Management of large indoor point clouds: an initial exploration

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

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

Commission IV, WG IV/7

KEY WORDS: Point Cloud, Indoor, Level of Detail, Oracle, Data Management, Benchmark, Morton Curve

ABSTRACT:

Indoor navigation and visualization become increasingly important nowadays. Meanwhile, the proliferation of new sensors as well as the advancement of data processing provide massive point clouds to model the indoor environment in high accuracy. However, current state-of-the-art solutions fail to manage such large datasets efficiently. File based solutions often require substantial development work while database solutions are still faced with issues such as inefficient data loading and indexing. In this research, through a case study which aims to solve the problem of intermittent rendering of massive points in the context of indoor navigation, we devised and implemented an algorithm to compute the continuous Level of Detail (cLoD) where geometric and classification information are considered. Benchmarks are developed and different approaches in Oracle are tested to learn the pros and cons. Surprisingly, the flat table approach could be very efficient compared with other schemes. The crucial point lies in how to address priority of different dimensions including cLoD, classification and spatial dimensions, and avoid unnecessary scanning of the table. Writing results either to the memory or the disk constitutes major part of the time cost when large output is concerned. Conventional solutions based on spatial data objects present poor performance due to cumbersome indexing structure, inaccurate selection and additional decoding process. Besides, approximate selection in the unit of physical object is proposed and the performance is satisfactory when large amount of data is requested. The knowledge acquired could prompt the development of a novel data management of high dimensional point clouds where the classification information is involved.

1. INTRODUCTION

A paradigm shift from outdoor to indoor spatial services has taken place in recent years, as the complexity of buildings increases and people spend most of the time indoors. Due to high accuracy and efficiency of data collection techniques, point clouds are utilized more frequently for indoor 3D modelling processes. In January 2018, the National Institute of Standards and Technology (NIST) of US initialized a public safety research program of which the topic was the collection of indoor point clouds. The intention was to build a standard prototype for indoor point cloud models as point clouds may become the basis for indoor applications for the next generation. Such agendas would definitely result in the harvesting of massive point data.

Unlike most conventional point cloud with only X/Y/Z attributes, indoor point clouds provide more abundant information including colour, and also semantics, e.g. classification which plays an important role in visibility detection and navigation. These attributes could also be named as dimensions because conceptually, there is no difference between these two terms (Liu et al., 2018b). Every type of information such as sound and temperature could be perceived as one dimension for us to comprehend the world. *However, in terms of storage, two types of dimensions are identified. One type is called organizing dimension which could be utilized to cluster and index the data, e.g. X/Y/Z. The other is the property dimension such as color, intensity and classification which is not frequently queried.* Depending on applications, these two types of dimension are interchangeable. All dimensions together form the nD point clouds. However, practical experience indicates that current database management systems (DBMS) present critical problems to manage massive nD point clouds such as inefficient loading/indexing, lack of support of continuous Level of Detail (cLoD) and limited functionalities. Flat table-based approaches mostly suffer from full table scans for simple queries, while block based solutions cost enormous time for construction of data blocks as well as decoding them for data extraction. Besides, the concept of LoD which serves as a general way for processing big data is either missing in state-of-the-art solutions or implemented using traditional Octree structure which presents side effects such as visual artefacts during rendering (Liu et al., 2018a). Basically, blocks of points in various densities are shown in the same scene.

The research aims at exploring a model of data management for large indoor point clouds. This will be constructed for specific applications, i.e. visualization of the indoor environment considering navigation needs. The whole paper is divide into 6 sections. The first two sections introduce the background information. This is then followed by a description of benchmark applied in Section 3. LoD in the indoor environment is specifically discussed in Section 4. Then by implementing different data schemas and testing in Oracle, results are presented and analysed in Section 5. The lessons learnt as well as research directions in the future are concluded in the end.

2. RELATED WORK

A data management solution normally starts from a conceptual model which guides the organization of various information. Hagedorn et al. (2009) proposed an indoor LoD model taking account of geometry, semantics and appearance of indoor objects for the purpose of indoor route visualization. The LoD model put forward includes 4 levels. The first two levels are based on 2D floor plans with different geometric accuracy and details of topology. The last two levels utilize a 3D vector model. The level LoD-3 only presents doors and windows in addition to the floor plans while LoD-4 cover all objects in the 3D model with highest accuracy.

As regard to point cloud management, lots of approaches exist. Most of them are file-based solutions such as LAS/LAZ (ISPRS, 2011), HDF (Folk et al., 2011), while other vendors adopt their own formats. Additional sorting and indexing to create block data structures for efficient querying have to be manually performed. Yet the scalability with large data cannot be guaranteed.

In contrast, the DBMS does not have such problems. State-of-the-art solutions are split into two types according to the storage model. Take Oracle as an example. One is the flat table approach where all dimensions are stored equally in each record for a point. Psomadaki (2016) developed an approach based on Oracle Index-Organized Tables (IOT) for the management of large dynamic point data. The index node was a Morton key (Figure 1) by encoding X/Y/Z or X/Y/T together, whereas other dimensions were stored as normal attributes in the database. By performing a benchmark, the best approach in the research concerns an equal treatment of the spatial and temporal dimensions in the Morton key. The other advanced solution is utilization of the SDO_PC data type which groups points into blocks which could be indexed by organizing dimensions. Van Oosterom et al. (2015) implemented this approach for managing and querying the AHN2 dataset consists of only xyz information. Results indicated that the block approach did not have noticeable scaling effects when increasing the data size. The query accuracy is at point level, so that a significant overhead would be incurred due to the need to unpack the blocks for checking whether individual points were within the query regions and also when dumping the selected points in the results table. What was kept in the cache were the points instead of the blocks. When the query was repeated, the blocks had to be read again, which confined the scalability.

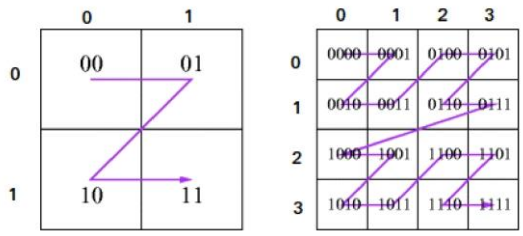

Figure 1. The first and second order of the Morton curve. In each cell, the corresponding key is shown.

A comprehensive benchmark test as the last step is needed to assess the performance of different solutions as well as to prompt further optimization. Van Oosterom et al. (2015) designed and implemented a benchmark for large point clouds management, after collecting user requirements (Suijker et al., 2014). The testing dataset AHN2 elaborates totally 640 billion points with 12 TB size in LAS files. Various platforms and data organizing approaches were tested including PostgreSQL flat table, PostgreSQL block, Oracle flat table, Oracle block, Oracle Exadata, MonetDB and LAStools. To exploit the scalability, the benchmark was decomposed into several stages with different data size, i.e. mini (20 million points), medium (20 billion points), and full benchmark (640 billion points). Besides, two parallel query processing algorithms were presented and partly tested to learn the improvement of performance.

3. BENCHMARK

As is mentioned, the large nD point cloud data is the focus, and an open dataset is utilized. From the query list provided by van Oosterom et al. (2015), relevant processes of indoor applications as well as the maintenance of data are selected for the benchmark testing.

3.1 Data

The Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) (Armeni et al., 2016), containing totally 273,608,340 points, is collected by a 3D camera (Figure 2). The dataset is split into 6 areas, corresponding to 6 folders. Each area (folder) is further divided into separate rooms (subfolder) such as conference room, hallway, office, etc. Inside a folder, different objects are stored individually using text files, with desk_1 and floor_1 for example, as the file name. Each text file contains 6 fields and they are x, y, z, R, G and B. The spatial range is (-37.928, -26.078, -2.645, 29.927, 46.056, 6.576) in (lower left corner, upper right corner).

Figure 2. The S3DIS model

As the semantic information like classification of objects, identifier of objects and room types is encoded into file names in the original dataset, to manage all the information into a DBMS, more dimensions should be added to each point record explicitly. They are:

1. *Classification* such as floor, door and stairs
2. *RoomType* such as hallway and office
3. *RoomID*, a numeric identifier for each room
4. *ObjectID*, a numeric identifier for each object
5. *LoD*, a numeric value for each point indicating the importance and it is a continuous field.

3.2 Queries

1. Multi-resolution/LoD selections. For example, select top 1% of the points, i.e. the first 1% most important points.
2. Simple 2D range/rectangle filters of various sizes.
3. Selection on other dimensions such as the colour and the classification.
4. The K nearest neighbours’ search.
5. Attribute statistics including minimum, maximum, average, median, and count.
6. Update of point geometries, e.g. some small changes to many points.
7. Computations of areas of implied surface by point clouds.
8. Computations of volumes below surfaces.
9. Deletion of portions of points (0.1%, 1%, 5%, 10%, 25%).

4. LOD COMPUTATION

Conventional LoD is comprised of discrete layers, for instance, the 4-layer LoD model mentioned in Section 2. For indoor point clouds, this entails visualizing the indoor environment with several distinct densities. The sudden change between different layers does not keep in line with human’s visual perception. Hence the cLoD concept is developed to gradually visualize points at different scales, which can make the rendering process more smoothly and naturally. In the cLoD structure, every point is assigned an importance value indicating its ranking among the whole dataset. So there is no discrete data layers as before.

Unlike the 3D indoor models represented by vectors, the indoor point cloud model does not address too much on topology. In this research, when visualizing the point clouds for the navigation purpose, the computation of LoD only takes account of geometric and semantic information.

4.1 Geometric LoD

The method is to first construct a Btree, Quadtree or Octree structure for each object, and then within each level of the tree, the ranking of points is randomized. Afterwards, all levels are combined sequentially. So the final result is the same as the original data but with a different ranking of points. Specifically, more important points which are located at geometric centres are at the head of the list. In this method, the tree structure is implicitly embedded. Every point represents a level.

To determine whether Btree, Quadtree or Octree for organizing the points of an object, a *dimapprox* function is built. The basic idea is to first grasp the range of x, y and z of the object where 90% of points fall respectively. Then the ranges are compared. If the largest range is 10 times larger than the second, which implies a linear feature, then the object is regarded as a 1D object and Btree will be applied. If the second largest range is 10 times larger than the third, which indicates a surface, then the object will be simulated in 2D and Quadtree will be employed. Otherwise, the Octree will be adopted. In this way, computing efficiency can be improved and geometric features can be captured more accurately. As an illustration, a 2D wall is presented in Figure 3.

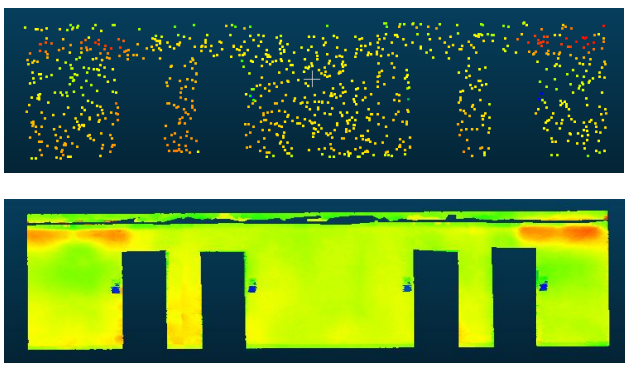

Figure 3. Representation of a wall in a small scale and large scale by controlling number of points to render after the cLoD sorting.

As the objective is the visualization, a purely random ranking could also be applicable to compute the cLoD. That is, by assigning a random number between 0 to 1 to each point, sort the points using the number. Unless the number of points to visualize is too small, the object could be recognized. The computational load becomes less than that of the tree-based approach.

4.2 Semantic LoD

The geometric LoD cannot express the importance of objects for navigation. Large objects like wall and floor would be represented by much more points than small objects like window and door by choosing a same range of cLoD values. Hence, three classes are established to represent the importance level. The first class (most important) includes 'stairs', 'floor', 'window' and 'door'. They are essential for navigation. The second class contains objects like 'wall', 'ceiling', 'clutter', etc. The third class refers to 'sofa', 'bookcase', 'board', 'table' and 'chair'. They are movable objects which are less significant for routing.

The semantic LoD method is then to assign a unique importance value to each point. Basically, after the geometric sorting, a random series based on the uniform distribution between 0 and 1 will be generated. Then the random numbers are sorted and each of them is attached to the end of a point record as the initial LoD value. However, the LoD values belonging to objects in the second class will be multiplied by 0.9, while for the third class a factor 0.8 is applied. In the end, all points from different objects are mixed and sorted according to the LoD value. The head of the final point list would be occupied by points in the first class and at the geometric centres. Figure 4 shows a demonstration of a conference room by utilizing the geometric and semantic LoD method. It clearly presents that when a small amount of points are selected, only the first class objects are rendered.

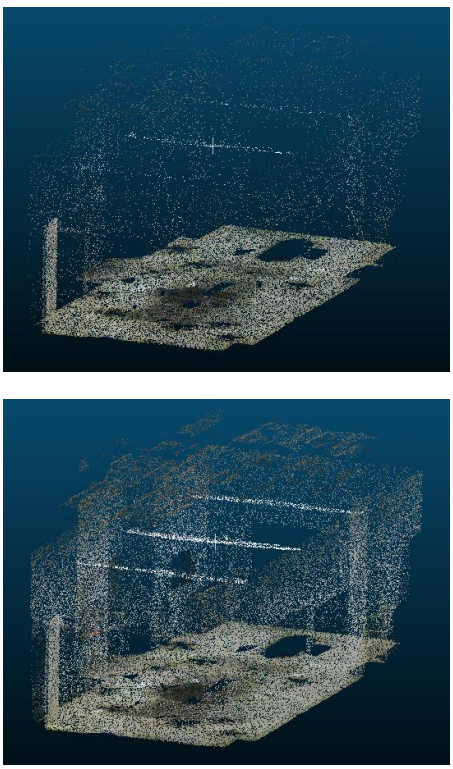

Figure 4. Visualization of a conference room using the indoor cLoD model. The scenes contain first 50 000, 100 000 points respectively sorted by the cLoD value.

5. TEST RESULTS AND DISCUSSION

To get more understanding for managing and querying the indoor point cloud data, possible approaches have been implemented in Oracle. Data are stored in a normal flat table, a nested table, a table with a spatial index, SDO_PC blocks and an IOT respectively (schemas are provided in the Appendix). Queries executed only include spatial selection, LoD selection and classification selection. Each query is executed several times with cache flushed until the response time converges to a stable value. The time measurement starts from sending the SQL command, while ends with storing the results into a Python variable, i.e. an in-memory object. The SDO_PC solution is an exception as it writes results into another table. The aim of the testing is to examine the possible direction to

realize an efficient data management. The test platform is a HP DL380p Gen8 server with 2 × 8-core Intel Xeon processors, E5-2690 at 2.9 GHz, 128 GB of main memory, a RHEL6 operating system. The disk storage is a 41 TB SATA 7200 rpm in RAID5 configuration. Advanced disks such as SSD is also available but they are not utilized.

5.1 Flat table

As every dimension might be queried, so no index is created. The table contains following fields:
X, Y, Z, R, G, B, RoomType, RoomID, Classification, ObjectID, LoD

All field use the same data type, NUMBER. With SQL loader, the data can be directly imported into a flat table (Table 1). Its storage space is 17,141 MB and utilizes 548,495 Oracle logical blocks.

Due to the structure, whatever the selection is, a full table scan will be executed. All queries tested are listed in Table 1. These queries are applied for testing other solutions as well. The first COUNT query is executed to learn the speed of a full table scan. The second query is concerned with LoD selection, and 4 levels are set to assess the scalability (Figure 5) which is linear. Similarly, spatial selections are executed. The tendency also shows a linear scalability and time cost can be modelled by 9 x 10^{-6} n + 6.54 where n is the number of points returned.

From the execution plan, all these queries experience the full table scan. The time cost therefore includes table scanning, if-statements for filtering and writing into the memory. The linear tendency of the real measurements verifies this assertion. With a constant representing full table scan and if-statements, the main component of time cost lies in the writing part.

Query	Number of records returned	Time cost (s)
1. select COUNT(*) from s3disflat	1	15.2
2. select * from s3disflat where lod>0.99/0.95/0.8/0.5	657,295/ 3,287,515/ 31,730,001/ 122,433,484	17.72/ 39.57/ 277.92/ 1042.3
3. select * from s3disflat where x>12.075 and x<20.13 and y>15.267 and y<19.3 and z>0 and z<2 / (-9.072, 17.307, 1, 3.208, 23.557, 2.8) / (4.7, 10.9, -1, 19.2, 31.6, 3.8) / (-20.32, -5.9, -2, 21.2, 27.5, 2)	1,126,508/ 6,117,966/ 13,914,061/ 98,991,886	16.5/ 59.8/ 125.93/ 861.3

Table 1. Query performance on S3DISFLAT

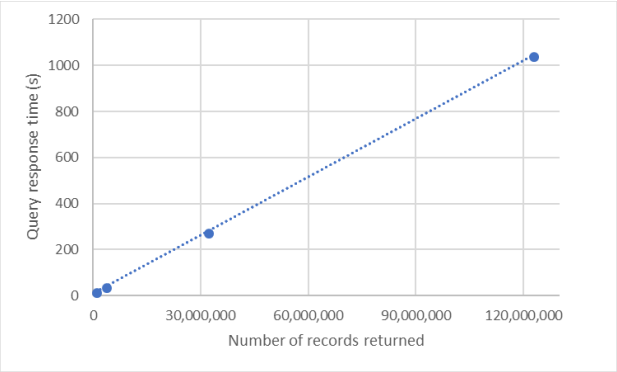

Figure 5. LoD selection (Query 2) at different levels. The tendency shows a linear scalability and Time = 8 x 10^{-6} n + 11.73, where n is the output size.

5.2 Nested table

Oracle supports the nested table, i.e. the data type of a field in a table can be another table. By observing the data, we find that the fields including *RoomType, RoomID, Classification, ObjectID* cause much redundancy. So, a solution is to store these fields into a base table and by adding a *Point_tab* field of which the data type is a table, other information of points could be stored (Figure 6). *Geom_extent* is the spatial bounding box of each object where a 3D R-tree index is created. Spatial queries will be formulated into an intersection operation with the bounding boxes. What is returned is a set of points of whole objects, i.e. an approximate answer. This is reasonable as for visualization, it is vague to show only part of an object. As LoD is another dimension frequently queried, a B-tree index is created on the sub tables. After population, the base table contains 9,833 records. The total storage space is 21,366 MB, out of which 5,246 MB is used to store the B-tree index.

RoomType	*RoomID*	*Class*	*ObjectID*	*Geom_extent*	*Point_tab*
Num	Num	Num	Num	SDO_GEOMETRY	POINTER

LoD	*X*	*Y*	*Z*	*R*	*G*	*B*
Num	Num	Num	Num	Num	Num	Num

Figure 6. Schema of the S3DIS nested table, S3OBJ_NEST, containing a base table and many sub tables.

Query	Number of records returned	Time cost (s)
1. select COUNT(*) from (select u.* from S3OBJ_NEST, table(S3OBJ_NEST.point_tab) u)	1	25.3
2. select u.* from S3OBJ_NEST,table (S3OBJ_NEST.point_tab) u where u.lod>0.99/0.95/0.8/0.5	657,295/ 3,287,515/ 31,730,001/ 122,433,484	5/ 33.27/ 207.14/ 767.4
3. select roomtype, roomid, class, objected, u.* from S3OBJ_NEST,table (S3OBJ_NEST.point_tab) u where u.lod>0.99/0.95/0.8/0.5	657,295/ 3,287,515/ 31,730,001/ 122,433,484	74.66/ 114.86/ 342.6/ 1092.48
4. select u.* from S3OBJ_NEST,table (S3OBJ_NEST.point_tab) u where u.R=223/142/ Roomtype = 2006	188,704/ 2,117,209/ 1,161,301/	14/ 26.7/ 7.54
5. SELECT u.* FROM s3obj_nest s, table(s.point_tab) u WHERE SDO_ANYINTERACT (s.geom_extent, SDO_GEOMETRY(3008,NULL, NULL,SDO_ELEM_INFO_ARRAY(1,1007 ,3), SDO_ORDINATE_ARRAY(12.075,15.267, 0, 20.13,19.3,2))) = 'TRUE' / (-9.072, 17.307, 1, 3.208, 23.557, 2.8) / (4.7, 10.9, -1, 19.2, 31.6, 3.8) / (-20.32, -5.9, -2, 21.2, 27.5, 2)	4,036,284/ 14,195,028/ 22,893,270/ 129,250,908	27.6/ 92.96/ 149/ 836.6

Table 2. Query performance on S3OBJ_NEST

Similar to the flat table approach, the second query, LoD selection presents a linear tendency with the formula: Time = 6 x 10^{-6} n + 7.71. The execution plan shows that only indexed range scan is performed. The slope is smaller as there are only 7 dimensions written out. However, when the *RoomType* and the other attributes in the base table are added (i.e. the 3rd query), the execution plan incorporates nested loops to join dimensions and the time cost becomes larger.

Without index, the selection on the R (color) dimension of Query 4 incurs the full table scan which indeed takes longer than the selection on the indexed LoD dimension. However, the increase of the output scale could make the writing process dominant. Hence, the time cost does not vary much from that of LoD when more points are selected. Besides, if the dimension

selected resides in the base table, the process, in contrast, can be very fast. Only full table scan of the base table (which nearly has no cost) and the writing phase take up the time.

With regard to the spatial selection, i.e. Query 5, the original full table scan for three dimensions turns into a spatial selection using a spatial index. Yet the number of records for searching decrease as well, i.e. from full records into records in the base table. On the whole, the approximate spatial selection on the S3OBJ_NEST table is faster than full table scan selections on the S3SDISFLAT table (Figure 7). However, in all cases, the approximate selections contain more points (Figure 8). On the one hand, this is a result from selections of complete objects; on the other hand, since the bounding box is utilized for intersection, more objects might get involved. Sometimes the outliers are even more than the accurate result itself. Nonetheless, the ratios of outliers (the extra number of points returend divided by the number of points from approximate selection) are: 0.72, 0.57, 0.39 and 0.23, which presents a decreasing trend as the output size grows. Consequently, in the tail of the curve of time cost (Figure 8), we observe that the approximate selection surpasses the accurate selection. It implies that the time spent on full table scan becomes more significant than the additional time to write extra records into the memory.

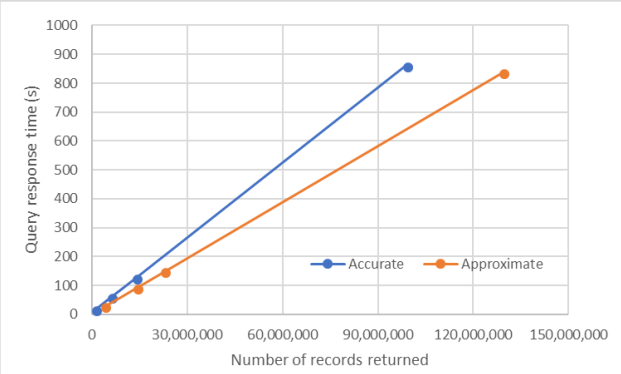

Figure 7. Comparison between the accurate spatial selections on the S3SDISFLAT table and the approximate selections on the S3OBJ_NEST table.

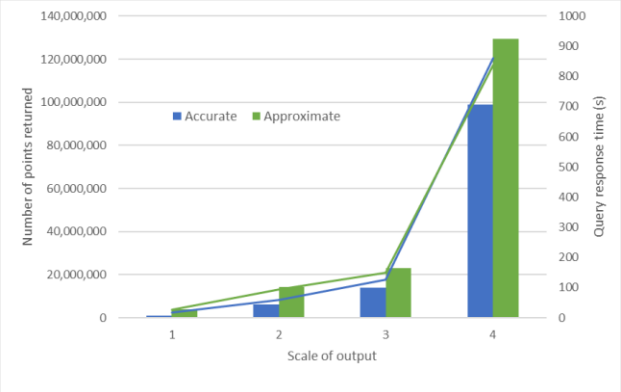

Figure 8. Comparison of the number of points returned from spatial selections on the S3SDISFLAT table presented as bars and the S3OBJ_NEST table, together with the query execution cost expressed using curves.

In summary, the nested table structure complexes the query process to some extent, selections to join the dimensions in the base table and sub tables for instance. The advancement from the B-tree index on LoD is insignificant although it facilitates queries with small output. It could be an alternative to order the data according to the cLoD value and then use ROWNUM to only scan parts of the sub tables to reduce the time cost. The access pattern of the LoD dimension is always to select certain continuous portion of the points. The approximate method presents its advantage when the time cost on the full table scan becomes inevitably significant with large output. Besides, it is crucial to notice that the importance of each dimension is well addressed by such a data structure, i.e. CLASS > SPATIAL > LoD > Normal.

5.3 Table with a spatial index

Compared to the S3DISFLAT, the xyz information is stored in a SDO_GEOMETRY object, i.e. SDO_GEOMETRY(3001, null, MDSYS.SDO_POINT_TYPE(*X*, *Y*, *Z*), null, null) in the S3DISGEOM table. After it, a 3D R-tree is created on the geometry column, while the index creation cost 3 hours and 9 minutes to finish. The other dimensions are normal attributes. The table size is 19,175 MB, occupying 611,142 Oracle logic blocks, while the spatial index size is 22,011 MB.

For the LoD selection and the spatial selection, initial tests present high time cost. Hence, to reduce the workload for writing, only XYZ are selected (Table 3). The linear model of the LoD selection is: Time = 4 x 10^{-5} n + 17.6, which entails the query execution is much slower than that of S3DISFLAT. The main reason is that in the writing part, additional cost is spent on converting binary geometry objects into plain coordinates.

It could be observed that the spatial selections based on SDO_INSIDE fail to fetch the correct number of points. In addition, the process takes huge amount of time to execute. As is indicated in the table, the inefficient spatial computation comprises the body. Besides, previous developers also proposed that an index may not work out if the query was not selective enough (e.g. query data distributed across the whole storage). It could be much slower than a full table scan. The R-tree index created in this solution is very large. With such a cumbersome index, searching the pointers and then extracting the data are less efficient than a direct full table scan.

Query	Number of records returned	Time cost (s)
select COUNT(*) from s3disgeom	1	17.3
select t.x,t.y,t.z from s3disgeom c, table(SDO_UTIL.GETVERTICES(c.point_geom)) t where lod>0.99/0.95/0.8/	657,295/ 3,287,515/ 31,730,001	41.54/ 136.3/ 1165
select t.x, t.y, t.z from s3disgeom c, table (MDSYS.SDO_UTIL.getvertices(c.point_geom)) t where SDO_INSIDE (c.point_geom, SDO_GEOMETRY(3008, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1007,3), SDO_ORDINATE_ARRAY(12.075,15.267,0,20.13,19.3,2))) / (-9.072, 17.307, 1, 3.208, 23.557, 2.8) / (4.7, 10.9, -1, 19.2, 31.6, 3.8)	911,090/ 5,541,547/ 13,163,720	549.87 (538.92)/ 2917.9 (2761)/ 6659.6 (6239.4)

Table 3. Query performance on S3DISGEOM. Time measurements in brackets are roughly the time to perform the SDO_INSIDE computation.

5.4 SDO_PC blocks

The schema is analogous to the nested table (Figure 9). In this solution, the whole dataset is divided into blocks/BLOBs of 10,000 points. So each object might be stored into several blocks. In the block table S3DISBLKTAB, the column

BLK_EXTENT stores the bounding box of each block. On top of it, a 3D R-tree index is created. The solution costs 32.5 hours to finish, which includes the creation and population from a stage table like S3DISFLAT. 32,525 blocks are created in the end. The total storage size is 21,417 MB, which is 1/4 more than that of the nested table. This is mainly because normally in the last block of each object, only a part is used to store the points.

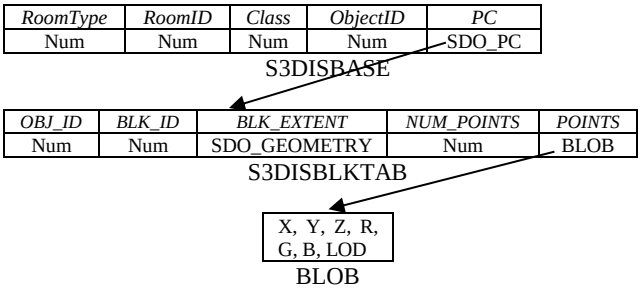

Figure 9. Schema of the SDO_PC solution. It includes the base table S3DISBASE, the block table S3DISBLKTAB, and the BLOBs for real storage of points.

Queries are implemented in SQL scripts (Appendix) with SDO_PC tools which is the standard operation in this solution. The output is an Oracle flat table and it is difficult to store results as an in-memory object.

From Table 4, the LoD selections take enormous amount of time to execute, which is mainly caused by unpacking each block for filtering, as was indicated by van Oosterom et al. (2015). Additional queries are tested to learn the output writing rate, and it turns out that in this case, the writing part is negligible compared to the unpacking process.

Type	Range	Number of records returned	Time cost (s)
LoD query	>0.99 >0.95	- -	>10,000 >10,000
Spatial query	(12.075,15.267,0,20.13,19.3,2) (-9.072, 17.307, 1, 3.208, 23.557, 2.8) (4.7, 10.9, -1, 19.2, 31.6, 3.8) (-20.32, -5.9, -2, 21.2, 27.5, 2)	1,155,519/ 6,140,715/ 13,916,132/ 99,028,009	112.5/ 485.4/ 808.2/ 9,380

Table 4. Query performance of the SDO_PC solution.

The spatial query is more efficient thanks to the spatial index. However, still considerable time is spent on decoding the blocks and the successive scanning to return the precise results. The performance deteriorates severely in the last query. It should be noted that the spatial operators utilized causes inaccurate selections once again.

The schema of this solution is similar to the nested table, but unpacking blocks is needed. It is not preferred considering the LoD selection where each BLOB has to be unpacked for filtering. But this is inevitable as no operators is available for creating a 4D geometry (i.e. *BLK_ENTENT*) used for indexing. A substitute is to utilize *X/Y/LoD* as the bounding box to index. However, Z dimension plays a more important role in the indoor environment than outdoor. Additionally, the solution also has the problem of long loading process.

5.5 Index-Organized Table (IOT)

The Oracle IOT is an index integrated data structure. The primary key and non-key column data stored within the same B-tree structure in leaf nodes. The logical model is however, still a table. Changes to the table, for example, adding new rows, or updating or deleting existing rows, result only in updating the index. It is the fact that most of the time, X/Y/Z/LoD would be selected together. Using pySFC (Meijers, 2017), we encode X/Y/Z/LoD into a Morton key which is the primary key of the IOT. Specifically, X, Y and Z values are multiplied by 1,000 to become integers. The cLoD value are expressed as the ranking number. Then by interleaving the bits of the four dimensions (Psomadaki, 2016), the Morton key could be derived. As the length of bits after encoding could exceed the limit of the NUMBER type, so VARCHAR is used to store the key. Other dimensions keep the same as that of the S3DISFLAT. The total storage size is 21,179 MB.

Type	Range	Search depth/Number of ranges	Number of records returned	Time cost (s)
LoD query	>0.99 >0.95 >0.8 >0.5	57/105 56/51 53/61 51/59	688,127/ 3,342,335/ 31,981,567/ 123,731,967	8.8 + 36.36/ 22.69 + 193/ 217.2 + 2,019/ 775.6 + 7,848.5
Spatial query	(12.075,15.267,0,20.13,19.3,2) (-20.32, -5.9, -2, 21.2, 27.5, 2)	52/261 52/261	273,608,340/ 273,608,340	3001.7 + (>10,000)/ 2950.5 + (>1,0000)

Table 5. Query performance of the IOT solution. The first part of time cost is query time including writing into the memory, while the second part is the decoding cost.

The searching process is based on the key, so the original dimension spans have to be translated into the ranges in the one-dimensional key. Due to the LoD structure embedded in the Morton code (in this case, every four bits represent a level), it is possible to control the computational depth (Table 5) to generate the ranges. It implies that a larger depth corresponds more ranges, i.e. the result selected would be more accurate. In the end, these ranges are inserted into the WHERE clause in a SQL command to execute. Initial tests indicate too many ranges in the WHERE clause would be very inefficient. For all queries, the computation of ranges takes less than 1 second.

In the tests, the writing results contain all dimensions except LoD. As Table 5 shows, the querying process takes less time than that of the S3DISFLAT thanks to the index although more points are returned. This is more obvious when the output is small where writing is not the body. However, more time is spent on decoding. In fact, the decoding process constitutes another linear factor in the performance in addition to writing results. The current implementation of decoding is of complexity O(n) where n is the output size. If we intend to achieve the O(logN) (N refers to the input data size) performance of the B-tree, the n should be below of level of 10^6 if N does not exceed 10^{30}. Otherwise, the decoding process would become dominant. It implies that for large point clouds management, the additional encoding/decoding is not a rationale choice unless a new computational framework would be directly built on the encoded data.

With respect to the spatial queries, due to large time cost, the smallest spatial query and the largest spatial query are used for testing. As can be seen from the table, both queries return the whole dataset as the result. In real experiments, larger search depth is tried, but the query execution takes too much time to return the result. The inefficient spatial querying is radically caused by the value distribution of different dimensions encoded in the Morton key. Basically, the LoD dimension is a long series with distinct values for each point, which makes it

superior in the Morton key. The query then becomes extracting a thin slice from a large 4D cube, which is bound to be inefficient. The continuous LoD dimension may thus be removed from the Morton key encoding, but LoD selection then becomes a problem. Hence, employing the appropriate unit which controls the resolution and range of the data representation in the Morton code is a critical issue in which a proper balance has to be determined.

There are two major advantages in applying an IOT approach. First, it utilized the B tree to organize the whole storage based on a combined column SFC key where several dimensions are involved. Second, SFC is leveraged to group points together so that relationships such as spatial neighbouring could be sustained. So the selection could be more consecutive instead of too many intermittent retravel from the memory. Besides, the structure is more compact compared with the separate table and index approach with less pointers stored. This has direct influence on querying process where the “join” between index and table is more efficient.

6. CONCLUSIONS AND FUTURE WORK

This research is an exploration of a state-of-the art solution to manage large indoor point clouds for visualization in the navigating mode. First the cLoD which could help improve the performance of large data visualization is introduced into data management. Then by establishing a benchmark and utilize various Oracle solutions to test, we gained essential insights which indicate possible directions to further develop a nD PointCloud structure (Liu et al., 2018a):

1. Flat table with B-tree index and plain scanning could be efficient enough for management of large point clouds. A trick is how to realize the priority of different dimensions. Current unsatisfactory performance is mainly caused by the long writing process. In case of visualization, a possible solution is to transfer the data retrieved directly to GPU for rendering. Then smart caching strategies might be applied to buffer a part of the query results into memory in parallel. Besides, the superfluous table scanning could also be improved. An intuitive approach is to avoid unnecessary scanning by harnessing the cLoD dimension of which the access pattern is normally extraction of a continuous portion of the storage. Another option is to develop an approximate accessing scheme.

2. The approximate querying shows its superiority when the output size is large. The nested table approach is based on objects, i.e. semantic information, while the IOT approach controls the accuracy of query through the search depth. In general, the probability density function (Kraska et al.,2017) could be utilized to locate the targets roughly in the storage, for example, development of a nD histogram based on the distribution of points to refine the ranges.

3. Use of spatial data types is not recommended. In the experiments, none of the spatial computation could return accurate results. This might be related to the tolerance settings in the metadata table used for executing queries. Also such an approach relies on spatial indexes which turn out to be very cumbersome. The index size is even larger than original data and it works inefficiently. Lastly, additional encoding/decoding processes are required. The decoding is a linear factor in the time cost, which undermines any gain of time from smart searching of the data. This point is not only confined in spatial data types, but also applies to the Morton coding. In the research, a physical bound of 10^6 for the IOT approach is derived. If the output size exceeds it, the decoding process would become dominant. However, such a threshold could be improved by developing best practices, e.g. implementation inside the DBMS. Then the approach can be acceptable for example, in virtual realize equipment where the memory size is limited. Accordingly, research focus could shift to the LoD computation to determine the most representative points.

With knowledge acquired, the research could be extended to integrate point cloud models of cities where classification information is also involved. As the object is the minimum unit, a semantic index structure could be established, e.g. name of town and then the city’s name. A high dimensional point cloud management could then be developed as a case study.

Apart from novel data management, more aspects could be improved. Current implementation including Python query and encoding/decoding scheme, the SDO_PC and IOT structure is not optimal. Benchmarks applied are simple and are not verified and discussed with industrial professionals. Scalability is not fully incorporated. The computation of LoD still originates from an engineer thinking, instead, visual perception/computer vision should also be taken into account. The argument on whether utilize a cLoD or discrete LoD reserve to be solved.

ACKNOWLEDGEMENTS

The funding of this research comes from the Chinese Scholarship Council (CSC) and Fugro. Thus, their support is greatly acknowledged.

REFERENCES

Armeni, I., Sener, O., Zamir, A.R., Jiang, H., Brilakis, I., Fischer, M. and Savarese, S., 2016. 3D semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1534-1543.

Folk, M., Heber, G., Koziol, Q., Pourmal, E. and Robinson, D., 2011, March. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases* (pp. 36-47). ACM.

Hagedorn, B., Trapp, M., Glander, T. and Döllner, J., 2009, May. Towards an indoor level-of-detail model for route visualization. In *Mobile Data Management: Systems, Services and Middleware, 2009. MDM'09. Tenth International Conference on* (pp. 692-697). IEEE.

ISPRS, 2011. Las 1.4 format specification. Technical report, The American Society for Photogrammetry And Remote Sensing https://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf (13 May 2018).

Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. 2017. The Case for Learned Index Structures. arXiv preprint arXiv:1712.01208.

Liu, H., van Oosterom, P., Meijers, M. and Verbree, E., 2018a. Towards 10^{15}-level point clouds management - a nD PointCloud structure. In *Proceedings 21th AGILE Conference on Geographic Information Science, Lund, Sweden*.

Liu, H., van Oosterom, P., Tijssen, T., Commandeur, T. and Wang, W., 2018b. Managing large multidimensional hydrologic

datasets: A case study comparing NetCDF and SciDB. *Journal of Hydroinformatics*, https://doi.org/10.2166/hydro.2018.136.

Meijers, M., 2017. Functionality to use Space Filling Curves inside PostgreSQL for clustering and indexing. https://bitbucket.org/bmmeijers/pysfc

NIST. 2018. Notice of funding opportunity NIST public safety innovation accelerator program (PSIAP) – point cloud city. Technical report, National Institute of Standards and Technology https://www.nist.gov/sites/default/files/documents/2018/01/05/2018-nist-psiap-pc2_nofo.pdf (13 May 2018)

Psomadaki, S. 2016. *Using a space filling curve for the management of dynamic point cloud data in a relational DBMS*. Master’s thesis, Delft University of Technology, the Netherlands.

Suijker, P.M., Alkemade, I., Kodde, M.P. and Nonhebel, A.E., 2014. User requirements massive point clouds for eSciences (WP1). http://www.pointclouds.nl/docs/User_Requirements%20_MPC.pdf (13 May 2018)

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

APPENDIX

SQL for table creation

Flat table:

```
CREATE TABLE S3DISFLAT (X number, Y number,
Z number, R number, G number, B number,
RoomType number, RoomID number, Class
number, ObjectID number, LoD number);
```

Nested table:

```
CREATE TYPE Point_type as object (X number,
Y number, Z number, R number, G number, B
number, LoD number);

CREATE TYPE Point_table IS TABLE OF
point_type;

CREATE TABLE S3OBJ_NEST (RoomType number,
RoomID number, Class number, ObjectID
number, Geom_extent SDO_GEOMETRY, Point_tab
Point_table) NESTED TABLE Point_tab STORE
AS Points;

CREATE INDEX bb_idx ON S3OBJ_NEST
(Geom_extent) INDEXTYPE IS
MDSYS.SPATIAL_INDEX PARAMETERS
('sdo_indx_dims=3');

CREATE INDEX nested_tab_idx ON Points (LoD);
```

Table with a spatial index:

```
CREATE TABLE S3DISGEOM (Point_geom
SDO_GEOMETRY, R number, G number, B number,
RoomType number, RoomID number, Class
number, ObjectID number, LoD number);

CREATE INDEX point_idx ON S3DISGEOM
(Point_geom) INDEXTYPE IS
MDSYS.SPATIAL_INDEX PARAMETERS
('sdo_indx_dims=3');
```

SDO_PC blocks:

```
CREATE TABLE S3DISBASE (RoomType number,
RoomID number, Class number, ObjectID
number, PC SDO_PC);

CREATE TABLE S3DISBLKTAB AS select * from
mdsys.sdo_pc_blk_table;

CREATE TABLE tmp_heap as (select X as
VAL_D1, Y as VAL_D2, Z as VAL_D3, R as
VAL_D4, G as VAL_D5, B as VAL_D6, LoD as
VAL_D7 from S3DISFLAT;

PC := SDO_PC_PKG.init('S3DISBASE ', 'PC',
'S3DISBLKTAB', 'blk_capacity=10000',
mdsys.sdo_geometry(3008, null, null,
mdsys.sdo_elem_info_array(1,1007,3),mdsys.s
do_ordinate_array(-37.928, -26.078, -2.645,
29.927, 46.056, 6.576)), 0.0001, 7, null);

SDO_PC_PKG.create_pc(PC, 'tmp_heap',  null);
```

IOT:

```
CREATE TABLE S3DISIOT (SFC varchar2(200), R
number, G number, B number, RoomType number,
RoomID number, Class number, ObjectID
number, CONSTRAINT SFC4D_PK PRIMARY KEY
(SFC)) ORGANIZATION INDEX;
```

SDO_PC query script

LoD:

```
INSERT into PC_RES SELECT query_points.X,
query_points.Y, query_points.Z from
table(SDO_PC_PKG.CLIP_PC(each.pc,
SDO_GEOMETRY(3008, null, null,
SDO_ELEM_INFO_ARRAY(1,1007,3),
SDO_ORDINATE_ARRAY(-37.928, -26.078, -2.645,
29.927,
46.056,6.576)),SDO_MBR(SDO_VPOINT_TYPE(0,0,
0,LoDmin), SDO_VPOINT_TYPE
(256,256,256,1)),null,null))
query_blocks,table(SDO_UTIL.GETVERTICES(SDO
_PC_PKG.TO_GEOMETRY(query_blocks.points,
query_blocks.num_points,3,null)))
query_points; (LoDmin equals
0.99/0.95/0.8/0.5)
```

Spatial:

```
INSERT into PC_RES SELECT query_points.X,
query_points.Y, query_points.Z from
table(SDO_PC_PKG.CLIP_PC (PC, SDO_GEOMETRY
(3008, null, null, SDO_ELEM_INFO_ARRAY (1,
1007, 3),
SDO_ORDINATE_ARRAY (Xmin, Ymin, Zmin, Xmax,
Ymax, Zmax)),null,null,null)) query_blocks,
table(SDO_UTIL.GETVERTICES(SDO_PC_PKG.TO_GE
OMETRY (query_blocks.points,
query_blocks.num_points,3,null)))
query_points; (Spatial range are
(12.075,15.267,0,20.13,19.3,2) / (-
9.072,17.307, 1, 3.208, 23.557, 2.8) / (4.7,
10.9, -1, 19.2, 31.6, 3.8)) / (-20.32, -5.9,
-2, 21.2, 27.5, 2)
```