ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume VI-4/W1-2020, 2020
3rd BIM/GIS Integration Workshop and 15th 3D GeoInfo Conference, 7–11 September 2020, London, UK

AN OPTIMIZED SFC APPROACH FOR ND WINDOW QUERYING ON POINT CLOUDS

H.Liu*, P. Van Oosterom, M. Meijers, E. Verbree

Faculty of Architecture and the Built Environment, Delft University of Technology, the Netherlands
(H.Liu-6, P.J.M.vanOosterom, B.M.Meijers, E.Verbree)@tudelft.nl

Commission IV, WG IV/1, IV/7

KEY WORDS: Point Clouds, Query, SFC, Histogram, Distribution, B+-Tree, Oracle, Morton Curve

ABSTRACT:

Dramatically increasing collection of point clouds raises an essential demand for highly efficient data management. It can also facilitate modern applications such as robotics and virtual reality. Extensive studies have been performed on point data management and querying, but most of them concentrate on low dimensional spaces. High dimensional data management solutions from computer science have not considered the special features of spatial data; so, they may not be optimal. A Space Filling Curve (SFC) based approach, PlainSFC which is capable of nD point querying has been proposed and tested in low dimensional spaces. However, its efficiency in nD space is still unknown. Besides that, PlainSFC performs poorly on skewed data querying. This paper develops HistSFC which utilizes point distribution information to improve the querying efficiency on skewed data. Then, the paper presents statistical analysis of how PlainSFC and HistSFC perform when dimensionality increases. By experimenting on simulated nD data and real data, we confirmed the patterns deduced: for inhomogeneous data querying, the false positive rate (FPR) of PlainSFC increases drastically as dimensionality goes up. HistSFC alleviates such deterioration to a large extent. Despite performance degeneration in ultra high dimensional spaces, HistSFC can be applied with high efficiency for most spatial applications. The generic theoretical framework developed also allows us to study related topics such as visualization and data transmission in the future.

1. INTRODUCTION

In point cloud data, each point contains multiple dimensions. Dimension and attribute are equivalent terms, and each represents a specific type of information. Apart from routinely concerned spatio-temporal dimensions, i.e. X/Y/Z/T, other dimensions such as Level of Detail (LoD), classification and identity are also indispensable. In specific fields, points may carry more information. For instance, in hydraulic modelling, a point may also record the flow direction and speed, sediment concentration, and other fluid parameters. For certain applications, several dimensions can be queried in a combined way. Take indoor navigation in a Virtual Reality (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 LoD which represents importance of objects. Besides, people should be able to see things through windows which belong to a specific class. Then, a query concerned with XYZ, LoD and classification will form the basis to realize the visualization. Another example is the prevalent Building Information Modeling (BIM), which frequently uses XYZ, LoD and Time. In the future, as information continues to grow, we will expect to see more dimensions involved in queries. Generally, we call them nD queries.

Current point data management solutions are mostly file based systems, and formats include LAS/LAZ, PCL, TEXT and vendor specific schemas. The integration of different data sources thus becomes a critical problem. Also, sorting and indexing need additional data structure design and implementation. Aimed at generic purposes, Database Management Systems (DBMS) on the other hand avoid those issues to a large extent. Oracle and PostgreSQL offer state-of-the-art flat table and block approaches (Van Oosterom et al., 2015). Flat table 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 on one or several columns of flat tables, but it is only favourable for limited types of queries. If columns requested change, 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 boundary blocks for filtering to achieve accuracy at the individual point level. It is the fact that if the blocks do not fit in the query geometry well, the process can be very inefficient due to the unpacking and filtering processes. To make things worse, it only supports a limited number of dimensions to be indexed, e.g., at most 3 for Oracle SDO_PC (Oracle, 2013b).

Efficient solutions for generic nD points querying have been proposed and implemented in computer science (Berchtold et al., 1996, Berchtold et al., 1998, Ooi et al., 2000). Advanced data modelling and partitioning strategies have been applied, which results in significant performance gain. However, most data structures were not adapted and optimized to spatial applications. The characteristics of spatial data have not been considered, including number of dimensions, typical query shapes, specific distributions of commonly used dimensions such as XYZ and LoD, and spatio-temporal coherence embedded in the data (Section 2).

Nonetheless, some of the studies did take the peculiarity of spatial data into account (Böhm et al., 2001, Zhang et al., 2014). Spatial Filling Curve (SFC), as an advanced technique to cluster and access the data, has been considered and used. SFC approaches have been adapted and improved for point data management (Wang, Shan, 2005, Zhang et al., 2014, Van Oosterom et al., 2015). Specifically, Van Oosterom et al. (2015) presented 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 multidimensional points and queries into a one-dimensional SFC

space so that one-dimensional indexing structure such as the B+-tree could be used. Note that only *organizing dimensions* which are used to cluster and index the data, e.g., X/Y/Z/T are transformed and mapped. The other *property dimensions* are affiliated, such as color, intensity and classification, which are not frequently used in the SQL WHERE clause. These two types of dimensions are interchangeable depending on applications. PlainSFC has been studied since then (Martinez-Rubi 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. Low dimensional solutions may perform significantly worse in high dimensional spaces (Weber et al., 1998), which is known as the "curse of dimensionality".

As the foundation for many operations on point clouds, such as visualization, data downloading and analysis, nD window queries are executed frequently. We perform this research to improve its querying efficiency. We made two main contributions in this paper:

- We devised and implemented *HistSFC*, which improves the efficiency of PlainSFC for querying inhomogeneously distributed data. HistSFC utilizes a HistTree recording an nD histogram of the data to address the problem.
- We set up theoretical basis for nD window querying on massive points, based on PlainSFC and HistSFC. Influence of correlation between dimensions was considered. In addition, we conducted soundly designed tests to verify the theory.

The rest of the paper is organized as follows: Section 2 identifies features of spatial window querying, which distinguish it from general window querying. Section 3 explains structures and mechanisms of PlainSFC and HistSFC in detail. The mathematical foundation to quantify the performance of both approaches is established in Section 4. This is then followed by experimental evaluation consisting of both simulations and real data tests in Section 5. The final section concludes the paper.

2. FEATURES OF SPATIAL WINDOW QUERIES

Unlike general nD vectors used in multimedia or data mining (Böhm et al., 2001, Weber et al., 1998), every dimension of nD point clouds has a physical or semantic meaning. The dimensionality of spatial window queries is thus limited, instead of an ultra high number. Besides, the dimensionality is also confined by the data volume. For example, a typical source for point clouds is laser scanning. Suppose the most accurate Airborne Laser Scanning (ALS) can return 1 point per cm^2. As the area of US is $9.8\times10^6\ km^2$, then, 9.8×10^{16} points can be collected. We assume a selective query should at least reach a 10 percent selectivity on each dimension. Then the expected number of points retrieved is as follows:

$$E(k) = N \times 10^{-m} \approx 10^{16-m}$$

where k is the output size, and m refers to the dimensionality of the query box. To guarantee at least one point is retrieved, m should be less than 16.

Data distribution along each dimension is of various types in point clouds: despite uniform distribution of XY and time, Gamma distribution (e.g., Z and LoD (Van Oosterom, 2019)) and normal distribution such as colour and intensity are also common. Non-uniform data distribution in nD space poses higher requirement for efficient querying.

Besides, we also note that the majority of current spatial applications still focus on several dimensions. They are, for instance, spatio-temporal dimensions, LoD and so on. The implication is that if the solution is organized by n dimensions, then it should also perform efficiently for mD queries ($m < n$). In addition, spatial window queries are not always cubic, based on previous investigation (Van Oosterom et al., 2015, Meijers, van Oosterom, 2018, Guan et al., 2018), we list some typical non-cubic querying windows (Table 1):

Buffer of a curve, e.g., a river or a road
Diagonal rectangle for an inclined feature
Arbitrary geometry, e.g., footprint of a municipality
View frustum for perspective view selection
3D XYLoD rectangle for zooming in and out, with LoD much thinner than XY dimensions
Trajectory of an object, i.e., a 4D window query on XYT and identity dimensions 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

Table 1. Typical non-cubic window queries

Nonetheless, cubic queries are still executed frequently, e.g., in quality checking (Pavlovic et al., 2018), performance testing (Pajić et al., 2018) and k Nearest Neighbor (kNN) search.

These features have been largely considered in the experiments in Section 5.

3. PLAINSFC AND HISTSFC

Among all SFCs, Morton curve is commonly studied and practiced due to the simplicity of mapping functions (Morton, 1966, Wang, Shan, 2005). It is based on interleaving the bits from the coordinates. By truncating the Morton key, we can construct the upper levels in the SFC hierarchy. In Figure 1, the four points residing in the lower left box all start with 0000. If we truncate the last two bits of their keys, they could be represented by a point for which the key equals 0 in the upper level. That is to say, SFC contains an implicit 2^n-tree, where n is the dimension number.

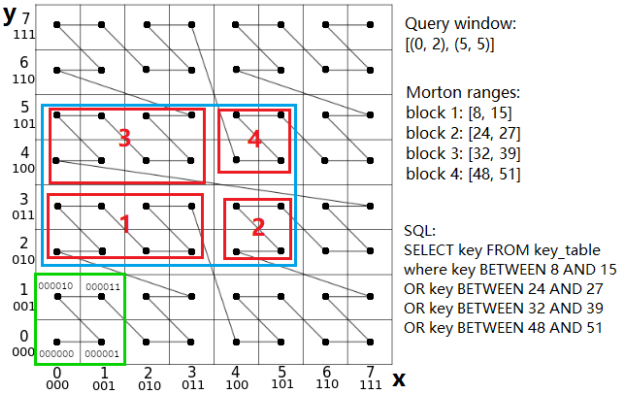


Figure 1. Mapping a 2D window to 1D Morton ranges

3.1 PlainSFC

In this paper, PlainSFC is implemented using a Morton curve. For compact and non-redundant storage of real point data, each

point is encoded as a full resolution SFC key, a combination of all organizing dimensions. Property dimensions are attached to the SFC key. Such a full resolution key can be decoded directly to the original coordinates. Figure 2 presents the workflow of PlainSFC. For storage, after computing SFC keys, PlainSFC adopts the B+-tree structure to manage them. Oracle has implemented B+-tree in Index-Organized Tables (IOT) (Oracle, 2013a). The key and other property dimensions are stored in leaf nodes present in a normal table, while the internal structure uses the SFC keys to organize and index the storage. IOT is used throughout the paper.

With regard to querying, PlainSFC adopts two filters to process (Figure 2). PlainSFC uses SFC at different levels to approach the query window, by decomposing the Minimum Bounding Box (MBB) of the dataset recursively. As is illustrated in Figure 1, the query algorithm first examines whether MBB of the dataset intersects the query window. If they intersect, the MBB would be decomposed into 4 quadrants, and the spatial relationship between each quadrant and the query window would be assessed again. During the process, if a quadrant is totally inside the query window, the corresponding 1D range of the quadrant would be exported directly without further decomposition. Near the query boundary, the process continues to the maximum depth defined. In this way, PlainSFC maps the query window from the original multidimensional space into the SFC ranges. It then searches the data using the ranges, e.g. by using a SQL WHERE clause for selection (Figure 1).

A large depth during the decomposition will cause huge number of ranges generated, slowing down the first filter. A small depth, however, results in a rough result, where the error comes from coarse boundary cells. This is also unacceptable because it moves the burden to the second filter for an accurate answer.

3.2 HistSFC

A crucial bottleneck of PlainSFC is that part of the ranges generated may actually contain few or even no points when the data distribution is skewed. For instance, ALS points mostly lie on the ground, and the number decreases sharply as elevation grows. On the one hand, this implies extra time is spent on keeping refining cells containing few points, and generating empty ranges; on the other hand, large quantities of non-dense and empty ranges increase the time cost of data selection later. We thus propose using HistTree to improve the quality of ranges and thereafter the whole querying performance. We build HistTree as an additional structure to represent the data distribution (Liu et al., 2020), and use it to optimize the range computation.

As Figure 3 shows, the HistTree records the point count for each SFC node at different level. If the count exceeds the threshold of the tree, it will be partitioned into SFC nodes in a lower level. A height field is used in a HistTree node to distinguish different nodes, because branch nodes at different levels may possess identical keys. A HistTree node actually represents the MBB of a quadrant, but it contains neither points nor pointers to points. Thus, HistTree is a compact structure which can be stored in a normal table.

When querying, unlike PlainSFC that adopts a fixed depth for recursive decomposition of nodes, HistSFC employs HistTree which is adaptive to point density. Starting from the root node, the extent of each node can be computed using its height and the key (Figure 3). Then, by performing intersecting computation between branch nodes and the query window iteratively, the function retrieves all relevant nodes, and abandons non-overlapping nodes. Part of the result are branch nodes which locate totally inside the query window. The searching process stops at these nodes and exports their ranges. The other resultant nodes are leaf nodes which fall on the boundary of the query window. The leaf nodes can be further refined using the fixed decomposition approach adopted by PlainSFC. The process stops when the number of ranges reaches the maximum number allowed. In this way, with same number of ranges, the accuracy of the first filter is expected to be improved significantly when data distribution is non-uniform.

However, what accuracy that the first filter can reach, and how effective the HistTree performs in the nD space remain unclear. We express the accuracy using False Positive Rate (FPR):

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

The first filter leverages the B+-tree for querying, which is efficient, while the second filter is a linear scanning process and is adversely affected by a large FPR. Therefore, FPR is a key indicator to evaluate the performance of PlainSFC and HistSFC. Next section quantifies how FPR changes with dimensionality.

4. MATHEMATICAL FOUNDATION

The mathematical derivation uses symbology listed in Table 2.

Notation	Description
n	Number of organizing dimensions
m	Number of dimensions in querying
N	Number of points in input
k	Number of points in output (i.e., exact answer)
k'	Number of points returned by the first filter
D_i	The i^{th} dimension
F_i, f_i	Cumulative and Probability Density Function (CDF and PDF) of the i^{th} dimension
M_i	The maximum range of the i^{th} dimension
r_i	The length of data region in a biased dimension
T	Maximum number of ranges for querying
l	The edge length of the mD query box
Δl	The length of a SFC cell for each dimension

Table 2. Notations used in the formulation

Low dimensional key solutions are inefficient for high dimensional queries, as the first filter may return huge amount of false positive points due to the whole span of dimensions not existing in the key. So, we adopt high dimensional key storage. However, the efficiency to perform low dimensional queries may then decrease, as the involvement of more dimensions in the key undermines the clustering of dimensions concerned in the query. So, this section focuses on the FPR of m-nD queries ($m \leq n$). It demonstrates the main idea with two specific cases: A 2-nD query and an n-nD query. To simplify the proof, we employ a hyper-cube query.

Problem statement: A point cloud with size N is the input, and the query window is a hyper-cube with an edge length of l. What is the FPR to perform a 2-nD query and an n-nD query, with constant T, where $2 \leq n < +\infty$?

Theorem 1: For both queries, the FPR of PlainSFC rises with the increase of n, when points are uniformly distributed.

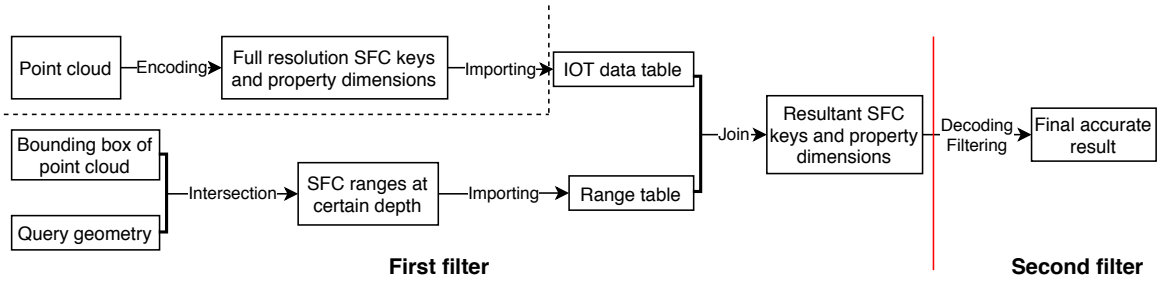


Figure 2. The loading and querying procedure of PlainSFC, divided by the dash line

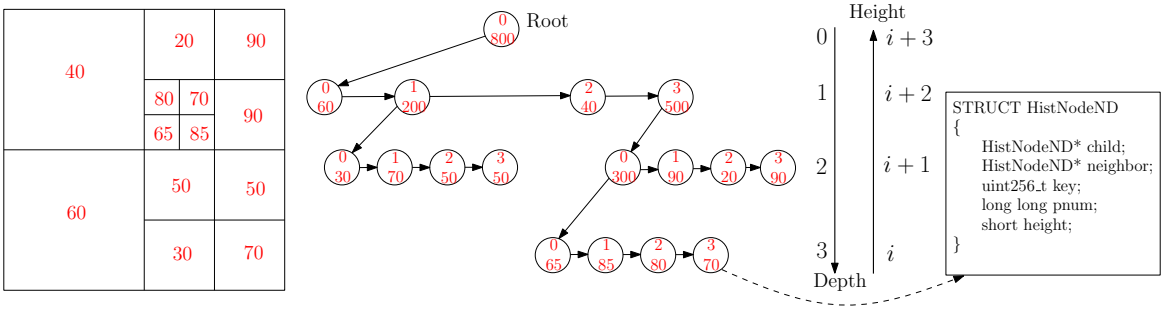


Figure 3. A 2D HistTree example, where the threshold is 100; left: point counting, middle: pointer structure of HistTree, with each node storing a SFC key and number of points, right: structure of a HistTree node

Proof: For 2-2D query, the output size is,

$$k = N \int_{a}^{a+l} \int_{b}^{b+l} f_1 \cdot f_2 \, dD_1 \, dD_2$$
$$= N(F_1(a+l) - F_1(a))(F_2(b+l) - F_2(b))$$

where (a, b) refers to the lower-left anchor of the query window. Suppose every dimension has same length M (, $\forall i > 0, M_i = M$), the length of the SFC cell at the given depth is Δl:

$$\Delta l = \frac{l}{\sqrt{T}} \tag{2}$$

SFC grid normally does not match the query window exactly. For genericness, we assume the boundary of the query window splits SFC cells intersecting the boundary into halves. We will retrieve more points than needed:

$$E(k' - k) = N \left(F_1 \Big|_{a-\frac{\Delta l}{2}}^{a} F_2 \Big|_{b-\frac{\Delta l}{2}}^{b+l+\frac{\Delta l}{2}} + F_1 \Big|_{a+l}^{a+l+\frac{\Delta l}{2}} F_2 \Big|_{b-\frac{\Delta l}{2}}^{b+l+\frac{\Delta l}{2}} + F_1 \Big|_{a}^{a+l} F_2 \Big|_{b-\frac{\Delta l}{2}}^{b} + F_1 \Big|_{a}^{a+l} F_2 \Big|_{b+l}^{b+l+\frac{\Delta l}{2}} \right)$$

Assume $\Delta l \ll l$, according to Equation 1,

$$FPR = \frac{E(k' - k)}{k} \geq \frac{F_1(a + l + \frac{\Delta l}{2}) - F_1(a - \frac{\Delta l}{2})}{F_1(a + l) - F_1(a)} + \frac{F_2(b + l + \frac{\Delta l}{2}) - F_2(b - \frac{\Delta l}{2})}{F_2(b + l) - F_2(b)} - 2 \tag{3}$$

Then we generalize the derivation, for 2-nD query. We could then get the same mathematical expression as Equation 3. However, Δl changes:

$$\frac{\Delta l_n}{\Delta l_{n-1}} = \left(\frac{M^2 T}{l^2} \right)^{\frac{1}{n(n-1)}} \tag{4}$$

As $MT > l$ ($M \geq l$ and $T > 1$), then $\Delta l_n > \Delta l_{n-1}$. That is to say, for 2D query, the FPR of a higher dimensional organization will always be larger than that of a lower dimensional organization. For m-nD query, where $2 < m < n$, we could draw the same conclusion.

With respect to n-nD query, if the lower-left anchor of the query window is $(a_1, a_2 \ldots a_n)$, then,

$$FPR \geq \sum_{i=1}^{n} \frac{F_i(a_i + l + \frac{\Delta l}{2}) - F_i(a_i - \frac{\Delta l}{2})}{F_i(a_i + l) - F_i(a_i)} - n \tag{5}$$

Hence, with the increase of dimensionality, the FPRs of both 2-nD query and n-nD query grow. □

Theorem 2: Starting from 1D, when an organizing dimension added is non-uniform, the HistTree will be more effective in a higher dimensional space under a certain dimensionality, for both types of query.

Proof: The exact FPR of HistSFC depends on specific data distribution. Without losing generality, we adopt a partial uniform model to approximate a non-uniform dimension (Figure 4). In this model, the data portion is r_i, which implies that from δ to $\delta + r_i$ (where δ refers to the offset from 0), data obeys uniform distribution in D_i. It is apparent that when $r_i = M$, D_i becomes a uniform dimension.

We define effectiveness of HistTree by

$$e = \frac{FPR_{PlainSFC}}{FPR_{HistSFC}} \tag{6}$$

When the 2D querying box is totally inside the data region where points are uniformly distributed, there will be no gain from HistTree. So e equals 1. When the querying box partially intersects the data region (Figure 5), for both PlainSFC and HistSFC,

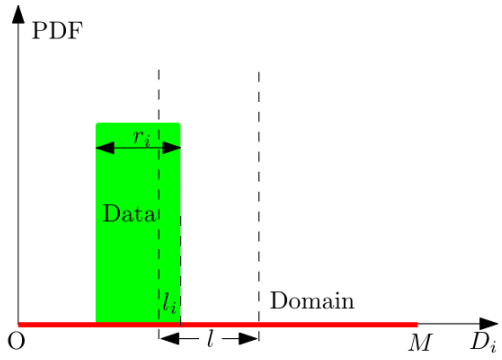


Figure 4. PDF of a non-uniform dimension illustrated by the partial uniform model

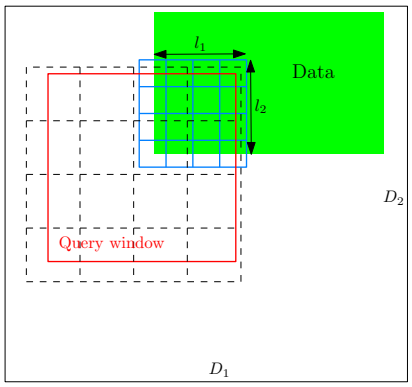


Figure 5. 2D window querying based on PlainSFC grid (dashed) and HistSFC grid (blue), T = 16

$$FPR = \frac{E(k'-k)}{k} \approx \frac{f_1 \cdot f_2 \left(l_1 \frac{\Delta l}{2} + l_2 \frac{\Delta l}{2}\right)}{f_1 \cdot f_2 \cdot l_1 l_2} = \frac{\Delta l(l_1+l_2)}{2 l_1 l_2} \tag{7}$$

The Δl, however, changes from $\frac{l}{\sqrt{T}}$ in PlainSFC to $\sqrt{\frac{l_1 l_2}{T}}$ in HistSFC. This indicates that HistSFC decreases FPR. In addition, if the 2D query box does not intersect the data region, then FPRs for both approaches equal 0. We also assign 1 to e, to express no gain from HistTree. However, this does not mean PlainSFC functions equivalently as HistSFC. In fact, HistSFC generates no range for selection as it learns the data distribution from HistTree, while PlainSFC still generates T ranges and retrieves no points after selection. So PlainSFC takes additional time.

We then generalize the deduction to 2-nD query. We assume $l \leq r_i$ $(0 < i \leq n)$, because in reality, users normally know the range of data and will not query a larger window than it. As Table 3 indicates, for both intersection cases, since $0 < r_i \leq M$ and $l_i \leq l$, then $e \geq 1$. In other words, HistTree always has a positive effect. When the 2D window is inside the data area, we could derive the following:

$$\frac{e_n}{e_{n-1}} = \left(\frac{\prod_{i=3}^{n-1} r_i}{r_n^{n-1}}\right)^{\frac{1}{n(n-1)}} \tag{8}$$

So, when $r_n < (\prod_{i=3}^{n-1} r_i)^{\frac{1}{n-1}}$, the effectiveness of HistTree

Probability	Query position	e
$\frac{(r_1-l)(r_2-l)}{M^2}$	Inside data area	$(\prod_{i=3}^{n} \frac{r_i}{M})^{-\frac{1}{n}}$
$\frac{2l(r_1+r_2-l)}{M^2}$	Partially intersects data area	$(\frac{l^2}{l_1 l_2} \prod_{i=3}^{n} \frac{M}{r_i})^{\frac{1}{n}}$
$\frac{M^2-2r_1r_2+(r_1-l)(r_2-l)}{M^2}$	No data area	1

Table 3. Effectiveness of HistTree for 2-nD query, $l \leq \min_{2<i\leq n} r_i$

Probability	Query position	e
$\frac{\prod_{i=1}^{n}(r_i-l)}{M^n}$	Inside data area	1
$\frac{\prod_{i=1}^{n}(r_i+l)}{M^n} - \frac{\prod_{i=1}^{n}(r_i-l)}{M^n}$	Partially intersects data area	$(\frac{l^n}{\prod_{i=1}^{n} l_i})^{\frac{1}{n}}$
$1 - \frac{\prod_{i=1}^{n}(r_i+l)}{M^n}$	No data area	1

Table 4. Effectiveness of HistTree for n-nD query, $l \leq \min_{2<i\leq n} r_i$

increases. We assume $\frac{r_i}{M}$ are all random variables between 0 and 1, then $P(r_n < (\prod_{i=3}^{n-1} r_i)^{\frac{1}{n-1}}) = (\frac{n-1}{n})^{n-3}$. It is a monotonically decreasing function. We could derive that when $n < 9$, such probability is larger than 0.5. To put it another way, when we are querying data within 8D space, there is a high probability that e for 2D box selection rises all the way. Similarly, when partial intersection occurs, within 5D, HistTree can perform more effectively in higher dimensional spaces.

Using the same method, we get results for n-nD query (Table 4). In all cases, $e \geq 1$. We could also derive that within 5D, the probability that HistTree becomes increasingly effective for n-nD query is high, when partial intersection occurs.

Current derivation is based on the partial uniform model. Now we present a weak deduction when this model is changed to curve-shaped distribution such as the normal distribution. A consequence is that the point density inside data area becomes varying. Suppose the number of points inside the query window is still the same, then FPR of PlainSFC is unchanged. For HistSFC, based on the original grid, the following algorithm will be applied:

Algorithm 1: Query optimization for a point cloud with varying density

if *the density of dense point area is* 2^n *times larger than that of the sparse area* **then**
  HistSFC uses smaller cells to select dense point area, while employees larger cells for sparse area (Figure 6);
else
  HistSFC uses original grid;
end

We assume the density of the sparse area is d points per cell. In the sparse area, 2^n original cells merge into 1 cell, which results in $2^{n-1}d$ false positive points. In contrast, in a dense area, one cell is decomposed into 2^n cells: half of them are outside the query window, so $0.5 \times 2^n d$ false positive points are removed. By executing Algorithm 1, the FPR of HistSFC reduces further. However, when n grows until $k < 2^n$, HistSFC stops optimization, because the cells selected will not change. □

Theorem 3: For both queries, correlation involved in the data decreases the FPR of HistSFC, compared to data without correlation.

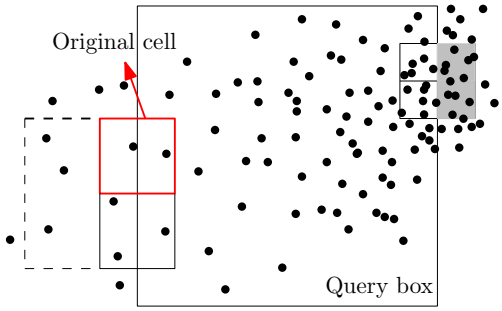


Figure 6. Various cell sizes computed by HistSFC when point density is variable in 2D data. The left two cells are merged into one bigger cell, while the right cell are split into four smaller ones with two totally out of the query box

Proof: We assume in a 2D dataset, D_1 and D_2 are correlated: $cov(D_1, D_2) = \int_0^M \int_0^M (D_1 - \mu_1)(D_2 - \mu_2) f(D_1, D_2)\, dD_1\, dD_2$, where μ refers to the mean value. Suppose $f(D_1, D_2) = C$, where C is a constant; then $cov(D_1, D_2) = 0$. However, as D_1 and D_2 are correlated, $cov(D_1, D_2) \neq 0$, so $f(D_1, D_2)$ must be a variable inside the domain. Then by adopting Algorithm 1, HistSFC performs more effectively when dimensions are correlated. This applies to nD data as well. □

Theorem 1 indicates the FPR of PlainSFC continues to grow as n keeps increasing. Theorem 2 only shows the effectiveness of HistTree without discussing HistSFC itself. We now introduce a lemma when HistSFC performs a 2-nD query:

Lemma: The FPR of HistSFC, on the whole, increases with n for the 2-nD query.

Proof: It is not difficult to deduce that for the 2-nD query, the FPR of HistSFC follows Equation 7. However, Δl changes:

$$\Delta l = \left(\frac{l_1 l_2 \prod_{i=3}^{n} r_i}{T} \right)^{\frac{1}{n}}$$

$$\frac{\Delta l_n}{\Delta l_{n-1}} = \left(\frac{T r_n r_n}{l_1 l_2} \cdot \frac{r_n^{n-3}}{\prod_{i=3}^{n-1} r_i} \right)^{\frac{1}{n(n-1)}} \tag{9}$$

In Equation 9, $\frac{T r_n r_n}{l_1 l_2} > 1$. As $\frac{r_1}{M}, \frac{r_2}{M}, \cdots \frac{r_n}{M}$ are all independent variables between 0 and 1, then,

$$E\left(\frac{r_n^{n-3}}{\prod_{i=3}^{n-1} r_i} \right) = \frac{E(\frac{r_n}{M}{}^{n-3})}{E(\prod_{i=3}^{n-1} \frac{r_i}{M})} = \frac{2^{n-3}}{n-2} \geq 1, \forall n \geq 3$$

That is, Δl increases with n, so does the FPR. □

Theorem 4: When n exceeds a certain value, k' from PlainSFC and HistSFC will be a constant for a m-nD query, and FPR is not influenced by n thereafter.

Proof: It is the fact that when n increases until $2^{n-m} \geq T$, PlainSFC will only return the ranges of nodes at the first depth. For example, when $n = 16$, $m = 2$ and $T = 10,000$, the first decomposition of the root node in PlainSFC will results in 2^{16} child nodes. For a 2-nD query, there are 3 types of intersection between the query box and child nodes. First, when the query box totally falls in one quadrant of the 2 dimensions it queries, then one fourth of child nodes of the root will be selected. In this case, $\frac{1}{4} \times 2^{16} > T$. So PlainSFC will not search deeper in the tree, and will return all the points that reside in the quadrant. Besides, the query box can also intersect with half of all child nodes or all child nodes. As n keeps increasing, the returned number of points will remain unchanged. This is because the involvement of other dimensions does not influence the partition of the 2 dimensions queried in the SFC hierarchy. We could derive the following for 2-nD query (assume uniform distribution):

$$E(k') = N \frac{l^2}{M^2} + \frac{N}{2}\left(\frac{2l}{M} - \frac{2l^2}{M^2} \right) + \frac{N}{4}\left(1 - \frac{2l}{M} + \frac{l^2}{M^2} \right) \tag{10}$$

In selective queries, $l \ll M$:

$$E(k') \approx \frac{N}{4} + \frac{l}{2M} \tag{11}$$

Analogously, we derive the limit of m-nD query:

$$E(k') \approx \frac{N}{2^m} + \frac{Nml}{2^m M} \tag{12}$$

As k is not influenced by n, the FPR will not be determined by n either. We could also derive that when points are distributed inhomogeneously, $E(k')$ is the same (by assuming $N_1, N_2 ... N_{2^n}$ in each child node). In the case $\frac{l}{M} > 0.5$, $E(k')$ will be N.

HistSFC, however, will not select nodes with no points inside. Thus, it is likely that when $2^{n-m} \geq T$, the number of actual child nodes selected, t, is less than T. Then, when we add one organizing dimension into the storage, the number of child nodes selected will become $2t$. So, if n keeps increasing, until the number of nodes at the first depth selected by HistSFC is larger than T, HistSFC will return the same result as PlainSFC. Therefore, HistTree only postpones the threshold of n when the limit occurs. □

We then present a significant example derived from Theorem 4:

Example: For $n = 16$, $m = 4$, $T \leq 4096$, $\frac{l_i}{M_i} < 0.1$ $(0 < i \leq 4)$, according to Equation 12,

$$E(k') \approx 0.0875N$$

A common assumption used frequently (Weber et al., 1998) is that when the index can reduce the scan to 10% of total data, then the index is effective. According to this, when we perform a 4-16D query or with even more dimensions in the query window, then either PlainSFC or HistSFC works effectively. Based on previous theorems, when $n < 16$ and $m \geq 4$, PlainSFC and HistSFC will always be effective, with a proper T.

5. EXPERIMENTAL EVALUATION

This section presents simulations conducted to verify the theorems, and also shows the performance of PlainSFC and HistSFC using real data. The ideal simulation (IdealSim) builds ideal situations assumed in the proof of the theorems. The realistic simulation (RealSim) aims to simulate data and queries in real cases. Both simulations present FPR for the 2-nD query and the n-nD query, using PlainSFC and HistSFC. Afterwards, the real case test uses an ALS dataset, and evaluates the performance in real spatial applications.

5.1 IdealSim

IdealSim generates 1 million point records with 16 dimensions. Every dimension ranges from 0 to 4096, meaning $0 < M \leq 4096$. Each dimension follows the partial uniform model randomly generated, where $45 \leq r_i \leq 3177$ ($1 \leq i \leq 16$). The threshold of all HistTrees is set to 100, to fully exploit the histogram. IdealSim then generates 100 2-nD query windows randomly with an edge length of 410; another 100 n-nD query windows are generated with an edge length of 2000 for the performance test. A large edge length can guarantee that points will also be selected in high dimensional spaces. To make full use of the SFC hierarchy for searching, we set T to 100,000. Figure 7 - 9 show the results.

FPRs for the 2-nD query and the n-nD query are presented in Figure 7 and 8, respectively. In both cases, FPR of HistSFC is much lower than that of PlainSFC, especially in higher dimensional spaces. Besides, FPRs of both approaches rise as n goes up, while PlainSFC reaches an upper bound after 10D, in the 2-nD query. This corresponds to the limit indicated in Theorem 4 (Section 4). On the other hand, for the same query, FPR of HistSFC is smaller than 10 until 12D. This is a significant sign that HistSFC can function efficiently in high dimensional spaces. Figure 7 also shows that the overall fluctuation of PlainSFC is stronger than that of HistSFC, especially when dimensionality becomes larger. In fact, the stable performance of HistSFC is also an advantage, as it alleviates worst cases. This pattern also happens in the n-nD query. In Figure 8, the gap between the two approaches is more evident. Due to the drastic increase of PlainSFC, its FPR can reach 1,836 in 16D.

In Figure 9, the effectiveness of HistTree for the n-nD query rises all the way until 10D, where it starts to drop. The decrease occurs a bit later for the 2-nD query. However, Theorem 2 predicts that after 5D, the effectiveness will most probably drop. The wrong prediction might be attributed to the narrow data span from D_6 to D_9, where the r_i are 45, 849, 69 and 646, respectively. Specifically, due to the large window size (i.e., 2000) adopted in the n-nD query, PlainSFC deteriorates much more significantly; HistSFC, on the other hand, is not influenced much. 2-nD queries are also affected by the narrow data span, but the change is not so drastic.

5.2 RealSim

RealSim builds an independent dataset and a correlated dataset (Table 5), to investigate the effect of correlation within the data on FPR. Each dataset contains 1 million points with 6 dimensions. Each dimension ranges from 0 to 1,048,576 (i.e., 2^{20}). All dimensions either obey Normal distribution $\mathcal{N}$ or Gamma distribution Γ. PDFs of all dimensions are depicted in Figure 10. In the correlated dataset, the correlation coefficient between D_1 and D_2 is 0.85, while it is 0.61 between D_1 and D_4, and 0.64 between D_2 and D_4. The threshold to build HistTrees is still 100. For the 2-nD query, RealSim applies a box query, with an edge length equal to 1% - 5% of a dimension range. 100 querying boxes which have point inside are tested for the evaluation. RealSim then randomly generates 100 nD rectangles which must contain 0.1% - 1% of total points, for the n-nD query. This is done to guarantee sufficient points selected in high dimensional spaces. We set T to 1000 in all cases, as the dimensionality concerned is not high. Figure 11 - 14 present the results.

	Independent dimensions	Correlated dimensions	Stretch factor
D_1	$\mathcal{N}(2^{19},\ \sqrt{3} \cdot 2^{17})$	$\mathcal{N}(2^{19},\ \sqrt{3} \cdot 2^{17})$	1
D_2	$\mathcal{N}(2^{19},\ 2^{18})$	$D_1 + \mathcal{N}(0,\ 2^{17})$	1
D_3	$\Gamma(1, 2)$	$\Gamma(1, 2)$	2^{17}
D_4	$\mathcal{N}(2^{18},\ \sqrt{1.48} \cdot 2^{17})$	$0.2D_1 + 0.3D_2 + \mathcal{N}(0,\ 2^{17})$	1
D_5	$\Gamma(2, 3)$	$\Gamma(2, 3)$	2^{15}
D_6	$\Gamma(10, 2)$	$\Gamma(10, 2)$	2^{16}

Table 5. Distributions of different dimensions in RealSim

Both types of query are executed on the independent dataset and the correlated dataset. The overall FPRs (Figure 11 and 13) follow similar patterns as has been discussed in Section 5.1. The correlation inside data causes further degeneration of PlainSFC, while it has less influence on HistSFC, particularly for n-nD queries. As a result, the HistTree is more effective on correlated data queries than independent ones. This is more significant in n-nD queries, in Figure 14. In Figure 12 and 14, the odd pattern that effectiveness at 2D is the highest is caused by the small FPRs. That is, HistSFC most of the time can return very accurate answer, with few false positive points. So, according to Formula 6 (Section 4), this will result in a large value of e. The formula may thus be improved in the future. Despite this, the effectiveness continues to grow as n goes up, which keeps in line with the theory.

5.3 Real data test

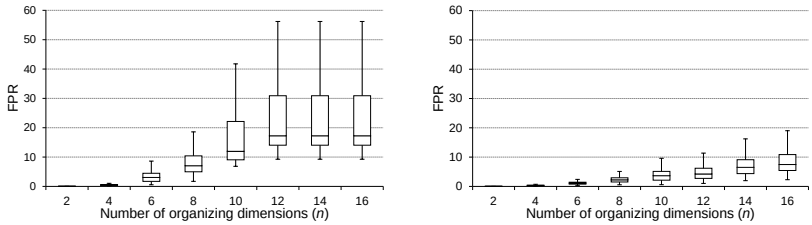
Real data test is based on AHN2, the Dutch national terrestrial point cloud. The testing data is part of AHN2 with XYZ coordinates, containing 2 billion points. Its MBB is [13427.64, 363052.95, -3.57; 22000, 380507.68, 100] in spatial reference system Amersfoort / RD New, EPSG:28992. To prompt visualization of the data, we also added a continuous LoD (cLoD) dimension. The cLoD value of a point actually represents the importance of the point during rendering. A smaller value corresponds to a higher importance (Liu et al., 2018, Van Oosterom, 2019). We computed cLoD using the method introduced in (Van Oosterom, 2019), and it obeys an exponential distribution, similar to D_3 in RealSim. The final range of cLoD dimension is (0, 12000) after stretching. CLoD receives more frequent access than Z in most spatial applications, e.g. in modelling and processing. Therefore, we built 3D(-key) solutions based on X, Y and cLoD organizing dimensions and attached Z as a property dimension. We also built 4D solutions with all dimensions in the key. HistTrees were created with threshold 10,000, to avoid a large occupancy in the memory.

Considering common query scale (size), as well as typical applications, we devised and selected 5 representative queries among others (Table 6):

Q1	XY MBB [16671.1, 370494; 16896.76, 370735.45]
Q2	XYLoD MBB [15281.52, 378658.19, 0; 18320.86, 380248,9000]
Q3	XYLoD MBB [16664.54, 370486.56, 0; 17997.76, 372036.45, 9000]
Q4	XYZ MBB [16902.29, 365439.3, 30; 19999, 367189.4, 100]
Q5	XYZLoD MBB [16902.29, 365439.3, 1.5, 0; 19999, 367189.4, 100, 9000]

Table 6. AHN2 queries

Q1 selects points in the region of a small town. Q2 locates in the coastal area, where half of the XY-plane is sea with no points.



(a) PlainSFC (b) HistSFC

Figure 7. FPR for 2-nD query in IdealSim

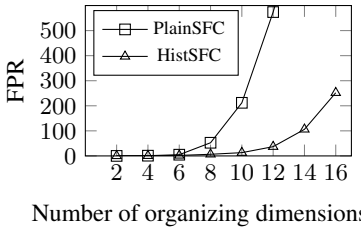


Figure 8. Mean FPR for n-nD query in IdealSim

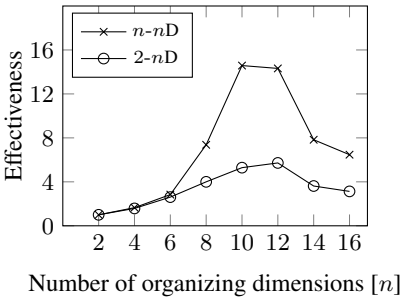


Figure 9. Effectiveness of HistTree of 2 query types in IdealSim

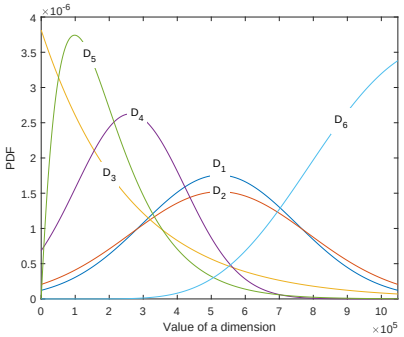


Figure 10. PDFs of different dimensions in RealSim

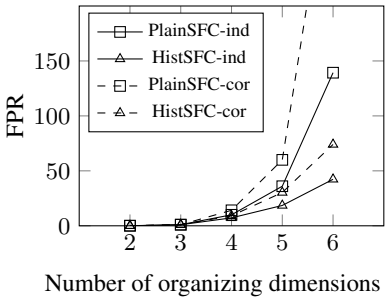


Figure 11. Mean FPR for 2-nD query in RealSim

Q3 locates in the urban area, and contains similar number of points as Q2 (Table 7). Q4 aims to select ultra high objects. It actually has no upper constraint in Z; so, we use the largest Z value as the boundary to form a cube. Q5 is a typical ground filtering query, which removes massive ground points acquired by LiDAR, and only reserves objects above the ground. For all queries, we set $T = 1000,000$, as the dataset is large and our server is efficient enough to process such amount of ranges. Table 7 lists the results.

For Q1, FPR of HistSFC solutions is lower than that of PlainSFC solutions, thanks to the finer SFC ranges adopted. However, as points are distributed evenly on the XY-plane, the gap between two 3D solutions is insignificant. Comparing Q2 and Q3, HistTree works more effective for Q2, when 3D solutions are employed. This is because Q2 (coastal region) contains more vacant area than Q3, which increases inhomogeneity of the data. However, the advantage of HistSFC caused by uneven distribution on the XY-plane diminishes using 4D solutions. 3D solutions do not encode Z into SFC keys. Consequently, for Q4 and Q5, points with any Z value which fall into the XY or XYLoD range will be selected. This causes huge FPRs, especially for Q4 where the selectivity on Z is only 0.09%.

For all cases, HistSFC solutions improves the accuracy, compared with PlainSFC solutions. Besides, despite that 3D solutions are still preferable for low dimensional queries, 4D solutions can handle more types of queries without significant deterioration. This is more evident when the query selects dimensions not existing in the 3D keys.

6. CONCLUSIONS

This paper presents how PlainSFC and HistSFC behave in the nD space for window queries. Specifically, we developed theorems which prove that up until a certain dimensionality, the FPR of PlainSFC continues to grow as dimensionality rises. By simulating various nD point clouds, tests verified this pattern. As PlainSFC performs poorly to process queries on skewed data, we developed the HistSFC approach. Both theory and practice indicate that HistSFC can effectively decrease FPR of all kinds of window queries. The effectiveness keeps growing within a certain dimensionality, and this is even more evident when dimensions correlate to each other inside the data. Additionally, we found that HistSFC delivers more stable performance than PlainSFC in all dimensional spaces. It can alleviate worst cases significantly. Through tests on real data, we also assert that when the dimensions queried do not exist in the key, the performance will deteriorate significantly.

When applying HistSFC in practice, a proper T is essential. It can be acquired by conducting simple selection on flat tables until an optimum is found, as the ranges are stored in a flat table for joining. The FPR of the solution should be controlled as low as possible: as a larger FPR is inevitable when dimensionality is high, we suggest only putting selective dimensions in the key. Logs may be used to determine the organizing dimensions.

Generally, the methodology developed, including the mathematical framework based on random variable distributions and the simulations, can be applied to research in other contexts. For example, nD spatial data caching, streaming and rendering all need optimal data organization. In the future, we plan to perform a more comprehensive benchmark test, elaborating more solutions. Also, the data can be more generic, not only Lidar points. We could also extend the data structures to address other spatial operations such as kNN search and querying with irregular geometries.

ACKNOWLEDGEMENTS

The authors would like to thank the Chinese Scholarship Council (CSC) and Fugro for supporting this research. Gratitude also goes to Ordnance Survey GB and 1Spatial for sponsoring the publication of this paper. Besides, grateful thanks are also extended to Rod Thompson for his help in editing the paper.

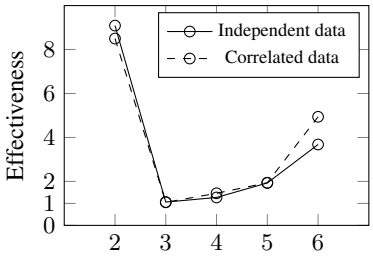


Figure 12. Effectiveness of HistTree for 2-nD query in RealSim

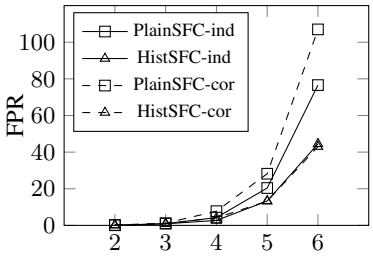


Figure 13. Mean FPR for n-nD query in RealSim

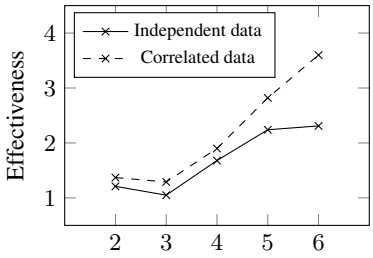


Figure 14. Effectiveness of HistTree for n-nD query in RealSim

Query	Query window		Exact answer	3D PlainSFC	3D HistSFC	4D PlainSFC	4D HistSFC
Q1	XY	Output (k')	717,342	749,694	744,476	984,613	802,676
		FPR	-	4.51%	3.78%	33.95%	9.2%
Q2	XYLoD	Output (k')	498,286	514,772	503,162	607,855	536,087
		FPR	-	3.31%	0.98%	22%	7.59%
Q3	XYLoD	Output (k')	505,101	517,296	512,597	596,790	539,851
		FPR	-	2.41%	1.48%	18.15%	6.88%
Q4	XYZ	Output (k')	85,090	100,608,118	100,217,866	645,819	186,101
		FPR	-	1.181×10^3	1.177×10^3	659%	119%
Q5	XYZLoD	Output (k')	524,730	1,594,469	1,577,980	1,767,020	1,386,908
		FPR	-	204%	201%	237%	164%

Table 7. FPRs of PlainSFC and HistSFC for AHN2 querying

REFERENCES

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. *Very Large Data-Bases*, 28–39.

Böhm, C., Berchtold, S., Keim, D. A., 2001. Searching in High-Dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases. *ACM Comput. Surv.*, 33(3), 322–373. https://doi.org/10.1145/502807.502809.

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). https://doi.org/10.3390/ijgi7080327.

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. https://aircconline.com/ijdms/V12N3/12320ijdms02.pdf.

Liu, H., van Oosterom, P., Meijers, M., Verbree, E., 2018. Towards 10^{15}-level point clouds management-a nD PointCloud structure. *Proceedings of the 21th AGILE International Conference on Geographic Information Science*, Association of Geographic Information Laboratories for Europe (AGILE).

Martinez-Rubi, O., Van Oosterom, P., Gonçalves, R., Tijssen, T., Ivanova, M., Kersten, M. L., Alvanaki, F., 2015. Benchmarking and improving point cloud data management in MonetDB. *SIGSPATIAL Special*, 6(2), 11–18.

Meijers, M., van Oosterom, P., 2018. Clustering and indexing historic vessel movement data with space filling curves. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 42(4), 417–424. https://doi.org/10.5194/isprs-archives-XLII-4-417-2018.

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

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, 2013a. Indexes and Index-Organized Tables. `https://docs.oracle.com/database/121/CNCPT/indexiot.htm#CNCPT721`.

Oracle, 2013b. SDO_PC_PKG Package (Point Clouds). `https://docs.oracle.com/database/121/SPATL/sdo_pc_pkg-package-point-clouds.htm#SPATL172`.

Pajić, V., Govedarica, M., Amović, M., 2018. Model of Point Cloud Data Management System in Big Data Paradigm. *ISPRS International Journal of Geo-Information*, 7(7), 265.

Pavlovic, M., Sidlauskas, D., Heinis, T., Ailamaki, A., 2018. Quasii: query-aware spatial incremental index. *21st International Conference on Extending Database Technology (EDBT)*, 325–336.

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.

Van Oosterom, P., 2019. From discrete to continuous levels of detail for managing nd-pointclouds. `http://nd-pc.org/documents/vario-nD-PC-v7.pdf`.

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. https://doi.org/10.1016/j.cag.2015.01.007.

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

Weber, R., Schek, H.-J., Blott, S., 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *VLDB*, 98, 194–205.

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.