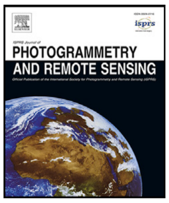
ISPRS Journal of Photogrammetry and Remote Sensing 224 (2025) 1–18

Contents lists available at ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

journal homepage: www.elsevier.com/locate/isprsjprs



An nD-histogram technique for querying non-uniformly distributed point cloud data

Haicheng Liu [a,*], Zhiwei Li [a], Peter van Oosterom [b], Martijn Meijers [b], Chuqi Zhang [a]

[a] China South-to-North Water Diversion Group Water Networks Intelligent Technology Co., Ltd., Beijing, China
[b] Delft University of Technology, Delft, The Netherlands

A R T I C L E   I N F O

Keywords:
nD point clouds
Space filling curves
Data structures
Histograms
Skewed distributions

A B S T R A C T

Point cloud data contains abundant information besides XYZ, such as Level of Importance (LoI) and intensity. These non-spatial dimensions are also frequently used and queried. Therefore, developing an efficient nD solution for managing and querying point clouds is imperative. Previous researchers have developed PlainSFC that maps both nD points and queries into a one-dimensional Space Filling Curve (SFC) space and uses B+-tree for indexing. However, when computing SFC ranges for selection, PlainSFC subdivides the nD space mechanically to approach the query window without considering the point distribution. Then, excessive ranges are generated in vacant areas, and ranges generated in dense point areas are coarse. Consequently, a large number of false positives are selected, slowing down the whole querying process.

This paper develops a new solution called HistSFC to resolve the issue. HistSFC builds an nD-histogram which records point data distribution, and uses it to compute ranges for selecting data. Also, this paper discovers a novel statistical metric, Cumulative Hypercubic Coverage (CHC), to measure the uniformity of the point cloud data. Theory is established and it indicates that the nD-histogram is more beneficial when CHC is smaller. Thus, CHC can be used to guide the building of HistSFC. In addition, the paper conducts simulations and benchmark tests to examine the improvement on PlainSFC. It turns out that using the nD-histogram can decrease the false positive rate by orders of magnitude. HistSFC is also evaluated against state-of-the-art solutions. The result shows that HistSFC leads the performance in nearly all the tests.

1. Introduction

Point clouds are increasingly used in spatial related domains, from terrain modeling (AHN, 2014), forest estimation (Neuville et al., 2021), trajectory analysis (Zheng et al., 2009) to recently emerged semantic labeling (Tchapmi et al., 2017), virtual reality (Blanc et al., 2020), and autonomous driving (Chen et al., 2021). 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 modeling, a point may also record the flow direction and speed, sediment concentration, and other dimensions.

1.1. nD-PointCloud

We propose the term nD-PointCloud to cover the point cloud data containing multidimensional information. nD-PointCloud can be an independent 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. That is, many applications can be addressed with only nD-PointCloud. Besides, nD-PointCloud's advantage also lies in the ultra high accuracy which may be decreased when converting to rasters. Moreover, nD points are intuitive to interact with and convenient to analyze.

The dimension plays an important role in nD-PointCloud's support of different applications. Spatio-temporal dimensions are normally the fundamental dimensions for analysis considering the majority of applications, while the classification dimension is essential for semantic analytical purposes. Continuous Level of Importance (cLoI) (van Oosterom et al., 2022) can be additionally used to express importance

* Corresponding author.
E-mail address: liuhaicheng@csnwd.com.cn (H. Liu).

https://doi.org/10.1016/j.isprsjprs.2025.03.014
Received 11 October 2024; Received in revised form 5 March 2025; Accepted 13 March 2025

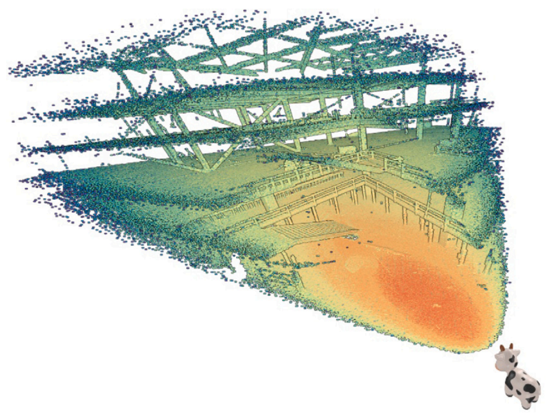


Fig. 1. Third-person view of a cLoI sample, from Schütz et al. (2019).

and alleviate the computational workload. Schütz et al. (2019), van Oosterom et al. (2022) add cLoI dimension to the data for querying, to smoothly and efficiently visualize large volumes of LiDAR points. When doing perspective view selection, all nearby points are selected, while fewer faraway points with restricted cLoI values are selected (Fig. 1). These dimensions can also be jointly used to support 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 cLoI. 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, cLoI and classification will form the query for navigation. As information continues to grow, more dimensions are expected in queries. Generally, we call them nD queries.

1.2. Motivation of the research

However, since the body of current spatial applications is still under 3D, most software for spatial data management adopts 2D/3D organization, e.g., Oracle's SDO_PC package (Oracle, 2019), PostgreSQL's pgPointcloud extension (Ramsey, 2020) and PDAL (PDAL-Contributors, 2018). To efficiently execute nD queries, all concerned dimensions are suggested to be used for clustering and indexing. van Oosterom et al. (2015) developed a prospective SFC mapping-based clustering and indexing framework, which we will call *PlainSFC*, for the sake of convenience for referencing. Basically, PlainSFC maps both multi-dimensional points and queries into a one-dimensional SFC space so that one-dimensional indexing structure such as the B+-tree can be used. PlainSFC distinguishes two types of dimensions. The *organizing dimension* is used to cluster and index the data, e.g., X, Y, Z and Time. They are transformed and mapped to the SFC space. The other *property dimensions* are affiliated to the SFC key, such as color and return number, which are not frequently used in the SQL WHERE clause.

The superiority of PlainSFC has been verified with different use cases (Psomadaki, 2016; Guan et al., 2018; Meijers and van Oosterom, 2018). However, these studies only use PlainSFC for managing and querying points within 4D. Our practical experiments indicate that PlainSFC performs inefficiently in higher dimensional spaces, especially when the data distribution is skewed. This is because PlainSFC adopts fixed recursive decomposition of the SFC space to generate SFC ranges for selection. This will result in a large amount of ranges containing no data when data is non-uniformly distributed. This increases the memory and time cost to compute ranges. It also increases I/O because of coarse ranges for selection in dense point areas.

This paper develops a technique to resolve this issue. The paper focuses on a common query type, the window query. It refers to a hyperrectangular query region formed by multiple dimensions. Unlike previous studies which propose their techniques and only demonstrate using specific test cases (Berchtold et al., 1998; Ooi et al., 2000), this paper provides a theoretical analysis of point data distribution and the effectiveness of the technique developed. In summary, the paper has the following contributions:

- We develop a novel solution called HistSFC that adopts an nD-histogram structure to overcome the limitations of PlainSFC caused by skewed data distribution. HistSFC achieves significant efficiency improvement in the window query on massive point cloud data. Query algorithms and optimizations are developed based on HistSFC.
- We theoretically demonstrate the effectiveness of HistSFC by proposing and using a statistical metric, Cumulative Hypercubic Coverage (CHC). We prove that CHC can measure the uniformity of the point cloud data and that it can indicate the performance gain by using HistSFC. Thus, CHC can be used to guide the building of HistSFC.
- We perform realistic benchmark tests on PlainSFC, HistSFC and other state-of-the-art solutions based on two novel applications: one is exploring 4D laser scanning point clouds; the other is analyzing flood risk using the 8D point cloud representation of hydraulic modeling results.

The remainder of the paper is organized as follows: Section 2 summarizes previous studies on organizing and indexing nD-PointCloud. Section 3 introduces the PlainSFC structure which is the basis for development. Section 4 describes HistSFC and the corresponding algorithms. This is followed by Section 5 which introduces CHC, including theoretical analysis and simulation results. Section 6 compares PlainSFC, HistSFC with other state-of-the-art solutions based on real applications. Section 7 concludes the paper.

2. Related work

Plenty of studies have been carried out to investigate the optimal data structures to manage point cloud data. Here, we categorize the techniques into three categories — R-tree and variants, the 2^n-tree and B+-tree. Besides, strategies to deal with skewed point data distribution are also discussed.

2.1. R-tree and variants

The 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 R-tree based solutions normally group points into blocks clustered by a specific order (e.g., Hilbert-R tree), and build index on these blocks. To further improve R-tree, variants including the R*-tree (Beckmann et al., 1990), SR-tree (Katayama and Satoh, 1997) and X-tree (Berchtold et al., 1996) are developed to decrease the overlap ratio between blocks, support frequent updates, and resolve nD queries. An R-tree based solution is tested in Section 6.4.

2.2. 2^n-Tree

2^n-tree represents an indexing category which evenly splits all dimensions in an iteration until the leaf node level. A leaf node normally refers to a block of points. In 2D, the 2^n-tree refers to Quadtree while in 3D, it becomes Octree. Potree, the most prevalent software for visualizing point clouds uses the Octree to organize data. CloudCompare, another software frequently used to analyze point clouds adopts the Octree index to facilitate analytical functions. To improve the performance on querying non-uniformly distributed data, adaptive data blocks can be adopted. That is, a threshold is to constrain the node's capacity so that a node with less number of points will not be split further (Wang and Shan, 2005).

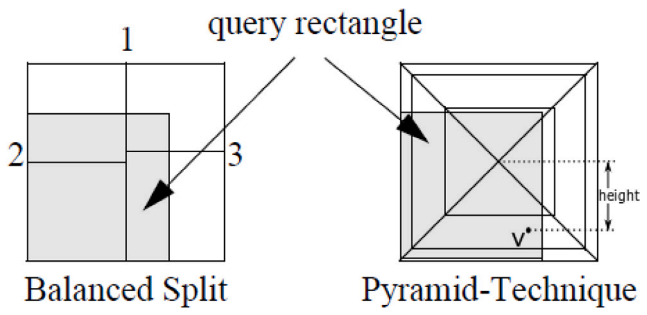


Fig. 2. Pyramid data partitioning, adapted from Berchtold et al. (1998).

2.3. B+-tree

B+-tree (Comer, 1979) is a variant of the B-tree which is widely used for indexing one-dimensional data. Unlike B-tree, the leaf nodes of the B+-tree are connected by pointers. Therefore, besides the top-down traversal, a leaf node can also be visited from its prior leaf node. B+-tree forms the basis for many nD-PointCloud solutions, such as Pyramid-Technique (Berchtold et al., 1998), iMinMax(θ) (Ooi et al., 2000) and Size Separation Indexing (SSI) (Zhang et al., 2014). For example, in high dimensional spaces, the Pyramid-Technique avoids excessive access to data pages by partitioning data into pieces which cater to the shape of hyper-cubes (Fig. 2). The approach maps each nD point into a one-dimensional space according to the pyramid piece the point belongs to and the height in the pyramid. All the resultant one-dimensional keys are then managed by a B+-tree structure to be indexed. The extended Pyramid-Technique improves Pyramid-Technique for handling non-uniformly distributed data by shifting all points to the cluster center of the data. In the benchmark test (Section 6.4), the Pyramid-Technique is examined.

2.4. Histograms

As a common technique to improve the querying efficiency given non-uniform data distribution, histograms are largely used in major DBMSs (e.g., statistics collection module). 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, 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 et al., 2021b). For instance, Achakeev and Seeger (2012) build a convenient spatial histogram based on R-tree, mainly for managing rectangular or point objects. 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 is another nD-histogram (Eavis and Lopez, 2007), and is basically a truncated version of the R-tree. It optimizes the nD-histogram using a k-uniformity metric which utilizes kd-tree to measure the uniformity inside a leaf node of the histogram. There are also other techniques such as STHoles (Bruno et al., 2001) and STHistogram (Roh et al., 2010) that are not based on R-tree. All these studies indicate the benefits of using nD-histograms for querying.

3. PlainSFC

This section introduces PlainSFC, which provides the preliminaries of our solution.

3.1. Terminology

When introducing data structures and queries, we use *node* and *range*. Fig. 3 illustrates them in 2D, where 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.

3.2. Basic settings

Fig. 4 presents the workflow 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 a table for indexing. Property dimensions are attached to each key. Based on this organization, PlainSFC utilizes Oracle Index-Organized Tables (IOTs) (Oracle, 2013) to manage the data. The internal data structure of IOT is a B+-tree, integrating the index and data storage.

For querying, PlainSFC adopts a two-step filtering mechanism. The first filter uses the Morton hierarchy to approximate the query window and derive the ranges. Take Fig. 5(a) 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. After this, the first filter exports all ranges into a range table and joins it with the IOT for selection (Fig. 4). This is followed by the second filter which conducts post-processing including decoding and point-wise checking to complete the query.

3.3. Time complexity

The querying time of PlainSFC includes two parts, the first filter and the second (Fig. 4). The time cost of the first filter comprises range computation, database fetching and other processes such as database initialization and communication. On the whole, the first filter costs $\mathcal{O}(r \log N)$ time, where r is the number of ranges generated and it reaches the threshold r_{max} by default; N represents the number of points stored. The expression is derived because range computation costs $\mathcal{O}(r)$ time and searching r ranges from IOT costs $\mathcal{O}(r \log N)$ time. The time cost of the second filter mainly covers the I/O cost of reading point data from the disk and post-processing. The cost of these processes is mostly determined by k' which is the result size from the first filter. Specifically, the I/O cost maximally covers $\mathcal{O}(k')$ I/Os. The post-processing time is bounded by $\mathcal{O}(nk')$. n refers to the dimensionality of the key, assuming the dimensions in the point cloud data are all used as the organizing dimensions.

The efficiency of the solution mainly depends on the performance of the first filter. An optimal first filter should on the one hand process fastly, while on the other hand return a small k'. This can alleviate I/O and post-processing in the second filter. We introduce False Positive Rate (FPR) to indicate the accuracy of the first filter (Eq. (1)). A large

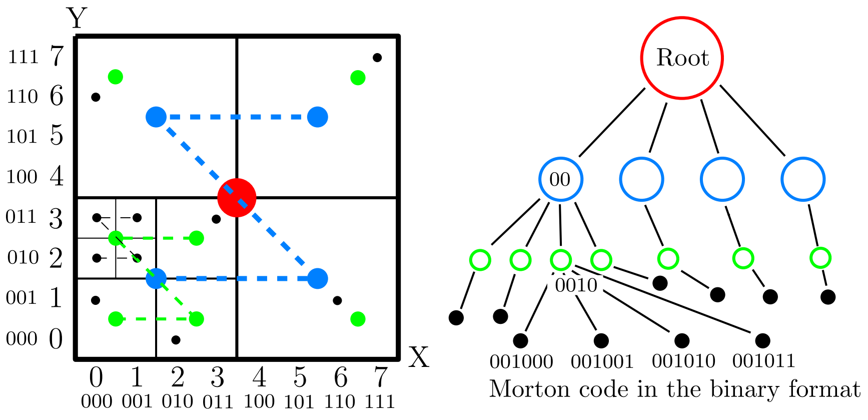


Fig. 3. Implicit Morton hierarchy: black dots are real points to be managed, while colored dots are Morton branch nodes at different levels.

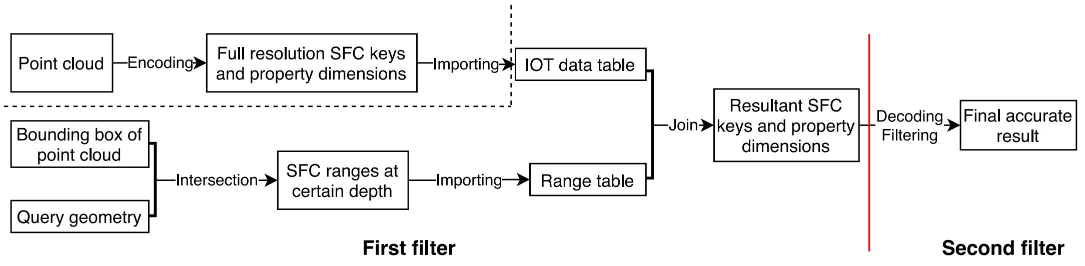


Fig. 4. The loading and querying procedure of PlainSFC, separated by the dash line.

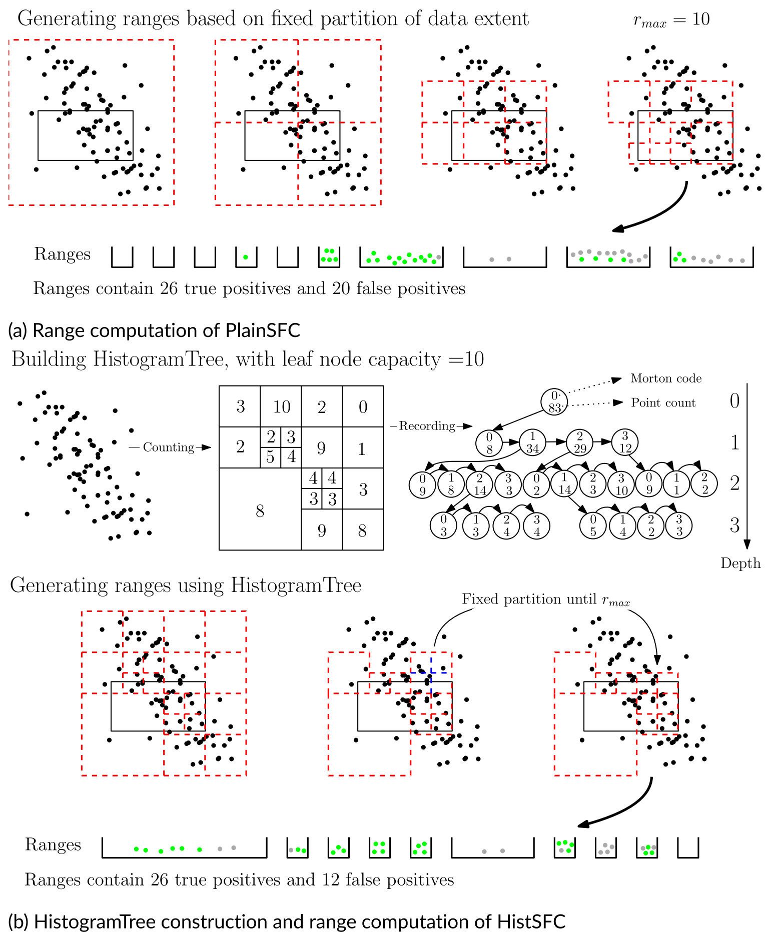

Generating ranges based on fixed partition of data extent
$r_{max} = 10$
Ranges
Ranges contain 26 true positives and 20 false positives
(a) Range computation of PlainSFC
Building HistogramTree, with leaf node capacity =10
Counting
Recording
Morton code
Point count
Depth
Generating ranges using HistogramTree
Fixed partition until r_{max}
Ranges
Ranges contain 26 true positives and 12 false positives
(b) HistogramTree construction and range computation of HistSFC

Fig. 5. Illustration of range computation in 2D, where r_{max} refers to the maximum number of ranges used for querying.

FPR means a coarse first filter and a slow second filter. Although FPR calculated may be larger than 1, it is an appropriate metric to indicate the querying efficiency (Section 6.4).

$$FPR = \left| \frac{k' - k}{k} \right|, \tag{1}$$

where k refers to the final output size.

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. In such cases, PlainSFC generates a large number of ranges without any points inside. This means that the budget for ranges that actually select points decreases. Consequently, ranges in dense point regions cannot be refined sufficiently. This increases FPR, and the overall querying efficiency declines.

4. Principles of an nD-histogram method

To resolve the defects of PlainSFC, a distribution-aware method is needed for computing ranges. The idea is to optimize the range computation to generate finer ranges where the point density is high, while generate coarser ranges where points are sparsely distributed. To this end, this section develops an nD-histogram solution — HistSFC, as described in the following.

4.1. HistogramTree

We assume all dimensions in the point cloud data are used as the organizing dimension, for the ease of explanation. We implement the nD-histogram by using a tree structure — HistogramTree. The C++ data structure of a node in HistogramTree is expressed as

```
STRUCT HistNodeND {HistNodeND *child; HistNodeND
     *neighbor; uint_256 key; long long
   pointcount; short height;}
```

`pointcount` records the number of points inside a node. If the number exceeds a threshold, i.e., the capacity of a leaf node, then the node is decomposed into 2^n children. `height` is used to distinguish different nodes, because branch nodes at different levels may possess identical keys. 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 window. It is also compact and is normally stored in a flat table. So, it is convenient to be loaded into the memory for querying. By using HistogramTree, the number of vacant ranges can be greatly diminished (Fig. 5(b)). We call the new solution HistSFC. HistSFC builds HistogramTree using HistSTREAM algorithm which is described in Appendix A.

4.2. HistSFC querying

HistSFC employs HistogramTree to compute ranges (Fig. 6). Starting from the root node, by performing intersection between the HistogramTree and the query window iteratively, the function retrieves all relevant nodes to build the range table (Fig. 4). Nodes inside the query window are immediately added to the range table without further processing. Nodes on the boundary containing few points 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. In fact, some nodes in the pool intersect with the query window by a large proportion. Then, most points inside these nodes are likely to be within the query window as well. By contrast, other nodes intersecting the query window by a small portion may introduce many false positives. Consequently, HistSFC computes the intersection ratio of each node which equals the volume of intersection divided by the volume of the node. These nodes are then ranked to be subdivided. The process stops when the refinement pool is empty or the number of ranges reaches the threshold. The rest of the querying process remains the same as PlainSFC.

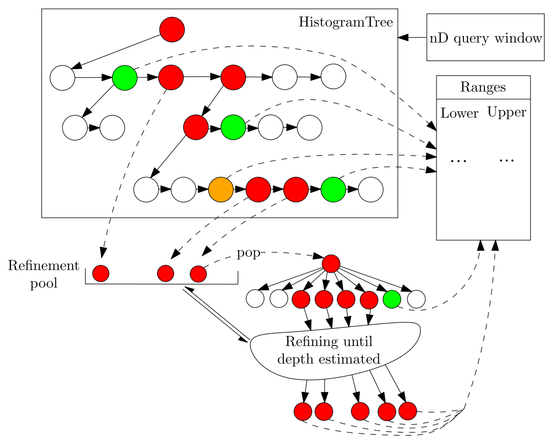


Fig. 6. Range computation using HistogramTree: with respect to the query window, green nodes are inside; red nodes are on the boundary; orange nodes are on the boundary but with few points; white nodes are outside. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.3. Parallel decoding

Although HistSFC decreases FPR compared to PlainSFC (Eq. (1)), the decoding process of the second filter can still be time consuming if the result contains many keys. To address this issue, HistSFC adopts parallel technique for decoding. HistSFC distributes the ranges generated by the first filter to different processors so that each processor executes a part of the query and decodes the result. As each range contains a different number of points due to skewed data distribution, the actual workload can be unbalanced among processors. To improve this, HistSFC ranks the ranges according to their lengths, assuming that the length represents the number of points inside. This is reasonable as the result is computed using the nD-histogram. Then, each range is randomly assigned to a processor in the processor pool. Each processor handles all its ranges and fetches the point data on the disk.

5. Cumulative hypercubic coverage

In principle, HistSFC improves the query performance given skewed data distribution. However, to what extent can HistSFC improve the performance? How is this related to specific data distributions? As the dimensionality of point clouds is always limited, we conduct a theoretical analysis to investigate the effectiveness of HistogramTree. This section proposes a metric called Cumulative Hypercubic Coverage (CHC) to quantify the uniformity of nD points. Then, the section analyzes the relationship between CHC and the effectiveness of HistogramTree in querying. In addition, simulation is performed to learn how the effectiveness of HistogramTree changes with CHC.

5.1. Definition

The idea of CHC comes from a basic question: is it possible to quantify the uniformity of a set of points on a 2D plane? We may think of using area to indicate the uniformity. This is because when the points are spread over the plane, the area seems to be larger than the case when the points are clustered. However, as a point has no area, to quantify such a measure, we need to build cells with area around each point for evaluation. Besides, as the goal is to measure uniformity, we also want the area measure to be independent from the number of points (i.e., N) and only relate to the distribution. To achieve this, the cell size should be varied according to N. We choose the square as the shape of each cell, which becomes the hypercube in nD space. We name the cell created for the measurement the *associated cell* for each point.

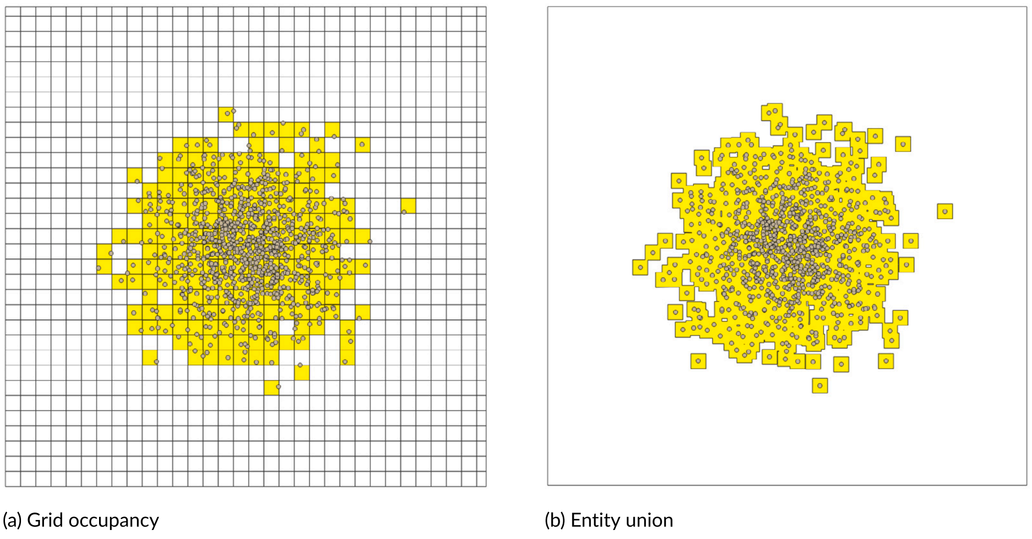


Fig. 7. Two approaches computing the 2D cumulative hypercubic coverage.

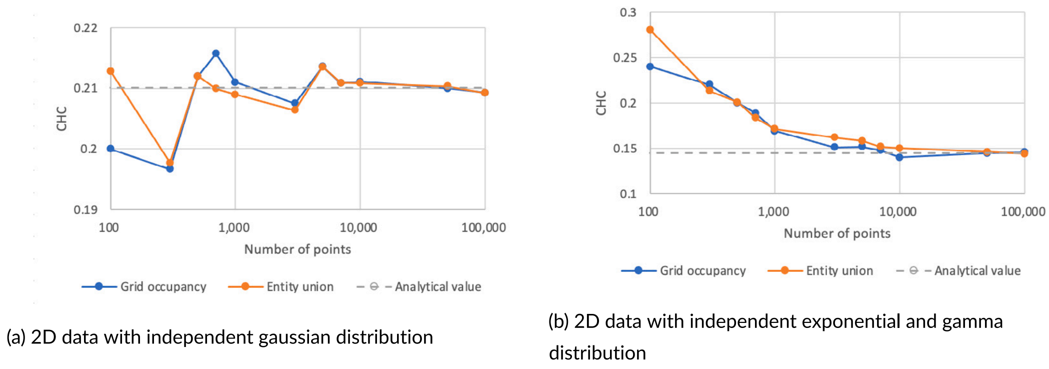


Fig. 8. Numerical simulation of CHC using two computing approaches.

Definition. Given a set of nD points, cumulative hypercubic coverage refers to the sum of standardized hypercubic volume of all associated cells after deduplication.

We provide two approaches to compute CHC, and later prove that they approach to the same expectation when N becomes arbitrarily large. 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:

Grid occupancy: We divide the domain into N nD-cells with equal size (Fig. 7(a)). 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, $CHC = \frac{1}{V}\frac{\text{count of occupied cells}\cdot V}{N} = \frac{\text{count of occupied cells}}{N}$.

Entity union: For each point p, we use it as the center to build an nD-cell c (Fig. 7(b)) of which the edge length at Di equals $\frac{E_{Di}}{\sqrt[n]{N}}$. Then, $CHC = \frac{hypervolume(\bigcup c)}{V}$.

Theorem 1. *Given a point set following a specific distribution, when N becomes arbitrarily large, the expectation of grid occupancy is a constant which is only determined by the joint Probability Density Function (PDF).*

Theorem 2. *The expectation of entity union converges to the same constant as grid occupancy, when N becomes arbitrarily large.*

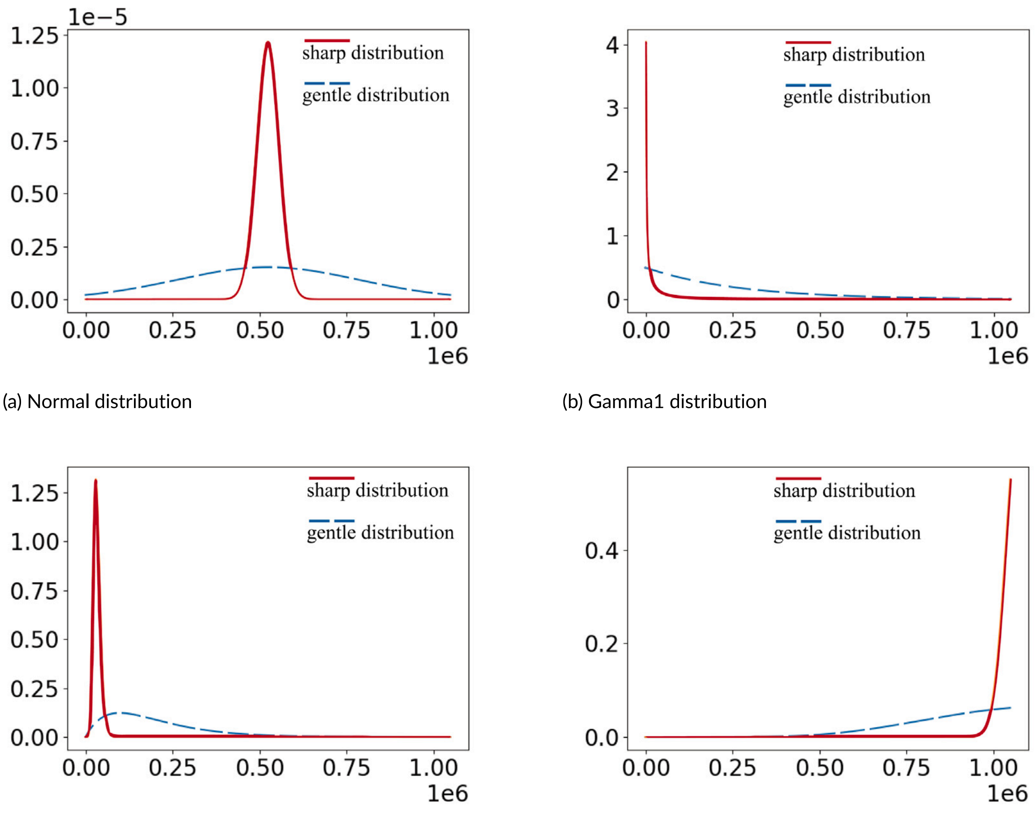
Appendix B provides the proof. We also perform simulations to illustrate the theorems. In Fig. 8(a), the CHC of a 2D point cloud is computed. The dimension X and Y are independent from each other, both following $\mathcal{N}(0.5, 0.1)$ (shown in Fig. 7). Fig. 8(b) shows the CHC of another 2D point cloud, where X follows exponential distribution $E(12.5)$ and Y follows gamma distribution $\Gamma(2, 0.08)$. The figures indicate that the CHC values computed by both approaches gradually converge to the true value which can be computed (Eq. (6) in Appendix B).

Compared with entity union, grid occupancy is more convenient and efficient to compute. For example, a DBMS flat table PC stores N points with columns $d1$, $d2$, $\dots dn$, corresponding to different dimensions. The SQL command "SELECT COUNT(ct)/N from (SELECT COUNT(*) AS ct from PC group by TRUNC(d1/el_{D1}), TRUNC(d2/el_{D2}), ...TRUNC(dn/el_{Dn}))" can be used to derive CHC. el_{Di} refers to the edge length of a cell in Di, which equals $\frac{E_{Di}}{\sqrt[n]{N}}$.

Uniformity metrics for point clouds have been proposed before (Gunzburger and Burkardt, 2004; Ong et al., 2012). Compared with them, CHC is computed based on accumulating the hypercubes, which keeps consistent with the querying strategy of HistSFC. Therefore, CHC can be an appropriate metric to explore the relationship between the effectiveness of HistogramTree and the uniformity of data.

5.2. Effectiveness of nD-histogram

As mentioned, there exist a fraction of the ranges computed by PlainSFC containing no points. We define an *effective range* as a range exported by PlainSFC that contains at least one point, no matter whether the point is inside the query window or not. Assume the



(a) Normal distribution (b) Gamma1 distribution

(c) Gamma2 distribution (d) Gamma3 distribution

Fig. 9. Simulated probability density functions.

capacity threshold of HistogramTree is 1 (i.e., the highest precision). Obviously, HistSFC only returns effective ranges. Then, we use Eq. (2) to measure the effectiveness of HistogramTree with respect to a query. Theorem 3 shows how the effectiveness is related to CHC.

$$E_{hist} = \frac{\text{Number of ranges exported by PlainSFC}}{\text{Number of ranges exported by HistSFC}} \tag{2}$$

Theorem 3. *Given a point cloud containing N points. Suppose the points can move freely in the domain so that the uniformity is changing. For a window query, the expected E_{hist} is a monotonically decreasing function of CHC.*

The proof is provided in Appendix B. We derive a general pattern from Theorem 3: given two nD point clouds A and B where $CHC_A > CHC_B$, for a large number of window queries, $\overline{E_{hist}}(A) < \overline{E_{hist}}(B)$. That is, with the decrease of CHC, the benefit of using HistSFC increases. When CHC approaches 0, E_{hist} becomes arbitrarily large and the vacant ranges generated by HistSFC will be much less than that of PlainSFC. The smallest E_{hist} equals 1, which happens when points follow a chessboard distribution (i.e., CHC = 1). In that case, HistogramTree is not needed.

5.3. Realistic simulation

To further evaluate the effectiveness of HistogramTree, we conduct another experiment with realistically simulated data. Using simulation, we guarantee that CHC values of the data generated can be of different orders of magnitude. We first collect point clouds from different sources such as indoor laser scanning and Airborne Laser Scanning (ALS), and study the distributions of various dimensions involved. This test simulates 6 dimensions with different distribution types that we derived

Table 1
Distributions designed in the realistic simulation.

	Gentle	Sharp
Uniform	$U(0, 2^{20})$	$U(0, 2^{20})$
Normal	$\mathcal{N}(2^{19}, 2^{18})$	$\mathcal{N}(2^{19}, 2^{17})$
Gamma1	$\Gamma(1, 2) \times 2^{17}$	$\Gamma(0.05, 1) \times 2^{17}$
Gamma2	$\Gamma(2, 3) \times 2^{15}$	$\Gamma(10, 0.1) \times 2^{15}$
Gamma3	$\Gamma(10, 2) \times 2^{16}$	$\Gamma(820, 0.02) \times 2^{16}$

(Table 1 and Fig. 9). The value of each dimension is based on 20 bits, ranging from 0 to 2^{20}.

The test builds 10 data sets in 3D, 4D, 5D and 6D, respectively. Each data set contains 10^7 points. The first two dimensions of these data sets always follow the uniform distribution, while other dimensions are generated by randomly choosing the distribution from Table 1. Then, the test adopts the regular procedure to build HistogramTree based on capacity threshold: 3D and 4D solutions use 100 as the capacity, while 5D and 6D adopt 1000, instead. When querying, the test randomly generates 500 nD query windows with varying edge lengths at each dimension. The maximum number of ranges for 3D and 4D querying is 1000; while that for 5D and 6D querying is 10,000. These settings are typical given corresponding data. We adopt a variant of E_{hist} to evaluate the effectiveness of HistogramTree, which is E'_{hist} (Eq. (3)). It is more appropriate because PlainSFC and HistSFC returns the same number of ranges in this experiment:

$$E'_{hist} = \frac{FPR_{PlainSFC}}{FPR_{HistSFC}} \tag{3}$$

where $FPR_{PlainSFC}$ and $FPR_{HistSFC}$ stand for FPR of PlainSFC and HistSFC, respectively.

Fig. 10 presents the medians of E'_{hist} from all the tests. On the whole, HistogramTree works more effectively when the data set possesses a

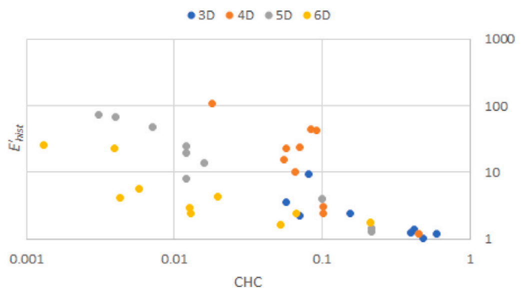


Fig. 10. Relationship between CHC and E'_{hist} derived from the realistic simulation.

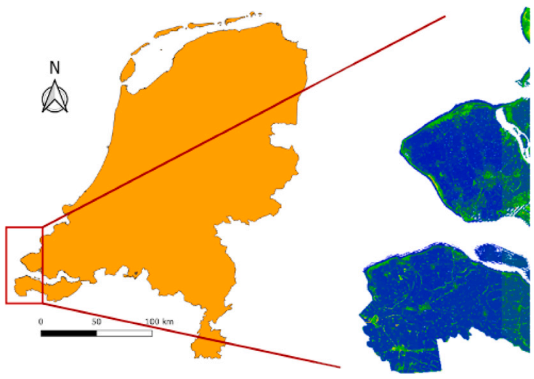


Fig. 11. The AHN2 sample used for benchmarking.

smaller CHC. The results indicate that when the CHC value is smaller than 0.1, HistogramTree becomes essential to use. This is because in many cases, the FPR can be decreased by orders of magnitude, especially when CHC is below 0.01. In practice, as CHC is convenient to compute based on the occupancy grid, developers are suggested to first measure the CHC before building the HistogramTree. If a point cloud is too large, random sampling can be performed to derive CHC since it is only influenced by the point distribution.

6. Experimental evaluation

After acquiring the convincing results of HistSFC by theoretical analysis and simulation, we evaluate the performance in practice. This section elaborates benchmark results on real data. Section 6.1 lists the other state-of-the-art solutions for benchmarking. Section 6.2 describes two use cases on which the benchmark is based. Section 6.3 compares HistSFC and PlainSFC. This is then followed by an overall comparison including HistSFC and state-of-the-art solutions in Section 6.4. Section 6.5 discusses the results and provides more aspects of the use of HistSFC. 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.

6.1. State-of-the-art solutions

Appendix C presents the querying process of state-of-the-art solutions for benchmarking. The final results of all solutions are stored as C++ in-memory objects, before being exported to the disk.

(Extended) Pyramid-Technique Section 2.3 describes the principle of both Pyramid-Technique and extended Pyramid-Technique. They adopt the same architecture for querying.

PostGIS The pgPointcloud extension (Ramsey, 2020) can maximally support two organizing dimensions, which performs inefficiently in higher dimensional queries. Thus, it is not used in the benchmark test.

Table 2
Storage size of AHN2 data sets on the disk (GB).

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

Table 3
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%

Instead, we implemented a 4D solution based on the 4D MultiPoint geometry. The original "M" dimension is replaced by cLoI (Section 6.2.1). To keep in line with HistSFC, a MultiPoint object which corresponds to a leaf node of HistogramTree is created to store point data. Such an object can be regarded as a 4D block. Then, a 4D R-tree is built on all MultiPoint objects for indexing.

SDO_PC Oracle SDO_PC solution (Oracle, 2019) partitions point cloud data into 2D blocks and adopts Hilbert R-tree for indexing. In the flood data test (Section 6.2.2), we use X and Y dimension to organize data and create blocks.

6.2. Use cases

We investigate two use cases: the AHN2 exploration in Section 6.2.1 concerns a 4D point cloud, while the flood risk querying in Section 6.2.2 uses an 8D data set.

6.2.1. AHN2 exploration

AHN2 is an ALS point cloud recording the terrain elevation of the Netherlands (AHN, 2014), with a density of 6 — 10 points/m^2. We cropped a sample which locates at the southwestern part of the Netherlands (Fig. 11), containing 10 billion points. The XYZ bounding box is [13427.6, 359007.3, −8.8; 38000, 415990.9, 119.7] in spatial reference system Amersfoort/RD New, EPSG:28992. cLoI is used to represent the importance of a point. It improves the performance on visualizing large point clouds where less important points may not have to be rendered (Fig. 12). We add the cLoI dimension into AHN2 as the fourth organizing dimension.

In order to learn the scalability of different solutions, we split the data into five vertical slices from west to east. Starting from the first piece which is Data set 1, by adding one more slice each time, five different data sets are built. Table 2 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.

For querying, we devise the query window sizes and locations by considering query logs and testing requirements (e.g., diverse selectiveness of queries defined by Eq. (4)). The selectiveness of the 5 query windows used for benchmarking is presented in Table 3.

$$selectiveness = \frac{\text{Number of points within the query range}}{\text{Total number of points}} \tag{4}$$

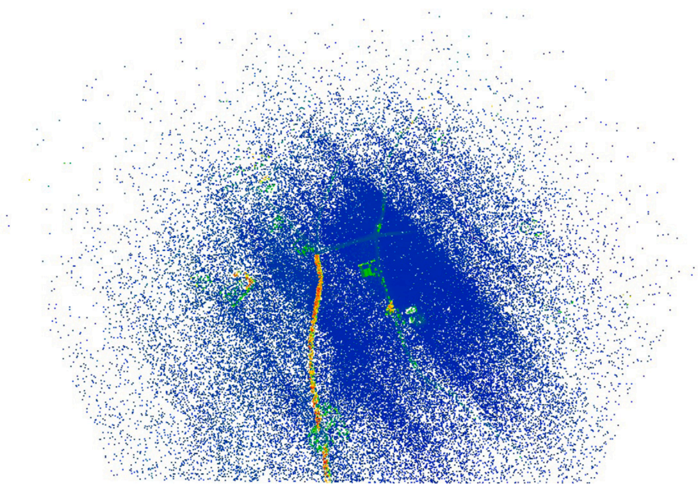


Fig. 12. A bird-view selection of the AHN data using cLoI. Only important points are rendered on the periphery.

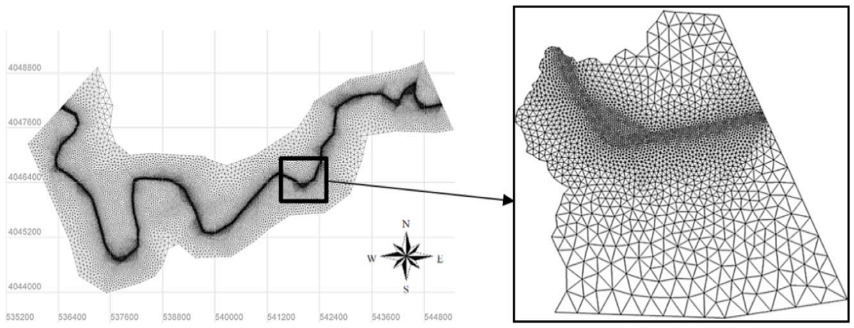


Fig. 13. A typical flood modeling grid along a river, from Gharbi et al. (2016).

6.2.2. Flood risk querying

Flood risk mapping projects generate huge amounts of modeling 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 mesh covering the modeling basin (Fig. 13). The modeling results are then used for making various maps such as the maximum inundation map and inundation duration map, in a following step. The water authorities collect these final products, and use them for decision making. However, a large part of original modeling results are omitted because they are cumbersome to manage, analyze and present. This certainly has drawbacks: the products are "static" and no more details can be derived; the maps fail to address new requirements.

In fact, any specific flood map can be expressed and formed by a type of query. 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. Due to the irregular grid (Fig. 13), 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-PointCloud database using all these relevant dimensions. This section demonstrates this with a use case in China (Liu et al., 2021a).

The project models 8 cases (i.e., initial conditions), and each case simulates 720 time steps with 30-min resolution. In total, we get $59{,}680 \times 720 \times 8 = 343{,}756{,}800$ points in an 8D space composed by case ID, X, Y, Z, time, depth, velocity and direction. To explore the scalability, we divide 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 4 lists the storage size of different solutions. HistSFC's size includes IOT and HistogramTree.

Table 4
Storage size of flood data sets on the disk (GB).

Data set	Number of points	Raw TEXT	HistSFC	SDO_PC
1	42,969,600	2.69	1.67 + 0.002	3.63
2	85,939,200	5.41	3.34 + 0.004	7.26
3	171,878,400	10.8	6.67 + 0.009	14.1
4	343,756,800	20.7	12.9 + 0.017	33.3

In practice, the flow direction is infrequently used for risk analysis compared with other dimensions. So, we set flow direction as the only property dimension when building HistSFC. We choose to implement SDO_PC because Oracle is widely used to manage water data. So, it is very convenient to be used directly for point clouds. Besides, HistSFC is also implemented in Oracle and this can achieve fair comparison in terms of architecture.

According to practical experience and potential needs, we devised 4 queries for testing (Table 5). As these queries all concern case 1, the execution using different data sets will return the same results.

6.3. HistSFC vs PlainSFC

This section investigates to what extent HistSFC improves the performance of PlainSFC. We tested different size of HistogramTree to

Table 5
Flood queries used for benchmarking.

Query	Description	Org. Dimensions	Num. Points
DEPTH3 m	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) within 24 h, 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 rectangular area), in case 1	caseID, depth, X, Y	170,417

Table 6
Memory consumption of HistogramTree in AHN2 test (MB).

Data set	Capacity threshold		
	1,000	10,000	100,000
1	135	11.6	1.18
2	265	23.2	2.38
3	531	46.8	4.77
4	1,408	124	11.9
5	2,615	242	24.1

Table 7
False positive rate using different HistogramTrees in AHN2 test.

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	0.100	0.124	0.285
Large	0.028	0.033	0.063	0.136

Table 8
Memory consumption of HistogramTree in the flood data test (MB).

Data set	Capacity threshold		
	1,000	5,000	10,000
1	13.8	2.70	1.74
2	27.4	5.38	3.62
3	54.7	10.8	7.13
4	105	20.5	13.6

Table 9
False positive rate using different HistogramTrees in the flood data test.

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

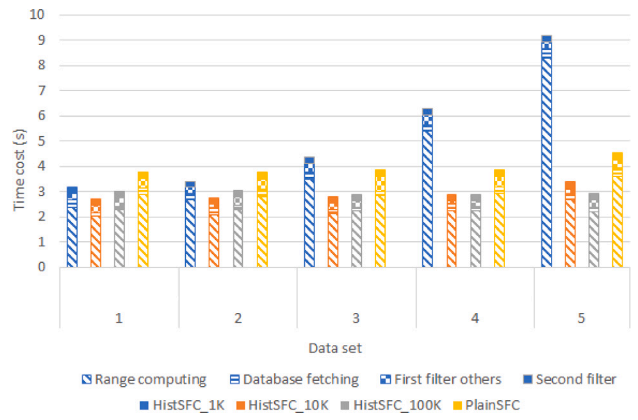


Fig. 14. Time cost of SmallC query using different HistogramTrees.

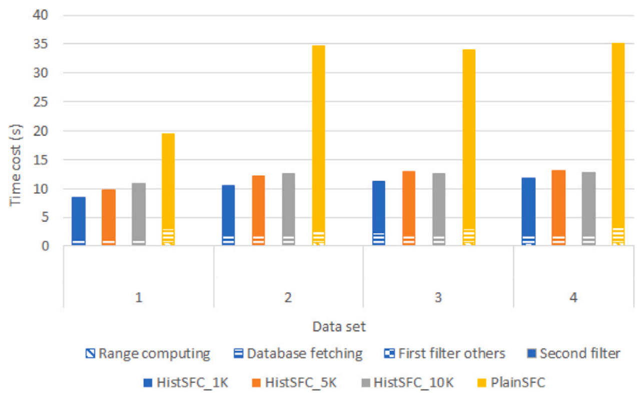


Fig. 15. Time cost of ARRIVAL24h using different HistogramTrees.

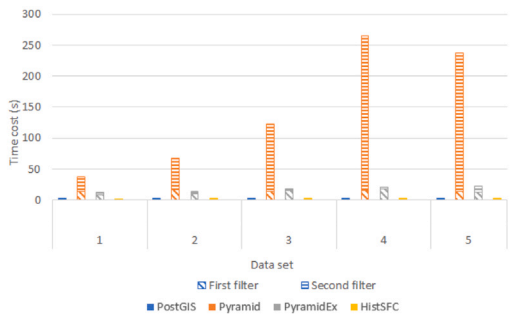
learn how this influences the querying efficiency. We adopted neutral number of ranges for querying. That is, 1 million for AHN2 test and 100,000 for the flood data test.

Table 6 presents the memory consumption in AHN2 test. Table 7 presents FPR of different solutions. HistSFC_1K refers to HistogramTree with a capacity threshold of 1000. Overall, using HistogramTree significantly decreases FPR of PlainSFC. HistSFC_1K performs the best, and it at least decreases FPR by half compared to PlainSFC.

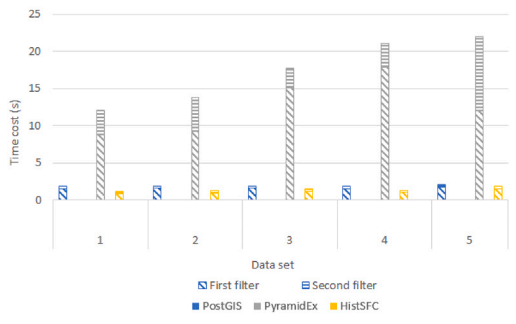
Table 8 presents the memory cost of HistogramTrees in the flood data test. As the data set is smaller than AHN2, smaller node capacities are tested. As the tables show, HistogramTrees' sizes are much smaller here than in the AHN2 experiment. They are all below 100 MB. Table 9 presents FPRs for flood queries. The FPR can be decreased by orders of magnitude after using HistogramTree. In fact, HistogramTree's effectiveness is closely related to CHC, where the CHC of AHN2 data is 0.0095 and that of the whole flood data is 0.0006758.

Fig. 14 shows the querying time cost of different solutions in the AHN2 SmallC test. Appendix D provides exact time measurements. Other tests present analogous patterns. The time cost is composed by the first filter time and the second (Section 3.3). On the whole, HistSFC_10K and HistSFC_100K perform the best. PlainSFC follows behind, while HistSFC_1K ranks last. In most cases, the gap between PlainSFC and HistSFC_10K or HistSFC_100K is not large. The main reason lies in the high uniformity of the data, and the effectiveness of HistogramTree is limited. In contrast to the favorable performance in FPR (Table 7), HistSFC_1K degrades remarkably in time cost as data size increases. This is mainly caused by traversing the huge HistogramTree in the first filter which takes an enormous amount of time. Besides, HistogramTree first has to be loaded into memory to use. The loading process of HistSFC_1K of Data set 5 can take 150 s, which is unacceptable.

Fig. 15 shows the time cost of the flood ARRIVAL24 h test as a representative. Appendix D provides exact time measurements. HistSFC_1K, HistSFC_5K and HistSFC_10K respond 2× to 5× faster than PlainSFC. This is due to higher FPRs (Table 9) of PlainSFC which increases I/O and decoding time cost. Compared to the AHN2 test, HistogramTree



(a) Overall



(b) Omitting Pyramid

Fig. 16. Time cost of SmallA query using state-of-the-art solutions.

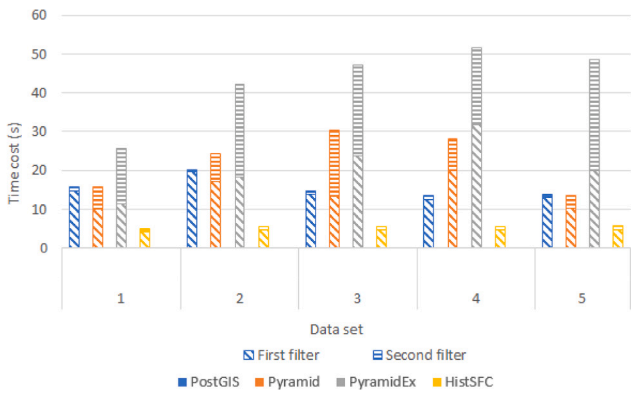


Fig. 17. Time cost of SmallB query using state-of-the-art solutions.

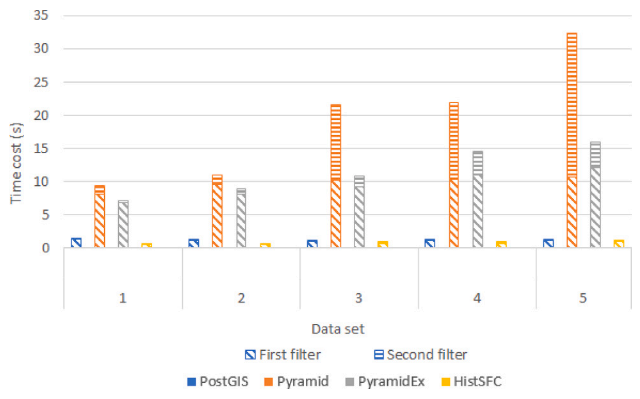


Fig. 18. Time cost of SmallC query using state-of-the-art solutions.

works more effectively for this non-uniformly distributed data. The time cost of HistSFC solutions do not vary significantly. In the following, we choose HistSFC_5K as the optimal solution because it balances the performance and resource consumption.

6.4. HistSFC vs State-of-the-arts

To further learn HistSFC's performance, we perform benchmark tests to compare HistSFC and other solutions using AHN2 and the flood data. In the AHN2 test, the leaf node capacity of HistogramTree is set to 10,000 (Table 6), and the number of ranges for querying is 100,000. These parameters are acquired by extra experimenting and tend to be optimal. For PostGIS, the capacity of MultiPoint object is 10,000 points. Figs. 16–20 present the time cost of different solutions. Appendix D provides all related time measurements. For all solutions, we implement parallel post-processing with 32 processors.

From Figs. 16–20, HistSFC always takes the least time to execute. PostGIS ranks behind, while Pyramid and PyramidEx (i.e., extended Pyramid-Technique) are the slowest solutions. More specifically, in SmallA and SmallC, PostGIS spends slightly more time than HistSFC. However, SmallB is different, where HistSFC is 3× to 5× faster than PostGIS. 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. For these two queries, HistSFC spends most of the time on the second filter while less than 2 s on the first filter. With respect to scalability, PostGIS scales constantly as input data size increases. HistSFC presents a slightly increasing pattern in terms of scalability. This is mainly attributed to the growing HistogramTree's size which causes more traversing time.

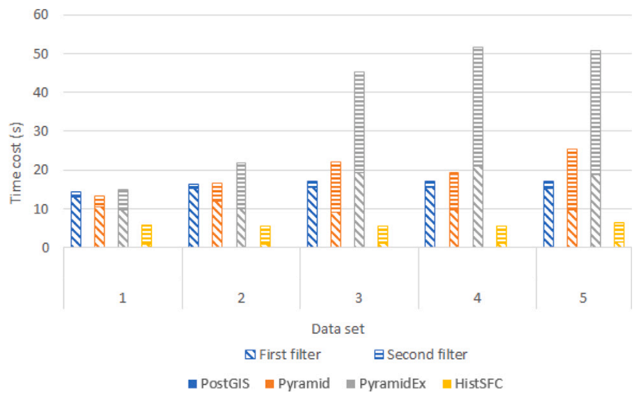


Fig. 19. Time cost of Medium query using state-of-the-art solutions.

Pyramid and PyramidEx spend much more time on querying, and the performance fluctuates significantly. This is caused by the large and changing FPR (Table 10). 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. This brings large number of false positive points, increasing FPR. On the other hand, the FPR may also decrease when the input data size increases. 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. PyramidEx may not always outperform Pyramid. In SmallB, PyramidEx returns higher FPR for certain query windows, which is caused by the window positions. Besides the time cost, the final output of Pyramid and PyramidEx may

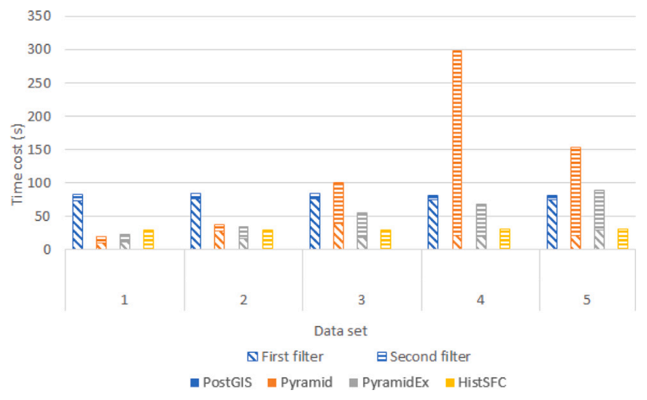


Fig. 20. Time cost of Large query using state-of-the-art solutions.

Table 10
False positive rate of state-of-the-art solutions in AHN2 test.

	Data set	SmallA	SmallB	SmallC	Medium	Large
HistSFC	All	0.33	1.92	0.16	0.18	0.08
PostGIS	All	1.57	7.14	0.66	0.36	0.16
Pyramid	1	1739.32	336.69	32.85	14.77	5.73
	2	3328.40	677.63	84.20	29.58	13.34
	3	5739.03	862.65	708.46	54.45	30.07
	4	11 345.91	612.88	585.33	33.65	50.16
	5	13 568.89	170.72	1405.94	67.47	35.60
PyramidEx	1	255.80	689.05	20.43	23.87	6.40
	2	338.27	1148.89	41.91	44.66	9.36
	3	408.23	1083.39	84.84	86.67	12.12
	4	465.24	1266.48	263.35	101.90	14.34
	5	757.63	1733.64	432.85	139.94	20.01

not be accurate. This is because the pyramid value is not so precise to avoid errors at the query boundaries.

In the flood data test, HistSFC uses 5000 as the leaf node capacity (Table 8) and 100,000 ranges for querying. SDO_PC adopts 5000 as the block capacity. In Fig. 21, from DEPTH3 m to EXTENTmax, HistSFC responds significantly 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. However, SDO_PC shows superior performance in HOUSErisk. This is because the query uses XY range, which caters to the strength of SDO_PC which implements efficient 2D intersection. Only a few blocks are retrieved in HOUSErisk, and incurs small I/O cost. By contrast, HistSFC adopts 7D data organization with one dimensions as the property dimension, which means more nodes are examined for intersection. This also introduces more false positive points with velocity below 0.5 or outside the road, increasing the FPR. Overall, the test indicates that SDO_PC is only preferable for 2D spatial queries, while HistSFC is advantageous in queries concerning different combinations of dimensions.

6.5. Discussion

The benchmark tests show that HistSFC is the most favorable solution from all tests. HistSFC efficiently generates accurate SFC ranges for selection using HistogramTree, and does not need to unpack blocks due to its IOT storage. PostGIS performs very efficiently in retrieving blocks (i.e., MultiPoint objects), and thus functions efficiently for queries with small output. However, with more blocks selected, PostGIS spends significantly more time on unpacking them. Yet, both PostGIS and SDO_PC fail to address high dimensional queries efficiently due to their storing strategies. Pyramid and PyramidEx are originally devised for very high dimensional hypercubic window queries. However, they perform less efficiently in our experiment. The specific pyramid decomposition of the space causes very large FPRs.

As the input data size rises, HistogramTree's memory cost goes up (Tables 6 and 8). We also observe that the growing HistogramTree's size causes increasing time cost of the first filter. However, the size (e.g., 242 MB of HistSFC_10K) is moderate given current hardware settings, and the induced time cost in the first filter is insignificant unless we adopt HistSFC_1K. In fact, the optimal size of HistogramTree depends on the data and the implementing environment, and can be derived by benchmarking in general. Considering prevalent settings of hardware and typical applications of AHN2, HistogramTree is suggested to be kept under 1 GB. If the data size continues to increase, we may develop a block based HistSFC. That is, we group points to blocks as the leaf nodes of IOT. Then, to reduce the block unpacking workload, we can adopt a B+-tree to organize and index the data inside each block. Then, HistSFC is able to retrieve and unpack only the corresponding part of the block involved in a query by adopting the internal index. This remains to be future work. Besides, we suggest using CHC as an initial indicator to employ HistSFC. The simulation shows that when CHC is greater than 0.1, PlainSFC is also appropriate to use (Section 5.3).

To efficiently use PlainSFC and HistSFC, it is crucial to determine the organizing dimensions. In fact, the set of organizing dimensions balances the variety of queries and efficiency. That is, if we want to add more organizing dimensions to support more types of queries, the efficiency of every type of query somewhat declines due to the high dimensionality of the node. This is because high dimensional nodes generate a large number of child nodes by partitioning once, but r_{max} is confined by the memory size. So, the selected nodes may not be refined sufficiently to reduce false positives. Thus, it is significant to perform a systematic analysis of the application and only use dimensions that are queried frequently to organize data. For spatial applications, we should first consider using X and Y as organizing dimensions. Next, the Z, time, cLoI, and classification may be used in addition. It is likely that we may not foresee all possible applications in advance, and a crucial new application requires conducting queries on property dimensions. Then, we suggest performing benchmark tests to compare different data organizations, e.g., by adopting different organizing dimensions considering query frequency and selectiveness (Eq. (4)). In the end, we may need to reorganize the data storage or build another copy to achieve the best performance. More knowledge of nD-PointCloud data organization can be acquired from Meijers and van Oosterom (2018), Psomadaki (2016), Liu et al. (2021c).

7. Conclusions

To improve the range computation of PlainSFC on window queries, this paper develops an nD-histogram approach — HistSFC. It builds a HistogramTree structure and uses it to generate more accurate ranges for selection. The paper discovers a statistical metric, CHC, to quantify the uniformity of point data. It is revealed that CHC is only determined by the data distribution regardless of the number of points. Theory shows that HistogramTree works more effectively on data sets which possess larger CHC values. Both simulation and benchmark tests indicate that when the CHC value is below 0.01, HistSFC can reduce the FPR of PlainSFC by at least 50%. As a consequence, the time cost decreases evidently. The performance gain becomes more significant as CHC further decreases. HistSFC is also compared with state-of-the art solutions. Overall, HistSFC responses the fastest. In certain cases, it takes less than 10% of the time of the others. It presents fine scalability as well as stable performance in nearly all querying tests. Related code of the research can be assessed at https://github.com/rencailhc/HistSFC.

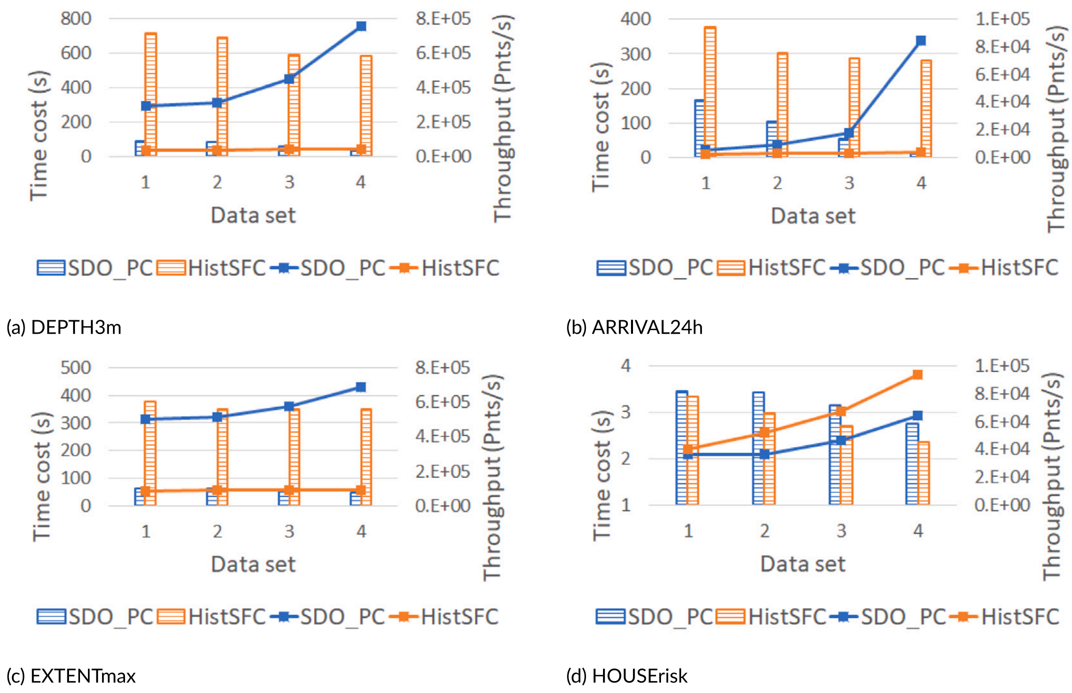


Fig. 21. Time cost (lines) and throughput (bars) of different solutions on flood querying.

CRediT authorship contribution statement

Haicheng Liu: Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Conceptualization. **Zhiwei Li:** Validation, Resources, Project administration, Investigation, Funding acquisition. **Peter van Oosterom:** Supervision, Methodology, Conceptualization. **Martijn Meijers:** Validation, Supervision, Investigation. **Chuqi Zhang:** Writing – review & editing, Validation.

Declaration of competing interest

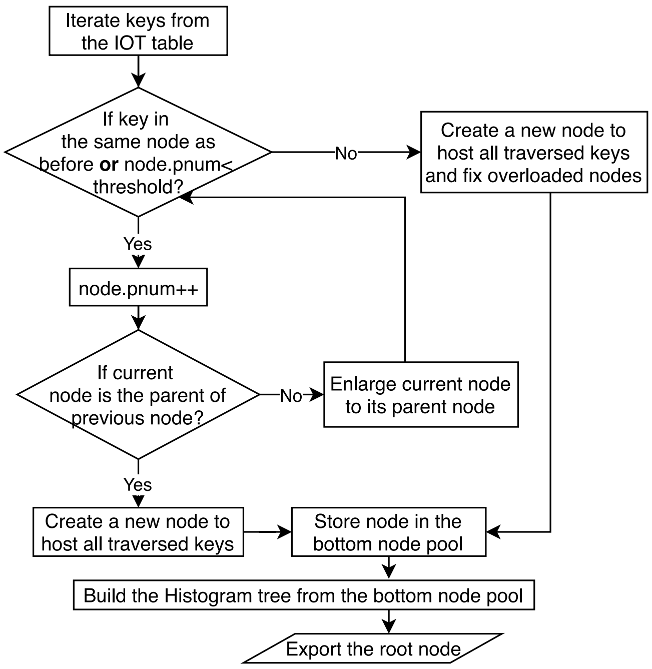
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank Rodney Thompson, Jianping Wang, Zhida Chen, and Jiong Xie for their help in preparing the manuscript.

Appendix A. HistogramTree construction

Fig. 22 presents the procedure of building HistogramTree, called HistSTREAM. Basically, HistSTREAM reads sorted Morton keys sequentially and meanwhile computes a node that could be the parent of all traversed keys. This stops until reading a key that belongs to the sibling or parent of the current node. The process will also stop when the number of keys traversed exceeds the leaf node's capacity. Then, HistSTREAM returns to the beginning of this traversal and creates the nodes. HistSTREAM continues and repeats such process until the scanning of IOT is completed. Then, all nodes created are aggregated till the root node. 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 data sorting has been done and IOT has been built. Besides, the memory usage mainly depends on the leaf node capacity, and will not keep increasing as the input size grows. Moreover, HistSTREAM can be a fully streaming process.



Iterate keys from the IOT table
If key in the same node as before or node.pnum< threshold?
No
Create a new node to host all traversed keys and fix overloaded nodes
Yes
node.pnum++
If current node is the parent of previous node?
No
Enlarge current node to its parent node
Yes
Create a new node to host all traversed keys
Store node in the bottom node pool
Build the Histogram tree from the bottom node pool
Export the root node

Fig. 22. HistSTREAM.

Appendix B. Proof of CHC related theorems

Theorem 1. *Given a point set following a specific distribution, when N becomes arbitrarily large, the expectation of grid occupancy is a constant which is only determined by the joint Probability Density Function (PDF).*

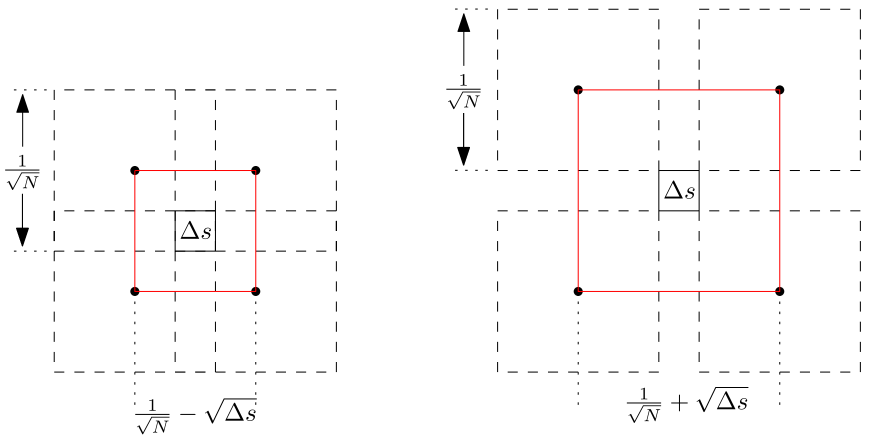

$\frac{1}{\sqrt{N}}$
Δs
$\frac{1}{\sqrt{N}} - \sqrt{\Delta s}$
$\frac{1}{\sqrt{N}} + \sqrt{\Delta s}$

Fig. 23. The region used to compute $E(s)$ which is the red box, with lower bound on the left, upper bound on the right.

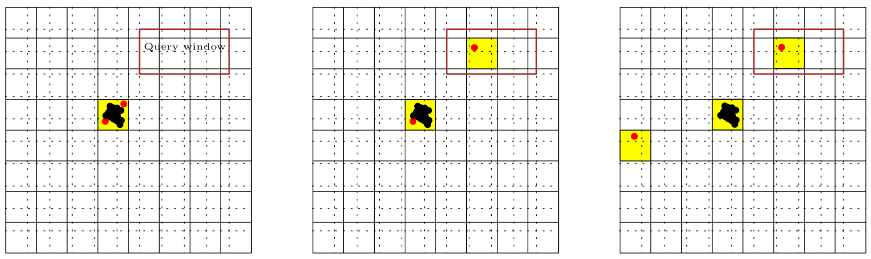


Fig. 24. The change of effective ranges with respect to CHC, where solid lines constitutes the occupancy grid, while dash lines indicate the subdivision of space. Red rectangle refers to the query window. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

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:

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

where $F_{c(x,y)}$ is the cumulative probability in c, and $1 - F_{c(x,y)}$ is the probability that a point does not fall into c. $\overline{f_{c(x,y)}}$ refers to the average probability density in c. When $N \to \infty$, the size of c becomes arbitrarily small, we have

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

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

Since

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

when $N \to \infty$, we derive

$$E(CHC) = \iint_{\Omega} P(\sigma)\, d\sigma = 1 - \iint_{\Omega} e^{-f(x,y)}\, dx\, dy \tag{5}$$

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

This can be easily extended to the nD unit hypercubic domain, where f_n represents the joint Probability Density Function (PDF) and is a continuous function in the domain:

$$E(CHC) = 1 - \int \cdots \int_{\Omega} e^{-f_n}\, dv \tag{6}$$

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. *The expectation of entity union converges to the same constant as grid occupancy, when N becomes arbitrarily large.*

Proof. The difficulty to compute CHC using entity union lies in computing the overlapping area of different cells. To solve this, instead of a cell, we focus on a small box region in the domain that may be covered by the union of cells. We still assume that the point set is distributed in a 2D unit domain. The area of the box region is denoted by Δs, where $\frac{1}{N} \gg \Delta s$. We use two bounds to derive the expectation of the area of which Δs is covered, denoted by $E(s)$. In Fig. 23, the dashed box refers to an nD-cell c, when a point falls into the red box, Δs is considered to be covered.

Apparently, the lower bound omits the region where a cell partially intersects Δs, while the upper bound elaborates all intersection cases. This yields

$$P_{\sigma_L} \Delta s \le E(s) \le P_{\sigma_U} \Delta s$$

where P_{σ_L} refers to the probability that a point falls into the region of the lower bound, while P_{σ_U} refers to that of the upper bound. We have,

$$P_{\sigma_L} = 1 - \left(1 - F_{\sigma_L(x,y)}\right)^N =$$
$$1 - \left(1 - (\frac{1}{\sqrt{N}} - \sqrt{\Delta s})^2 \cdot \overline{f_{\sigma_L(x,y)}}\right)^N$$

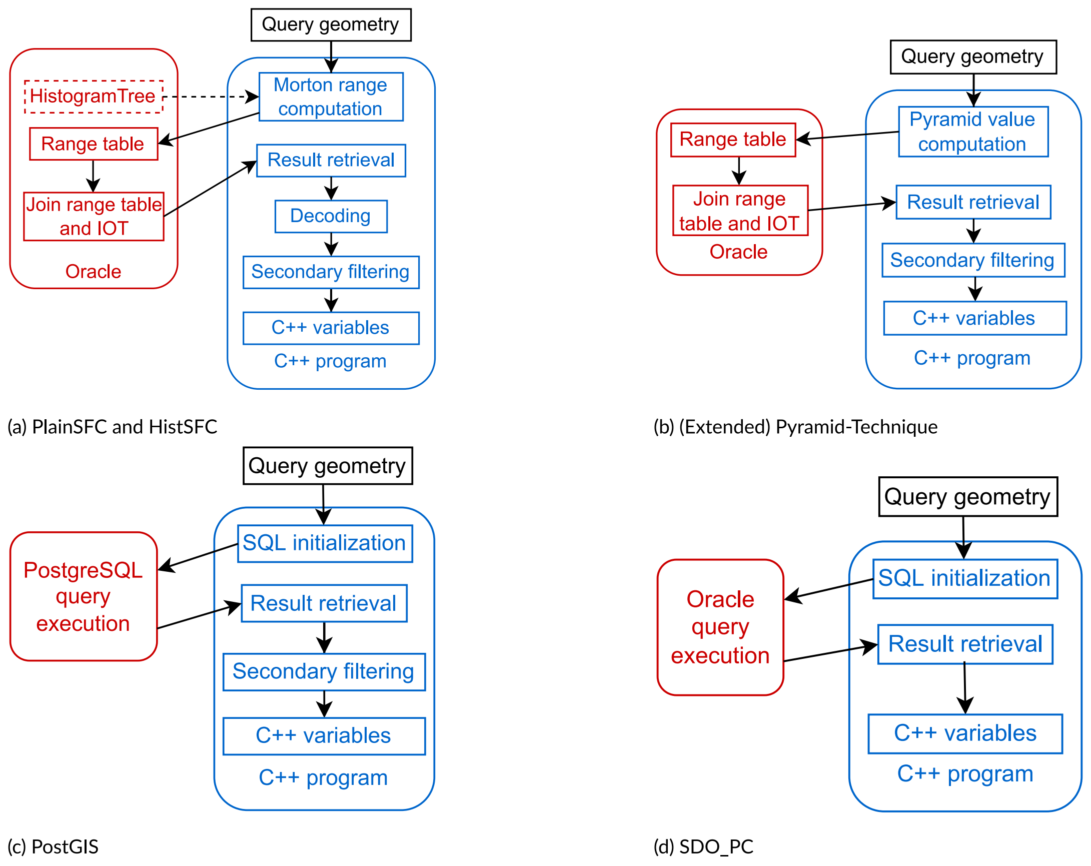


Fig. 25. Querying process of different solutions.

$$P_{\sigma_U} = 1 - \left(1 - (\frac{1}{\sqrt{N}} + \sqrt{\Delta s})^2 \cdot \overline{f_{\sigma_U(x,y)}}\right)^N$$

When $N \to \infty$, then, $\Delta s \to 0$, the size of σ_L and σ_U becomes arbitrarily small, we then derive

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

$$\lim_{N\to\infty} P_{\sigma_U} = 1 - e^{-f(x,y)}$$

As

$$E(CHC) = \sum E(s)$$

so

$$\sum_{i=1}^{\frac{1}{\Delta s}} P_{\sigma_L} \Delta s \le E(CHC) \le \sum_{i=1}^{\frac{1}{\Delta s}} P_{\sigma_U} \Delta s$$

When $N \to \infty$, and $\Delta s \to 0$, we also derive

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

where Ω refers to the unit domain.

This is the same as Eq. (5). Analogously, we can extend the derivation of CHC's expectation to the nD domain, and the expression is the same as Eq. (6).

Theorem 3. *Given a point cloud containing N points. Suppose the points can move freely in the domain so that the uniformity is changing. For a window query, the expected E_{hist} is a monotonically decreasing function of CHC.*

Proof. We first build the occupancy grid for each point set (Fig. 24). Assuming l is the search depth, as N may not equal 2^{nl} which is the number of subspaces, we set l to $\left\lceil \frac{\log_2 N}{n} \right\rceil$. We assume the query window totally matches the boundaries of SFC cells at l.

In Fig. 24, 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 result. When a point moves from the central grid cell to another, which means CHC increases, E_{hist} decreases due to a higher probability of encountering a point inside (middle sub-figure). When more points move out of 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 grid, but reside in the same SFC cell. This is due to the mismatch of these two cell sizes. 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 a monotonically decreasing function of CHC.

Appendix C. Querying process of tested solutions

See Fig. 25.

Appendix D. Benchmarking results

See Tables 11–14.

Table 11
Time cost of different processes in the AHN2 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 12
Time cost of different processes in the flood 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 13
Time cost of state-of-the-art solutions in AHN2 test (second).

	Data set	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	1.56	0.24	14.76	0.97	1.35	0.13	13.00	1.52	72.21	9.54
	2	1.63	0.23	19.61	0.62	1.07	0.10	15.18	1.19	76.52	6.97
	3	1.61	0.22	13.93	0.77	1.06	0.13	15.47	1.77	75.56	7.98
	4	1.55	0.26	12.33	1.19	1.13	0.13	15.38	1.82	73.52	6.72
	5	1.79	0.26	12.94	0.78	1.07	0.16	15.31	1.77	74.20	6.95
Pyramid	1	13.41	24.61	9.86	5.78	8.07	1.36	10.40	2.94	9.33	10.52
	2	17.11	50.06	17.08	7.20	9.45	1.52	11.90	4.57	27.67	9.24
	3	14.42	107.89	13.29	17.18	10.19	11.39	9.17	12.95	39.28	61.04
	4	15.80	248.82	19.85	8.35	10.36	11.56	9.75	9.53	21.59	275.63
	5	14.55	222.50	10.26	3.22	10.68	21.64	9.65	15.87	21.27	131.43
PyramidEx	1	8.80	3.32	11.39	14.20	6.81	0.37	9.57	5.23	11.61	11.19
	2	9.26	4.53	18.35	23.80	8.01	0.85	9.96	11.77	16.78	16.92
	3	15.03	2.67	23.76	23.33	9.18	1.59	19.26	25.95	18.70	35.99
	4	17.80	3.21	31.85	19.75	10.96	3.51	21.12	30.39	19.95	47.34
	5	11.93	10.09	19.79	28.84	12.05	3.94	18.89	32.02	28.67	59.84
HistSFC	1	0.81	0.32	4.14	0.83	0.46	0.28	0.83	5.02	0.73	28.53
	2	0.94	0.32	4.71	0.83	0.45	0.28	0.95	4.61	1.05	28.13
	3	1.20	0.31	4.72	0.83	0.67	0.30	1.17	4.35	0.90	28.41
	4	1.02	0.31	4.83	0.84	0.69	0.28	1.08	4.49	1.11	29.03
	5	1.49	0.32	4.84	0.86	0.93	0.28	1.55	4.92	1.32	29.20

Table 14
Overall throughput of state-of-the-art solutions (points per second).

	Data set	SmallA	SmallB	SmallC	Medium	Large
PostGIS	1	128,545	17,541	162,478	232,510	277,205
	2	124,053	13,635	204,623	206,137	271,442
	3	126,363	18,768	202,378	195,770	271,318
	4	127,480	20,406	189,131	196,237	282,443
	5	112,997	20,110	195,770	197,569	279,282
Pyramid	1	6,069	17,651	25,431	252,914	1,141,608
	2	3,435	11,364	21,869	204,873	614,112
	3	1,886	9,056	11,112	152,565	225,891
	4	872	9,784	10,944	175,118	76,251
	5	973	20,467	7,420	132,213	148,419
PyramidEx	1	19,036	10,779	33,405	228,171	994,088
	2	16,737	6,546	27,055	155,295	672,559
	3	13,030	5,860	22,255	74,651	414,402
	4	10,983	5,348	16,564	65,523	336,784
	5	10,479	5,674	15,001	66,294	256,044
HistSFC	1	280,363	65,771	490,425	3,334,862	8,400,003
	2	240,103	57,831	523,620	2,782,259	7,240,641
	3	190,065	57,746	352,156	2,516,689	7,869,169
	4	222,292	56,435	343,087	2,674,231	7,572,070
	5	153,826	56,423	256,216	1,938,472	6,844,822

References

Achakeev, D., Seeger, B., 2012. A class of R-tree histograms for spatial databases. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems. pp. 450–453.

AHN, 2014. Actueel hoogtebestand Nederland. Retrieved 2025-03-26, from https://www.ahn.nl/.

Bamba, B., Ravada, S., Hu, Y., Anderson, R., 2013. Statistics collection in oracle spatial and graph: Fast histogram construction for complex geometry objects. Proc. VLDB Endow. 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. In: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data. pp. 322–331.

Berchtold, S., Böhm, C., Kriegel, H.-P., 1998. The pyramid-technique: Towards breaking the curse of dimensionality. In: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data. pp. 142–153.

Berchtold, S., Keim, D.A., Kriegel, H.-P., 1996. The X-tree: An index structure for high-dimensional data. In: Proceedings of VLDB. San Francisco, CA, USA, pp. 28–39.

Blanc, T., El Beheiry, M., Caporal, C., Masson, J.-B., Hajj, B., 2020. Genuage: visualize and analyze multidimensional single-molecule point cloud data in virtual reality. Nature Methods 17 (11), 1100–1102.

Bruno, N., Chaudhuri, S., Gravano, L., 2001. STHoles: A multidimensional workload-aware histogram. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. pp. 211–222.

Chen, S., Liu, B., Feng, C., Vallespi-Gonzalez, C., Wellington, C., 2021. 3D point cloud processing and learning for autonomous driving: Impacting map creation, localization, and perception. IEEE Signal Process. Mag. 38 (1), 68–86.

Comer, D., 1979. Ubiquitous B-tree. ACM Comput. Surv. 11 (2), 121–137.

Eavis, T., Lopez, A., 2007. rK-Hist: an R-tree based histogram for multi-dimensional selectivity estimation. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management. pp. 475–484.

Gharbi, M., Soualmia, A., Dartus, D., Masbernat, L., 2016. Comparison of 1D and 2D hydraulic models for floods simulation on the medjerda riverin Tunisia. J. Mater. Environ. Sci. 7 (8), 3017–3026.

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 Int. J. Geo-Inf. 7 (8), 19.

Gunzburger, M., Burkardt, J., 2004. Uniformity Measures for Point Sample in Hypercubes. Tech. Rep., Florida State University, USA, Retrieved 2025-03-26, from https://people.sc.fsu.edu/~jburkardt/publications/gb_2004.pdf.

Guttman, A., 1984. R-trees: A dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data. pp. 47–57.

Katayama, N., Satoh, S., 1997. The SR-tree: An index structure for high-dimensional nearest neighbor queries. ACM SIGMOD Rec. 26 (2), 369–380.

Liu, Q., 2009. Approximate query processing. In: Liu, L., Özsu, M.T. (Eds.), Encyclopedia of Database Systems. Springer US, Boston, MA, pp. 113–119.

Liu, H., van Oosterom, P., Mao, B., Meijers, M., Thompson, R., 2021a. An efficient nD-point data structure for querying flood risks. In: The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLIII-B4-2021, Copernicus GmbH, pp. 367–374.

Liu, Q., Shen, Y., Chen, L., 2021b. LHist: Towards learning multi-dimensional histogram for massive spatial data. In: 2021 IEEE 37th International Conference on Data Engineering. IEEE, pp. 1188–1199.

Liu, H., Thompson, R., van Oosterom, P., Meijers, M., 2021c. Executing convex polytope queries on nD point clouds. Int. J. Appl. Earth Obs. Geoinf. 105, 102625.

Meijers, M., van Oosterom, P., 2018. Clustering and indexing historic vessel movement data with space filling curves. In: ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLII-4, Copernicus GmbH, pp. 417–424.

Neuville, R., Bates, J.S., Jonard, F., 2021. Estimating forest structure from UAV-mounted LiDAR point cloud using machine learning. Remote. Sens. 13 (3), 352.

Ong, M.S., Kuang, Y.C., Ooi, M.P.-L., 2012. Statistical measures of two dimensional point set uniformity. Comput. Statist. Data Anal. 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. In: Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 166–174.

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. Comput. Graph. 49, 92–125.

van Oosterom, P., van Oosterom, S., Liu, H., Thompson, R., Meijers, M., Verbree, E., 2022. Organizing and visualizing point clouds with continuous levels of detail. ISPRS J. Photogramm. Remote. Sens. 194, 119—-131.

Oracle, 2013. Indexes and index-organized tables. Retrieved 2025-03-26, from https://docs.oracle.com/database/121/CNCPT/indexiot.htm#CNCPT721.

Oracle, 2019. SDO_PC_PKG Package (Point Clouds). Retrieved 2025-03-26, from https://docs.oracle.com/en/database/oracle/oracle-database/19/spatl/SDO_PC_PKG-reference.html.

PDAL-Contributors, 2018. PDAL Point Data Abstraction Library. http://dx.doi.org/10.5281/zenodo.2556738.

Psomadaki, S., 2016. Using a Space Filling Curve for the management of dynamic point cloud data in a Relational DBMS (Unpublished master's thesis). Delft University of Technology, The Netherlands.

Ramsey, P., 2020. pgPointcloud - A PostgreSQL extension for storing point cloud (LIDAR) data. Retrieved 2025-03-26, from https://pgpointcloud.github.io/pointcloud/.

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. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. pp. 627–638.

Schütz, M., Krösl, K., Wimmer, M., 2019. Real-time continuous level of detail rendering of point clouds. In: 2019 IEEE Conference on Virtual Reality and 3D User Interfaces. VR, IEEE, pp. 103–110.

Tchapmi, L., Choy, C., Armeni, I., Gwak, J., Savarese, S., 2017. SEGCloud: Semantic segmentation of 3D point clouds. In: 2017 International Conference on 3D Vision. 3DV, pp. 537–547.

Wang, J., Shan, J., 2005. Space filling curve based point clouds index. In: Proceedings of the 8th International Conference on GeoComputation. pp. 551–562.

Zhang, R., Qi, J., Stradling, M., Huang, J., 2014. Towards a painless index for spatial objects. ACM Trans. Database Syst. 39 (3), 1–42.

Zheng, Y., Zhang, L., Xie, X., Ma, W.-Y., 2009. Mining interesting locations and travel sequences from GPS trajectories. In: Proceedings of the 18th International Conference on World Wide Web. WWW '09, Association for Computing Machinery, New York, NY, USA, pp. 791–800.