HISTSFC: OPTIMIZATION FOR NDMASSIVE SPATIAL POINTS QUERYING

HaichengLiu[1], Peter van Oosterom[1], Martijn Meijers[1], XuefengGuan[2], Edward Verbree[1], Mike Horhammer[3]

[1]Faculty of Architecture and the Built Environment,
Delft University of Technology, Delft, the Netherlands
[2]Wuhan University, Wuhan, China
[3]Oracle Inc, Nashua, USA

ABSTRACT

Space Filling Curve (SFC) mapping-based clustering and indexing works effectively for point clouds management and querying. It maps both points and queries into a one-dimensional SFC space so that B+-tree could be utilized. Based on the basic structure, this paper develops a generic HistSFC approach which utilizes a histogram tree recording point distribution for efficient querying. The goal is to resolve the issue of skewed data querying. Besides, the paper proposes an agile method to compute a continuous Level of Detail (cLoD), and integrates it into HistSFC to support smooth rendering of massive points. Results indicate that for range queries, HistSFC decreases the False Positive Rate (FPR) of selection by maximally 80%, compared to previous approaches. It also performs significantly faster than the state-of-the-art Oracle SDO_PC solution. With improved performance on visualization and k Nearest Neighbour (kNN) search, HistSFC can therefore be used as a new standard solution.

KEYWORDS

Point Clouds, Histogram, Space Filling Curve, Benchmark, nD

1. INTRODUCTION

Point clouds become increasingly used in geomatics: for one thing, the prosperity is due to the advent of all kinds of sensors for data acquisition, to name a few, LiDAR, GPS and 3D cameras; for another, new applications such as robotics and Virtual Reality (VR) need the support of highly accurate spatial data. Considering the repeated scans of the same area periodically, data collected can be billions, even trillions (10^{12}) of points [1]. The Dutch national digital elevation model (AHN2) contains 640 billion points collected by Airborne Laser Scanners (ALS). Also, it is pre-calculated that the Dutch national terrestrial point cloud could achieve 35 trillion points by adopting dense image matching on streetphotos.

Large data size aside, the realm also experiences a revolution that due to various needs, applications are stepping to nD space. Apart from routinely concerned spatio-temporal dimensions, other dimensions such as Level of Detail (LoD), classification and identity constitute indispensable part in the data. Consequently, the incorporation of more dimensions in the data organization also becomes imperative. Dimension and attribute are equivalent terms. In terms of storage, two types of dimensions are identified. One type is called *organizing dimension* which is utilized to cluster and index the data, e.g. X/Y/Z/Time. The other is the *property dimension* such as colour, intensity and classification which is not frequently used in the SQL WHERE clause. Depending on applications, these two types of dimension are interchangeable. Specifying organizing dimensions crucial for efficient datamanagement.

As one of the common dimensions in spatial data, LoD which represents the importance of each point is often leveraged to balance accuracy and efficiency of applications. For visualization, software normally renders data at a certain LoD and if more details are required, then points with high LoD values would be loaded and rendered. Current 3D point cloud viewers mostly rely on Octree data structure to realize LoD, and each level in the Octree represents a LoD. This treatment has a negative effect that points pop up in blocks (Figure 1). Also it consumes large memory, because extra points are buffered, as blocks are selected instead of accurate query region. To guarantee a natural and smooth rendering process, the discrete LoD (dLoD) structure of Octree should be replaced by a continuous LoD (cLoD)structure.

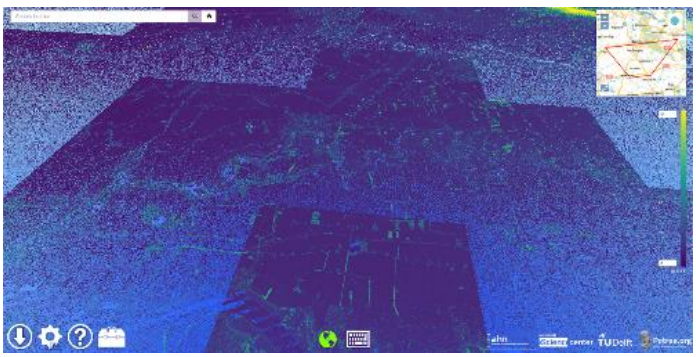

Figure 1. Block patterns introduced by the Octree structure

To facilitate massive nD point clouds querying and visualization, we design and implement a novel HistSFC framework including data structures and advanced algorithms in this paper. It is based on Space Filling Curve (SFC) mapping-based indexing [1, 2] which maps both points and queries into a one-dimensional SFC space so that one-dimensional indexing structure such as the B+-tree could be utilized. Throughout the paper, we use the term PlainSFC to represent normal SFC mapping-based indexing. The querying time cost of PlainSFC is $O(r + r\log N + k')$, where r is the number of SFC ranges generated for querying; N is the input size, while k' refers to the output size. However, the range generation process implemented before [3, 4] is mechanical and does not take point distribution into account, and this will result in a large r that undermines the performance, when points distribute inhomogeneously in the nD space. HistSFC utilizes a secondary histogram tree (HistTree) index to resolve the issue. Moreover, it supports range queries and advanced spatial functionalities through the compound structure, and high efficiency has been achieved. To summarize, the research contributionsare:

1. We devised and implemented an nDHistTree which applies an variable depth tree structure adapting to point density. It effectively decreases r and thus improves the performance.

2. The main querying process of HistSFC lies in an overlap detection between HistTree nodes and the query window. We then identify three types of nodes according to their spatial relationship with the query window, i.e. outside, inside and boundary. The boundary nodes influence the False Positive Rate (FPR) of selection significantly, which then impacts the performance of a secondary filtering. We thus developed an adaptive range generation method to refine boundary nodes and thus optimized the whole queryingprocess.

3. We put forward a generic algorithm for cLoD computation and integrated cLoD dimension into data organization, which is a significant improvement that benefits major Geographic Information system (GIS) applications. Tests indicate that with regard to visualization, cLoD could on the one hand produce more representative results, and on the other hand decrease data volume, compared todLoD.

4. We established mathematical equations to realize realistic 3D perspective view selection. We additionally devised a k Nearest Neighbour (kNN) algorithm. Both functions are implemented based on exact match between HistTree nodes and irregular query geometries including nD spheres and cones, to take full advantage ofHistSFC.

5. Benchmark tests demonstrate the superiority of HistSFC over PlainSFC and Oracle SDO_PC solution [5]. Besides, for multidimensional queries, a solution with 4 organizing dimensions can be preferable to a solution with 3 organizing dimensions and a propertydimension.

The rest of the paper is organized as follows: Section 2 introduces PlainSFC and basic terminology used in the paper. Then, Section 3 presents HistSFC, including the foundation of HistTree and its interaction with different dimensions, especially cLoD. Two spatial functions based on HistSFC are described in Section 4. This is then followed by Section 5 which presents practical tests and evaluations. The last two sections review previous work and conclude.

2. PLAINSFC

SFC is a curve that passes by all basic units in a multidimensional space only once. It reserves spatial relationships of objects it covers and has been widely used to index spatial data [6]. Among all SFCs, Morton curve (also called Z-order or N-order curve) is commonly studied and practiced due to the simplicity of mapping functions [7, 8]. It is based on interleaving the bits from the coordinates. Points are then sorted according to the SFC codes to be grouped together, while their spatial relationship still retains.

By truncating the Morton key, we can construct upper levels in the SFC hierarchy. In Figure 2, the four points residing in the low left box all starts with 0000, and when we truncate the last two bits of their keys, they could be represented by a point of which the key equals 0 in the upper level. That is to say, SFC contains an implicit 2^d-tree, where d is the number of organizing dimensions.

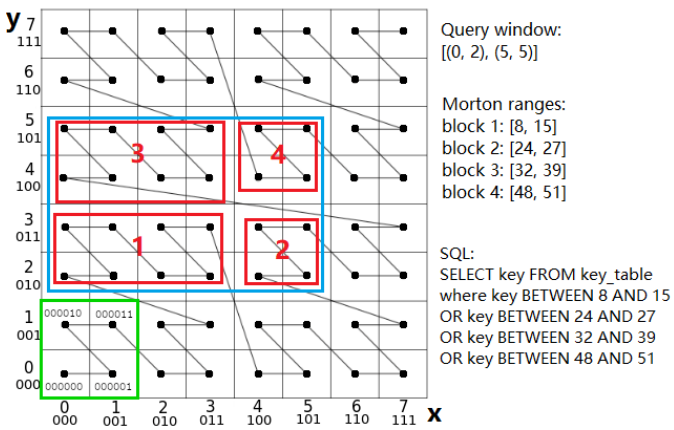

Figure 2. The conversion of a 2D query window into 1D Morton ranges

For compact and non-redundant storage of real point cloud data, each point is encoded as a full resolution SFC key, followed by property dimensions. The SFC key consists of a combination of all organizing dimensions. Mostly, values of the organizing dimensions contain decimals, so all values are scaled up to integers for encoding. Such a full resolution key can be decoded to the original coordinates, which is impossible for partial resolution keys [2] without additional storage

of dimension values. Besides, due to uniqueness of each full resolution key, they can be used as the primary key in tables which is indexed more efficiently[9].

Figure 3 presents the work flow of PlainSFC. For storage, after computation of SFC keys, PlainSFC adopts the B+ tree structure to manage them. Oracle has implemented B+ tree in Index-Organized Tables (IOT) [10]. The key and other property dimensions are stored in leaf nodes present in normal table, while the internal structure utilizes the SFC keys to organize and index the storage. The main advantage of IOT is that the index and data are combined together instead of two separate structures. Compared with storage using normal index structures, it can save at least 2× space usage. The advantageous IOT technique is used to realize the B+ tree structure throughout the paper.

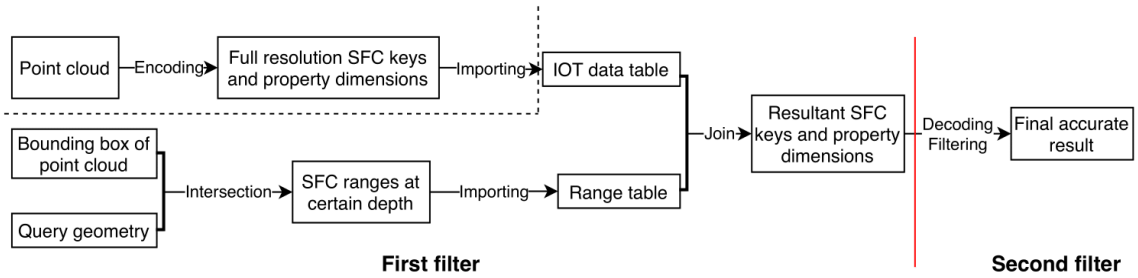

Figure 3. The loading and querying procedure of PlainSFC, divided by the dash line

As regard to querying, PlainSFC adopts two filters to process (Figure 3). PlainSFC first maps the query window from original multidimensional space into 1D SFC ranges. It then searches the data using the ranges. In practice, these ranges could be inserted into a SQL WHERE clause for selection. 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 2, 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. This process goes on recursively until a certain depth which is predefined. Some of the final ranges may still be neighbors which were divided in the refining process. Some are close to each other with small amount of outliers in between. These ranges will be glued in the end to reduce total number of ranges for querying, while introducing insignificant error. As each range corresponds to a continuous storage on disk, lots of jumping access to the disk can thus be avoided.

During the range computing process, as Figure 4 shows, if a quadrant is totally inside the query geometry, the range would be exported directly without further decomposition. Near the query boundary, the process continues until the maximum depth defined. A large depth would cause huge number of ranges generated, slowing down the first filter. A small depth however results in rough query result, where the error comes from coarse boundary cells (Figure 4). This is also unacceptable because it moves the burden to a secondary filter for an accurate answer. Such an optimal depth could be derived by performing benchmark tests, while other depths around the optimum might also be tolerable [4]. After computation of ranges, PlainSFC exports them into a range table and then joins it with IOT data table to select SFC keys and possibly other property dimensions. A secondary filtering would be conducted in a following step. For certain applications, the filtering may be unnecessary, e.g. visualization.

The time complexity of the whole querying process is $O(r + r\log N + k')$, where k' refers to the output size of the first filter before the final output k ($k' \geq k$). When points are distributed evenly in the space, the time complexity could be approximated by $O(k\log N + k)$ because B+ tree on

SFC keys make a continuous range querying take O(log$N$ + k_i) (note: $\sum k_i = k$) time to accomplish, and the upper boundary of r is k. The individual k in the time complexity covers decoding, filtering and writing processes which are slow in practice. Techniques such as parallelization might be conducted to address the issue. Besides, in most GIS applications, the scale of k is limited. So, in general, if N is big enough, say 10^{15}, the advantage of PlainSFC will be very significant compared with the flat table approach which takes O(N + k) time. Thebottleneck of the approach lies in cases where points are inhomogeneously distributed in the space. This happens frequently when points are in 3D or higher dimensional spaces. For instance, ALS points mostly locate on the ground, i.e. biased distribution in Z dimension. The consequence is a large r from the first filter, maybe much larger than k, as it elaborates large amount of vacuum ranges with no points inside. By increasing the query geometry size, the influence of r becomes dominant.

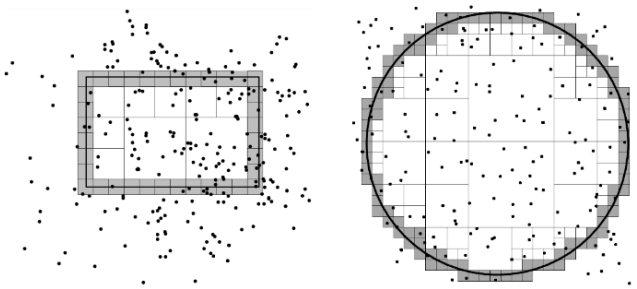

Figure 4. Recursive partition of the MBB of data to match different query geometries, left: rectangle, right: sphere (adapted from [11]), normally with large SFC cells inside and small SFC cells on theboundary

3. HISTSFC

Part of the ranges generated by PlainSFC may actually contain no points when data distribution is skewed. On the one hand, this implies extra time is spent on generating empty ranges; on the other hand, large amount of empty ranges increases the time cost of data selection. 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 data distribution:

- It is built during data loading process or afterit.
- It adopts a pointer-based structure (Figure5).
- It serves to guide the generation of SFC ranges forquerying
- It could be automatically updated when the data table ischanged.
- It is a generic data structure that could be applied for nD pointclouds

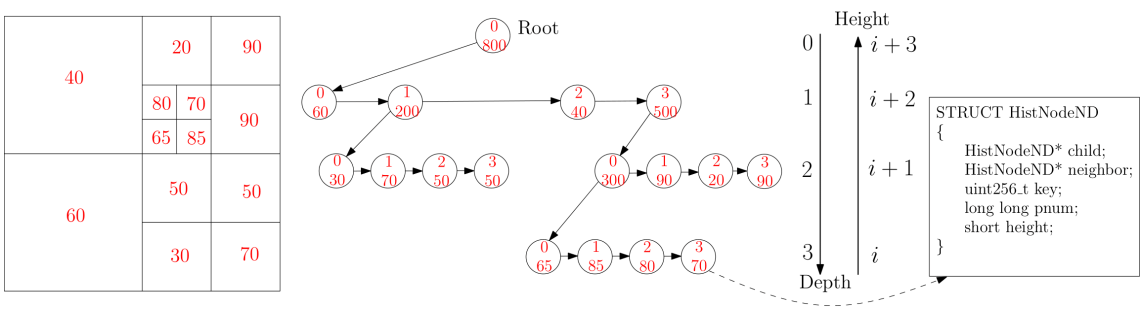

Figure 5. A 2D HistTree example, with threshold being 100; left: point counting, middle: pointer structure of HistTree, with each node storing a SFC key and number of points, right: structure of a HistTree node

As is shown in Figure 5, 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. From the pointer of the parent SFC node, all its children can be visited. A height field is used in a HistTree node to distinguish different nodes, because branch nodes at different levels may possess identical keys. Alternatives exist, for instance, MD-Hbase [12] employs the longest prefix of the binary SFC key to represent a region. For example, 1100** in 2D space covers cells from 110000 to 110011. It is convenient to retrieve the parent node from a child node by truncating the key, e.g. the parent of 1100** is 11****. However, CHAR type might be implemented to support such schema, which is inefficient for computation and storage compared to a NUMBER type which is implemented in HistSFC. 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.

3.1. HistTreeConstruction

As prior investigation, we developed and compared several options to build HistTree, including top-down, bottom-up and streaming approaches.

The top-down approach first creates a root node to cover the whole point cloud. Then it builds nodes below by reading the point list once per level. The time complexity is thus O(N log$N$). On the contrary, The bottom-up approach starts creating HistTree nodes from raw points, and incrementally builds nodes at upper levels. It thus incurs high memory consumption, while the time complexity is O(N). An improved approach is to construct nodes from a middle level (ML). That is, it first determines an appropriate ML where the number of points contained by most nodes is around the threshold. It then directly builds all HistTree nodes at that level. This is followed by a checking step in which the approach picks up those overloaded ML nodes, and then reads the point list once more to fix them using a top-down approach. After it, the top part of HistTree is constructed from ML by adopting a merge-collapse procedure. The challenge of this approach lies in devising a robust algorithm to compute the height of ML. Considering variety and complexity of different datasets, manual interventions may still be needed. Moreover, the memory cost still scales linearly as Ngrows.

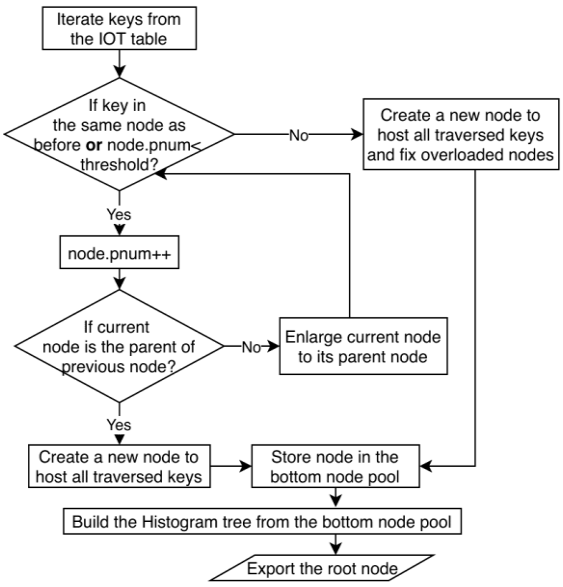

Figure 6. HistIOT approach to build HistTree

The streaming approach HistIOT, as is shown in Figure 6, utilizes the IOT to build HistTree. Basically, HistIOT reads sorted keys sequentially and meanwhile computes a key PK that could be the parent of all traversed keys. HistIOT stops until it encounters a key that belongs to the sibling or parent of PK, and the number of keys traversed exceeds the threshold. Then HistIOT returns to the beginning of this traversal and creates child nodes of PK. Afterwards, HistIOT continues and starts a new traversal. When the scanning of IOT is completed, nodes are aggregated till the root node. HistIOT has to be implemented after data loading. The order of the keys is essential to HistIOT, and it thus costs O(N log$N$) time. But as the IOT is a part of HistSFC, sorting the keys is indispensable. Meanwhile, the memory usage is around a constant in theory. Another advantage is that once the IOT is built, HistIOT will be a fully streaming process. The actual performance of the algorithm is assessed in Section 5.3.

3.2. cLoDComputation

LoD is a critical factor which is always concerned with modelling. It is a general way to break down massive computation to a solvable scale. Computation can be controlled at a certain LoD and if more details or higher accuracy is required, points at lower levels would get involved for processing. LoD is also related to importance of points, and a small LoD value corresponds to high importance.

As analysed before, a cLoD could improve users' comprehension and accuracy of spatial applications compared with dLoD. We then generalize previous random sampling method [3, 13] to compute cLoD values, by adapting conventional d-ary tree structures to continuous levels [14].

In the 1D binary tree, 2D Quadtree and 3D Octree, the value of a level is an integer. To convert them into cLoD, we first split integer levels into half levels, i.e. one refinement as indicated in Figure 7. After the refinement, the distribution of number of points still remains a geometric series, while the common ratio changes from $2^d$to $2^{d/2}$, where d is the number of organizing dimensions. Assume the levels have been refined $i$times where $i\rightarrow\infty$, then the Probability Density Function (PDF) of a decimal level l is

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

where L is the original maximum level, and l refers to the level of LoD. The Cumulative Distribution Function (CDF) is:

$$F(l) = \frac{2^{l(d-1)} - 1}{2^{(d-1)(L+1)} - 1} \tag{2}$$

by adopting a random generator U which exports values between 0 and 1, it is possible to assign the cLoD value to each point based on:

$$cLoD = l = \frac{\ln\left((2^{(d-1)(L+1)} - 1)U + 1\right)}{(d-1)\ln 2} \tag{3}$$

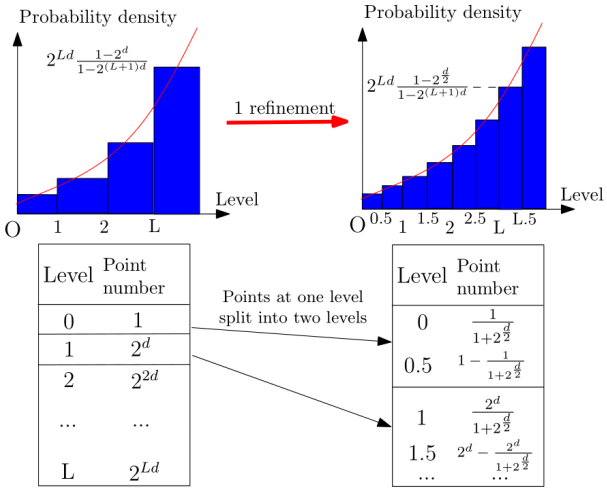

Figure 7. Conversion from dLoD distribution to cLoD distribution

In GIS applications, LoD dimension is queried and used more frequently than Z dimension. Queries normally cover the whole range of Z dimension as users are not keen on absolute Z values. On the other hand, LoD is an effective way to sample data and could have adaptive implementations according to different use cases, e.g. zoom in and out of a large scene. From this point of view, it is reasonable to manage cLoD as an organizing dimension. More importantly, due to the exponential distribution, cLoD could benefit from HistTree and thus it is deeply integrated into HistSFC. When managing a LiDAR point cloud for instance, we could encode cLoD together with XY dimensions in a 3D key and build a 3D HistSFC solution with Z as an additional property dimension. However, cLoD needs to span a similar range as XY, so that there is no priority on any dimension inside the key when selecting. This could be achieved by scaling the result from Equation (3). Since the specific computation is transparent to users, selections on cLoD are ought to be adjusted at backends according to the actual use.

If Z or another dimension (depending on data) is still needed, and 4D queries are executed frequently, then a 4D-key HistSFC solution can be built. The HistTree could contribute more because Z values are also unevenly distributed in most cases, e.g. points align along horizontal surface. This effect is verified in Section 5.6 for generic purposes.

3.3. Querying

The query is executed with the aid of HistTree (Figure 8). Unlike PlainSFC which adopts a fixed depth for recursive decomposition of nodes, HistSFC employs a flexible strategy which is adaptive to point density. It balances two thresholds to control the querying process. One is an overall FPR which refers to the expected FPR of selection without a secondary filter. The setting depends on applications, and visualization for example could tolerate a large FPR. The other threshold is the maximum number of output ranges. It is mainly determined by the hardware environment considering performance. Besides, it relates to r in the whole time complexity (Section 2).

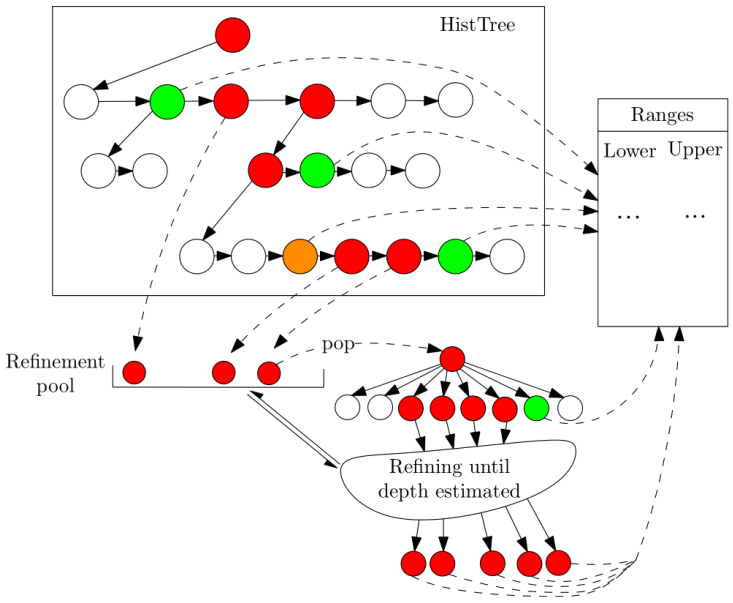

Figure 8. Range generation using a 3D HistTree; green nodes: within query geometry, red nodes: intersecting query geometry; white nodes: outside query geometry.

Starting from the root node, the extent of each node could be computed using its height and SFC key. Then by performing overlap computation between branch nodes and the query geometry 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 geometry. The searching process would stop at these nodes and export their ranges. The other part of resultant nodes are leaf nodes which fall on the boundary of the query geometry. As regard to this portion of nodes, two functions are performed:

1. During the overlap computation, the intersection type of each node is recorded and stored in the node. The intersection type refers to the number of dimensions of the specific node intersecting with the query geometry (Figure9).
2. A depth estimation function that deduces the depth that each leaf node has to reach to satisfy the overall predefinedFPR.

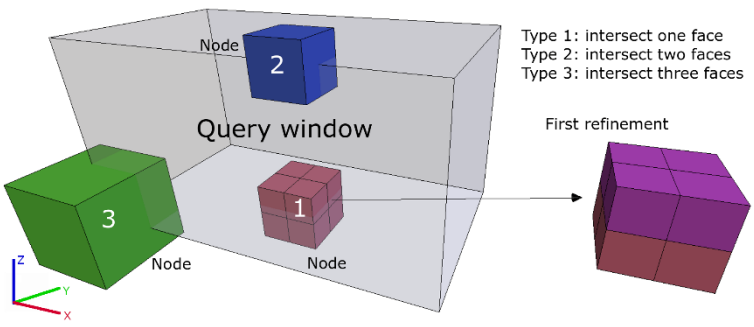

Figure 9. Intersection types between HistTree nodes and the query window in 3D space

The depth estimation function first estimates the number of points inside the query geometry. It is the sum of the number of points covered by the inner nodes and a portion of points from boundary leaf nodes. Such portion is determined by the intersection type of the node. The

subtraction of the number of points estimated from the total number of points in all retrieved nodes (*T*) is the number of extra false positive points (*E*). If *E/T* is larger than the FPR threshold, then the boundary leaf nodes would be refined (Figure 9). Otherwise, ranges of all leaf nodes involved would beexported.

When it comes to the refinement, the depth estimation function computes the specific depth each boundary leaf node should be refined to, to reach the FPR threshold. The improvement of selection accuracy at each refinement step varies for different types of boundary nodes (Figure 9). This is because each refinement decomposes the node into 8 children, while the number of children that need to be further refined depends on the intersection type of the original node. As is shown in Figure 9, the node of type 1 will be cut by half after the first refinement, i.e. only the upper 4 children will be further refined and other 4 are abandoned. The decomposition of the node only works along Z dimension in this case. Consequently, each refinement will improve the accuracy by a factor of 2. Similar principle applies to nodes of other intersection types. The depths computed are therefore different from eachother.

It is likely that a boundary leaf node is refined to a very deep level due to high point density of the specific node. This can cause crucial degradation in performance, especially in high dimensional spaces where each decomposition generates large quantities of child nodes. To avoid this, the refinement algorithm adopts a breadth-first structure and also uses the threshold of maximum number of ranges to confine the recursion. After it, the final ranges are joint with the IOT data table, and then the function returns the result afterdecoding.

4. ADVANCED SPATIALFUNCTIONS

Apart from regular range queries, the framework could also support more advanced spatial functions. In this section, we describe the algorithmic foundation of two common spatial operations, namely perspective view selection and kNN searching, and present how they could benefit from the data structure. Performance tests are performed in Section 5.

4.1. Perspective ViewSelection

As regard to visualization of point clouds, perspective view selection is a basic operation both indoors and outdoors. Not only spatial dimensions, but also LoD contributes to the function, because perceptually, only most important points could be seen when they are distant. Besides, visual perception also consider eccentricity and speed of movement [15]. Eccentricity refers to that for objects at the same distance, the objects directly in front should be shown clearer than the ones locate aside. Velocity also matters because details fade away as the observer moves faster [16], and it is especially critical in VR. In the following realization of the 3D perspective view selection, we demonstrate the idea of how to integrate distance and eccentricity into the mathematical expressions, while reserve velocity for future research. So the general constraints adopted are:

1. All points locate inside a 3D view frustum (Figure10).
2. The maximum LoD value of points selected decreases as distanceincreases.
3. When at the same distance, points on two sides have a lower maximum LoD value compared to those locate in the center of theview.

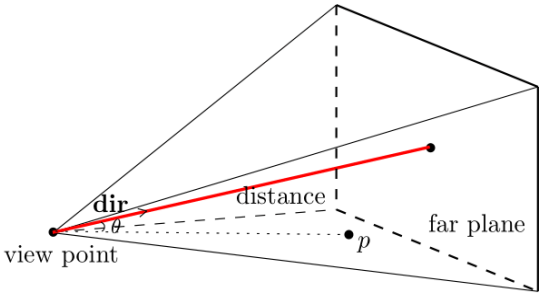

Figure 10. 3D view frustum

The first constraint incurs cutting off half planes:

$$a_i x + b_i y + c_i z \leq d_i (i = 0 \rightarrow 5) \tag{4}$$

, where a_i, b_i, $c_i$and $d_i$are constants, and they can be deduced based on the view point, direction and maximum view distance. Constraint 2 can be achieved by

$$\begin{gathered} \sqrt{(x-u)^2 + (y-v)^2 + (z-w)^2} \leq a_6 - b_6 LoD \\ 0 \leq LoD \leq \frac{a_6}{b_6} \end{gathered} \tag{5}$$

, where (u, v, w) are the coordinates of the viewing point. In order to elaborate constraint 3, the right part of Equation (5) is multiplied by a $cos(\theta)$ factor (Figure 10), i.e. $cos(\theta)$ = (**dir**·p)/(|**dir**|·|p|), where **dir**is the normalized direction vector. Then,

$$\begin{gathered} (x-u)^2 + (y-v)^2 + (z-w)^2 \leq (\mathrm{dir}_x(x-u) + \\ \mathrm{dir}_y(y-v) + \mathrm{dir}_z(z-w))(a_6 - b_6 LoD) \\ 0 \leq LoD \leq \frac{a_6}{b_6} \end{gathered} \tag{6}$$

Equation (6) represents a twisted 4D cone. Then, every point that satisfies both Equations (4) and (6) appears in the perspective view. Such a generic mathematical framework can be applied to both dLoD and cLoD implementations. But in order to create more realistic scenes as well as reduce processing load, we focus on the cLoD solution. Rather than individual point computation, the function leverages HistTree to check whether 4D nodes intersect two 4D geometries defined by Equation (4) and (6) respectively.

HistTree nodes that are totally inside both 4D geometries will be exported directly. For the leaf nodes intersecting both geometries partly or containing them, a secondary filter will scan them and acquire accurate result. In most cases, Z dimension does not influence the result significantly as the vertical vision span can be as large as 120°. Only objects nearby with a large height will be cut by the view. Then the 3D perspective view selection can be simplified by employing a subset of the equations. Moreover, to ease implementation, Equation (4) can be approached by half planes despite minor errors introduced.

4.2. kNN

The basic definition of kNN search is: given a point P, find its k nearest neighbors in a 3D point cloud PC. kNN has been researched broadly for point clouds. Key examples include noise

removal [17], registration [18] and change detection [19]. Oracle SDO_PC provides SDO_PC_NN function [20] to search for nearest k points in a point cloud given a certain point in the 3D space. kNN algorithms based on PlainSFC have also been proposed and achieved high efficiency [21, 22]. In fact, HistSFC approach could also prompt kNN using customized approach. The process is presented in Figure 11.

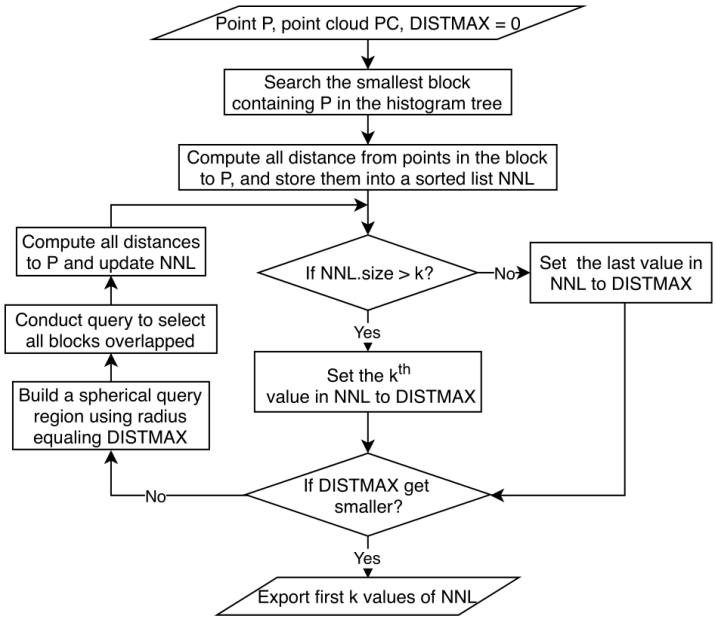

Figure 11. Execution path of kNN using HistSFC

The output structure is a key-value container for neighbors. The key is the distance, while value is the neighbor point. The container adopts a binary tree structure. In every iteration, a spherical query is executed. Then all distance values of points in HistTree nodes which intersect the ball are computed and exported to the container. When the size of the container is larger than *k*, the k^{th}key value of the container will be the diameter for searching in the next iteration. Otherwise, the last key value will be the diameter. The searching process ends when the diameter starts to decline. We assume diameter would not be the same for consecutive iterations as the probability is small enough to benegligible.

In the implementation, points checked in a new iteration will exclude those checked before, to decrease the cost of decoding which might be significant when the output is large. In the worst case, *k* HistTree nodes get involved in the searching process, so the time complexity is O(*k* log*N* + *k*). Besides, the adaptive refinement described in Section 3.3 could be applied to improve the performance.

5. EXPERIMENTALEVALUATION

Benchmark tests are performed to examine the real cost of algorithms in the paper. Experiments are designed to answer the following key questions:

- What is the actual cost of HistIOT (Section 3.1) to buildHistTree?
- How much improvement has been brought by HistSFC compared toPlainSFC?
- Is a 4D HistSFC solution preferable to a 3D HistSFC approach with an affiliated propertydimension?

- Compared with current state-of-the-art, how fast is the HistSFCapproach?
- Is cLoD really more suitable for the visualization of massive points thandLoD?

5.1. Data

The testing data is part of AHN2 with XYZ coordinates, containing in total 10 billion points. Its minimum bounding box is [13427.64, 363052.95, -8.54; 37999.99, 415990.94, 100] in spatial reference system Amersfoort/RD New, EPSG:28992. To support LoD, by applying Equation (3) and setting L to 11 which is a level capable to hold all points in an Octree, we compute a cLoD value for each point as the fourth dimension. The final range of cLoD dimension is (0, 12000) after stretching, which is of the same orders of magnitude as X and Y dimensions. Such a design balances dimensions inside a Morton key so that the selectivity of each dimension does not vary significantly.

As is mentioned, cLoD receives more frequent access than Z in most GIS applications. Therefore, we build 3D(-key) IOT solutions based on X, Y and cLoD organizing dimensions and attach Z as a property dimension. In 4D IOT solutions, XYZ and cLoD are encoded in the Morton key, while Z is scaled 10 times larger to increase its selectivity. Besides, Oracle SDO_PC solution applying a plain R-tree model will be implemented as a reference. In Oracle 12c, it functions the best by utilizing 2D blocks indexed by XY organizing dimensions, while Z and cLoD are property dimensions. This strategy is then implemented. In fact, Oracle provides an option to build a dLoD structure on the blocks, but for fair comparison, it isexcluded.

In order to learn the scalability of different approaches, we split the data into 5 equal vertical slices from west to east, i.e. each contains 2 billion points. Starting from the first piece, by adding one more piece each time, 5 different data stores are built. Table 1 presents the storage size of different solutions. Raw TEXT refers to point records with 4 fields stored in TEXT files.

Table 1. Storage size of 3D solutions (in GB)

Number of points	Raw TEXT files	IOT	SDO_PC
2,000,000,000	61.4	46.9	86.5
4,000,000,000	123.1	90.6	171.2
6,000,000,000	184.9	136.6	256.8
8,000,000,000	246.8	182.8	342.4
10,000,000,000	308.4	231.1	428

As 4D IOT solutions scale analogously to 3D IOT solutions, we only build it at the largest level, i.e. 10 billion points to compare with the corresponding 3D solution. The 4D IOT occupies 199.7 GB on the disk. It can be seen from Table 1 that SDO_PC solution is 2× larger than IOT approaches. This is because on the one hand, during Morton encoding for IOT, all points are shifted to the origin, so the actual values stored are drastically decreased. On the other hand, an additional ID column gets involved in SDO_PC Binary Large Objects (BLOB) and no compression is applied, which results in large storage of the SDO_PC solution. Oracle actually offers secure file compression with various compression levels, but it is not applied as it may cause queries to slow down.

5.2. Benchmarkframework

The benchmark test is performed on "testbed" server: a HP DL380p Gen8 server with 2 × 8-core Intel Xeon processors, E5-2690 at 2.9 GHz, 138 GB of main memory, a RHEL6 operating

system. The disk storage is a 41 TB SATA 7200 rpm in RAID6 configuration. A C++ framework is developed and used for the test (Figure 12). It is built specifically to maintain communication cost as equal as possible for all solutions. In plain English, the queries used for benchmarking are as follows:

- *Q1: selection of all points within a XY MBB [16671.1, 370494; 16896.76,370735.45]*
- *Q2: selection of all points within a XYLoD MBB [15281.52, 378658.19, 0; 18320.86, 380248.44,9000]*
- *Q3: selection of all points within a XYLoD MBB [16664.54, 370486.56, 0; 17997.76, 372036.45,9000]*
- *Q4: selection of all points within a XYZ MBB [16902.29, 365439.3, 30; 19999, 367189.4,999]*
- *Q5: selection of all points within a XYZLoD MBB [16902.29, 365439.3, 1.5, 0; 19999, 367189.4, 999,9000]*
- *Q6: selection of the nearest 30,000 points for a point [18011.7, 374867, 11500] in the XYLoDspace*
- *Q7: perspective view selection starting from a point [15093, 370758.2, 1] in the XYZ space, direction is north and the maximum distance is1km*

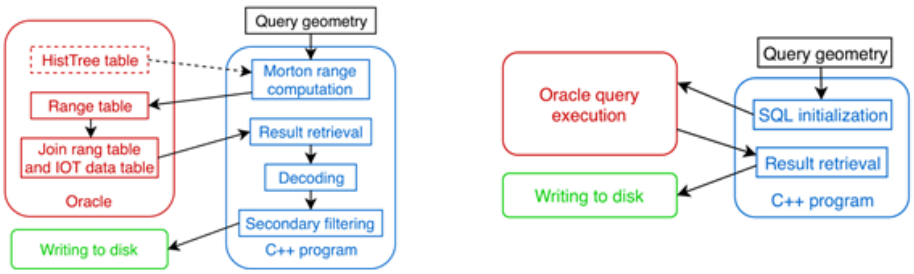

Figure 12. Query execution process for benchmarking: left: PlainSFC and HistSFC; right: SDO_PC

Their locations are depicted on a 2D map (Figure 13). All query regions fall in the west part and every data store elaborates them. In Q4 and Q5, the upper boundary of Z is 999, which indicates the query on Z is half open, i.e. ≥ 30 or 1.5. The output size of all queries span from 10^4 to 10^6, which keeps consistent with previous studies [1, 3]. For one thing, these are typical query sizes in GIS applications; for another, it avoids too large output so that the decoding and writing will not dominate the whole querying process.

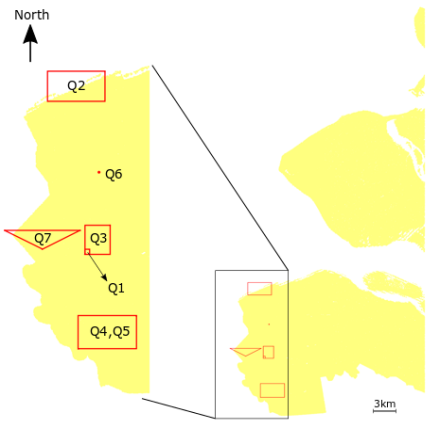

Figure 13. Query locations in the 2D scope of the test data, south west part of the Netherlands

In the tests, each query is executed 5 times, and the final figure for the query response time is the average of the middle 3 records with the lowest and highest records removed for each solution. The cache of both Oracle and the operating system are purged after each execution, so that what is measured can reflect the real time complexity instead of O(1) retrieval from the cache.

5.3. Exp1: HistTreeconstruction

HistIOT reads the sorted keys sequentially and when it overpasses the threshold, it will return to the beginning of the specific round to build HistTree nodes. In the implementation, every time HistIOT starts a new building session (i.e. after the last construction round), it stores every key scanned into a pool for fixing later. It utilizes 10,000 as the threshold to build 3D (XYLoD) HistTree. The size of HistTree of the five IOT stores range from 37 MB to 180 MB on the disk. Figure 14 shows the time cost and maximum memory consumption of HistIOT.

Time cost includes reading keys from IOT and HistTree construction in memory. In Figure 14, the time cost rises in a linear manner smoothly. When it comes to peak memory consumption, it increases all the way while converges to a constant in the end. The erratic shape of the curve is caused by processing specific irregular data tiles which result in different capacity of pools. As could be deduced, the peak memory usage will stay steady if no larger irregular data tiles are encountered. This is the major advantage of HistIOT when processing massive points.

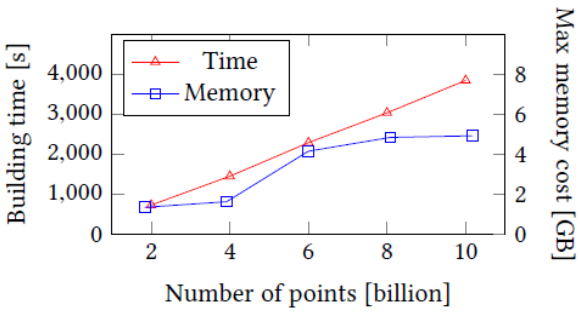

Figure 14. Time and memory cost of HistIOT

5.4. Exp2: FPR from the first queryingfilter

We perform experiments to examine whether HistSFC is an improvement of PlainSFC, from two aspects: (1) FPR from the first filter which is hardware independent; (2) query performance, presented in following sections. For computation of FPR, we adopt FPR = $|(k' - k)/k|$, where k' is the approximate result and k is the exact answer[23].

In this experiment, Q1 to Q5 are executed based on 3D PlainSFC, 3D HistSFC, 4D PlainSFC and 4D HistSFC solutions (Table 2). The secondary filter is turned off so that the result indicates the quality of SFC ranges. As mentioned in Section 3.3, HistSFC adopts two constraints to control the querying process: a FPR for refinement and a maximum range number threshold. PlainSFC employs a depth parameter indicating the depth the decomposition should reach, and also the maximum range number. We set the maximum range number of both approaches to 1 million, because by prior tuning, it is approximately an equilibrium between time cost on computing such number of ranges on "testbed" server and the accuracy of selection. Then we use a fairly small FPR for HistSFC and a large searching depth for PlainSFC so that both solutions generate 1 million ranges and reach the most accurate state. Table 2 shows the results. Please note that the FPR parameter driving the HistSFC is different from the FPR in the table which describes the

accuracy of the firstfilter.

Table 2. FPRs of PlainSFC and HistSFC for AHN2 querying, with 1 million ranges

Query	Query window		Exact answer(k)	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%

For Q1, FPR of HistSFC solutions is lower than that of PlainSFC solutions, thanks to the finer SFC ranges adopted. However, as points distribute 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) elaborates 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. The effectiveness of HistTree for Q2 and Q3 are both around 3 then. 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 is0.09%.

For all cases, HistSFC solutions improves the accuracy of the first filter, 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 elaborates dimensions not existing in the 3D organizations.

5.5. Exp3: performance of 3Dsolutions

In this experiment, we explore the scalability of 3D HistSFC and 3D PlainSFC by referencing the SDO_PC approach. The query execution time includes range computation, IOT selection using the ranges, decoding, secondary filtering and writing. We do not take HistTree loading time into account as the design of HistSFC assumes that once loaded, the HistTree would reside in memory for later use. After all, HistTree is originally devised as a compact structure with little memory cost. We set the FPR to 10% in the refinement process of HistSFC, so that it could stop decomposing ranges before reaching the maximum range number threshold (i.e. 1 million) for the purpose ofefficiency.

Figure 15 shows the performance of the three solutions for Q1 and Q2 respectively. Generally speaking, all solutions scale constantly with input size. For both queries, HistSFC takes the least time for querying, and it is followed by PlainSFC and SDO_PC. Specifically, in Q1, HistSFC spends around 800 to 1000ms for range computation, 950ms for IOT selection, 900ms for secondary selection (i.e. decoding + filtering) and 2000ms for writing. PlainSFC spends about 1450ms for range computation, which is mostly the extra time cost compared to HistSFC. This is mainly due to more ranges generated by PlainSFC. HistSFC produced 445,153 ranges, almost half of that of PlainSFC. The high time cost of SDO_PC is majorly owing to the long process to

unpack the blocks to retrieve actual coordinates. In Q2, the relative gap between HistSFC and PlainSFC becomes larger, which is caused by the lower FPR and less ranges of HistSFC. Additionally, SDO_PC degrades significantly because in Q2, the XY region grows, so more blocks are unpacked. Besides, all blocks retrieved incorporate the whole cLoD range as SDO_PC solution utilizes 2D blocks. Thus, large amount of false positive points decrease the performance. This can be improved by employing a 3D blocking approach, which we can compare to, once SDO_PC fully supportsit.

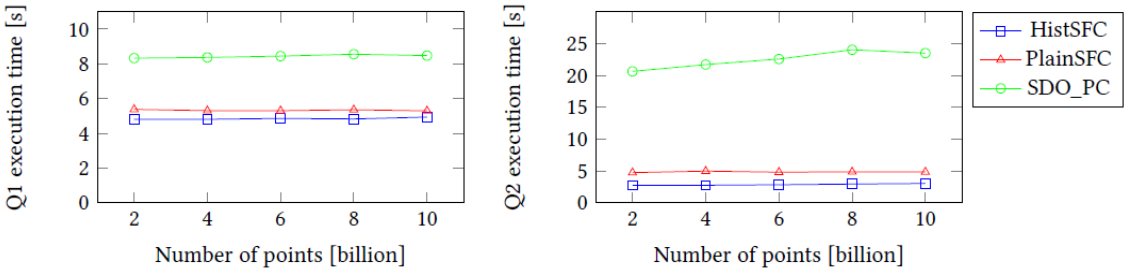

Figure 15. Performance of 3D solutions on Q1 (left) and Q2 (right)

5.6. Exp4: 3D HistSFC vs 4DHistSFC

The scalability of 4D approaches also remains steady for different queries, which is omitted for conciseness. This experiment compares the efficiency of 3D and 4D SFC solutions based on the largest store, i.e. entire sample. As opposed to the claim that Z is queried less frequently in GIS applications than other dimensions, this experiment involves Z to explore generic querying performance.

As Figure 16 presents, in general, HistSFC has superiority over PlainSFC, attributed to the low FPR. In particular, the largest gap happens in Q5, where 3D HistSFC is 1.42× faster than 3D PlainSFC. Current kNN implementation relies HistTree for querying, so PlainSFC is absent in Q6. With regard to Q4, most Z values fluctuating at 0, while points above 30m occupies only 0.08% of all points inside the XY range. So the linear scanning to filter Z after XY selection using the B+ tree dominates the whole time cost. Unlike Q4, Q5 uses 1.5 as the lower boundary of Z, which is close to the ground. So the FPR caused by Z is greatly reduced. Q4 indicates that 3D solutions may not always perform better than 4D solutions in terms of efficiency.

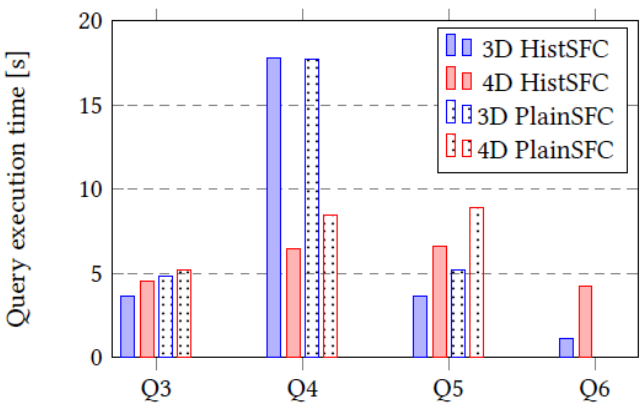

Figure 16. Response time of 3D and 4D SFC approaches

3D HistSFC reaches its best performance in Q6, i.e. kNN searching. It should be noted that the the 3D space refers to XYLoD, to cater to the data organization of 3D HistSFC. In fact, XYZ kNN is executed more frequently. To realize it, current 3D HistSFC should apply XY NN searching and meanwhile compute the 3D Euclidean distance with Z involved. In the worst case, it will conduct full table scan. However, this is not the case for 4D HistSFC which can cast 4D nodes into any 3D spaces without losing relevant dimensional information. So from the perspective of generality, 4D HistSFC holds the superiority.

To sum up, if 4D queries are the routine, then a 4D HistSFC should be the choice as it provides more generic access to the data without much performance degradation. In all other cases, a 3D HistSFC is more applicable for 3D or higher dimensional queries.

5.7. Exp5: dLoD vs cLoD

Q7 realizes a 3D perspective view selection. Through it, we demonstrate how cLoD could prompt visualization of massive points. The query is executed using all stores of 3D HistSFC. The refinement process is turned on for the sake of performance. It turns out that all 3D HistSFC stores take around 1900 ± 200ms to accomplish Q7.

Figure 17 presents the visualization of the view based on the dLoD structure and cLoD structure. The dLoD structure is not a strict Octree, as it is realized by selecting all 3D HistTree leaf nodes that intersect the view and LoD frustums and exporting them directly. Nevertheless, it still reflects the sharp boundary issue. The figure clearly shows the natural and smooth visual perception realized by cLoD. The output size drops from 500,446 points to 100,441 points, which also reduces I/O cost for visualization.

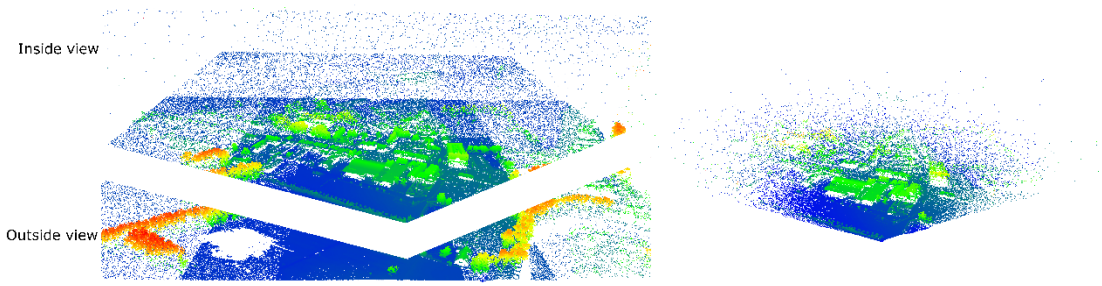

Figure 17. Visualization using different LoD schemes, left: dLoD scene manually split into two parts, with in total 500,446 points; right: cLoD scene with 100,441 points

6. RELATEDWORK

Efficient solutions for generic nD points querying have been proposed and implemented in computer science [24-27]. Among them, Pyramid-Technique [25] and iMinMax(θ) [27] scheme are two of the most referenced. 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. The approach maps each nD point into a one-dimensional space according to the pyramid the point belongs to and its height in the pyramid. All the resultant one-dimensional keys are then managed by a B+-tree structure to be indexed. The Pyramid-Technique is initially devised for hyper-cube shaped queries. Thus, when the query shape is non-hyper-cube such as a long rectangle, the efficiency will decrease significantly. The iMinMax(θ) is another technique to map nD points into 1D keys. It also uses B+-tree for indexing. For an nD point (x_1, x_2,… x_n), the

approach first locates the minimum or maximum dimensional value, x_{min} or x_{max}, and the corresponding dimension D_{min} or D_{max}. Then it computes the 1D key by using either $D_{min} + x_{min}$ or $D_{max} + x_{max}$. In this way, a point is first mapped to a partition (n partitions in total) based on the dimension component in the key. Then it is ordered within the dimension according to the extreme dimensional value. When querying, iMinMax(θ) decomposes the nD query window into n sub-query windows for selection. The time complexity also grows linearly as n increases [26].

Pyramid-Technique and iMinMax(θ) both separate index and data storage. Also, the keys are computed only for the indexing, while original records have to be traced through pointers stored together with keys. On the other hand, the SFC key in PlainSFC contains all the information of a point, and the information can be retrieved from the key by decoding. Consequently, PlainSFC saves significant amount of storage. Moreover, PlainSFC does not need mapping from the index to the original data for selection, which is more efficient. The second advantage concerns approximate selection. For visualization, the result from the first filter of PlainSFC is sufficient. However, neither Pyramid-Technique nor iMinMax(θ) takes spatial coherence into account for data management. Hence, the rough selection may return points which are distant from the query window. This may confuse the user when points are visualized. Due to the same reason, Pyramid-Technique nor iMinMax(θ) also incur high FPR for low dimensional queries. Block based approaches such as X-tree [28] and SR-tree [29] support spatial coherence, but they suffer from long block unpacking process for querying. This is a problem indicated in[1].

Querying skewed point clouds has also been addressed to some extent [2, 8], previously. The solution proposed in [8] first decomposes the 3D point cloud into blocks according to Hilbert curve [30, 31], and inside each block points are grouped into a BLOB. All blocks have a same capacity (similar to the threshold of HistTree) and thus, the block span is adaptive to the point density. When querying, it utilizes an Octree to retrieve the blocks that overlap with the 3D query window, and then conducts a secondary filtering to get final result. That is, the efficiency of the solution is directly related to the block capacity. HistSFC however could continue a refining process from leaf nodes, which improves flexibility and efficiency. Another study [2] targets at spatial objects with nonzero extents, and it handles skewed data issue based on the centroids of objects. It utilizes the CDF to transform original skewed data into uniformly distributed data, and then uses a PlainSFC for management and querying. However, the transformation is an expensive operation and thus inapplicable to large scale point clouds. The research of learned index structure [32] indicates that range index models such as B tree are CDFs which take a key as input and predicts position in a sorted array. Histograms can also be used as CDF, as a replacement of B tree according to the work. However, our paper proposes and verifies that they could be combined into a compound indexing structure and achieve higher efficiency. Besides, as a common technique to deal with non-uniform data distribution, histograms are largely used in major DBMSs for querying (statistics collection) because they incur little run-time overhead and produce low-error estimates with compact storage [23]. In particular, Oracle Spatial and Graph has developed state-of-the-art solutions to build spatial histograms for query optimization purposes [33], where HistTree can be integrated seamlessly.

With regard to cLoD, Cura [13] indicates that an ideal LoD should be homogeneous in space, insensitive to density variations, regular, efficient and deterministic. The study puts forward an implicit LoD scheme (MidOc) that uses the order of points to form a cLoD, i.e. by buffering different amount of points, users are at different LoD levels. For generic purposes, MidOc only considers geometry and picks up points closest to centres of Octree cells to constitute different layers of the LoD. Inside each level, points can be ordered following various strategies. Based on this work, an indoor cLoD which elaborates semantic information is also developed [34]. The work specifically concentrates on indoor navigation and points of accessible objects such as doors and windows get a higher place in the ranking. Zheng et al. [35] also develop a priority

ordering index which is actually the cLoD value for visualization of big datasets. The tests show that the approach is more efficient at determining representative points than random sampling. To realize gradual rendering of scenes in VR, Schutz et al. [36] first sample points in dLoD chunks into a vector array using spacing information. Then inside GPU, points are filtered according to the view frustum and target spacing in VR. However, the approach suffers from limited memory size of GPU. In comparison with method above, the advantage of our cLoD lies in rapid computation, generic usage and applicability to arbitrarily large point clouds.

7. CONCLUSIONS AND FUTUREWORK

This paper introduces HistSFC, an optimization for previous PlainSFC for querying. It harnesses HistTree to produce SFC ranges with low FPR and thus improves querying efficiency. This advantage is especially remarkable when points are unevenly distributed in 3D or higher dimensional spaces where multiple irregular dimensions exist. In addition, we propose a method to fast compute cLoD and then integrate it into the data organization to serve queries. This is a query optimization from the perspective of spatial cognition. With such data structure, 3D perspective view selection and 3D kNN functions are realized and they achieve high efficiency. Visualization experience has been improved in terms of both quality and quantity (i.e., less points) using cLoD. We also explore the impact of dimensionality of the SFC key on the performance of HistSFC. We therefore assert that the data organization should cater to the dimensionality of applications. If 4 dimensions are queried frequently, not necessarily at the same time, then a 4D HistSFC is preferable to a "3D + 1D" solution as the 4D model is generic to any lower dimensional queries with acceptable performancedegradation.

Limitations still exist, we therefore list the following as future work:

- Execute a more comprehensive yet strict benchmark test, e.g. more query types, larger data size and more solutions such as Oracle SDO_PC with a hybrid model stored in IOT.
- Study systematically on deciding the threshold of leaf nodes in HistTree as well as tuning the maximum number of ranges forquerying.
- Explore the generality of HistSFC by elaborating different dimensions from other data sources, e.g. time and identity dimensions from trajectorypoints.
- Develop a sound Oracle solution for HistSFC to reduce communication, and realize HistTree in the queryoptimizer.
- Adapt HistSFC to distributed computing platforms to resolve real big dataissue.
- Improve the method of cLoD computation by considering specific needs, e.g. incorporate semantic information and velocity for visualization inVR.
- Upgrade the data organization of HistSFC to facilitate data streaming in a server-client architecture, e.g. state-of-the-art blockingstrategy.
- In terms of storage, investigate compression techniques that does not cause significant deceleration of querying, e.g. LAZ format and Oracle secure filecompression.

ACKNOWLEDGEMENTS

The funding of this research comes from the Chinese Scholarship Council (CSC) and Fugro. They are greatly acknowledged. Besides, sincere gratitude also goes to Marianne de Vries for the help with Oracle implementation, and Dongliang Peng for his input to the development of novelalgorithms.

REFERENCES

[1] P. van Oosterom, O. Martinez-Rubi, M. Ivanova, M. Horhammer, D. Geringer, S. Ravada, T. Tijssen, M. Kodde, and R. Goncalves, "Massive point cloud data management: Design, implementation andexecution of a point cloud benchmark," Computers & Graphics, vol.49, pp. 92–125, 2015. [Online].Available: https://doi.org/10.1016/j.cag.2015.01.007.

[2] R. Zhang, J. Qi, M. Stradling, and J. Huang, "Towards a painless index for spatial objects," ACM Transactions on Database Systems (TODS), vol. 39,no. 3, 2014.

[3] X. Guan, P. van Oosterom, and B. Cheng, "A parallel n-dimensional space-filling curve library and its application in massive point cloud management," ISPRS International Journal of Geo-Information, vol. 7, no. 8, Aug. 2018. [Online]. Available: https://doi.org/10.3390/ijgi7080327

[4] M. Meijers and P. van Oosterom, "Clustering and indexing historic vessel movement data with space filling curves,"International Archivesof the Photogrammetry, Remote Sensing & Spatial InformationSciences,vol. 42, no.4, pp. 417–424, 2018. [Online]. Available: https://doi.org/10.5194/isprs-archives-XLII-4-417-2018

[5] Oracle,SDO_PC_PKGPackage(Point Clouds), 2013. [Online]. Available:https://docs.oracle.com/database/121/SPATL/sdopcpkg-package-point-clouds.htm#SPATL172.

[6] J. K. Lawder, "The application of space-filling curves to the storage and retrieval of multi-dimensional data," Ph.D. dissertation, University of London,2000.

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

[8] J. Wang and J. Shan, "Space filling curve based point clouds index," in Proceedings of the 8th International Conference on GeoComputation, 2005, pp.551–562. [Online]. Available: http://www.geocomputation.org/2005/WangJ.pdf

[9] P. van Oosterom, M. Meijers, E. Verbree, H. Liu, and T. Tijssen, "Towards a relational database spacefilling curve (sfc) interface specification formanaging nd-pointclouds," in Geoinformationssysteme 2019, Beiträgezur 6. MünchnerGI-Runde, (T. Kolbe, R. Bill, and A. Donaubauer, Eds.), Munich. Mar 2019, pp. 61–71.

[10] Oracle,IndexesandIndex-OrganizedTables,2013. [Online]. Available: https://docs.oracle.com/database/121/CNCPT/indexiot.htm#CNCPT721.

[11] S. Psomadaki, "Using a space filling curve for the management of dynamic point cloud data in a relational DBMS," Master's thesis, Delft University of Technology, 2016. [Online]. Available:http://resolver.tudelft.nl/uuid:c1e625b0-0a74-48b5-b748-6968e7f83e2b

[12] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi, "MD-hbase: design and implementation of an elastic data infrastructure for cloud-scale location services," Distributed and Parallel Databases, vol. 31, no. 2, pp. 289–319, 2013. [Online]. Available: https://doi.org/10.1007/s10619-012-7109-z

[13] R. Cura, "Inverse procedural street modelling: from interactive to automatic reconstruction," Ph.D. dissertation, Université Paris-Est Marne-la-Vallée,France, Mar. 2016.

[14] P. van Oosterom, "From discrete to continuous levels of detail for managing nd-pointclouds," Jun. 2019. [Online]. Available: http://nd-pc.org/documents/vario-nD-PC-v7.pdf

[15] T. Ohshima, H. Yamamoto, and H. Tamura, "Gaze-directed adaptive rendering for interacting with virtual space," in Proceedings of the IEEE 1996Virtual Reality Annual International Symposium. IEEE,1996, pp. 103–110.

[16] M. Reddy, "Perceptually modulated level of detail for virtual environments," Ph.D. dissertation, University of Edinburgh. College of Science and Engineering., Edinburgh, UK, Jul. 1997. [Online]. Available: http://hdl.handle.net/1842/505

[17] J. Sankaranarayanan, H. Samet, and A. Varshney, "A fast k-neighborhood algorithm for large point-clouds," in Eurographics Symposium on Point-Based Graphics, M. Botsch, B. Chen, M. Pauly, and M. Zwicker, Eds. The Eurographics Association, 2006, pp. 75–84.

[18] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter, "Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration,"Journal of Software Engineering for Robotics, vol. 3, no. 1, pp. 2–12, 2012.

[19] D. Girardeau-Montaut, M. Roux, R. Marc, and G. Thibault, "Change detection on points cloud data acquired with a ground laser scanner,"International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences, vol. 36, no. 3, pp. 30–35, 2005.

[20] Oracle,SDO_PC_PKGPackage(PointClouds):SDO_PC_NN,2018. [Online].

Available:https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/SDOPCPKG-reference.html#GUID-4C3EB18C-A53A-4671-A4A4-D1224E4A6F63.

[21] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "idistance: An adaptive B+-tree based indexing method for nearest neighbor search,"ACM Transactions on Database Systems (TODS), vol. 30, no. 2, pp. 364–397, 2005.

[22] M. Connor and P. Kumar, "Fast construction of k-nearest neighbor graphs for point clouds,"IEEE transactions on visualization and computer graphics, vol. 16, no. 4, pp. 599–608, 2010.

[23] Q. Liu, Approximate Query Processing. Boston, MA: Springer US, 2009, pp. 113–119. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9534

[24] R. Bayer, "The universal b-tree for multidimensional indexing: General concepts," inInternational Conference on Worldwide Computing and Its Applications. Springer, Berlin, Heidelberg, Mar 1997, pp. 198–209. [Online]. Available: https://doi.org/10.1007/3-540-63343- X48

[25] S. Berchtold, C. B hm, and H.-P. Kriegal, "The pyramid-technique: towards breaking the curse of dimensionality," in ACM SIGMOD Record, vol. 27,no. 2.ACM, 1998, pp. 142–153.

[26] R. Zhang, B. C. Ooi, and K.-L. Tan, "Making the pyramid technique robust to query types and workloads," in Proceedings. 20th International Conference on Data Engineering. IEEE, 2004, pp. 313–324.

[27] B. C. Ooi, K.-L. Tan, C. Yu, and S. Bressan, "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, 2000, pp. 166–174.

[28] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The x-tree: An index structure for high- dimensional data," in Very Large Data-Bases, 1996, pp. 28–39.

[29] N. Katayama and S. Satoh, "The sr-tree: An index structure for high-dimensional nearest neighbor queries," ACM SIGMOD Record, vol. 26, no. 2, pp.369–380, 1997.

[30] D. Hilbert, "Ueber die reellenZügealgebraischer Curven," Mathematische Annalen, vol. 38, no. 1,pp.115-138, 1981.

[31] A. R. Butz, "Alternative algorithm for hilbert's space-filling curve," IEEE Transactions on Computers, vol. C-20, no. 4, Apr 1971, pp. 424–426.

[32] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in Proceedings of the 2018 International Conference on Management of Data. ACM, 2018, pp. 489–504.

[33] B. Bamba, S. Ravada, Y. Hu, and R. Anderson, "Statistics collection in oracle spatial and graph: Fast histogram construction for complex geometry objects," Proceedings of the VLDB Endowment, vol. 6, no. 11, 2013, pp. 1021–1032.

[34] H. Liu, P. van Oosterom, M. Meijers, and E.Verbree, "Management of large indoor point clouds: An initial exploration," in ISPRS – International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Copernicus GmbH, Sep 2018, vol. XLII-4, pp. 365–372.

[35] Y. Zheng,Y. Ou, A. Lex, and J. M. Phillips, "Visualization of big spatial data using coresets for kernel density estimates," in2017 IEEE Visualization in Data Science (VDS), Oct 2017, pp. 23– 30.

[36] M. Schütz, K. Krösl, and M. Wimmer, "Real-time continuous level of detail rendering of point clouds," in2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR). IEEE, 2019, pp. 103–110.