Towards a relational database Space Filling Curve (SFC) interface specification for managing nD-PointClouds

Peter VAN OOSTEROM, Martijn MEIJERS, Edward VERBREE, Haicheng LIU and Theo TIJSSEN

Section GIS technology, Faculty of Architecture and the Built Environment, TU Delft
E-Mails: P.J.M.vanOosterom@tudelft.nl, B.M.Meijers@tudelft.nl, E.Verbree@tudelft.nl, and H.Liu-6@tudelft.nl

Abstract
In this paper we propose to treat point clouds as a first-class representation (similar to vector or raster representations), with the nD-PointCloud as the solution for this, offering deep integration of space, time and scale. For efficiency reasons spatial indexing and clustering of these large point clouds is extremely important and this is obtained based on a Space Filling Curved (SFC). In order to get beyond the current state of the art of storing/ managing point clouds in files, a DBMS solution is presented (with all benefits: integration with other data types, scalability, multi-user, transaction support, etc.). Finally, a DBMS SFC interface specification for point clouds is proposed.

1 Introduction

Gridded (raster) or object (vector) models are state of the art to represent spatio-temporal features as for instance in urban planning, geo-information or human and social geography. Point clouds are emerging as a data acquisition technique, but not used throughout the whole processing chain (model storage, spatial analysis and visualization). Typically, current models are organized in a fixed number of levels of importance (detail/scale), introducing limitations as fixed level choices and data density jumps between levels. We propose an added dimension to space and time that will facilitate continuous levels of importance (scale). The goal is to construct a novel nD-PointCloud model for handling massive multi-dimensional (nD) point cloud data sets, representing space, time, scale and added information (colour, material properties, velocity etc.). The nD-PointCloud as serious alternative to raster of vector models offers unique new possibilities: more natural representations (e.g. vegetation), more detail and no conversion loss enabling better spatial analysis (e.g. flow computation or routing). Based on a novel use of high-resolution nD space filling curves deep integration of space, time and scale as basis for data organization is realized and applied for big data (towards 10^{15} points). By enabling operations directly on the raw point cloud data, nD-PointCloud largely avoids the extract-transform-load hurdle, which is an increasing problem in using big data. This enables major advances in domains requiring lossless spatio-temporal data of extremely high accuracy, such as applications of scanned surface models and moving object (trajectory) models, which are used as Proof-of-Principle. If successful, then nD-PointCloud will become the most used model (more than vector or raster), enabling progress in research fields like water management, land administration, vegetation monitoring, building modelling, transportation and mobility.

Deep integration allows analysis and visualization which would otherwise not be possible. Furthermore, this deep integration is key to efficient solutions; e.g. nD points represented via space filling curves, such as Morton or Hilbert curves (Guan et al 2018). Conceptually this deep integration was applied for the spatial and temporal dimensions in Hägerstrand's space-time cube (1970) to represent and visualize trajectories of moving objects (in space over time) and being able to decide if multiple trajectories meet in space and time (e.g. when tracking the attendance of people in a music concert when the individuals have positioning devices) or if trajectories represent people visiting the same place but at different moments in time (e.g. people shopping in the same market at various hours of the day). In a similar manner Meijers and van Oosterom (2012) propose the deep integration of spatial and scale (continuous Level of Detail) dimensions for vector data to provide solutions for efficient overviews, smooth zooms, and deriving uniform-scale views and mixed-scale (perspective or fish-eye) views: the space-scale cube. nD-PointCloud aims at being a generic approach, applicable to many domains.

In the majority of today's applications, point clouds are managed using file solutions with specific formats (e.g. ASPRS LAS). Typical file-based solutions include desktop applications (usually vendor-specific) and command-line executables, like Rapidlasso's LasTools (mixed-source) or the Point Cloud Abstraction Library (PDAL) (an open source project). The work-flow includes reading one or more files, processing the data and writing files back to the user. These traditional file-based solutions have major drawbacks, such as data isolation, data redundancy, and application dependency. DataBase Management Systems (DBMSs) do not have these drawbacks. The database community, commercial and open source, provides point cloud specific data structures; e.g. Oracle (Spatial and Graph) and PostgreSQL (PostGIS). The DBMS-based solutions can be categorized into two types (van Oosterom et al 2015): the block and flat-table models. In the flat-table model, points are directly stored in a database table, one row per point, resulting in tables with many rows (Wang and Shan 2005). This flat-table model is easy to implement and flexible for query and manipulation, but multi-dimensional clustering and indexing is a challenge (Martinez-Rubi et al 2014). For this purpose, the Space

Filling Curve (SFC) is introduced. This results in the nD-PointCloud as a new datatype, which is described in more detail in Section 2. A first proposal for the DBMS interface specification for nD-PointCloud is given in Section 3. Finally, conclusions and future work can be found in Section 4.

2 The nD-PointCloud as a new datatype

The nD-PointCloud approach (model with accompanying theory and technology) will be a radically new line, compared to today's gridded (raster, voxel) or object (vector, polyline/ polygon/ polyhedron) representations (Molenaar 1998) that work with discrete number of levels of importance (detail/scale). The continuous Level of Importance (cLoI) support is provided via its deep integration with space and time, i.e. their combination into a single representation. The *n* organizing dimensions, usually a subset of space, time and cLoI, define the location of a point in the *nD* Organizing Space. In addition, and depending on the specific domain and data set, there are *m* additional property dimensions (e.g. colour, flow direction, velocity, surface normal, different spectral channels, temperature, pressure, classification, object identity, light intensity, uncertainty, content descriptors, etc.), which together conceptually define points in a $k=n+m$ dimensional space. Deep integration offers a range of benefits: spatio-temporal consistent models, efficient selection (on organizing dimensions; e.g. simultaneous selection based on space and time search predicates), new types of spatio-temporal computations, use of continuous importance, and integration of data from multiple sources.

2.1 Discrete vs. continuous levels

There is a serious drawback in current models as their discrete number of levels will give density difference artefacts. The current state of the art is organizing point clouds in discrete LoD, or data pyramids (Arikan et al 2014, van Oosterom et al 2016) as illustrated in Figure 1 for the 2D case. This organization allows fast spatial searching including LoD selection: the further away from viewer the lesser points selected, i.e. the higher level blocks/points (Figure 3-right). These representations offer a discrete set of levels, i.e. multiple map-scales (Jones et al. 1996, Kilpelainen 1997, Friis-Christensen and Jensen 2003). Discrete levels have well known restrictions; e.g. in perspective views (more detail nearby, less detail further away), the abrupt transition between various levels is often disturbing to the human eye, in spatio-temporal analysis/computation just the predefined discrete levels are available and 'optimally detailed' representation may not be available. There is a continuous point cloud approach proposed by Microsoft (2014), but this has severe drawbacks as it is not based on using cLoI as an organizing dimension and the solution is less general (focuses on visualization of 3D point clouds).

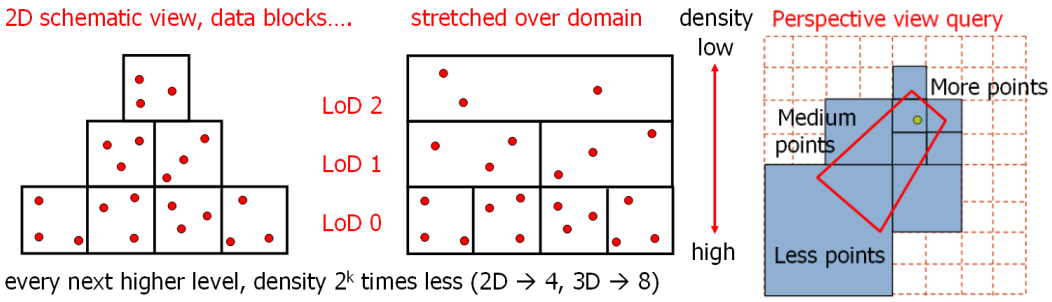


Figure 1: Discrete LoD (scale/importance), left/middle: symbolic 'side view' of 2D point cloud data blocks with height now LoD; right: 'top view' for selecting point cloud data blocks for perspective query

The human eye is very sensitive to this disturbing effect as can be noticed in Figure 2, but the same applies to 'suboptimal' analysis based on similar multi-level organization. More gain is possible by taking this strategy to the limit using an infinite number of levels, which is what the proposed cLoI approach addresses; see Figure 3. This is one of the key features of my nD-PointCloud model: cLoI becomes a true continuous dimension and it is used in the data organization: clustering, indexing (point cloud in space-time-importance data blocks) for fast retrieval and transmission. Regarding the time dimension, current solutions usually construct independent data structures for each time snapshot of the scanned surface. This disables efficient spatio-temporal analyses. The nD-PointCloud approach which deeply integrates space, time and importance aims at better enabling this kind of analyses. Changes between different moments in time can be more efficiently detected and analysed (up to its maximum available resolution), i.e. directly on point cloud data without down-sampling to grid or other representations.

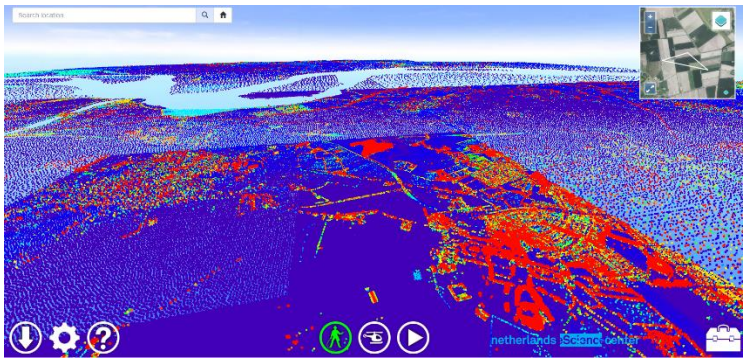


Figure 2: Potree webviewer with AHN2 data (640 billion points) showing the discrete LoD (note the data density differences)

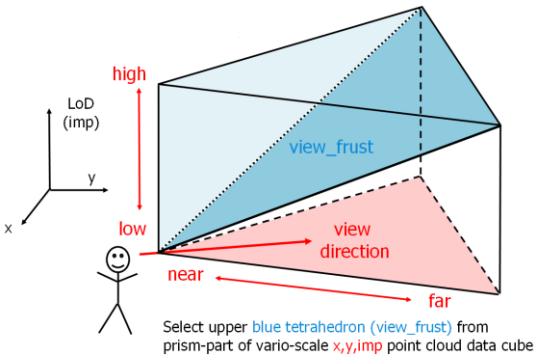
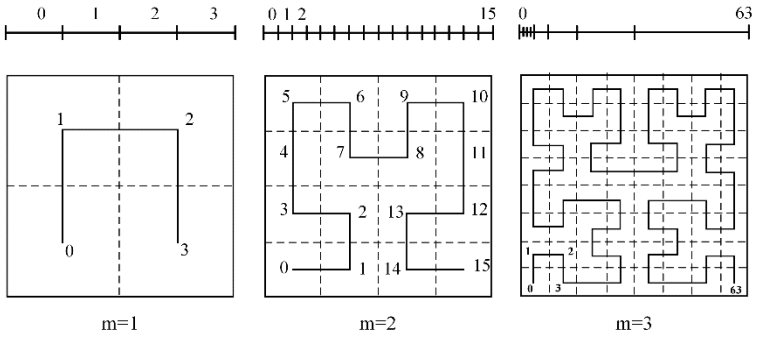


Figure 3: Perspective view: further away from viewer, gradually only points with higher cLoI

2.2 Space Filling Curves for organizing nD-PointClouds

Space filling curves (SFCs) such as Morton or Hilbert curves (Butz 1971, van Oosterom 1999, Lawder and King 2001, Bader 2012, Guan et al 2018) are proposed as a foundation for spatio-temporal representation. SFCs have the ability to cluster points close in reality, close on the curve. In mathematics, a space-filling curve is a continuous bijection from the hypercube in nD space to a 1D curve, i.e. C : R^n → R (Jaffer 2014). The nD hypercube is of the order m if it has a uniform side length 2^m (see Figure 4). Analogously, the curve C also has an order m and its length equals to total number of 2^{n*m} cells.



The method of SFCs will enable the integration of all the organizing dimensions into a single value. The design options that need to be researched and analysed for their impact, include: what are the organizing dimensions (used for SFC computation), what are the other attributes, what type of organization (relative scaling of organizing dimensions encoding, exact type of SFC, e.g. Morton or Hilbert curve, explicit data partitioning parameters such as clustering/ blocking size), what compression, etc.

Figure 4: The illustration of 2D Hilbert curves with different orders

The proposed nD-PointCloud approach supports efficient clustering, indexing, compression, data selection, transmission and use of huge point cloud data sets; towards 10^{15} points. One might end-up in organizational spaces with five or more (generally n) organizing dimensions with m attributes as property dimensions. The right balancing of organizing dimensions and other attributes when designing the data model is one of the research challenges. Given the huge volumes of data, parallel processing will be employed. The parallel approaches enable an nD-PointCloud analysis where serial approaches are not working (e.g. memory limits). Due to the 'quadrant' recursive partitioning of space, the SFCs may be used to subdivide a large problem into smaller parts suitable for parallel processing.

2.3 Using the SFC for multidimensional queries

Multidimensional selections using SFCs require a modified query algorithm that takes into account the SFC organization. This means that the query geometry needs to be translated into a number of continuous runs on the curve. Because all the above-mentioned queries correspond to a kind of multi-dimensional range query, we make use of the relationship between the SFC and the quadtree (van Oosterom and Vijlbrief 1996, Gargantini 1982) or 2^n trees for higher dimensions. The maximum depth of the tree affects: (1) the number of SFC ranges that compose the query, and (2) the approximation of the query geometry. Only requesting higher levels will give coarser 2^n tree cells, resulting in additional points. The query procedure starts with filtering (and can be optionally followed by exact refinement computation). The filtering recursively checks the 2^n tree cells that intersect with the query region, up to a specific depth, producing a mixture of big and small ranges returned, with the smaller ones located mostly near the boundary (Figure 5 left). The cells are then translated into the equivalent SFC ranges and the neighbouring ranges on the curve are merged without add-

ing any tree cell space (Figure 5 middle). The direct neighbour merging can create non-rectangular ones. The ranges are further merged with nearby (but non-direct) neighbours in case they exceed a specified maximum amount (Figure 5 right). Merging of non-direct neighbours will always result in additional tree cell space added to the original situation. The returned ranges are used for fetching the data. Despite merging of ranges, the result is an approximation which is very accurate, compared to the bounding box.

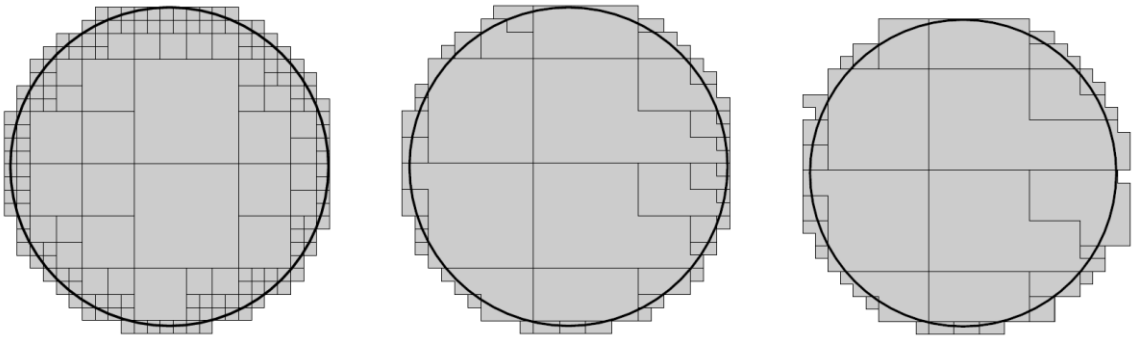


Figure 5: The different steps in the preparation of the filter step: (left) Tree cell identification, 176 cells, (middle) direct neighbour merging, 42 cells and, (right) merging to maximum number of 30 cells

3 DBMS interface specification for nD-PointClouds

This paper presents data structures and functions required to manage n-dimensional (nD) point cloud data in a DBMS. The system (interface) is designed to be flexible, easy to use and efficient at the same time. To some extent these are contradictory requirements, this makes it relatively difficult to implement the system in such a way that these requirements are well balanced (the optimal solution may be context dependent, but we propose a generic solution). The nD-PointClouds can be of quite different nature or background but are all within the scope of this specification. So, on one side we want to represent nD-PointClouds which are a result of scanning surfaces (LiDAR, MBES, dense matching images, etc.), but we also want to represent nD-PointClouds which are the result of tracing moving objects.

In this section the general ideas and interface will be described, many details can be found in earlier papers which describe the research and partial implementation related to management of nD-PointClouds in a DBMS. (van Oosterom et al, 2015) introduce the challenge to manage massive point cloud data in a database, present a benchmark, and propose to use a space filling curve (SFC) for efficient 3D point cloud data management with test case the AHN2 data. (Psomadaki et al, 2016) propose to use a SFC for the management of dynamic point cloud data and analyse organizations options: ranging from 2D (just xy in SFC key) to 4D (xyzt in key) with test case coastal monitoring data. The SFC computation was in both cases outside the Oracle database and implemented in Python (drawback a lot of data transfer between database and SFC programs). (Guan et al, 2018) implemented the nD SFC Library in C++ (more efficient than Python) and tested up-to 5D by adding also the importance dimension to the SFC key (xyzti), but expensive data transfer between database and external SFC software remained. In all of the above solutions the Oracle database with the Index Organized Table (IOT) was used, with several performance benefits: a. data is clustered (in order of key) -> faster, no memory jumping after range selection, b. single structure is more compact than 2 structures (table + index)-> faster, less reading, c. no combination of 2 structures ('join' of index and table needed) -> faster, less processing.

(de Vreede, 2016) implemented using Python the SFC key technique in combination with the MongoDB database and tested this with moving ship trajectory data of the Automatic Identification System (system). Unfortunately, MongoDB does not support clustering of data (as in IOT). (Meijers et al, 2017) implemented using Python and Rust the SFC key technique inside the PostgreSQL database and also tested this with AIS data (2D space and 1D time). PostgreSQL does support clustering the table according to order on the index (SFC based), but index and table are different structures, so not completely having the benefits of an IOT. Finally (Liu et al, 2018) present the ambition towards 10^{15}-level point clouds management based on a generic nD PointCloud structure with functionality inside the database (and benefiting from IOT). In addition to the cited publications most of the accompanying software is also available as open source; e.g. the Python code for the Dynamic Point Cloud is available at https://github.com/stpsomad/DynamicPCDMS, the C++ code for Morton/Hilbert encode/decode/range generation is available at https://github.com/kwan2004/SFCLib and the Python and Rust code for running inside PostgreSQL is available at https://bitbucket.org/bmmeijers/sfc-rs and https://bitbucket.org/bmmeijers/sfc-rs-ffi.

In general, all the above approaches store one point in one row (record, tuple) in the database and the organize the points well (according the nD SFC). For several reasons the next step could be to group points in blocks. When starting from the sorted individual points (with their SFC key) this is now rather easy. Some benefits of blocks are: nice communication unit between server and web-client (allowing caching), compression well possible because nearby points

will be in same block and their first part (most significant bits) are quite equal. However, in this document we limit ourselves to one point per row (and blocks are future work). Some of the lessons learnt from these experiences in various database environments are:

- Oracle IOT is elegant and efficient structure,
- full high-res key allows omitting storing the corresponding attributes (e.g. x or y as they can be decoded in full resolution from the SFC key),
- encoding/decoding, range generation outside database results in quite a bit of communication overhead,
- full high-res and higher dimensional keys results in large keys (for sure not fitting in 64 bits, so actual representation requires attention, e.g. use varchar or raw bytes),
- the PostgreSQL SFC key implementation within the database avoids massive data transfer in/out data when using the functions,
- the roles of organizing properties (e.g. x, y, time, importance called organizing dimensions) and other properties (non-organizing, call attributes) is rather arbitrary and context/use dependent,
- relative scaling of different organizing dimensions (space, time, importance) is needed to compute SFC (e.g. 1 meter = 1 second, but non-trivial as this influences the actual clustering). Finally,
- directly generating SFC ranges for different query geometry shapes has big benefits over simple bounding box approximation (more precise approximation will result in less false hits), but a high-resolution query geometry range generation in nD space may result in (too) many ranges especially when specifying just 2 dimensions of the key and the other open (implying they extend to infinite).

3.1 Handling nD points with a Space Filling Curve based organisation

Most databases cannot handle multi-dimensional (nD) point cloud data very well. Solutions are available for multi-dimensional spatial vector or raster data, but these are not very suitable for nD-PointClouds. Other databases exist that target multi-dimensional array data. The solution chosen here is to ‘map’ the multiple dimensions of the point to a single dimension. This takes the form of what will be called a SFC key. SFC is used because of optimal organisation of the dimensions when SFC encoding (Morton or Hilbert) is applied. The SFC is the key of the IOT (single structure).

The main idea for the point cloud management system described here is that fast access to sets of points can be achieved by means of a SFCkey. Each point of the point cloud is stored as a full resolution SFCkey, possibly followed by (other) point attributes. The SFCkey consists of a combination of dimensions. Dimensions are point attributes that play a role in the organisation (structuring) of the data, the SFCkey represents this organisation. Typical dimensions for point clouds are the spatial X, Y, Z dimensions, and time. Other potential dimensions are e.g. importance or level of detail. A point cloud is seen as any collection of points with spatial and other attributes, so in this sense a GPS track is also considered a point cloud. In this last case an ID could become a dimension. From these examples it is clear that also non-spatial attributes can be can be part of the SFCkey.

The dimensions are included in the SFCkey in such a way that points close together in n-dimensional ‘space’ (nD hypercube) also have a similar key. But obviously not all dimensions are equally prominent, depending on the relative scaling of the dimensions added to the SFCkey. So, the scaling, of dimensions have repercussions for the efficiency of the SFCkey for data retrieval. A consequence of this is that different applications might require different inclusion/scaling of dimensions, hence there exists a requirement for a flexible setup of dimensions and SFCkey.

A distinction can be made between full resolution keys and partial resolution keys. With full resolution keys the original attribute (= dimension) values can be calculated from the SFCkey without loss of information, in this case the values for the dimensions do not have to be stored explicitly. For partial resolution keys this is not possible, all values for dimensions and other attributes have to be stored in addition to the SFCkey (in this last case the SFCkey merely has the function of a cluster and index structure and implies group/blocks of points). The advantage of partial resolution keys is that less space is required for the SFCkey itself. However, when the key is used as primary key for the table in which the point data is stored, it must be unique (and not NULL). This is important when applying the SFCkey in combination with the IOT (as this needs a primary key). For partial resolution keys this is not the case and an option is to include another attribute to the key, or make the key high resolution. So, these are not considered (for the time being). Although not strictly necessary it can be an option to store all dimensions and other attributes explicitly, even if a full resolution SFCkey is used: This would make superfluous the use of a decoding function (less processing at the expense of more storage).

Queries on point clouds in the system described here use a two-stage query process similar to that often used for spatial data. Query geometries have to be translated to range queries on the SFCkey: the primary filter retrieves parts of the hypervolume by specifying a number of key ranges. If more fine ranges are specified the primary filter becomes more selective, but at the same time more ranges have to be processed. Less, but larger ranges, mean coarser resolution with as a result more points selected and as a consequence more work for the secondary filter. Somewhere an optimum will

exist for the number of ranges to generate for a query. Range generation is an important function the system has to provide to achieve good query performance.

To a large extent the implementation of the management of point cloud data in a DBMS can make use of existing data types, functions, structures, database tools, and so on. This is mainly due to the reduction of multiple dimensions to a single dimension. But some functions and data structures are specific for the proposed solution and have to be added to the DBMS, these are:

- Metadata for the point cloud dataset and the dimensions
- `SFC_ENCODE` to compute SFCkey from the multiple (n) dimensions using the metadata
- `SFC_DECODE` to compute the multiple (n) dimensions from the SFC key using the metadata
- `SFC_RANGES` to compute SFCkey ranges that cover the nD query geometry
- `SFC_CREATE` to set-up and load point cloud data (using metadata/encode) it IOT
- `SFC_DROP` to remove the point cloud and related IOT

The metadata is detailed below and is specifying the dimensions, their range and resolution, the actual encoding format of the SFCkey, etc. The encode, decode and range functions need the metadata to be able to work properly. In general, the SFCkey solution is rather generic and can be applied to different DBMSs. However, below we assume Oracle as platform amongst others because the availability of the IOT.

3.2 Summary of requirements

Three categories of requirements have been identified for the relational database SFC interface specification for managing nD-PointClouds: Flexibility requirements, Performance requirements, and Ease-of-use requirements. These will now be elaborated on below.

1. Flexibility requirements
   - support for multiple dimensions with flexible ordering, scaling
   - support for various types of SFC (e.g. Morton, Hilbert)
   - support for different encodings of SFCkey (e.g. number, character) for high-res large nD values needing more than 64 (or 128) bits
   - support for partial and full resolution keys
   - support for implicit (as part of SFCkey) or explicit (as additional attributes) storage of dimensions
   - higher dimensional query geometries of various types (more than just an nD box)

2. Performance requirements
   - data that will be used together must be stored together
   - processing related to storage and query of point cloud data should take place in the environment where the data is stored: in the database
   - data movements should be minimised (and data encode/decode conversions with SFC software should be inside database to get access to attribute in full resolution SFCkey)

3. Ease-of-use requirements
   - functionality must be available via SQL functions
   - despite all tuning options, use of functions should be straightforward (balance: one time tuning via the info in metadata, which may be a bit harder compared to the actual use of encode/ decode/ range functions as the later are used more often)
   - functionality should fit well with other spatial DBMS functionality (e.g. to perform second filter point in polygon test)
   - specifying a nD query geometry may be done via nD simplicial complexes or nD regular polytopes, but users are most likely unfamiliar with this, so simpler alternatives needed (e.g. extruded polygon/ polyhedron)
   - the number of SFC ranges is a crucial performance factor which user should be able to influence (comparable to resolution, depth in recursion, and involve glueing of ranges)

3.3 nD-PointCloud metadata and signatures of the SFC functions

The metadata contain the details of the nD space/hypercube as defined by the organizing dimension and their relative scaling, resolution and encoding. There are two types: one per dataset and the other per organizing dimension. In Oracle the metadata table per dataset could be called USER_PC_SFC_METADATA and contain the following attributes:

Dataset metadata

```
PC_ID                  number            system-generated, unique number
                                         (link to functions)
PC_NAME                varchar2(32)      name of point cloud dataset
PC_DESCRIPTION         varchar2(256)     description of point cloud
PC_TABLE               varchar2(32)      name final structured table IOT
STAGING_TABLE          varchar2(32)      name intermediate staging table
                                         (needed, may be virtual)
PC_DIMENSIONS          number            number of dimensions
PC_NUMBITS             number            number of bits per dimension
                                         with which to encode
SFC_TYPE               varchar2(32)      'Morton' or 'Hilbert'
SFC_ENCODING           varchar2(32)      'Number' or 'Character' or
                                         'Base32' or 'Base64' or 'Raw'…
KEY_TYPE               varchar2(32)      'Partial' or 'Full'
DIM_PROPERTIES         varray(PC_DIMENSIONS) of DIM_SPEC
SRID                   number            spatial reference of points in
                                         point cloud (in 2D or 3D)
TIME_EPOCH             datetime          start encoded time value (UTC)
TIME_UNIT              varchar2          unit time: year,week,day,hour
TIME_UCT_OFFSET        number            UTC offset
```

Note: These last 4 items belong to some specific DIM_SPEC. However, these items carry important meaning for whole point related to spatio-temporal reference. They might be specified per dimension (in DIM_SPEC), but this would introduce a bit of redundancy (e.g. 3 times same SRID for spatial dims X, Y, and Z). Also, the possible temporal dimension has special meaning. However, not sure if the above is indeed the best decision. What about importance (scale) is this also not having specific predefined meaning, which is needed for perspective view queries.

Dimension specific metadata via custom data type DIM_SPEC (one set of values for each dimension)

```
DIM.NAME               varchar2(32)      name of dimension
DIM.DESCRIPTION        varchar2(256)     description of dimension
DIM.OFFSET             number            value to be added for
                                         normalized dimension value
DIM.SCALE              number            scale factor for normalized
                                         dimension value
DIM.MINVAL             number            minimal dim value in dataset
DIM.MAXVAL             number            maximum dim value in dataset
DIM.TOL                number            tolerance associated with dim
```

Encode function

```
SFC_ENCODE (PC_ID, VAL_D1, VAL_D2, … , VAL_Dn)
Return: SFCkey
```

Note: The actual encoding depends on and must be in sync with the metadata (Morton/Hilbert, resolution, etc.)

Decode function

```
SFC_DECODE (PC_ID, SFCkey)
Return: VAL_D1, VAL_D2, … , VAL_Dn
```

Range generation function

```
SFC_RANGES (PC_ID, QUERY_GEOMETRY, [recursion depth,] [max_nr_ranges,] [type] …)
Return: table of SFCkey_LOW, SFCkey_HIGH values [,range_type]
```

Notes:

1. the SFC_RANGES are used for the primary filter in the query process (if needed a secondary filter should be applied with query geometry to get exact response)

2. if specified, recursion depth indicates the actual resolution (cell size of SFC ranges), and user should be careful because if resolution too high then many ranges will be generated
3. if specified, max_nr_ranges then this number of ranges will be returned. In case during computation more ranges are created (due to specified recursion depth), then ranges with smallest gap between them are merged together.
4. some options for the query geometry are: a. nD-hyperbox, b. nD-hypersphere, c. 2D polygon+extrusion min/max other dimensions, d. 3D polyhedron+extrusion min/max other dimensions, e. intersection nD-halfspaces (regular polytope), etc.
5. important special case may be the perspective view (gradually less points when further away from viewer), note this assumes the presence of importance dimension.
6. Some SFC ranges may be complete inside query geometry, some may be on boundary. With the 'type' parameter user may specify how these will be treated in return table

We could have two types of ranges: 1. Completely inside query geometry and 2. On boundary of query geometry. For points with their SFCkey in a range which is completely inside, there is no need to perform a secondary filter (which most likely is expensive to calculate), for sure point is inside. Only in case point has it SFCkey in range which is on the boundary of the query geometry the second filter step with exact query geometry has to be applied. How to include this in the interface of the sfc_ranges function? One option is to add overlap column to the result of sfc_ranges function that has either 'full' or 'partial'. Based on the overlap type, the join to the original table can be expressed with or without secondary filter. The parameter 'type' can have 2 values: added or not_added (last which is default. If it is requested to add the range type, then the return table has 3rd column indicating if range is completely inside or on boundary (alternative the type could be encode by a sign to ranges: ranges with + value are inside and ranges with – value are on boundary).

3.4 Loading data and example use of SFC functionality in queries

The typical data sources are CSV files or LAS/LAZ files, but many more types do exist depending on nature and background of point cloud (and associated attributes, such as colour (RGB), intensity, but also classification and object id as potential attributes). There may be several options to load the data:

1. Load direct from input files to final, structured table (this may not be the most efficient when the structure is an IOT, as then during adding data the Btree is continuously maintained)
2. Load from input files via staging table (heap) to final, structured table (this may be more efficient as structure is created in a bulk-manner, but an additional staging table is needed). It is to be investigated if this staging table can be virtual, i.e. the actual data is in the input file(s) which are mapped to a database table (without explicit storing/ copying but giving to the user the normal table interface/ feeling, e.g. by means of using a Foreign Data Wrapper, FDW).

Example use of the functionality for loading and assuming a staging table with the input data (option 2), after first filling metadata table (and during actual conversion to the IOT/SFC structure also sfc_encode function is used):

1. Filling metadata:

```
Insert into user_pc_sfc_metadata values (1, 'my test', 'a test point cloud',
'my_pc', 'test_pc_staging', 3, 21, 'Hilbert', 'Number', 'Full', 28992, -1,
'na', 0)
```

2. Loading a table:

```
Select sfc_create(pc_id);
```

The query geometry is converted to the relevant SFC ranges. Below first example query to just get the SFC ranges for a specific query geometry:

```
select * from sfc_ranges(pc_id, query_geom);
 min | max
-----+-----
 502 | 503
 507 | 508
 608 | 609
(3 rows)
```

This can be combined in a query with the pc data table in join to obtain the selected points according to the SFC ranges:

```
select sfc_decode(pc_id, key)
from my_pc pc, sfc_ranges(pc_id, query_geom) range
where pc.key between range.min and range.max
```

Next, an example with secondary filter for ranges on boundary (partial outside and outside):

```
Select sfc_decode(my_pc, key)
from pc_table pc, sfc_ranges(my_pc, query_geom, 'added') range
where (range.type='full' and pc.key between range.min and range.max)
   or (range.type = 'partial' and
       pc.key between range.min and range.max and
       overlaps(query_geom, geometry(sfc_decode(my_pc, key))))
```

Drop a point cloud:

```
Select drop_sfc_pc(pc_id);
```

4 Conclusions and some future work

In this paper we argued that point cloud should be treated as a first-class representation (similar to vector or raster representations). The nD-PointCloud was presented as the solution for this, offering deep integration of space, time and scale. For efficiency reasons spatial indexing and clustering of these large point clouds is extremely important and the Space Filling Curved (SFC) was introduced for this purpose. In order to get beyond the current state of the art of storing/ managing point clouds in files, a DBMS solution was presented (with all benefits: integration with other data types, scalability, multi-user, transaction support, etc.). Finally, a DBMS SFC interface specification for point clouds was proposed. This proposal will be further discussed within standardization organizations, such as the OGC and more specifically the Domain Working Group (DWG) on point clouds.

There is still a significant amount of future work ahead. We will now focus of the challenge of having rather precise queries without too many SFC ranges. As indicated earlier generating high resolution ranges for a query geometry is good for precision (reporting right points) but the amount of ranges may get so high that we can lose reliable performance. An example is when we have high resolution nD keys (e.g. with n=5) and specifying in the query geometry just 2 dimensions and then leave the other 3 dimensions open (nothing specified). This implies the open dimensions extend to infinity and will generate a lot of SFC ranges over there (most likely without points). Some improvements are possible here. Simplest solution: if we know the actual data values of the points in all dimensions of the key, we can (implicitly) add this knowledge to the query geometry. In this case we do not generate SFC ranges where we do not have data. However, we do need to keep track of the actual data and maintain statistics (at least min and max values per dimension) when loading or adding data. Also, the range generation may become data dependent, meaning that when one more point is inserted stretching the current actual data range, then the sfc_ranges function before and after the insert will return different ranges.

Taking this one step further than just keeping track of points or no points in a certain part of the nD space: even if there is 1 point (or a few points) in a certain part of the nD space, why generate many ranges if the data is so sparse over there. This is wasting ranges. So, the next level of refinement is having a histogram with the actual data distribution (density). How would such a histogram look like for the nD point cloud? It could be an nD array (corresponding to nD hypercube) of a certain resolution, but what resolution, if too coarse then it will be of little use when generating ranges, but if too detailed then it will become too heavy (a lot of overhead). A proposed, but still to be implemented and tested solution is to collect statistics at the 2^n –tree internal nodes during the actual loading and computing of the SFC keys that corresponds to the recursion of the SFC computations. At top level a single value contains a total count of all points in the data set. At second level the points are counted per sub-cell (if n=3 then 8 children, each with their own count and the sum of the 8 counts same value as parent count). If count is 0, then no need to represent this branch of the tree (and also no need to refine for its children) and nothing wasted on empty part of this histogram. With this recursive nD-histogram we can decide to descend or not when generating ranges for a certain query geometry based on the estimated number of points. So, if this is a low number of point (even at high level in tree/ recursion) we can stop refining the range and be happy with course range (big) cell as it does not contain many points. If there is still a high number of points then we should further refine this large range (heavy cell) into smaller cells even if we are at already medium or lower levels (apparently there is a local very high density of points, which justifies refining the ranges).

Acknowledgements

We would like to thank Mike Horhammer from Oracle and Xuefueng Guan from Wuhan University for their active participation in developing the presented ideas related to managing nD-PointClouds. We would further like to thank the TU Delft MSc Geomatics students doing point cloud projects, amongst others, Stella Psomadaki and Irene de Vreede.

Literature

Arikan, M., R. Preiner, C. Scheiblauer, S. Jeschke, M. Wimmer (2014), Large-Scale Point-Cloud Visualization through Localized Textured Surface Reconstruction. IEEE Transactions on Visualization and Computer Graphics, 20(9):1280-92.

Bader, M. (2012), Space-filling curves: an introduction with applications in scientific computing. Springer Science & Business Media.

Butz, A.R. (1971), Alternative algorithm for Hilbert's space-filling curve. IEEE Tr. on Computers, 424–426.

Friis-Christensen, C.S. and A. Jensen (2003), Object-relational management of multiply represented geographic entities. Cambridge, MA, USA, July 9–11, 2003. Proceedings of the Fifteenth International Conference on Scientific and Statistical Database Management.

Gargantini, I. (1982), An effective way to represent quadtrees. Commun. ACM 25(12), pp. 905–910.

Guan,X., P. van Oosterom, B. Cheng (2018), A Parallel n-dimensional Space-Filling-Curve Library and Application in Massive Point Cloud Management. In: ISPRS Int. J. Geo-Inf., MDPI AG, 7(8), 19 pages.

Hägerstrand, T. (1970). What about people in regional science?. Papers of the Regional Science Association 24 (1): 6–21.

Jaffer, A. (2014), Recurrence for Pandimensional Space-Filling Functions. arXiv preprint arXiv:1402.1807.

Jones C.B., D.B. Kidner, L.Q. Luo, G.L. Bundy, and J.M. Ware (1996), Database design for a multi-scale spatial information system. International Journal Geographic Information Science, Vol. 10, pp. 901–920.

Kilpelainen, T. (1997), Multiple representation and generalisation of geo-databases for topographic maps. PhD thesis. Finnish Geodetic Institute.

Lawder, J.K. and P.J.H. King (2001), Querying multi-dimensional data indexed using the Hilbert space-filling curve. SIGMOD Rec. 30, pp. 19–24.

Liu, H., P. van Oosterom, M. Meijers, E. Verbree (2018), Towards 10^{15}-level point clouds management - a nD Point-Cloud structure, In: Proceedings of the 21th AGILE Conference on Geographic Information Science, Lund, pp. 7.

Martinez-Rubi, O., M. Kersten, R. Gonçalves, R. and M. Ivanova (2014), A column-store meets the point clouds. FOSS4GEurope Academic Track.

Martinez-Rubi, O., S. Verhoeven, M. van Meersbergen, M. Schütz, P. van Oosterom, R. Gonçalves, Th. Tijssen (2015), Taming the beast: Free and open-source massive point cloud web visualization, In: Capturing Reality Forum 2015, Salzburg, 12 pages.

Meijers, M., W. Quak, P. van Oosterom (2017), Archiving AIS messages in a Geo-DBMS, In: Proceedings of the 20th AGILE Conference on Geographic Information Science (Arnold Bregt, Tapani Sarjakoski, Ron van Lammeren, Frans Rip, eds.), Wageningen University & Research, 3 pages.

Molenaar, M. (1998), An introduction to the theory of spatial object modelling for GIS, Taylor and Francis (CRC Press), London, p. 200.

Psomadaki, S., P. van Oosterom, Th. Tijssen, F. Baart (2016), Using a Space Filling Curve Approach for the Management of Dynamic Point Clouds, Chapter in: ISPRS Annals, IV-2/W1, pp. 107-118.

van Oosterom, P. and T. Vijlbrief (1996), The spatial location code. In: Proceedings of the 7th International Symposium on Spatial Data Handling, Delft, the Netherlands.

van Oosterom, P. (1999), Spatial Access Methods, Chapter T2.3 in Geographical Information Systems Principles, Technical Issues, Management Issues, and Applications (edited by Longley, Goodchild, Maguire en Rhind), Wiley, pages 385-400 (vol.1).

van Oosterom, P. and M. Meijers (2012). Method and system for generating maps in an n-dimensional space. Technische Universiteit Delft, assignee. Patent NL2006630C / WO 2012144893 A2. 26 Oct. 2012. (https://patents.google.com/patent/WO2012144893A2/en).

van Oosterom, P., O. Martinez-Rubi, M. Ivanova, M. Horhammer, D. Geringer, S. Ravada, T. Tijssen, M. Kodde, and R. Gonçalves (2015), Massive point cloud data management: design, implementation and execution of a point cloud benchmark. Computers & Graphics 49, 92–125.

van Oosterom, P., O. Martinez-Rubi, Th. Tijssen, R. Gonçalves (2016), Realistic Benchmarks for Point Cloud Data Management Systems, Chapter in: Advances in 3D Geoinformation, pp. 1-30.

de Vreede, I. (2016), Managing Historic Automatic Identification System data by using a proper Database Management System structure, Master's thesis, Delft University of Technology, pp. 90.

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