International Journal of Applied Earth Observations and Geoinformation 105 (2021) 102625

Contents lists available at ScienceDirect

International Journal of Applied Earth Observations and Geoinformation

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

Executing convex polytope queries on nD point clouds

Haicheng Liu [*], Rodney Thompson, Peter van Oosterom, Martijn Meijers

Faculty of Architecture and the Built Environment, Delft University of Technology, Delft, the Netherlands

A R T I C L E I N F O

Keywords:
nD point clouds
Polytope query
Spatial data structures
CPLEX
Perspective view selection

A B S T R A C T

Efficient spatial queries are frequently needed to extract useful information from massive nD point clouds. Most previous studies focus on developing solutions for orthogonal window queries, while rarely considering the polytope query. The latter query, which includes the widely adopted polygonal query in 2D, also plays a critical role in many nD spatial applications such as the perspective view selection. Aiming for an nD solution, this paper first formulates a convex nD-polytope for querying. Then, the paper integrates three approximate geometric algorithms – SWEEP, SPHERE, VERTEX, and a linear programming method CPLEX, developing a solution based on an Index-Organized Table (IOT) approach. IOT is applied with space filling curve based clustering and advanced querying mechanism which recursively refines hypercubic nD spaces to approach the query geometry for primary filtering. Results from experiments based on both synthetic and real data have confirmed the superior performance of SWEEP. However, the algorithm may lag behind CPLEX due to pessimistic intersection computation in high dimensional spaces. In a real application, by properly transforming a perspective view selection into a polytope query, the solution achieves a sub-second querying performance using SWEEP. In another flood risk query, SWEEP also leads the others. In general, the robust and efficient solution can be immediately used to address different polytope queries, including those abstract ones whose constraints on combinations of different dimensions are formed into a polytope model. Besides, the knowledge of high-dimensional computations acquired also provides significant guidance for handling more nD GIS issues.

1. Introduction

nD point clouds and nD queries become increasingly used nowadays. To smoothly and efficiently visualize large volumes of LiDAR point data, a continuous Level of Importance (cLoI) dimension is suggested to be added for points clustering and indexing (van Oosterom, 2019; Schütz et al., 2019). Thus, when doing an nD query such as the perspective view selection, nearby points will all be selected, while fewer faraway points with restricted cLoI values are selected. Point data and queries can be more generic. For example, in flood modelling, results are normally computed and stored in a 2D computational grid. When the grid cell is not a square, such as a triangle (Fig. 1), data storage and querying in the form of rasters would be cumbersome and inefficient. A possible solution is to extract the centriods of all cells and store the information including flow velocity, direction and inundation depth in these centriods. Flood risk analysis can then be performed by querying this nD point cloud using all relevant dimensions besides XYZ (Liu et al., 2021a).

Prevalent software for point processing such as Oracle spatial, PostGIS and PDAL (PDAL-Contributors, 2018) are initially developed to resolve 2D or 3D issues, they lack nD indexing support and do not provide nD operators. To provide an efficient nD solution, van Oosterom et al. (2015) developed an Index-Organized Table (IOT) approach to address nD window queries on massive point clouds (Section 3). The IOT approach on the one hand achieves high efficiency to cluster and index all related dimensions in the query, while on the other hand avoids the time-consuming block unpacking and filtering process of block-based approaches. However, the IOT approach does not address irregular query geometries which are also commonly used. The aforementioned perspective view selection is a typical example: instead of a constant range, the cLoI constraint changes according to the distance to the view point. Existing solutions encounter significant bottlenecks solving such queries, including poor practical performance, inaccurate results and insufficient verification with nD data (Section 2).

In this paper, we extend the IOT approach for more query geometries beyond orthogonal windows. The paper describes a search strategy which has been shown to provide good response for a particular class of search regions – convex polytopes – which while not universal, is common enough to be useful. The polytope studied here is a convex

* Corresponding author.
E-mail address: h.liu-6@tudelft.nl (H. Liu).

https://doi.org/10.1016/j.jag.2021.102625
Received 1 August 2021; Received in revised form 20 October 2021; Accepted 11 November 2021
Available online 8 December 2021

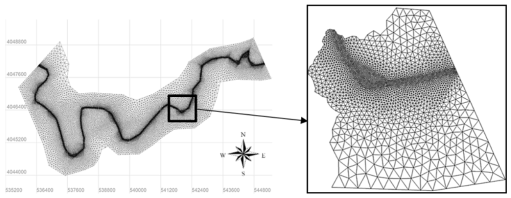


Fig. 1. A typical flood modelling grid. Image source: Gharbi et al. (2016).

geometry delimited by a set of half-spaces (Section 4.1). In summary, we made the following contributions:

1. Based on the IOT approach, we developed 3 geometric algorithms including SWEEP, SPHERE and VERTEX to detect intersection between a convex polytope and a hypercubic nD space. This forms the core to execute convex polytope queries on nD points.
2. We built two generic convex polytope geometries – the nD-simplex and nD-prism – to investigate and compare the performance of the algorithms and another linear programming method CPLEX. The querying tests from 2D to 10D have verified the theoretical complexity of these algorithms evidently.
3. We solved real world problems including perspective view selection and flood risk analysis by transforming the queries into nD polytope queries. They can then be accomplished efficiently using our solution.

The rest of the paper is organized as follows: Section 2 reviews and analyzes previous studies on polytope querying algorithms. Section 3 revisits the IOT approach briefly. Then, by first formulating an easy-to-use nD polytope for querying, Section 4 develops three generic algorithms to efficiently search point clouds with a polytope. Based on this, Section 5 evaluates all the algorithms by ideal experiments and real use cases. Section 6 concludes the paper.

2. Related work

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

The development of spatial indexing has facilitated the design and implementation of polytope querying. Based on the R-tree, Goldstein et al. (1997) developed two algorithms: one is the "simple" method which computes a scalar product indicating the minimum distance between a block and a half-space to examine whether an intersection happens; while the other method iteratively uses half-spaces to clip the R-tree block to detect intersection (Fig. 2). However, they discussed little about the performance in nD space, beyond testing a uniformly distributed 5D data set from the business domain. None of the

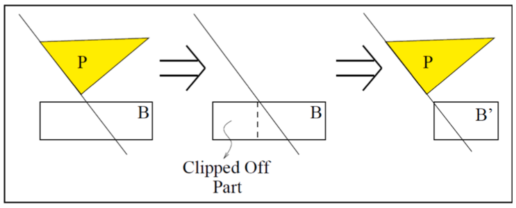


Fig. 2. One clipping operation where P refers to the polytope and B is an R-tree block, from Goldstein et al. (1997).

algorithms proposed distinguishes the type of intersection (inside or touching), leading to redundant intersection computation for a branch block totally inside the polytope. Kollios et al. (1999) developed an approximation algorithm to resolve trajectory searching problems. They also adapted and implemented algorithms of Goldstein et al. (1997) using the hB$^{\Pi}$-tree (Evangelidis et al., 1997) as a comparison. The result is that for 2D and 3D issues, their approximation algorithm works more efficiently.

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

Compared with those approaches, our IOT approach provides a true nD operator for executing convex polytope queries on nD points. It can be directly implemented and used in any database management systems that support the B+-tree structure. Besides, the solution always returns the correct result. It is also very efficient thanks to the Space Filling Curve (SFC) clustered data organization and the B+-tree indexing. Moreover, with optimizations such as the nD-histogram (Liu et al., 2020), the performance of querying on inhomogeneously distributed point data can also be improved significantly.

3. IOT approach

This section presents the overall architecture of the IOT approach. A detailed description can be found in (Liu et al., 2020). This forms the basis to realize the polytope querying algorithms described in Section 4.

3.1. Nomenclature

The key terminology used to illustrate the approach is introduced in the following sub-sections. Besides, the notations used in the paper is listed in Table 1.

3.1.1. Dimension

The dimensions discussed here possess either physical or semantic meanings that humans can perceive and interact with. In terms of data management, we identify two types of dimensions: *organizing dimensions* are used to cluster and index the data such as spatio-temporal dimensions; the other *property dimensions* that are not frequently used in the SQL WHERE clause are affiliated, such as color and intensity. These two types of dimensions are not fixed, and may be varied depending on applications.

3.1.2. Hypercube, node and range

In general, a cube refers to a 3D box with equal edge length. This

Table 1
Notations.

Notation	Description
n	Number of dimensions
m	Number of half-spaces in the polytope
N	Input size in the number of points
$k^{'}$	Output size from the first filter
k	Size of the accurate answer
r	Number of ranges generated
r_{max}	Maximum number of ranges (threshold)
B	Page capacity of storage

geometric concept extended into nD space becomes the hypercube.

Fig. 3 illustrates the node and the range in 2D. All points have integer coordinates. By truncating the last n bits ($n = 2$) of the points' Morton codes recursively, we derive Morton codes at upper levels. That is to say, the Morton codes of points implicitly contain a hierarchy which is equivalently a Quadtree structure. We can easily extend this scheme to higher dimensional spaces so that a Morton node refers to the corresponding node of a 2^n-tree. A branch node covers the nodes on the level below, and represents the extent of a hypercubic region (e.g., a block in the Quadtree). Thus, the branch node also indicates a range of Morton codes starting from the lower-left corner to the upper-right. A leaf node is not further subdivided.

3.2. Overview

Fig. 4 presents the workflow of our IOT approach. It applies two filters for querying, where the first filter computes the ranges for selecting keys while the second filter decodes the keys and conducts point-wise filtering to derive the final answer. Fig. 5 illustrates a 2D example of range computation in the first filter. In Section 4.2, we extended the range computing module to allow the transformation from an nD polytope query to 1D ranges.

4. Polytope querying

A convex polytope is defined as an nD geometry for which, given any 2 points within the region, every point along a straight line joining the points is also within the region. To use the polytope practically, this section first provides the mathematical formulation in Section 4.1. Then, novel intersection algorithms for polytope querying are developed in Section 4.2.

4.1. Mathematical formulation

A half-space is a division of space along a hyperplane (Fig. 6). Here it is defined as the set of points $\mathbf{x}$ such that $\boldsymbol{\omega}\cdot\mathbf{x} + \beta \leqslant 0$, where $\boldsymbol{\omega}$ is a unit vector ($\boldsymbol{\omega}\cdot\boldsymbol{\omega} = 1$) and β is a scalar. Note that the inequality is used here for compatibility with the conventions of computer representation software (3D) that the normal vector $\boldsymbol{\omega}$ is oriented so that it points to the outside of the solid object. A half-space can be denoted by the tuple ($\boldsymbol{\omega}$, β).

In 2D the term half-plane is sometimes used, defined by an infinite straight line – its only boundary. In 3D space, the half-space is defined and bounded by an infinite plane, while in higher dimensions the half-space represents all points on a particular side of an $(n-1)$ D hyperplane. In all cases, the dividing $(n-1)$D hyperplane has the definition $\boldsymbol{\omega}\cdot\mathbf{x} + \beta = 0$. A convex polytope is defined as the intersection of a finite set of half-spaces, where the boundaries may not be complete (Fig. 7):

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

where H_i is a set of m half-spaces.

Based on this formulation, we can test whether a point is within a half-space by evaluating $\boldsymbol{\omega}\cdot\mathbf{p} + \beta$: if the value is non-positive, the point $\mathbf{p}$ is within the half space. Since $\boldsymbol{\omega}$ and $\mathbf{p}$ are vectors of length n, the operation per point costs $\mathcal{O}(n)$ time. Then, computing the relationship between the point and the convex polytope, can cost $\mathcal{O}(mn)$ time per point. However, globally traversing all the points for selection is too costly and scales badly with the size of input. Consequently, we adopt the IOT approach to speed up the search.

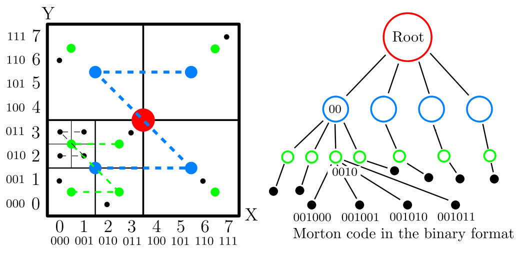


Fig. 3. Implicit Morton hierarchy: black dots are real points to be managed, while colored dots are Morton branch nodes at different levels.

4.2. Intersection algorithms

Given a set of $\boldsymbol{\omega}$ and β, we can then use the IOT approach to retrieve the result. The core of querying lies in the intersection computation between nodes and the convex polytope to generate ranges in the first filter. This section develops 3 geometric algorithms, and an additional linear programming solution provided by CPLEX. These algorithms return ranges which are then joined with the IOT, with a final filter performing the aforementioned point-in-polytope test.

4.2.1. SWEEP

SWEEP first identifies the “entry” and “exit” of a node with respect to a half-space (Fig. 8): imagine if the half-space were to be moved from a great distance away, towards and across the node so that ultimately the node is within the half-space; the entry and exit are the first and last vertices to cross the boundary. The categorization as entry/ exit is only true in relation to a single half-space, and must be re-appraised for others. Then, based on the distance between the half-space’s boundary and the entry or the exit, SWEEP determines if an intersection happens. Fig. 9 presents the whole workflow of SWEEP for one half-space.

Fig. 10 shows how SWEEP works with different nodes after several iterations of decomposition. Node N_1 is not within the half-space H_2, therefore it is external to polytope C, and can be dropped. N_2 is within all of H_1 to H_4, and its range can be exported. In the case of N_3, since it fulfils neither of these cases, it must be placed in a refinement pool before being accepted or rejected. The case of N_4 is significant, because it partially overlaps or falls within each half-space, but in fact it does not intersect C. We refer to this case as a False Positive Node (FPN) to be discussed later. In the process of searching the nodes from the refinement pool, sub-nodes at the next lower levels are processed (2^n of them). These are then applied to the same tests against C. Some sub-nodes are found to be internal, some to be on the boundary, and the rest external.

FPNs exist at crossings of half-spaces (e.g., N_4 in Fig. 10a and N_{42} in Fig. 10b). It is a practical proposition to ignore the problem, and allow FPNs to be processed as if they are true positive nodes. In Fig. 10, N_4, after first refinement, has three of its sub-nodes eliminated, leaving only N_{42} whose sub-nodes are eliminated at the next level. This appears to be a common event – that as a FPN is decomposed into sub-nodes at one level below, those sub-nodes are largely eliminated. With the second filter, points in any remaining FPNs will all be eliminated.

4.2.2. SPHERE and VERTEX

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

The SPHERE algorithm first computes the centre of a node. If the centre is in the half-space, or the Euclidean distance between the centre and the half-space is within half of the diagonal length of the node, the node will be selected. “inside” or “partial overlap” can be decided depending on the distance. SPHERE only needs a central point for intersection detection, which is favorable. However, as the distance computed is an upper bound, FPNs will be selected (N_4 in Fig. 11).

The VERTEX algorithm is more straightforward, as it examines every vertex of a node to determine whether the node intersects a half-space. If all vertices are outside, then no intersection happens. If all the vertices are in the half-space, the node is inside. For all other cases, a partial overlap is returned. The implementation is simple, but the algorithm degrades in high dimensional spaces because the number of vertices of a node grows exponentially with dimensionality. False detection will also arise at crossings of half-spaces (e.g., Fig. 10a), as with SWEEP.

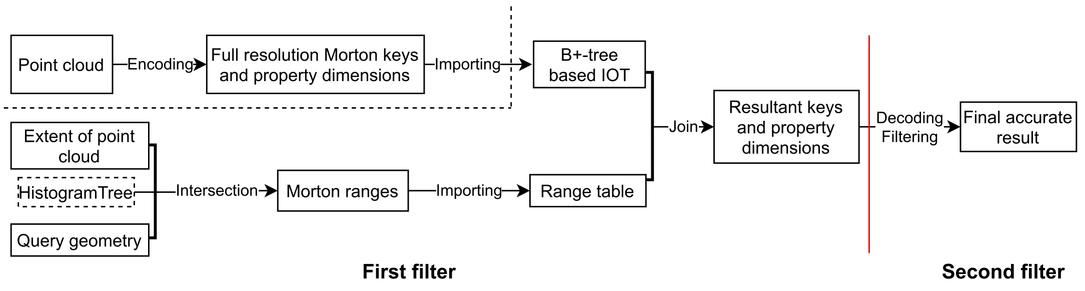


Fig. 4. The loading and querying procedure of the IOT approach.

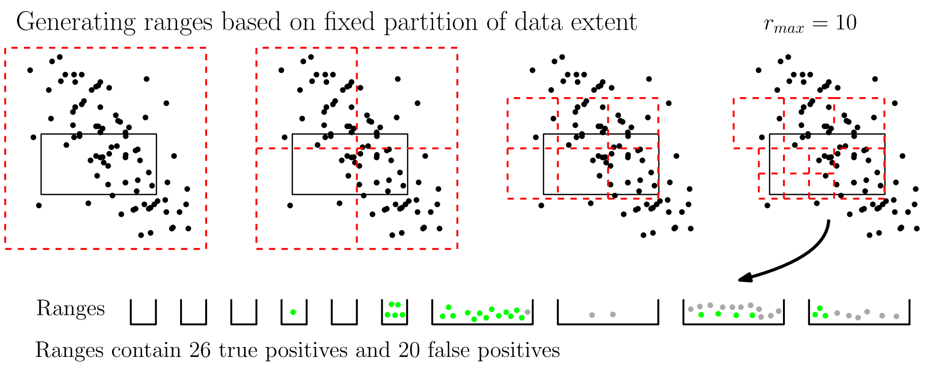

Generating ranges based on fixed partition of data extent
$r_{max} = 10$
Ranges
Ranges contain 26 true positives and 20 false positives

Fig. 5. Range generation of a window query using the principle of the Morton hierarchy. The data extent derived from Morton nodes is recursively decomposed to match the query window. This process continues until the number of ranges generated reaches the threshold r_{max}.

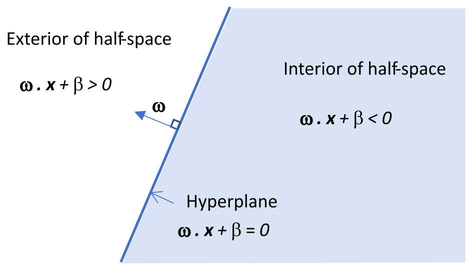
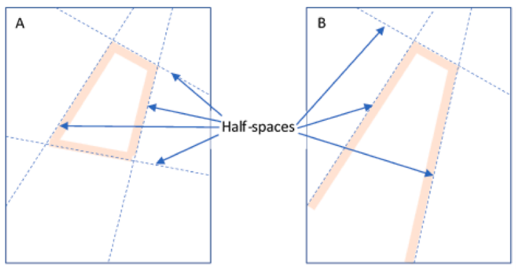


Fig. 6. The definition of a 2D half-space which can be generalised to nD.

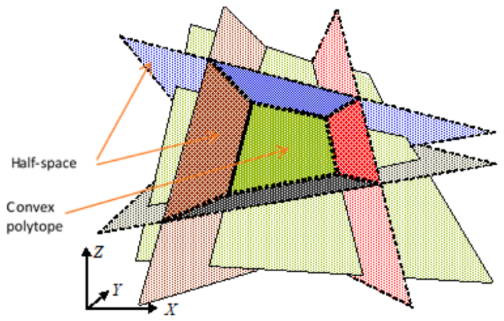
4.2.3. CPLEX

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

Fig. 12 presents the results of a 2D triangle query on a uniformly distributed point set, with a proper IOT setting. SPHERE contains all points selected by SWEEP which again contains points selected by CPLEX. VERTEX returns the same result as SWEEP. The result indicates a general pattern of k', which is SPHERE $\geqslant$ SWEEP (VERTEX) $\geqslant$ CPLEX. The false positive points are distributed along the boundaries and around the acute corners. Several factors influence the occurrence of these points, including the dimensionality, relative positions of the half-spaces to nodes, and r_{max} for querying. The actual performance of these algorithms depends on the specific settings and implementation, which is evaluated by experimenting in Section 5.



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



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

Fig. 7. Convex polytopes in 2D and 3D spaces. (See above-mentioned references for further information.)

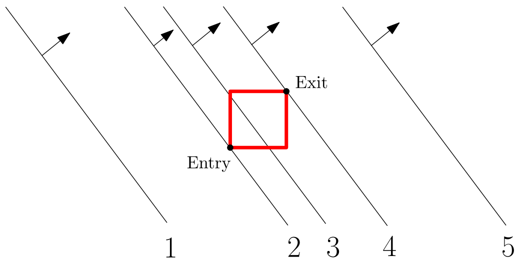


Fig. 8. The "sweeping" process: in (1), a half-space starts sweeping with the node outside; (2), (3), (4), the half-space sweeps over the node, where intersection happens; (5), the sweeping ends with the node inside the half-space.

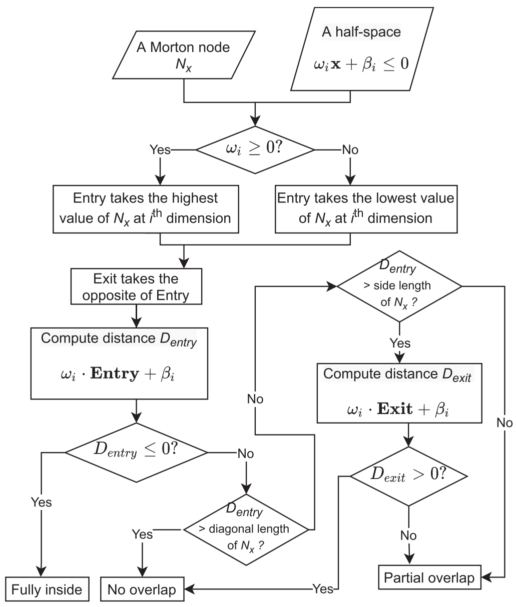

A Morton node N_x
A half-space $\omega_i \mathbf{x} + \beta_i \leq 0$
$\omega_i \geq 0$?
Yes
No
Entry takes the highest value of N_x at i^{th} dimension
Entry takes the lowest value of N_x at i^{th} dimension
Exit takes the opposite of Entry
Compute distance D_{entry} $\omega_i \cdot \mathbf{Entry} + \beta_i$
D_{entry} > side length of N_x ?
Compute distance D_{exit} $\omega_i \cdot \mathbf{Exit} + \beta_i$
$D_{entry} \leq 0$?
D_{entry} > diagonal length of N_x ?
$D_{exit} > 0$?
Fully inside
No overlap
Partial overlap

Fig. 9. The workflow of SWEEP.

4.3. Theoretical complexity

The total querying time based on the IOT approach is as follows:

$$T = T_{pre} + T_{io} + T_{post} \tag{1}$$

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

In SWEEP and SPHERE, T_{pre} is bounded by $\mathcal{O}(mnr\log_B N)$. In VERTEX, every vertex of a node has to be examined. Thus, its T_{pre} is bounded by $\mathcal{O}(2^n mnr\log_B N)$. T_{io} maximally covers $\mathcal{O}(\frac{k'}{B}+r)$ I/Os, while T_{post} is bounded by $\mathcal{O}(mnk')$. Once parallelism is applied, T_{post} becomes $\mathcal{O}(\frac{mnk'}{p})$, given p processors. Besides, all intersection algorithms introduce FPNs except CPLEX. So, k' can be varied. An optimal solution should balance the three cost terms. An accurate first filter with large r may cost more time, but it returns a small k' which alleviates I/O and post-processing in the second filter. For this purpose, we introduce False Positive Rate (FPR) to indicate I/O and the performance of second filter (Eq. 2):

$$FPR = \left|\frac{k' - k}{k}\right| \tag{2}$$

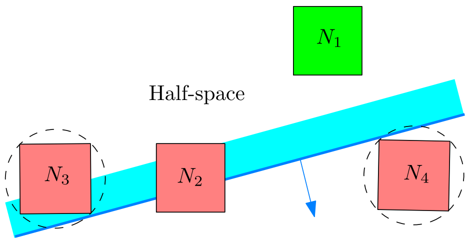


Fig. 11. Intersection detection using SPHERE: N_1 is inside; N_2 partially overlaps the half-space; N_3 and N_4 are falsely detected as partial overlap.

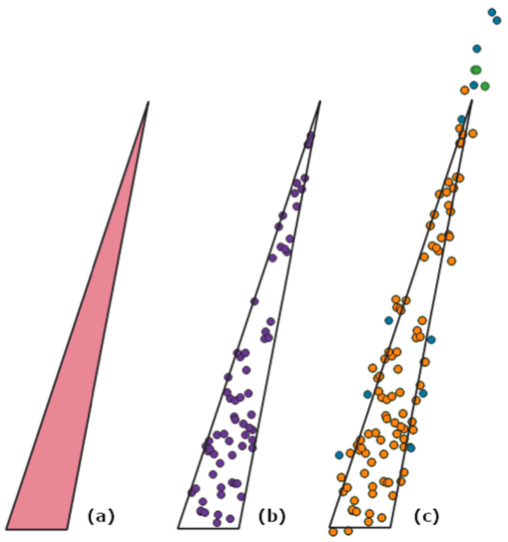


Fig. 12. Querying results from different algorithms: (a) querying geometry, (b) accurate result, (c) Overlapping results of CPLEX (orange), SWEEP (green) and SPHERE (blue), without a second filter.

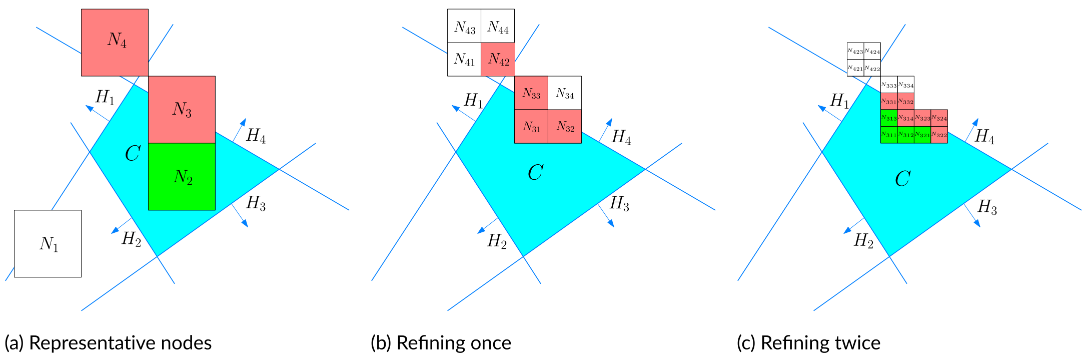


Fig. 10. SWEEP selection based on the IOT approach, with white nodes outside, green nodes inside and red nodes on the boundary.

Sometimes, FPR can be very large, e.g., in high dimensional spaces. Then, the portion of data selected by the first filter is also indicative (Eq. 3):

$$selectivity = \frac{k'}{N} \tag{3}$$

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

5. Experiments and analysis

This section describes experiments to evaluate the performance of different algorithms described above. Section 5.1 builds a regular nD-simplex model and an nD-prism model for testing. The nD-simplex is the simplest nD polytope, while the nD-prism is devised to investigate how the number of half-spaces influences the querying efficiency. Section 5.2 tests two real use cases and focuses on the time cost. One is the perspective view query, which is a basic operation to realize point clouds' visualization; the other is a flood risk query, based on modelling results.

The experimental platform is HP DL380p Gen8 server with 2 × 8-core Intel Xeon processors, E5-2690 at 2.9 GHz, 128 GB RAM, and 41 TB SATA 7200 rpm in RAID6 configuration. Solutions are implemented in Oracle 12.2. All tests are "cold", without caching.

5.1. nD-simplex and nD-prism tests

Both tests use a single thread, recording 3 indicators for evaluating the performance:

1. Selectivity of the first filter (Eq. 3).
2. Number of iterating cycles (i.e. for-loops) to generate ranges. A cycle of CPLEX means resolving the optimal problem once (Section 4.2.3). A cycle of SWEEP, SPHERE and VERTEX means computing the distance from a half-space to a point in the node.
3. Time cost of the first filter and the second filter. The first filter time corresponds to T_{pre} in Eq. 1, while the second filter takes $T_{io} + T_{post}$ to accomplish.

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

5.1.1. Synthetic data sets

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

5.1.2. nD-simplex query

As has been described in (Liu et al., 2021b), we mathematically formulated a regular nD-simplex model which has edges of equal length (Fig. 13) and comprises $n+1$ half-spaces. The model is totally inside the data domain, with each face oriented in a different direction. By modifying the parameters, we derive simplex models of the same volume at different dimensionality. In other words, a constant selectivity is achieved among all data sets, which facilitates analyzing the behaviour of querying algorithms. This experiment specifically builds simplexes for querying with a selectivity of 0.1%.

The experiment sets r_{max} to 10^6. This guarantees a low FPR in high dimensional spaces, without bloating the memory. CPLEX (two versions), SWEEP, SPHERE and VERTEX are tested. CPLEX#1 distinguishes

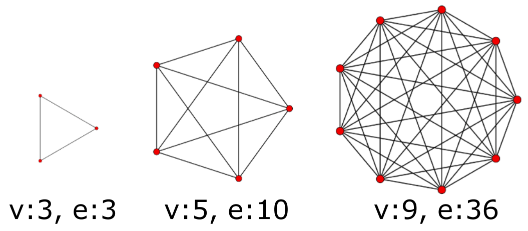


Fig. 13. Orthogonal projections of a 2D-simplex, 4D-simplex and 8D-simplex built, with the number of vertices (v) and edges (e). Image source: Wikipedia.

between two types of intersection: inside or partial overlap, while CPLEX#2 does not, meaning nodes inside the simplex are refined unnecessarily.

Tables 2–4 show the results. Over all, CPLEX#1 owns the lowest FPR, while SWEEP responses the fastest below 10D with CPLEX#2 fastest in 10D. The low FPR of CPLEX#1 leads to the smallest k' for post-processing. However, as CPLEX#1 spends significantly more time for each iteration for computing ranges, such advantage is insignificant until 10D (Table 4). CPLEX#2 presents analogous selectivity as CPLEX#1, but is faster. This is because CPLEX#1 decomposes the simplex to individual half-spaces to further compute inside or partial overlap, while ignoring this reduces about 50% computation (Table 3). Besides, it becomes less necessary to distinguish these two intersection types when n increases, as all nodes returned by the first filter tend to fall on the boundary (Table 5). This is because the number of nodes at each level of the Morton hierarchy increases exponentially with n, the final nodes selected mainly reside in higher levels with larger sizes. So, these nodes are more likely to partially intersect the simplex.

False positive points selected by CPLEX are caused by boundary nodes. On the other hand, in addition, SWEEP, SPHERE and VERTEX also select FPNs as they apply approximate intersection computation. So, larger k' are returned, which cause significant performance degradation in higher dimensions. SPHERE processes similar number of iterations as SWEEP, and is even faster in generating ranges. However, the ranges contain more false positive points, which undermines SPHERE's overall performance. Especially in 10D, SPHERE selects nearly the whole data set. VERTEX returns the same ranges as SWEEP, but takes much more iterations due to the multiple vertex-based distance computation. An odd pattern occurs that the iterations of SWEEP and SPHERE decline from 8D to 10D. This is because the simplex's boundary is very close to the boundaries of the data region in 10D (Liu et al., 2021b). So, the false detection related to this half-space is constrained by the data region.

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

5.1.3. nD-prism query

In theory, the number of half-spaces constituting the nD-polytope affects the time cost of range generation linearly (Section 4.3). This

Table 2
Selectivity of the first filters.

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

Table 3
Number of iterating cycles for computing ranges.

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

Table 4
Time cost (seconds) (first filter/second filter).

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

Table 5
Number of nodes selected by the first filter (inside/boundary).

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

The final number of ranges generated may be slightly more than 10^6, due to residual nodes exported after the decomposition stops.

section uses a simple nD-prism model to verify this. Inscribed regular polygons of a circle are used as the base (Fig. 14). To create the prism with $2f$ vertical faces, we apply a rotation formulation: suppose $\theta = \frac{\pi j}{f}$, where $j = -f+1, \ldots, f$. Then, we create each half-space with

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

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

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

The test uses the same uniform data sets from 2D to 10D. As the main patterns of performance are similar across different dimensionality, we only present 6D here. r_{max} is still 10^6. The approaches are the same as before. Figs. 15–17 show the results.

These figures indicate that for all solutions, the number of half-spaces influences the time cost of range computation linearly. CPLEX#2 holds the best scalability. It takes a constant number of iterations to compute ranges, and the number of half-spaces influences insignificantly on the time cost in each iteration. SWEEP and SPHERE hold the superiority over others in time cost of range computation. However, because of more FPNs, SPHERE takes more iterations than SWEEP. This gap becomes larger with larger dimensionality, which has been observed in 8D and 10D cases (Liu et al., 2021b). On the other hand, SWEEP returns no FPNs in this test, most likely due to the wide angle between two adjacent faces of the $8i$-gonal prisms.

5.2. Perspective view selection

After the ideal tests, we explored the applicability of the algorithms to real data, investigating perspective views on airborne laser scanning (ALS) points. This section begins by restating the query region as a convex polytope. Then, two kinds of perspective view are used to verify the performance of different algorithms.

5.2.1. Modelling 4D perspective view

The data used is AHN2, an ALS point cloud recording elevation of the whole Netherlands (AHN, 2014). The original data contains XYZ information only. We additionally add the cLoI dimension to achieve a smooth visualization, and avoid the "block" pattern with density shocks (Liu et al., 2020; Schütz et al., 2019). The computed cLoI values follow the exponential distribution (van Oosterom, 2019): the smaller the cLoI value, the more important the point is. In the 4D perspective view selection, all points should be within a 3D view frustum in XYZ. Also, to imitate a realistic scene, a hyperplane in the cLoI dimension ensures nearby points are all be rendered, while fewer faraway points are selected. This corresponds to our cognition. So, we build the 4D view model as follows:

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

$$\sqrt{(x-u)^2 + (y-v)^2 + (z-w)^2} \leqslant D - \frac{D \cdot cLoI}{cLoI_{max}} \tag{5}$$

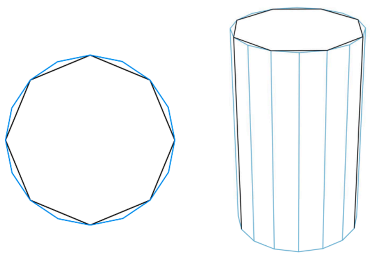


Fig. 14. The nD 8-gonal and 16-gonal prisms projected to 2D and 3D spaces.

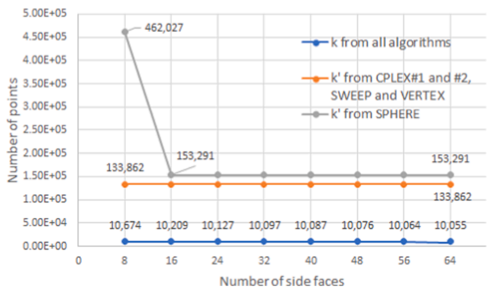


Fig. 15. Results from the first and second filters in the 6D-prism query.

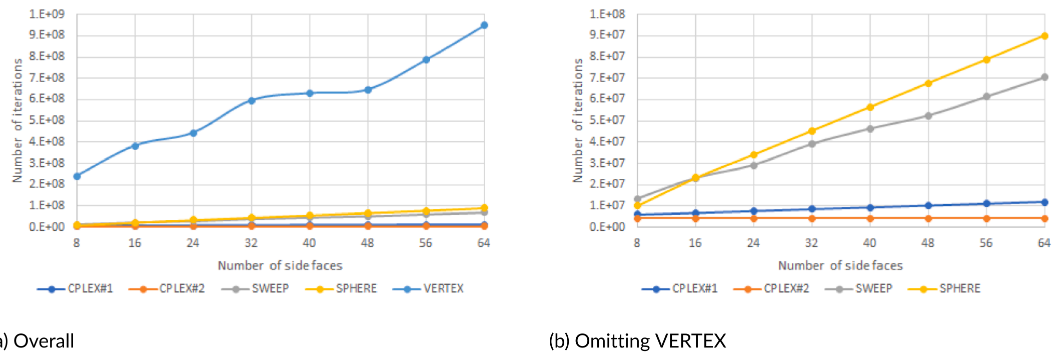


Fig. 16. Number of iterations of range computation in the 6D-prism query.

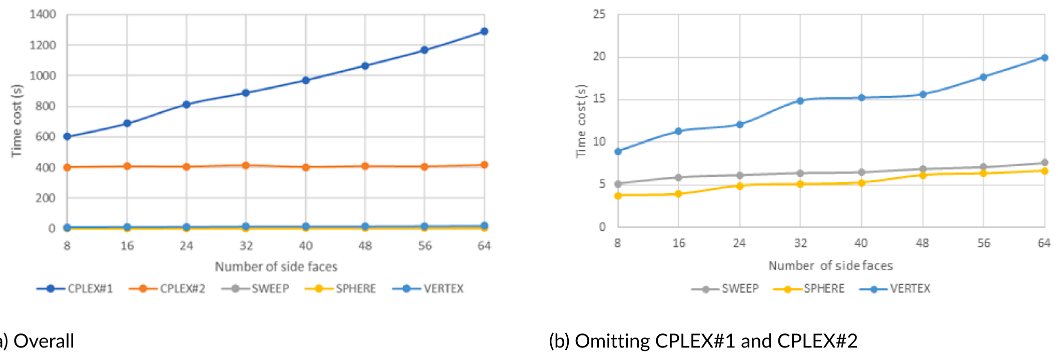


Fig. 17. Time cost of range computation in the 6D-prism query.

where Eq. 4 describes the 3D view frustum, while Eq. 5 defines the cLoI range. a_i, b_i, c_i and d_i are parameters based on the view point, direction and maximum view distance D. (u, v, w) represents the coordinates of the view point.

Based on this formulation, perspective views were selected using GEOM, SWEEP and CPLEX. GEOM detects intersection based on rigorous geometric computation. For example, Eq. 5 defines a 4D cone. To determine whether a 4D node intersects it, GEOM first computes the boundaries of the two geometries: the boundaries of the 4D cone are actually 2 3D balls and 6 3D cones, while that of the 4D node are 8 3D cubes. Then, GEOM detects intersection of these boundaries at the 3D space. Using GEOM, no FPNs will be detected, but the computation is non-trivial.

SWEEP uses a polytope for selection. So, Eq. 5 is approximated by linear equations. The left term in Eq. 5 represents a sphere, and can be approximated by 15 half-spaces surrounding the sphere (Fig. 18). Practically, the horizontal and vertical span of the vision could exceed 120° but cannot reach 180°. So, we rotate the central half-space by an interval of 30° both horizontally and vertically, to build the discrete approximation of the sphere.

CPLEX supports quadratic constraints and can be used without linearization. However, in practice, CPLEX collapses due to the high complexity of the quadratic problem (Eq. 5). Consequently, CPLEX uses the 4D polytope model established in the SWEEP approach. Note that the CPLEX approach implemented here distinguishes inside nodes as does CPLEX#1 in Section 5.1.

5.2.2. Benchmark tests

The test sample contains 2 billion points. GEOM, SWEEP and CPLEX all set r_{max} to 10^5 for querying. The tests include two view modes (Fig. 19): the close-up view simulates the situation where a user is standing looking around; the distant view imitates the bird-eye view located 800 m to 1 km high, looking at the ground. Each view test randomly generates 100 queries for benchmarking.

Fig. 20 shows the visuals of a typical close-up query. Fig. 21 presents the perspective view from another distant query covering the same region as Fig. 20. Fig. 22 presents the output size and FPR of both types of view selection based on all queries tested. Table 6 shows the average time cost of different approaches.

As indicated in Fig. 22b, SWEEP returns the most false positive points

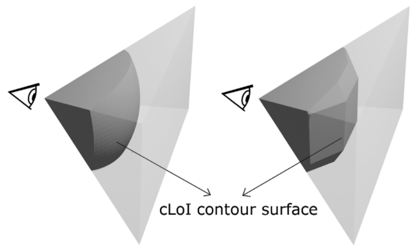


Fig. 18. Approximating cLoI range by half-spaces: all points on the contour surface have the same maximum cLoI value.

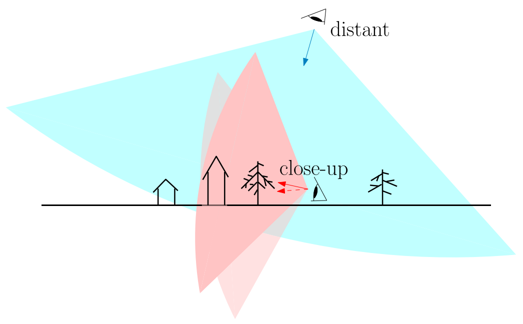
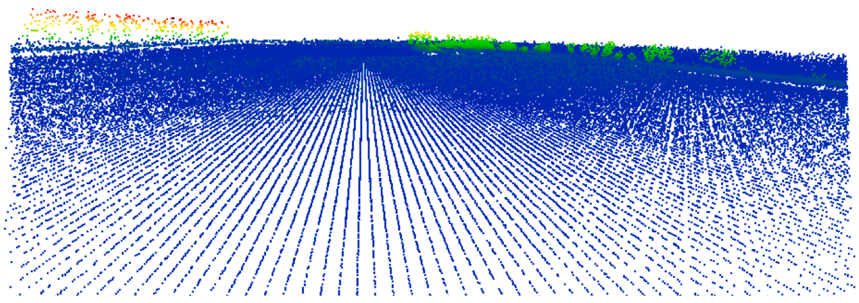
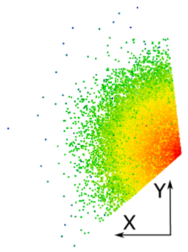


Fig. 19. Illustration of a distant view and two close-up views.



(a) Perspective view, colored by elevation



(b) Orthogonal projection, colored by maximum cLoI value

Fig. 20. Results from a close-up query.

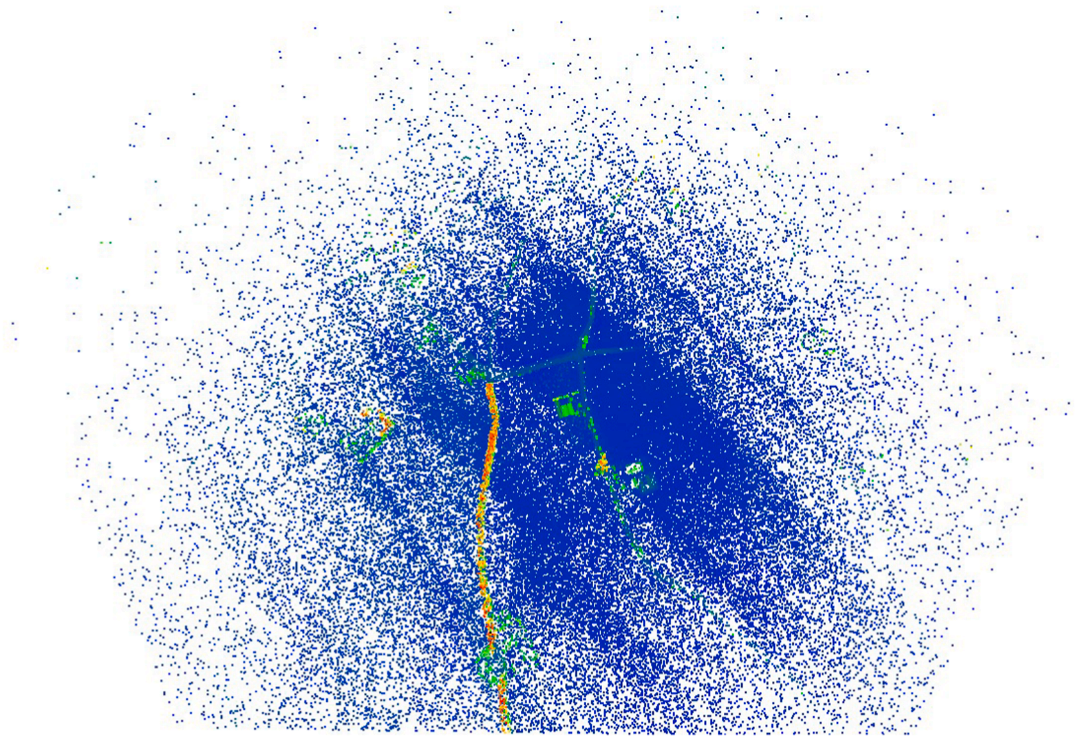
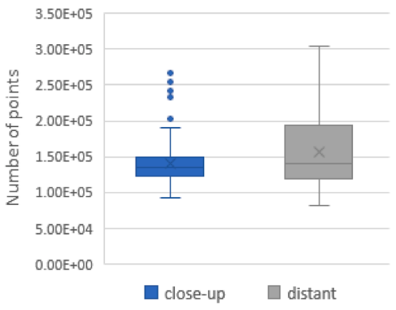
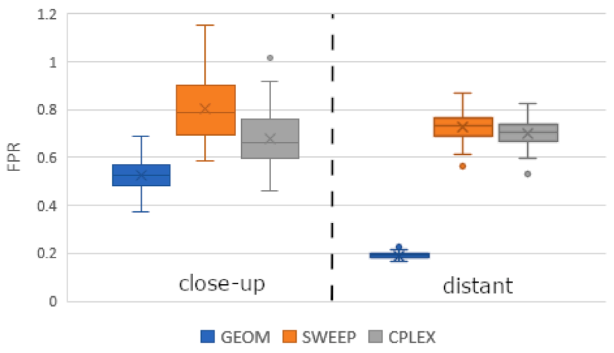


Fig. 21. Result from a distant query.



(a) Size of accurate answer



(b) False positive rate of different approaches

Fig. 22. Statistics of executing the 100 close-up and the other 100 distant queries.

Table 6
Average time cost of the perspective view queries (seconds).

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

from the first filter. Although SWEEP selects FPNs, the second filter times are not greatly affected - the FPNs' negative effect is limited for this application, which is consistent with the nD-simplex experiment (Table 2). GEOM holds the lowest FPR thanks to the accurate intersection between the original geometry (Eq. 5) and nodes. However, implementing is cumbersome, and can be impossible for other query geometries in high dimensional spaces.

Table 6 shows the superior performance of SWEEP, where the total time cost is below 1 s. In the first filter, range computation costs the most, while the others take constant time (< 0.2 s). Although CPLEX possesses a more accurate first filter than SWEEP, its range computation is 200$\times$ slower than SWEEP. This also agrees with previous results.

5.3. Flood risk query

This section presents another use case from the hydrology domain. The data set is generated by running flood models which is developed for the Niansi Levee in China (Liu et al., 2021a). The 2D fluid computational grid contains 59,680 triangular cells, which will be represented by their centroids. We modelled 8 cases including 4 locations of breach, combined with extreme rainfall with a return period of 20 and 50 years. Each case simulates 720 steps (corresponding to a 30-min resolution). So in total, we extract $59{,}680 \times 720 \times 8 = 343{,}756{,}800$ points, in an 8D space composed by case ID, X, Y, Z, time, depth, velocity and flow direction. The query is to select dangerous locations evaluated by human instability (, i.e., depth $\times$ velocity $\geqslant 2$) (Jonkman and Penning-Rowsell, 2008) in case 1.

Among the 8 dimensions, the flow direction is seldomly used for ad-hoc analysis. So, we set it as the property dimension when building the IOT, while used the other 7 dimensions for Morton key encoding. To employ SWEEP and CPLEX, we first converted the query into the polytope representation. It consists of a half-space indicating the case ID, and the other 7 half-spaces of which the points of tangency spread over the query boundary in the 2D projection (Fig. 23). As a comparison, we also built a customized GEOM approach. GEOM reports an intersection if the upper-right corner of a node is in the original geometry. An inside is returned if the lower-left corner is inside. r_{max} is set to 10^5, the same as the AHN2 test. We also changed the case ID to 5, which leads to a different result. The exact answer of case 1 contains 28,351 points, while that of case 5 contains 175,758 points. Table 7 presents the accuracy of different first filters, and Table 8 presents the time cost.

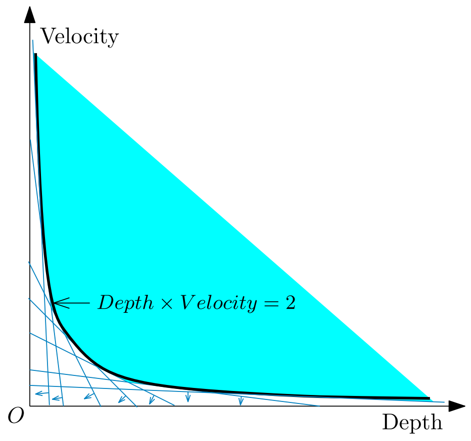


Fig. 23. Converting the constraint on human instability into a polytope model.

The large FPRs in both queries (Table 7) result from the data set's high dimensionality and the skewed distribution of the depth and velocity dimension. Both dimensions contain large amount of zero values which are selected by these algorithms. In case 5, SWEEP and CPLEX select less points than GEOM, applying a different path for node decomposition. So, using the polytope for approximation may not always lead to negative result. Thanks to similar k', SWEEP costs nearly the same time as GEOM (Table 8), but CPLEX still takes more time by an order of magnitude. Additionally, as GEOM needs hard-coded programming, it is still less applicable than the generic SWEEP.

5.4. Discussion

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

As shown in Tables 2 and 4, SWEEP becomes less competitive after 8D. FPNs occur at the acute corners where boundaries meet, and this occurs increasingly frequently in higher-dimensional simplexes. The nD-simplex is effectively a worst-case for the generation of FPNs, as SWEEP does not return any FPNs in the nD-prism test.

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

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

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

Table 7
Accuracy of the first filters in the flood query.

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

Table 8
Time cost of the flood risk query (seconds)

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

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

where p_i is one of the random points. These gradients serve as normal vectors of a set of hyperplanes which are what we need. In practice, for a specific query geometry, the user can pick a small set of points of tangency, P, which spread over the geometry to build the polytope. By iteratively performing benchmark tests and densify P, optimal polytope models can be acquired according to Eq. 1. Alternatively, users can apply a customized method if the query geometry can be modelled more efficiently.

6. Conclusions

Queries on nD point clouds include different types of query geometries. This paper concentrates on a specific type, the convex polytope. Based on an efficient IOT approach, the paper developed alternative polytope querying algorithms – SWEEP, SPHERE and VERTEX. They adopt different strategies for detecting intersection between a node and the polytope, where a node represents a hypercubic extent of a cluster of points. The linear programming method CPLEX is also implemented as a comparison. To learn their performance, the paper devised a representative nD-simplex and nD-prism query geometry to experiment. It turns out that SWEEP performs the best over all. The number of half-spaces influences the time cost of all approaches linearly. In a real application, the perspective view selection, SWEEP spends less than 1 second to accomplish the query. On the other hand, GEOM which computes intersection between a node and original curved geometries accurately is slower. Besides, SWEEP also performs competitively in a crucial flood risk query, which once again confirms the applicability of the polytope solution in practice. Moreover, based on this expressive, flexible and convenient framework, we are able to optimize and extend the algorithms further to address more comprehensive nD queries on point clouds. All the code concerned in this paper can be assessed at https://github.com/rencailhc/HistSFC.

CRediT authorship contribution statement

Haicheng Liu: Methodology, Software, Formal analysis, Writing – original draft. **Rodney Thompson:** Conceptualization, Methodology, Writing – original draft. **Peter van Oosterom:** Conceptualization, Writing – review & editing, Supervision. **Martijn Meijers:** Writing – review & editing, Validation.

Declaration of Competing Interest

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

Acknowledgements

The authors would like to thank the Chinese Scholarship Council (Grant No. 201708420146) and Fugro company for supporting this research. Besides, sincere gratitude also goes to Edward Verbree for his constructive suggestions on preparing the manuscript.

References

Agarwal, P.K., Arge, L., Erickson, J., Franciosa, P.G., Vitter, J.S., 2000. Efficient searching with linear constraints. J. Comput. Syst. Sci. 61, 194–216.
AHN, 2014. Actueel Hoogtebestand Nederland. https://www.ahn.nl/ (accessed: 2021-10-19).
Chazelle, B., 1989. Lower bounds on the complexity of polytope range searching. J. Am. Math. Soc. 2, 637–666.
Evangelidis, G., Lomet, D., Salzberg, B., 1997. The hB$^{\Pi}$-tree: a multi-attribute index supporting concurrency, recovery and node consolidation. VLDB J. 6, 1–25.
Gharbi, M., Soualmia, A., Dartus, D., Masbernat, L., 2016. Comparison of 1D and 2D hydraulic models for floods simulation on the Medjerda Riverin Tunisia. J. Mater. Environ. Sci. 7, 3017–3026.
Goldstein, J., Ramakrishnan, R., Shaft, U., Yu, J.-B., 1997. Processing queries by linear constraints. In: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 257–267.
Jonkman, S., Penning-Rowsell, E., 2008. Human instability in flood flows. J. Am. Water Resour. Assoc. 44, 1208–1218.
Khan, A., Yanki, P., Dimcheva, B., Kossmann, D., 2014. Towards indexing functions: Answering scalar product queries. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp. 241–252.
Kollios, G., Gunopulos, D., Tsotras, V.J., 1999. On indexing mobile objects. In: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 261–272.
Lima, R., 2010. IBM ILOG CPLEX - what is inside of the box? In: EWO Seminar. Carnegie Mellon University, Pittsburgh, PA, USA. http://egon.cheme.cmu.edu/ewo/docs/rlima_cplex_ewo_dec2010.pdf (accessed: 2021-10-19).
Liu, H., Oosterom, P.V., Mao, B., Meijers, M., Thompson, R., 2021a. An efficient nD-point data structure for querying flood risks. In: The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XLIII-B4-2021. Copernicus GmbH, pp. 367–374.
Liu, H., Thompson, R., van Oosterom, P., Meijers, M., 2021b. Fundamentals, implementations and experimental benchmarks of nD-polytype queries on point cloud data sets. Tech. rep. Delft University of Technology. http://www.gdmc.nl/publications/reports/GISt78.pdf (accessed: 2021-10-19).
Liu, H., van Oosterom, P., Meijers, M., Guan, X., Verbree, E., Horhammer, M., 2020. HistSFC: Optimization for nD massive spatial points querying. Int. J. Database Manage. Syst. (IJDMS) 12, 7–28.
Matoušek, J., 1992. Reporting points in halfspaces. Comput. Geom. 2, 169–186.
Matoušek, J., 1994. Geometric range searching. ACM Comput. Surv. (CSUR) 26, 422–461.
van Oosterom, P., 2019. From discrete to continuous levels of detail for managing nD-PointClouds. In: ISPRS Geospatial Week 2019. Enschede, the Netherlands. http://nd-pc.org/documents/vario-nD-PC-v7.pdf (accessed: 2021-10-19).
van Oosterom, P., Martinez-Rubi, O., Ivanova, M., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T., Kodde, M., Gonçalves, R., 2015. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. Comput. Graph. 49, 92–125.
PDAL-Contributors, 2018. PDAL Point Data Abstraction Library. https://doi.org/10.5281/zenodo.2556738 (accessed: 2021-10-19).
Schütz, M., Krösl, K., Wimmer, M., 2019. Real-time continuous level of detail rendering of point clouds. In: 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR). IEEE, pp. 103–110.
Thompson, R., 2007. Towards a Rigorous Logic for Spatial Data Representation. Ph.D. thesis, Delft University of Technology. Published as NCG Publications on Geodesy no. 65.
Wang, P., Ravishankar, C.V., 2013. Secure and efficient range queries on outsourced databases using Rp-trees. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE). IEEE, pp. 314–325.