data. For example, visualization supported by W3DS is based on synthesized images, which will cause the loss of information when rendering point data. Consequently, the aforementioned WPCS is still imperative to be realized.

BIBLIOGRAPHY

Achakeev, D., & Seeger, B. (2012). A class of R-tree histograms for spatial databases. *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, 450–453.

Agarwal, P. K., Arge, L., Erickson, J., Franciosa, P. G., & Vitter, J. S. (2000). Efficient searching with linear constraints. *Journal of Computer and System Sciences*, *61*(2), 194–216.

AHN. (2014). *Actueel Hoogtebestand Nederland*. Retrieved January 22, 2022, from https://www.ahn.nl/

Almoradie, A., Jonoski, A., Popescu, I., & Solomatine, D. (2013). Web based access to water related data using OGC WaterML 2.0. *International Journal of Advanced Computer Science and Applications, EnviroGRIDS Special Issue on "Building a Regional Observation System in the Black Sea Catchment"*, *3*(3), 83–89.

Bader, M. (2012). *Space-filling curves: An introduction with applications in scientific computing*. Springer Science & Business Media.

Bamba, B., Ravada, S., Hu, Y., & Anderson, R. (2013). Statistics collection in oracle spatial and graph: Fast histogram construction for complex geometry objects. *Proceedings of the VLDB Endowment*, *6*(11), 1021–1032.

Beckmann, N., Kriegel, H.-P., Schneider, R., & Seeger, B. (1990). The R*-tree: An efficient and robust access method for points and rectangles. *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, 322–331.

Berchtold, S., Böhm, C., & Kriegal, H.-P. (1998). The pyramid-technique: Towards breaking the curse of dimensionality. *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, 142–153.

Berchtold, S., Keim, D. A., & Kriegel, H.-P. (1996). The X-tree: An index structure for high-dimensional data. *Proceedings of VLDB*, 28–39.

Bhattacharjee, S. (2017). Automatic identification system (AIS): Integrating and identifying marine communication channels. *Marine Insight [Online]*.

Böhm, C., Berchtold, S., & Keim, D. A. (2001). Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, *33*(3), 322–373.

Brown, G., Nagel, C., Zlatanova, S., & Kolbe, T. H. (2013). Modelling 3d topographic space against indoor navigation requirements. In J. Pouliot, S. Daniel, F. Hubert, & A. Zamyadi (Eds.), *Progress and new trends in 3d geoinformation sciences* (pp. 1–22). Springer.

Bruno, N., Chaudhuri, S., & Gravano, L. (2001). STHoles: A multidimensional workload-aware histogram. *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 211–222.

Butz, A. R. (1971). Alternative algorithm for hilbert's space-filling curve. *IEEE Transactions on Computers*, *100*(4), 424–426.

Chazelle, B. (1989). Lower bounds on the complexity of polytope range searching. *Journal of the American Mathematical Society*, *2*(4), 637–666.

Comer, D. (1979). Ubiquitous B-tree. *ACM Computing Surveys (CSUR)*, *11*(2), 121–137.

Cura, R. (2016). *Inverse procedural street modelling: From interactive to automatic reconstruction* (Doctoral dissertation). Université Paris-Est Marne-la-Vallée. France.

Cura, R., Perret, J., & Paparoditis, N. (2017). 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.

Deibe, D., Amor, M., & Doallo, R. (2019). Supporting multi-resolution out-of-core rendering of massive lidar point clouds through non-redundant data structures. *International Journal of Geographical Information Science*, *33*(3), 593–617.

Dobos, L., Csabai, I., Szalai-Gindl, J. M., Budavári, T., & Szalay, A. S. (2014). Point cloud databases. *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, 1–4.

Eavis, T., & Lopez, A. (2007). rK-Hist: An R-tree based histogram for multi-dimensional selectivity estimation. *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, 475–484.

Elseberg, J., Magnenat, S., Siegwart, R., & Nüchter, A. (2012). Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics*, *3*(1), 2–12.

Evangelidis, G., Lomet, D., & Salzberg, B. (1997). The hB^{Π}-tree: A multi-attribute index supporting concurrency, recovery and node consolidation. *The VLDB Journal*, *6*(1), 1–25.

Faloutsos, C., & Roseman, S. (1989). Fractals for secondary key retrieval. *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 247–252.

Finkel, R. A., & Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta informatica*, *4*(1), 1–9.

Gaede, V., & Günther, O. (1998). Multidimensional access methods. *ACM Computing Surveys (CSUR)*, *30*(2), 170–231.

Geonovum. (2012). *Nederlandse richtlijn tiling, versie 1.1 [in Dutch]* (tech. rep.) [Available at https://www.geonovum.nl/uploads/standards/downloads/nederlandse_richtlijn_tiling_-_versie_1.1.pdf, accessed 2022-01-22]. Dutch Government.

Gharbi, M., Soualmia, A., Dartus, D., & Masbernat, L. (2016). Comparison of 1D and 2D hydraulic models for floods simulation on the medjerda riverin tunisia. *Journal of Materials and Environmental Science*, *7*(8), 3017–3026.

Girardeau-Montaut, D., Roux, M., Marc, R., & Thibault, G. (n.d.). Change detection on points cloud data acquired with a ground laser scanner. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XXXVI-3/W19*, 30–35.

Goldstein, J., Ramakrishnan, R., Shaft, U., & Yu, J.-B. (1997). Processing queries by linear constraints. *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 257–267.

Guan, X., van Oosterom, P., & Cheng, B. (2018). A parallel n-dimensional space-filling curve library and its application in massive point cloud management. *ISPRS International Journal of Geo-Information*, *7*(8), 19.

Gunzburger, M., & Burkardt, J. (2004). *Uniformity measures for point sample in hypercubes* (tech. rep.) [Available at https://people.sc.fsu.edu/~jburkardt/publications/gb_2004.pdf, accessed 2022-01-22]. Florida State University. USA.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, 47–57.

Hilbert, D. (1891). Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen*, *38*(3), 459–460.

Hofmann-Wellenhof, B., Lichtenegger, H., & Wasle, E. (2007). *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer Science & Business Media.

Isenburg, M. (2013). LASzip: lossless compression of LiDAR data. *Photogrammetric Engineering and Remote Sensing*, *79*(2), 209–217.

Isikdag, U., Zlatanova, S., & Underwood, J. (2013). A bim-oriented model for supporting indoor navigation requirements. *Computers, Environment and Urban Systems*, *41*, 112–123.

ISPRS. (2019). *LAS 1.4 format specification* (tech. rep.) [Available at http://www.asprs.org/wp-content/uploads/2019/07/LAS_1_4_r15.pdf, accessed 2022-01-22]. The American Society for Photogrammetry and Remote Sensing.

IUCN-Committee. (2019). *Guidelines for using the IUCN red list categories and criteria. version 14*. International Union for the Conservation of Nature, Gland, Switzerland.

Jiang, B. (2013). Head/tail breaks: A new classification scheme for data with a heavy-tailed distribution. *The Professional Geographer*, *65*(3), 482–494.

Jonkman, S., & Penning-Rowsell, E. (2008). Human instability in flood flows. *JAWRA Journal of the American Water Resources Association*, *44*(5), 1208–1218.

Kang, H.-K., & Li, K.-J. (2017). A standard indoor spatial data model—OGC IndoorGML and implementation approaches. *ISPRS International Journal of Geo-Information*, *6*(4), 25.

Katayama, N., & Satoh, S. (1997). The SR-tree: An index structure for high-dimensional nearest neighbor queries. *ACM SIGMOD Record*, *26*(2), 369–380.

Khan, A., Yanki, P., Dimcheva, B., & Kossmann, D. (2014). Towards indexing functions: Answering scalar product queries. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 241–252.

Kodde, M. (2010). The art of collecting and disseminating point clouds. In P. van Oosterom, G. Vosselman, T. van Dijk, & M. Uitentuis (Eds.), *Management of Massive Point Cloud Data: Wet and Dry* (pp. 9–15). Nederlandse Commissie voor Geodesie.

Kodde, M. (2016). *Dense image matching*. Retrieved January 22, 2022, from https://www.gim-international.com/content/article/dense-image-matching-2

Kollios, G., Gunopulos, D., & Tsotras, V. J. (1999). On indexing mobile objects. *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 261–272.

Koranne, S. (2011). Hierarchical data format 5: HDF5. *Handbook of open source tools* (pp. 191–200). Springer.

Kuhn, A., Huang, H., Drauschke, M., & Mayer, H. (2016). Fast probabilistic fusion of 3D point clouds via occupancy grids for scene classification. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, III-3*, 325–332.

Lawder, J. K. (2000a). *The application of space-filling curves to the storage and retrieval of multi-dimensional data* (Doctoral dissertation). University of London. UK.

Lawder, J. K. (2000b). *Calculation of mappings between one and n-dimensional values using the hilbert space-filling curve* (tech. rep.) [Available at https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.157.3680&rep=rep1&type=pdf, accessed 2022-01-22]. University of London. UK.

Li, J. (2020). *Manage 4D historical AIS data by space filling curve* (Master's thesis). Delft University of Technology. The Netherlands.

Lima, R. (2010). IBM ILOG CPLEX - what is inside of the box? [Available at http://egon.cheme.cmu.edu/ewo/docs/rlima_cplex_ewo_dec2010.pdf, accessed 2022-01-22]. *EWO Seminar*.

Liu, H., Oosterom, P. V., Mao, B., Meijers, M., & Thompson, R. (2021). An efficient nD-point data structure for querying flood risks. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XLIII-B4-2021*, 367–374.

Liu, H., van Oosterom, P., Meijers, M., & Verbree, E. (2020). An optimized SFC approach for nD window querying on point clouds. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, VI-4/W1-2020*, 119–128.

Liu, H., Thompson, R., van Oosterom, P., & Meijers, M. (2021). Executing convex polytope queries on nD point clouds. *International Journal of Applied Earth Observation and Geoinformation, 105*, 102625.

Liu, H., van Oosterom, P., Meijers, M., Guan, X., Verbree, E., & Horhammer, M. (2020). HistSFC: Optimization for nD massive spatial points querying. *International Journal of Database Management Systems (IJDMS), 12*(3), 7–28.

Liu, H., van Oosterom, P., Meijers, M., & Verbree, E. (2018a). Management of large indoor point clouds: An initial exploration. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XLII-4*, 365–372.

Liu, H., van Oosterom, P., Meijers, M., & Verbree, E. (2018b). Towards 10^{15}-level point clouds management-a nD PointCloud structure. *Proceedings of the 21th AGILE International Conference on Geographic Information Science*.

Liu, Q. (2009). Approximate query processing. In L. Liu & M. T. Özsu (Eds.), *Encyclopedia of database systems* (pp. 113–119). Springer US.

Liu, Q., Shen, Y., & Chen, L. (2021). Lhist: Towards learning multi-dimensional histogram for massive spatial data. *2021 IEEE 37th International Conference on Data Engineering*, 1188–1199.

Liu, T., Li, H., Lu, H., Cheema, M. A., & Shou, L. (2021). Indoor spatial queries: Modeling, indexing, and processing. *The 24th International Conference on Extending Database Technology*, 181–192.

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B., & Huebner, R. (2003). *Level of detail for 3D graphics*. Morgan Kaufmann.

Martinez-Rubi, O., Verhoeven, S., Van Meersbergen, M., Schûtz, M., Van Oosterom, P., Gonçalves, R., & Tijssen, T. (2015). Taming the beast: Free and open-source massive point cloud web visualization. *Capturing Reality Forum 2015*.

Masó, J., Pomakis, K., & Julià, N. (2010). *Web map tile service implementation standard, OGC 07-057r7* (tech. rep.). Open Geospatial Consortium Inc.

Matoušek, J. (1992). Reporting points in halfspaces. *Computational Geometry*, *2*(3), 169–186.

Matoušek, J. (1994). Geometric range searching. *ACM Computing Surveys (CSUR)*, *26*(4), 422–461.

Meagher, D. (1982). Geometric modeling using octree encoding. *Computer graphics and image processing*, *19*(2), 129–147.

Meijers, M., & van Oosterom, P. (2018). Clustering and indexing historic vessel movement data with space filling curves. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLII-4*, 417–424.

Meijers, M., van Oosterom, P., Driel, M., & Šuba, R. (2020). *International Journal of Cartography*, *6*(1), 152–176.

Moon, B., Jagadish, H. V., Faloutsos, C., & Saltz, J. H. (2001). Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on knowledge and data engineering*, *13*(1), 124–141.

Morton, G. M. (1966). A computer oriented geodetic data base and a new technique in file sequencing.

Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications*, *2*, 331–340.

NGA. (2015). *Sensor independent point cloud (SIPC) volume 1 design and implementation description document (DIDD) dated 2015-08-31 (LIWG ver 1.02)* (tech. rep.) [Available at https://nsgreg.nga.mil/doc/view?i=4206&month=2&day=5&year=2016, accessed 2022-01-22]. National Center for Geospatial Intelligence Standards (NCGIS), NGA.

Nishimura, S., Das, S., Agrawal, D., & El Abbadi, A. (2013). $\mathcal{MD}$-HBase: Design and implementation of an elastic data infrastructure for cloud-scale location services. *Distributed and Parallel Databases*, *31*(2), 289–319.

Ong, M. S., Kuang, Y. C., & Ooi, M. P.-L. (2012). Statistical measures of two dimensional point set uniformity. *Computational Statistics & Data Analysis*, *56*(6), 2159–2181.

Ooi, B. C., Tan, K.-L., Yu, C., & Bressan, S. (2000). Indexing the edges—a simple and yet efficient approach to high-dimensional indexing. *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 166–174.

Oracle. (2013). *Indexes and index-organized tables*. Retrieved January 22, 2022, from https://docs.oracle.com/database/121/CNCPT/indexiot.htm%5C#CNCPT721

Oracle. (2019). *SDO_PC_PKG Package (Point Clouds)*. Retrieved January 22, 2022, from https://docs.oracle.com/en/database/oracle/oracle-database/19/spatl/SDO_PC_PKG-reference.html

Otepka, J., Mandlburger, G., & Karel, W. (2012). The OPALS data manager — efficient data management for processing large airborne laser scanning projects. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, I-3*, 153–159.

PDAL-Contributors. (2018). *PDAL Point Data Abstraction Library*. Retrieved October 15, 2021, from https://doi.org/10.5281/zenodo.2556738

Poux, F., Neuville, R., Hallot, P., & Billen, R. (2017). Model for reasoning from semantically rich point cloud data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, IV-4/W5*, 107–115.

Psomadaki, S. (2016). *Using a space filling curve for the management of dynamic point cloud data in a relational DBMS* (Master's thesis). Delft University of Technology. The Netherlands.

Qi, J., Tao, Y., Chang, Y., & Zhang, R. (2018). Theoretically optimal and empirically efficient R-trees with strong parallelizability. *Proceedings of the VLDB Endowment, 11*(5), 621–634.

Ramsey, P. (2012). *Clustering on indices*. Retrieved October 22, 2021, from https://postgis.net/workshops/postgis-intro/clusterindex.html

Ramsey, P. (2020). *PgPointcloud - a postgresql extension for storing point cloud (LIDAR) data*. Retrieved October 22, 2021, from https://pgpointcloud.github.io/pointcloud/

Ravada, S., Horhammer, M., & Kazar, B. M. (2010). Point cloud: Storage, loading, and visualization. *Proceedings of National Science Foundation Tera Grid Workshop on Cyber-GIS, Washington DC, USA*.

Reddy, M. (1997). *Perceptually modulated level of detail for virtual environments* (Doctoral dissertation). University of Edinburgh. UK.

Remondino, F., Spera, M. G., Nocerino, E., Menna, F., Nex, F., & Gonizzi-Barsanti, S. (2013). Dense image matching: Comparisons and analyses. *2013 Digital Heritage International Congress (DigitalHeritage), 1*, 47–54.

Richter, R., & Döllner, J. (2014). Concepts and techniques for integration, analysis and visualization of massive 3D point clouds. *Computers, Environment and Urban Systems, 45*, 114–124.

Rieg, L., Wichmann, V., Rutzinger, M., Sailer, R., Geist, T., & Stötter, J. (2014). Data infrastructure for multitemporal airborne LiDAR point cloud analysis–examples from physical geography in high mountain environments. *Computers, Environment and Urban Systems, 45*, 137–146.

Roh, Y. J., Kim, J. H., Chung, Y. D., Son, J. H., & Kim, M. H. (2010). Hierarchically organized skew-tolerant histograms for geographic data objects. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 627–638.

Saarinen, J. P., Andreasson, H., Stoyanov, T., & Lilienthal, A. J. (2013). 3D normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments. *The International Journal of Robotics Research, 32*(14), 1627–1644.

Saloustros, G., & Magoutis, K. (2015). Rethinking HBase: Design and implementation of an elastic key-value store over log-structured local volumes. *14th International Symposium on Parallel and Distributed Computing*, 225–234.

Sankaranarayanan, J., Samet, H., & Varshney, A. (2006). A fast k-neighborhood algorithm for large point-clouds. *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, 75–84.

Schreijer, Y. (2021). *Visual insight into the temporal changes of sand patterns along the dutch coast* (Master's thesis). Utrecht University. The Netherlands.

Schütz, M., Krösl, K., & Wimmer, M. (2019). Real-time continuous level of detail rendering of point clouds. *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 103–110.

Schütz, M., Ohrhallinger, S., & Wimmer, M. (2020). Fast out-of-core octree generation for massive point clouds. *Computer Graphics Forum, 39*(7), 155–167.

Shi, Q., & Nickerson, B. (2006). *Decreasing radius k-nearest neighbor search using mapping-based indexing schemes* (tech. rep.) [Available at https://www.cs.unb.ca/tech-reports/documents/TR06_179_000.pdf, accessed 2022-01-22]. University of New Brunswick. Canada.

Skilling, J. (2004). Programming the hilbert curve. *AIP Conference Proceedings, 707*(1), 381–387.

Stive, M. J., De Schipper, M. A., Luijendijk, A. P., Aarninkhof, S. G., van Gelder-Maas, C., Van Thiel de Vries, J. S., De Vries, S., Henriquez, M., Marx, S., & Ranasinghe, R. (2013). A new alternative to saving our beaches from sea-level rise: The sand engine. *Journal of Coastal Research, 29*(5), 1001–1008.

Strobl, C. (2008). PostGIS. In S. Shekhar & H. Xiong (Eds.), *Encyclopedia of GIS* (pp. 891–898). Springer US.

Thompson, R. J. (2007). *Towards a rigorous logic for spatial data representation* (Doctoral dissertation) [Published as NCG Publications on Geodesy no. 65]. Delft University of Technology. The Netherlands.

Thomson, C., Apostolopoulos, G., Backes, D., & Boehm, J. (2013). Mobile laser scanning for indoor modelling. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, II-5/W2*, 289–293.

van der Maaden, J. (2019). *Vario-scale visualization of the AHN2 point cloud* (Master's thesis). Delft University of Technology. The Netherlands.

van Oosterom, P. (2005). Spatial access methods. In P. A. Longley, M. F. Goodchild, D. J. Maguire, & D. W. Rhind (Eds.), *Geographical information systems: Principles, techniques, management and applications (second ed.)* (pp. 385–400). Wiley.

van Oosterom, P. (2019). From discrete to continuous levels of detail for managing nD-PointClouds [Available at http://nd-pc.org/documents/vario-nD-PC-v7.pdf, accessed 2022-01-22]. *ISPRS Geospatial Week 2019*.

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

van Oosterom, P., Martinez-Rubi, O., Tijssen, T., & Gonçalves, R. (2017). Realistic benchmarks for point cloud data management systems. *Advances in 3D geoinformation* (pp. 1–30). Springer.

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

Virtanen, J.-P., Daniel, S., Turppa, T., Zhu, L., Julin, A., Hyyppä, H., & Hyyppä, J. (2020). Interactive dense point clouds in a game engine. *ISPRS Journal of Photogrammetry and Remote Sensing*, *163*, 375–389.

Vo, A.-V. (2017). *Spatial data storage and processing strategies for urban laser scanning* (Doctoral dissertation). University College Dublin. Ireland.

Wang, J., & Shan, J. (2005). Space filling curve based point clouds index [Available at http://www.geocomputation.org/2005/WangJ.pdf, accessed 2022-01-22]. *Proceedings of the 8th International Conference on GeoComputation*, 551–562.

Wang, L., Ma, Y., Yan, J., Chang, V., & Zomaya, A. Y. (2018). PipsCloud: High performance cloud computing for remote sensing big data management and processing. *Future Generation Computer Systems*, *78*, 353–368.

Wang, P., & Ravishankar, C. V. (2013). Secure and efficient range queries on outsourced databases using rp-trees. *2013 IEEE 29th International Conference on Data Engineering*, 314–325.

Wang, S., Bao, Z., Culpepper, J. S., & Cong, G. (2021). A survey on trajectory data management, analytics, and learning. *ACM Computing Surveys (CSUR)*, *54*(2), 1–36.

Wenzel, K. (2015). Dense image matching - challenges and potentials [Available at https://www.slideshare.net/KonradWenzel, accessed 2022-01-22]. *3D Virtual Reconstruction and Visualization of Complex Architectures 2015*.

Wijga-Hoefsloot, M. (2012). *Point clouds in a database: Data management within an engineering company* (Master's thesis). Delft University of Technology. The Netherlands.

Xie, H., Wu, B., & Zhao, Z. (2013). A novel organization method of massive point cloud [in Chinese]. *Remote Sensing Information*, *28*(6), 26–32.

Xu, S. (2015). *Classification and change detection in multi-epoch airborne laser scanning point clouds* (Doctoral dissertation). University of Twente. The Netherlands.

Xu, W., Kruminaite, M., Onrust, B., Liu, H., Xiong, Q., & Zlatanova, S. (2013). A 3D model based imdoor navigation system for hubei provincial museum. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XL-4/W4*, 51–55.

Ye, W., & Ai, T. (2017). SQL spatiotemporal query method for multidimensional trajectory data [in Chinese]. *Journal of Geomatics*, *42*(6), 16–20.

Zhang, L. (2020). *Visualization of point cloud models in mobile augmented reality using continuous level of detail method* (Master's thesis). Delft University of Technology. The Netherlands.

Zhang, R., Qi, J., Stradling, M., & Huang, J. (2014). Towards a painless index for spatial objects. *ACM Transactions on Database Systems (TODS)*, *39*(3), 1–42.

Zheng, Y., Ou, Y., Lex, A., & Phillips, J. M. (2021). Visualization of big spatial data using coresets for kernel density estimates. *IEEE Transactions on Big Data, 7*(3), 524–534.

Zheng, Y., Xie, X., Ma, W.-Y., et al. (2010). Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin, 33*(2), 32–39.

A

TIME COST OF AHN2 QUERYING

This appendix provides exact time measurements of the AHN2 benchmark test. The time cost of first filter includes range computing, database fetching, DBMS communicating and table creating. Specific explanation of the performance is provided in Section 5.1.

Table A.1: Time cost of different processes in the SmallA querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	7.37	3.14	0.89	4.62	0.26	4.88	47,254
	2	13.07	3.15	0.89	4.50	0.25	4.75	48,587
	3	26.02	3.46	0.81	4.73	0.25	4.98	46,315
	4	72.91	5.30	0.96	6.82	0.25	7.07	32,650
	5	173.50	6.23	1.50	8.30	0.25	8.55	26,993
HistSFC_10K	1	0.82	2.84	0.95	4.10	0.27	4.36	52,885
	2	1.34	3.09	0.84	4.53	0.27	4.80	48,081
	3	2.46	2.90	0.65	4.09	0.26	4.35	53,068
	4	6.21	3.06	0.64	4.19	0.26	4.45	51,886
	5	11.82	3.23	0.79	4.30	0.26	4.56	50,645
HistSFC_100K	1	0.20	2.96	0.84	4.20	0.30	4.50	51,253
	2	0.29	2.94	0.91	4.21	0.30	4.51	51,184
	3	0.38	2.83	0.73	4.16	0.30	4.47	51,677
	4	0.75	2.96	0.76	4.22	0.30	4.52	51,003
	5	1.38	2.85	0.68	4.14	0.29	4.42	52,168
PlainSFC	1	-	2.66	0.98	3.94	0.47	4.41	52,381
	2	-	2.65	0.80	3.86	0.47	4.33	53,338
	3	-	2.76	0.76	3.81	0.48	4.28	53,886
	4	-	2.72	0.87	3.86	0.47	4.33	53,264
	5	-	2.92	1.02	4.14	0.49	4.63	49,879

Table A.2: Time cost of different processes in the SmallB querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	6.70	2.73	4.37	7.61	0.50	8.11	34,004
	2	13.21	2.87	4.13	7.53	0.50	8.03	34,356
	3	26.04	3.24	4.41	8.18	0.50	8.68	31,776
	4	72.29	4.83	4.65	9.91	0.50	10.42	26,489
	5	151.85	9.56	4.43	14.48	0.53	15.01	18,382
HistSFC_10K	1	0.67	2.28	4.46	6.89	0.65	7.53	36,637
	2	1.29	2.30	4.41	6.75	0.60	7.36	37,503
	3	2.28	2.32	4.40	6.85	0.64	7.48	36,867
	4	6.54	2.60	4.26	6.93	0.64	7.57	36,453
	5	11.94	2.84	4.38	7.32	0.64	7.96	34,684
HistSFC_100K	1	0.21	2.54	4.80	7.58	0.67	8.25	33,448
	2	0.26	2.55	5.31	8.12	0.64	8.76	31,489
	3	0.37	2.55	5.28	8.01	0.67	8.68	31,776
	4	0.74	2.52	5.18	7.90	0.64	8.54	32,304
	5	1.39	2.47	5.31	8.07	0.64	8.71	31,688
PlainSFC	1	-	3.89	5.00	10.11	0.99	11.10	24,859
	2	-	3.80	5.06	10.74	0.99	11.72	23,536
	3	-	3.78	5.43	10.40	0.94	11.34	24,333
	4	-	3.83	5.80	10.40	0.94	11.34	24,326
	5	-	4.09	5.93	10.87	0.95	11.82	23,345

Table A.3: Time cost of different processes in the SmallC querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	7.00	2.38	0.31	2.94	0.25	3.19	75,107
	2	13.70	2.67	0.28	3.16	0.25	3.41	70,328
	3	29.26	3.45	0.30	4.12	0.26	4.38	54,715
	4	84.55	5.37	0.27	6.04	0.26	6.30	38,072
	5	145.89	8.28	0.37	8.92	0.26	9.18	26,127
HistSFC_10K	1	0.68	2.01	0.16	2.46	0.27	2.72	88,039
	2	1.31	2.07	0.28	2.49	0.26	2.75	87,334
	3	2.43	2.11	0.18	2.55	0.26	2.81	85,436
	4	6.28	2.26	0.27	2.62	0.26	2.88	83,299
	5	12.37	2.67	0.23	3.10	0.27	3.38	71,057
HistSFC_100K	1	0.21	2.28	0.14	2.75	0.27	3.02	79,384
	2	0.26	2.30	0.17	2.78	0.26	3.04	78,784
	3	0.40	2.23	0.14	2.63	0.27	2.89	82,867
	4	0.74	2.23	0.17	2.63	0.27	2.90	82,724
	5	1.41	2.19	0.16	2.65	0.26	2.91	82,327
PlainSFC	1	-	2.86	0.27	3.50	0.26	3.77	63,697
	2	-	2.80	0.29	3.48	0.28	3.76	63,849
	3	-	2.82	0.20	3.58	0.28	3.86	62,129
	4	-	2.91	0.27	3.60	0.27	3.88	61,889
	5	-	3.59	0.34	4.25	0.27	4.52	53,057

Table A.4: Time cost of different processes in the Medium querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	7.60	2.62	0.71	3.78	4.55	8.33	405,245
	2	14.93	3.23	0.78	4.25	4.67	8.92	378,477
	3	27.99	3.38	0.82	4.42	4.58	9.00	374,903
	4	84.33	6.40	0.76	7.46	4.59	12.05	280,120
	5	167.84	13.04	0.81	14.13	4.71	18.84	179,096
HistSFC_10K	1	0.72	2.70	0.69	3.82	5.10	8.91	378,647
	2	1.87	2.60	0.80	3.78	4.61	8.39	402,394
	3	2.80	2.58	0.71	3.74	4.73	8.47	398,686
	4	8.15	2.81	0.87	3.99	4.34	8.33	405,002
	5	13.09	3.00	0.69	4.14	4.66	8.80	383,335
HistSFC_100K	1	0.20	2.64	0.60	3.91	4.60	8.51	396,578
	2	0.27	2.64	0.57	3.84	4.24	8.09	417,373
	3	0.45	2.69	0.62	3.94	4.36	8.31	406,318
	4	0.83	2.63	0.62	3.82	4.32	8.15	414,248
	5	1.39	2.48	0.69	3.72	4.59	8.31	405,927
PlainSFC	1	-	2.48	0.73	3.57	5.51	9.08	371,724
	2	-	2.58	0.57	3.71	5.25	8.96	376,619
	3	-	2.56	0.68	3.74	4.98	8.72	387,116
	4	-	3.07	0.68	4.21	5.39	9.59	351,770
	5	-	2.98	0.57	4.13	5.12	9.24	365,207

Table A.5: Time cost of different processes in the Large querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	7.41	2.42	0.43	3.00	27.69	30.69	738,480
	2	17.76	2.90	0.49	3.58	26.49	30.07	753,632
	3	33.45	3.93	0.42	4.55	27.08	31.63	716,578
	4	88.18	7.65	0.44	8.30	27.09	35.40	640,276
	5	173.46	10.05	0.46	10.68	27.44	38.12	594,492
HistSFC_10K	1	0.87	2.53	0.75	3.61	27.43	31.04	730,082
	2	1.62	2.45	0.62	3.46	27.43	30.89	733,603
	3	2.95	2.53	0.60	3.53	27.41	30.94	732,560
	4	7.58	2.88	0.56	4.13	27.05	31.17	727,061
	5	14.02	3.17	0.68	4.17	28.47	32.64	694,381
HistSFC_100K	1	0.20	2.21	0.36	3.08	27.08	30.17	751,258
	2	0.29	2.28	0.44	3.00	27.05	30.05	754,183
	3	0.46	2.31	0.34	3.05	27.36	30.41	745,353
	4	0.96	2.40	0.46	3.12	27.19	30.31	747,640
	5	1.57	2.31	0.44	3.02	26.89	29.91	757,637
PlainSFC	1	-	2.92	0.43	4.03	31.05	35.08	646,025
	2	-	2.81	0.47	3.91	30.85	34.76	651,935
	3	-	2.86	0.45	3.93	30.85	34.78	651,672
	4	-	3.28	0.45	4.37	29.71	34.08	665,059
	5	-	3.73	0.93	4.85	31.35	36.19	626,177

Table A.6: Time cost of different processes in the SmallA querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_10K	1	1.04	0.03	0.89	0.94	0.45	1.39	166,358
	2	1.14	0.05	0.56	0.63	0.43	1.06	218,296
	3	2.24	0.08	0.84	0.94	0.46	1.40	165,404
	4	5.93	0.18	0.73	0.93	0.45	1.38	167,081
	5	11.74	0.34	1.43	1.79	0.45	2.24	103,101
Range_100K	1	0.86	0.30	0.44	0.81	0.32	1.13	203,474
	2	1.25	0.34	0.54	0.94	0.32	1.26	183,417
	3	2.51	0.37	0.76	1.20	0.31	1.51	153,112
	4	6.16	0.47	0.48	1.02	0.31	1.33	173,880
	5	11.74	0.63	0.80	1.49	0.32	1.81	127,480
Range_1M	1	0.82	2.84	0.95	4.10	0.27	4.36	52,885
	2	1.34	3.09	0.84	4.53	0.27	4.80	48,081
	3	2.46	2.90	0.65	4.09	0.26	4.35	53,068
	4	6.21	3.06	0.64	4.19	0.26	4.45	51,886
	5	11.82	3.23	0.79	4.30	0.26	4.56	50,645

Table A.7: Time cost of different processes in the SmallB querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_10K	1	0.96	0.06	3.65	3.73	2.96	6.69	41,230
	2	1.20	0.07	3.27	3.37	3.40	6.76	40,815
	3	2.27	0.10	3.92	4.04	3.52	7.56	36,501
	4	5.89	0.21	3.97	4.21	3.39	7.60	36,323
	5	11.72	0.44	4.11	4.58	3.23	7.80	35,369
Range_100K	1	0.66	0.38	3.62	4.14	0.83	4.96	55,627
	2	1.22	0.40	4.18	4.71	0.83	5.54	49,794
	3	2.36	0.42	4.16	4.72	0.83	5.55	49,696
	4	6.00	0.54	4.15	4.83	0.84	5.67	48,627
	5	12.03	0.77	3.93	4.84	0.86	5.70	48,414
Range_1M	1	0.67	2.28	4.46	6.89	0.65	7.53	36,637
	2	1.29	2.30	4.41	6.75	0.60	7.36	37,503
	3	2.28	2.32	4.40	6.85	0.64	7.48	36,867
	4	6.54	2.60	4.26	6.93	0.64	7.57	36,453
	5	11.94	2.84	4.38	7.32	0.64	7.96	34,684

Table A.8: Time cost of different processes in the SmallC querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_10K	1	0.72	0.06	0.11	0.18	0.33	0.52	463,865
	2	1.15	0.06	0.11	0.19	0.34	0.54	448,258
	3	2.54	0.16	0.13	0.31	0.32	0.63	382,485
	4	6.73	0.43	0.11	0.56	0.33	0.89	270,065
	5	11.95	0.46	0.15	0.64	0.32	0.95	251,382
Range_100K	1	0.67	0.26	0.14	0.46	0.28	0.74	323,205
	2	1.24	0.28	0.11	0.45	0.28	0.73	329,420
	3	2.41	0.32	0.29	0.67	0.30	0.97	248,002
	4	6.32	0.46	0.17	0.69	0.28	0.97	246,726
	5	12.09	0.67	0.19	0.93	0.28	1.21	198,361
Range_1M	1	0.68	2.01	0.16	2.46	0.27	2.72	88,039
	2	1.31	2.07	0.28	2.49	0.26	2.75	87,334
	3	2.43	2.11	0.18	2.55	0.26	2.81	85,436
	4	6.28	2.26	0.27	2.62	0.26	2.88	83,299
	5	12.37	2.67	0.23	3.10	0.27	3.38	71,057

Table A.9: Time cost of different processes in the Medium querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_10K	1	1.24	0.05	0.72	0.78	5.05	5.83	579,379
	2	1.30	0.09	0.51	0.63	5.16	5.79	583,183
	3	2.62	0.17	0.94	1.13	5.37	6.49	519,772
	4	6.11	0.25	0.75	1.02	5.32	6.34	532,316
	5	12.11	0.48	0.83	1.33	5.16	6.49	520,413
Range_100K	1	0.66	0.28	0.47	0.83	5.02	5.84	577,495
	2	1.39	0.34	0.52	0.95	4.61	5.56	606,993
	3	2.52	0.40	0.69	1.17	4.35	5.52	611,502
	4	6.13	0.49	0.51	1.08	4.49	5.57	606,447
	5	12.53	0.73	0.75	1.55	4.92	6.48	521,137
Range_1M	1	0.72	2.70	0.69	3.82	5.10	8.91	378,647
	2	1.87	2.60	0.80	3.78	4.61	8.39	402,394
	3	2.80	2.58	0.71	3.74	4.73	8.47	398,686
	4	8.15	2.81	0.87	3.99	4.34	8.33	405,002
	5	13.09	3.00	0.69	4.14	4.66	8.80	383,335

Table A.10: Time cost of different processes in the Large querying (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_10K	1	1.33	0.05	0.74	0.81	31.66	32.46	698,124
	2	1.20	0.06	0.58	0.67	30.90	31.57	717,917
	3	2.74	0.11	0.71	0.85	32.14	32.99	686,972
	4	6.73	0.26	0.69	0.98	32.09	33.06	685,435
	5	14.09	0.50	1.67	2.19	31.70	33.90	668,571
Range_100K	1	0.70	0.26	0.41	0.73	28.53	29.26	774,599
	2	1.25	0.26	0.73	1.05	28.13	29.18	776,749
	3	2.54	0.31	0.54	0.90	28.41	29.31	773,224
	4	6.64	0.47	0.58	1.11	29.03	30.14	751,981
	5	12.46	0.69	0.58	1.32	29.20	30.52	742,666
Range_1M	1	0.87	2.53	0.75	3.61	27.43	31.04	730,082
	2	1.62	2.45	0.62	3.46	27.43	30.89	733,603
	3	2.95	2.53	0.60	3.53	27.41	30.94	732,560
	4	7.58	2.88	0.56	4.13	27.05	31.17	727,061
	5	14.02	3.17	0.68	4.17	28.47	32.64	694,381

Table A.11: Time cost of state-of-the-art solutions (second)

	SmallA		SmallB		SmallC		Medium		Large	
	First filter	Second filter	First filter	Second filter	First filter	Second filter	First filter	Second filter	First filter	Second filter
PostGIS	1.56	0.24	14.76	0.97	1.35	0.13	13.00	1.52	72.21	9.54
	1.63	0.23	19.61	0.62	1.07	0.10	15.18	1.19	76.52	6.97
	1.61	0.22	13.93	0.77	1.06	0.13	15.47	1.77	75.56	7.98
	1.55	0.26	12.33	1.19	1.13	0.13	15.38	1.82	73.52	6.72
	1.79	0.26	12.94	0.78	1.07	0.16	15.31	1.77	74.20	6.95
Pyramid	13.41	24.61	9.86	5.78	8.07	1.36	10.40	2.94	9.33	10.52
	17.11	50.06	17.08	7.20	9.45	1.52	11.90	4.57	27.67	9.24
	14.42	107.89	13.29	17.18	10.19	11.39	9.17	12.95	39.28	61.04
	15.80	248.82	19.85	8.35	10.36	11.56	9.75	9.53	21.59	275.63
	14.55	222.50	10.26	3.22	10.68	21.64	9.65	15.87	21.27	131.43
PyramidEx	8.80	3.32	11.39	14.20	6.81	0.37	9.57	5.23	11.61	11.19
	9.26	4.53	18.35	23.80	8.01	0.85	9.96	11.77	16.78	16.92
	15.03	2.67	23.76	23.33	9.18	1.59	19.26	25.95	18.70	35.99
	17.80	3.21	31.85	19.75	10.96	3.51	21.12	30.39	19.95	47.34
	11.93	10.09	19.79	28.84	12.05	3.94	18.89	32.02	28.67	59.84
HistSFC	0.81	0.32	4.14	0.83	0.46	0.28	0.83	5.02	0.73	28.53
	0.94	0.32	4.71	0.83	0.45	0.28	0.95	4.61	1.05	28.13
	1.20	0.31	4.72	0.83	0.67	0.30	1.17	4.35	0.90	28.41
	1.02	0.31	4.83	0.84	0.69	0.28	1.08	4.49	1.11	29.03
	1.49	0.32	4.84	0.86	0.93	0.28	1.55	4.92	1.32	29.20

Table A.12: Overall throughput of state-of-the-art solutions (Points per second)

	SmallA	SmallB	SmallC	Medium	Large
PostGIS	128,545	17,541	162,478	232,510	277,205
	124,053	13,635	204,623	206,137	271,442
	126,363	18,768	202,378	195,770	271,318
	127,480	20,406	189,131	196,237	282,443
	112,997	20,110	195,770	197,569	279,282
Pyramid	6,069	17,651	25,431	252,914	1,141,608
	3,435	11,364	21,869	204,873	614,112
	1,886	9,056	11,112	152,565	225,891
	872	9,784	10,944	175,118	76,251
	973	20,467	7,420	132,213	148,419
PyramidEx	19,036	10,779	33,405	228,171	994,088
	16,737	6,546	27,055	155,295	672,559
	13,030	5,860	22,255	74,651	414,402
	10,983	5,348	16,564	65,523	336,784
	10,479	5,674	15,001	66,294	256,044
HistSFC	280,363	65,771	490,425	3,334,862	8,400,003
	240,103	57,831	523,620	2,782,259	7,240,641
	190,065	57,746	352,156	2,516,689	7,869,169
	222,292	56,435	343,087	2,674,231	7,572,070
	153,826	56,423	256,216	1,938,472	6,844,822

B

TIME COST OF FLOOD DATA QUERYING

This appendix provides exact time measurements of benchmark test on the flood data. The time cost of first filter includes range computing, database fetching, DBMS communicating and table creating. Specific explanation of the performance is provided in Section 5.2.

Table B.1: Time cost of different processes in the DEPTH3m (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	0.78	0.30	0.74	1.46	33.32	34.78	761,500
	2	1.50	0.31	1.06	1.77	34.63	36.40	727,548
	3	2.93	0.40	1.09	1.91	37.97	39.88	664,098
	4	7.24	0.52	0.96	1.89	40.82	42.71	620,051
HistSFC_5K	1	0.65	0.20	0.57	0.89	36.29	37.18	712,343
	2	0.36	0.21	0.74	1.07	37.47	38.54	687,152
	3	0.71	0.24	0.99	1.35	43.49	44.84	590,651
	4	1.14	0.25	1.39	1.77	43.41	45.18	586,245
HistSFC_10K	1	0.57	0.23	0.74	1.13	36.20	37.33	709,481
	2	0.31	0.24	0.84	1.17	40.71	41.87	632,489
	3	0.68	0.25	0.89	1.23	50.42	51.65	512,753
	4	1.17	0.25	1.00	1.33	53.15	54.48	486,109
PlainSFC	1	-	0.49	0.46	1.03	52.04	53.07	499,034
	2	-	0.52	0.47	1.07	102.85	103.92	254,854
	3	-	0.51	0.40	0.99	197.95	198.95	133,123
	4	-	0.53	0.61	1.22	202.28	203.50	130,142

Table B.2: Time cost of different processes in the ARRIVAL24h (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	0.90	0.25	0.86	1.20	7.22	8.42	109,887
	2	1.58	0.31	1.48	1.89	8.60	10.49	88,262
	3	3.18	0.36	1.97	2.42	8.92	11.34	81,609
	4	5.98	0.50	1.58	2.17	9.74	11.91	77,750
HistSFC_5K	1	0.24	0.26	1.19	1.51	8.27	9.77	94,719
	2	0.35	0.26	1.53	1.85	10.42	12.27	75,462
	3	0.63	0.29	1.58	1.92	10.97	12.89	71,815
	4	1.13	0.30	1.92	2.27	10.91	13.18	70,251
HistSFC_10K	1	0.20	0.32	1.16	1.51	9.43	10.94	84,623
	2	0.28	0.32	1.61	1.96	10.54	12.51	74,020
	3	0.49	0.33	1.68	2.04	10.52	12.56	73,690
	4	0.74	0.28	1.80	2.12	10.59	12.71	72,826
PlainSFC	1	-	0.66	2.40	3.14	16.34	19.48	47,513
	2	-	0.67	2.06	2.82	31.82	34.63	26,729
	3	-	0.68	2.30	3.08	30.98	34.05	27,185
	4	-	0.70	2.49	3.27	31.79	35.06	26,402

Table B.3: Time cost of different processes in the EXTENTmax (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	1.02	0.43	0.52	1.56	49.83	51.40	626,195
	2	1.61	0.48	0.49	1.59	50.76	52.35	614,760
	3	3.17	0.52	0.58	1.69	53.67	55.36	581,315
	4	5.80	0.73	0.55	1.89	54.96	56.84	566,209
HistSFC_5K	1	0.26	0.19	0.44	0.77	52.50	53.27	604,155
	2	0.40	0.20	0.51	0.85	56.67	57.51	559,574
	3	0.68	0.21	0.55	0.92	56.39	57.30	561,624
	4	1.14	0.23	0.76	1.14	56.14	57.28	561,840
HistSFC_10K	1	0.19	0.24	0.42	0.76	53.38	54.14	594,457
	2	0.32	0.26	0.52	0.90	57.21	58.11	553,873
	3	0.50	0.27	0.68	1.04	57.39	58.43	550,792
	4	0.84	0.28	0.72	1.10	58.26	59.35	542,227
PlainSFC	1	-	0.47	0.50	1.02	54.40	55.42	580,685
	2	-	0.46	0.71	1.22	108.93	110.15	292,185
	3	-	0.48	0.74	1.27	216.17	217.44	148,012
	4	-	0.48	1.52	2.05	220.23	222.28	144,785

Table B.4: Time cost of different processes in the HOUSErisk (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	0.90	0.28	1.17	1.48	0.78	2.25	75,707
	2	1.59	0.33	1.56	1.91	0.78	2.69	63,258
	3	3.03	0.38	2.15	2.56	0.78	3.34	51,084
	4	5.93	0.50	2.58	3.10	0.77	3.87	44,035
HistSFC_5K	1	0.24	0.27	1.08	1.38	0.82	2.19	77,674
	2	0.37	0.28	1.46	1.77	0.79	2.56	66,465
	3	0.66	0.31	1.91	2.24	0.78	3.02	56,504
	4	1.13	0.31	2.65	3.01	0.79	3.80	44,870
HistSFC_10K	1	0.23	0.34	1.14	1.50	0.96	2.46	69,388
	2	0.29	0.32	1.51	1.85	0.96	2.81	60,625
	3	0.50	0.35	1.98	2.35	0.91	3.26	52,355
	4	0.90	0.37	2.03	2.42	0.90	3.32	51,330
PlainSFC	1	-	0.74	1.09	1.85	2.36	4.22	40,412
	2	-	0.73	1.52	2.29	3.42	5.71	29,856
	3	-	0.75	1.84	2.62	3.44	6.07	28,098
	4	-	0.75	2.41	3.19	3.69	6.88	24,788

Table B.5: Time cost of different processes in the ROADrisk (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
HistSFC_1K	1	1.15	17.59	1.32	19.39	0.44	19.84	4
	2	1.62	17.61	2.01	20.06	0.44	20.49	4
	3	2.93	17.27	2.18	20.07	0.44	20.50	4
	4	7.01	17.79	2.54	20.77	0.44	21.20	4
HistSFC_5K	1	0.44	14.28	2.18	16.66	2.64	19.29	4
	2	0.41	14.36	1.43	15.98	2.58	18.56	4
	3	0.67	14.40	1.94	16.53	2.59	19.12	4
	4	1.20	14.36	1.83	16.37	2.59	18.96	4
HistSFC_10K	1	0.55	13.43	1.44	14.99	4.92	19.92	4
	2	0.31	13.52	2.20	15.84	4.69	20.53	4
	3	0.46	13.76	2.62	16.52	4.77	21.28	4
	4	0.97	13.78	3.27	17.17	4.69	21.86	4
PlainSFC	1	-	34.23	0.38	34.64	54.06	88.71	1
	2	-	34.20	0.46	34.69	109.86	144.55	1
	3	-	34.23	0.39	34.66	204.50	239.16	0.3
	4	-	34.22	0.44	34.69	204.96	239.64	0.3

Table B.6: Time cost of different processes in DEPTH3m (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_50K	1	0.25	0.08	0.55	1.04	35.31	36.35	728,689
	2	0.35	0.09	0.67	0.87	37.86	38.73	683,870
	3	0.63	0.11	0.86	1.09	43.89	44.97	588,905
	4	1.05	0.12	1.14	1.38	43.38	44.76	591,760
Range_100K	1	0.65	0.20	0.57	0.89	36.29	37.18	712,343
	2	0.36	0.21	0.74	1.07	37.47	38.54	687,152
	3	0.71	0.24	0.99	1.35	43.49	44.84	590,651
	4	1.14	0.25	1.39	1.77	43.41	45.18	586,245
Range_500K	1	0.48	1.71	0.81	2.66	36.39	39.05	678,248
	2	0.39	1.60	0.78	2.52	37.72	40.24	658,124
	3	0.78	1.71	1.07	2.92	41.29	44.21	599,068
	4	1.49	1.72	0.99	2.85	41.59	44.44	596,008

Table B.7: Time cost of different processes in ARRIVAL24h (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_50K	1	0.24	0.12	0.83	0.99	8.34	9.33	99,249
	2	0.38	0.13	1.23	1.40	10.05	11.45	80,882
	3	0.63	0.14	1.49	1.66	10.84	12.50	74,032
	4	1.14	0.16	2.05	2.25	10.89	13.14	70,454
Range_100K	1	0.24	0.26	1.19	1.51	8.27	9.77	94,719
	2	0.35	0.26	1.53	1.85	10.42	12.27	75,462
	3	0.63	0.29	1.58	1.92	10.97	12.89	71,815
	4	1.13	0.30	1.92	2.27	10.91	13.18	70,251
Range_500K	1	0.24	1.99	0.87	2.97	7.31	10.27	90,109
	2	0.43	1.90	1.49	3.51	8.06	11.56	80,049
	3	0.67	1.88	1.94	3.93	8.54	12.47	74,222
	4	1.12	1.89	1.98	3.98	8.88	12.86	71,965

Table B.8: Time cost of different processes in EXTENTmax (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_50K	1	0.24	0.07	0.41	0.62	52.27	52.89	608,472
	2	0.38	0.08	0.61	0.84	56.21	57.05	564,115
	3	0.67	0.11	0.61	0.89	56.20	57.09	563,769
	4	1.10	0.12	0.89	1.16	58.09	59.25	543,188
Range_100K	1	0.26	0.19	0.44	0.77	52.50	53.27	604,155
	2	0.40	0.20	0.51	0.85	56.67	57.51	559,574
	3	0.68	0.21	0.55	0.92	56.39	57.30	561,624
	4	1.14	0.23	0.76	1.14	56.14	57.28	561,840
Range_500K	1	0.23	1.60	0.47	2.20	50.30	52.50	613,027
	2	0.41	1.55	0.56	2.25	51.71	53.96	596,385
	3	0.70	1.63	0.56	2.33	55.65	57.98	555,114
	4	1.13	1.56	0.63	2.33	54.53	56.85	566,089

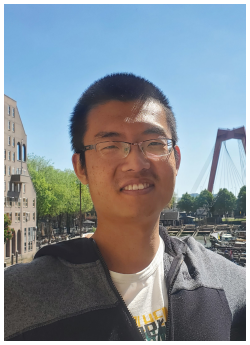
Table B.9: Time cost of different processes in HOUSErisk (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_50K	1	0.29	0.16	1.23	1.41	0.80	2.20	77,357
	2	0.38	0.16	1.43	1.61	0.80	2.41	70,712
	3	0.66	0.18	2.43	2.62	0.78	3.40	50,064
	4	1.16	0.21	2.55	2.77	0.80	3.57	47,682
Range_100K	1	0.24	0.27	1.08	1.38	0.82	2.19	77,674
	2	0.37	0.28	1.46	1.77	0.79	2.56	66,465
	3	0.66	0.31	1.91	2.24	0.78	3.02	56,504
	4	1.13	0.31	2.65	3.01	0.79	3.80	44,870
Range_500K	1	0.27	1.65	1.49	3.19	0.80	3.99	42,690
	2	0.38	1.64	1.57	3.25	0.80	4.05	42,099
	3	0.72	1.73	2.25	4.03	0.73	4.76	35,809
	4	1.22	1.76	2.25	4.05	0.79	4.84	35,217

Table B.10: Time cost of different processes in ROADrisk (second)

	Data set	HistogramTree loading	First filter			Second filter	Total time cost	Points per second
			Range computing	Database fetching	Total			
Range_50K	1	0.28	7.65	1.73	9.48	4.28	13.76	6
	2	0.40	7.86	1.75	9.70	4.22	13.91	6
	3	0.67	7.86	2.38	10.33	4.20	14.52	6
	4	1.11	7.83	2.27	10.19	4.20	14.39	6
Range_100K	1	0.44	14.28	2.18	16.66	2.64	19.29	4
	2	0.41	14.36	1.43	15.98	2.58	18.56	4
	3	0.67	14.40	1.94	16.53	2.59	19.12	4
	4	1.20	14.36	1.83	16.37	2.59	18.96	4
Range_500K	1	0.25	69.65	1.59	72.06	2.61	74.67	1
	2	0.37	70.11	2.13	73.06	2.44	75.49	1
	3	0.64	71.82	2.26	74.81	2.43	77.25	1
	4	1.07	70.37	2.26	73.41	2.42	75.83	1

CURRICULUM VITÆ



Haicheng Liu was born on April 23, 1991 in Jining, Shandong province, China. He grew up in Qufu, the hometown of Confucius. He was educated there before college. In 2008, he entered into Hohai University, pursing a bachelor degree in Hydrology and Water Resources Engineering. During this period of study, he got interested in big data and databases. So, after his graduation in 2012, he joined the Geomatics master program in TU Delft, learning different techniques to process spatial data. His master thesis “Comparing NetCDF and a multidimensional array database on managing and querying large hydrologic datasets: a case study of SciDB" was completed during an internship in Hydrologic B.V. in the Netherlands. In 2014, he received the master degree. After this, he returned to China, employed as a research engineer in Hohai University and later in Yangtze River Scientific Research Institute. He was involved in several national projects, including construction of highly accurate precipitation data, flood risk mapping, registration and certification of rural land use, etc.

After a few years of practical engineering, he decided to challenge himself further. With persistent passion for spatial data, he started his PhD research in 2017 in TU Delft under the supervision of Prof. Peter van Oosterom, Dr. Martijn Meijers and Ir. Edward Verbree. The topic is about nD-PointCloud data management and querying. However, he did not forget water. Based on his experience acquired during the flood modelling projects in China, he creatively proposed to use nD-PointCloud for managing modelling results to support more applications. In the meantime, the historic COVID-19 broke out, which significantly influenced the research and life. He carried on the research in the Netherlands until the accomplishment of his PhD dissertation in 2022. In the final phase of graduation, he got a job offer from Alibaba DAMO (Discovery, Adventure, Momentum and Outlook) Academy to develop spatial data computing platforms. He decided to join, along with his faith in data, water and life.

22#12

nD-PointCloud Data Management

Continuous Levels, Adaptive Histograms, and Diverse Query Geometries

Haicheng Liu

In the Geomatics domain, a point cloud refers to a data set that records the coordinates and other attributes of a huge number of points. Conceptually, each of the attributes can be regarded as a dimension to represent a specific type of information, such as time and Level of Importance (LoI). Drastically increasing collection of high dimensional point clouds raises essential demand for smart and highly efficient data management solutions. However, effective tools are missing. File-based solutions require substantial development of data structures and algorithms. Also, with such solutions, enormous effort has to be made to integrate different data types, formats and libraries. By contrast, state-of-the-art DataBase Management Systems (DBMSs) avoid these issues, because they are initially devised for generic use of data. However, DBMSs still present limitations on efficiently indexing non-uniformly distributed points, supporting continuous LoI, and operating high dimensional data. These problems motivate the PhD research which focuses on developing a new DBMS solution. It is aimed at efficiently managing and querying massive nD point clouds to support different types of applications.

A+BE | Architecture and the Built Environment | TU Delft BK