USING A SPACE FILLING CURVE APPROACH FOR THE MANAGEMENT OF DYNAMIC POINT CLOUDS

S. Psomadaki[a]*, P. J. M. van Oosterom[a], T. P. M. Tijssen[a], F. Baart[b]

[a] TU Delft, Faculty of Architecture and the Built Environment, Department OTB, 2600 GA Delft, - S.Psomadaki@student.tudelft.nl, (P.J.M.vanOosterom, T.P.M.Tijssen)@tudelft.nl
[b] Deltares, 2600 MH, Delft, the Netherlands - Fedor.Baart@deltares.nl

KEY WORDS: Point cloud data, Space filling curve, Spatio-temporal data, Benchmark, DBMS

ABSTRACT:

Point cloud usage has increased over the years. The development of low-cost sensors makes it now possible to acquire frequent point cloud measurements on a short time period (day, hour, second). Based on the requirements coming from the coastal monitoring domain, we have developed, implemented and benchmarked a spatio-temporal point cloud data management solution. For this reason, we make use of the flat model approach (one point per row) in an Index Organised Table within a RDBMS and an improved spatio-temporal organisation using a Space Filling Curve approach. Two variants coming from two extremes of the space - time continuum are also taken into account, along with two treatments of the z dimension: as attribute or as part of the space filling curve. Through executing a benchmark we elaborate on the performance -loading and querying time-, and storage required by those different approaches. Finally, we validate the correctness and suitability of our method, through an out-of-the-box way of managing dynamic point clouds.

1. INTRODUCTION

Since the introduction of the LIDAR technology in the 1960s, the volume of point cloud data has seen a rapid increase and it is anticipated that it will continue to increase exponentially in the years to follow. This growth is mainly the result of the developments in the point cloud acquisition technologies, the most important of which are: terrestrial and airborne laser scanning, mobile mapping, multi-beam echo-sound techniques (Otepka et al., 2013).

Over the last years many easy-to-use and inexpensive sensors mounted in mobile devices have become widely available. Examples of these devices are: Microsoft's Kinect sensor (Izadi et al., 2011), Google's Project Tango (Schöps et al., 2015), Structure from Motion (SfM) techniques (Westoby et al., 2012), etc. The advent of these technologies has allowed repeated scans of the same area on a regular basis, leading to massive spatio-temporal point clouds having both very high spatial and temporal resolution. However, the management and querying of these massive point clouds is a challenge (van Oosterom et al., 2015). The reason for this is the generally unstructured nature of the points (compared to raster data) and the multiple attributes that can be attached to them. Depending on the acquisition technique, point clouds can contain: time information, intensity, return number, number of returns, classification, colour etc. These attributes can be present in different combinations making even simple storage and selections non trivial.

In the majority of today's applications point clouds are managed using file solutions. 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). Within these solutions, the work-flow includes reading one or more files, processing the data and writing files back to the user.

The database community, commercial and open source, provides point cloud specific data structures. In particular, Oracle (Spatial and Graph) and PostgreSQL (PostGIS) follow a similar organisation technique for point cloud data. Their storage model is based on the physical reorganisation of the data into groups of spatially close points, called blocks (Ravada et al., 2010; Ramsey, 2014) and provides efficient management and query response times (van Oosterom et al., 2015; Cura et al., 2015). However, the data structures available in the Database Management Systems (DBMS), are not designed for applications with dynamic nature. This means that they consider point clouds as static objects, not including time as part of the organisation. This is a very important limitation as for specific applications time is as selective as the spatial component or needed in integrated space - time selections (change detection). These requirements, suggest that storing time as an attribute does not offer efficient query response times. Finally, the structures do not scale good with the accumulation of time dependent data. With such voluminous data, performance -in terms of loading and query time, as well as storage- is very important.

In this paper we investigate how effective time-varying point-clouds can be stored and queried in a relational DBMS. More specifically, we investigate different options of using a Space Filling Curve (SFC) to capture both time and space in one efficient data-containing index.

This paper is organised as follows: Section 2 gives an overview of the related literature on managing point clouds. Section 3 introduces the methodology used in this paper. Section 4 provides the implementation, followed by Section 5 and 6 where the benchmark description and results are given respectively. The paper ends with Section 7, where conclusions and the future work are discussed.

2. RELATED LITERATURE

2.1 Management of point cloud data in DBMS

Research on the management of point clouds has been ongoing for at least a decade. In the beginning, the already available data types (`POINT`, `MULTIPOINT`) were proposed for the management

*Corresponding author

of point clouds (Wijga-Hoefsloot, 2012; Höfle et al., 2006). Later on, it was argued that these approaches present significant drawbacks in terms of storage overhead, memory intensive operations and difficult updates and insertions (Ott, 2012).

The database community currently provides specific data structures suitable for the the management of point clouds. But apart of the *blocked* model available in Oracle and PostGIS, a second organisation is also possible. This is the *flat model*, where each point is stored separately in one row. This model is easy to be implemented in all database systems (van Oosterom et al., 2015), relational or not (Martinez-Rubi et al., 2014). Compared to each other, the block-based approach provides better scalability, less overhead per point and potentially good compression (van Oosterom et al., 2015) that is directly related to the block size. However, blocks are less efficient in terms of updating, and further insertions of points lead to overlapping blocks. This is not ideal when managing dynamic point clouds, as within the current structures the indexes are used independent from each other and the query optimiser will first filter on space and then on time. On the other hand, the flat model is easier with updates and insertions. An improved organisation of the flat model uses spatial clustering techniques and specifically SFCs. SFCs have the ability to cluster points close in reality, close on the curve. The improved organisation is introduced in van Oosterom et al. (2015) and extended in Martinez-Rubi et al. (2015) for the two spatial dimensions.

2.2 Spatio-temporal point cloud management

The organisation of point clouds has in its majority been focused only on the spatial dimensions of the points. This means that time is stored as an attribute, not taking part in the organisation of the points into blocks or into the SFC. However, point clouds are used for spatio-temporal analysis and therefore, an integrated space and time approach is needed when organising the points. Integrating space with time is, nonetheless, not an easy task. The challenge lies in the different semantics and nature of the two concepts. In the design phase, two aspects should always be taken into account: 1. the resolution of time, meaning how the line of time is partitioned. 2. the granularity of time, meaning at which level of the spatial phenomenon (point data) the time dimension is added. For example, time can be attached to the whole or subset of a dataset, or can be part of each spatial object (point).

Implementations of spatio - temporal point cloud databases can be found in the academia. More specifically, Fox et al. (2013) used a NoSQL database for managing time dependent point data. Their implementation is based on interleaving parts of the geohash, with parts of the string representation of time. A geohash is an implementation of the Morton SFC, ultimately generating a recursive quadtree of the world. The derived string is used as key for indexing the point data lexicographically. One disadvantage of the method is that it is very platform specific as it accommodates the column key requirements of the Accumulo database (key value store). At the same time, however, the system can provide efficient insert and update operations. Tian et al. (2015), following also a clustering approach, interleave the bits of the x,y and time dimensions to derive the Morton key which is later used to create a 3D Morton R-Tree inside a relational database. Their developed prototype gives efficient query response times, also under concurrent queries. However, their approach lacks the ability to further insert new data. Finally, a different approach is followed by Richter and Döllner (2014). Their developed system handles point cloud updates by using an ”incremental database update process”. For this they use change detection techniques to determine which parts of the new point cloud have changed since the previous state. Only the changed parts are then being inserted into the database. Their implementation although has reduced storage requirements and allows efficient change detection from one moment to another, makes it very hard to restore what happened at a specific moment as a number of change entries have to be applied to the initial state.

From the literature it becomes clear that a SFC approach is a logical way to proceed when managing dynamic point clouds. However, different options need to be investigated to find an optimal solution; specifically treating time (and z) as dimensions used in the SFC computation (or not) and the scaling of space and time.

3. METHODOLOGY

As explained in Section 2.1, the flat storage model is a very flexible solution for the management of point clouds, dynamic or not. The method can either be used as a final storage model or as an intermediate stage in order to efficiently create blocks. However, being able to efficiently ask questions to the database is directly proportional to the data structure and access methods used. Therefore, a clustering technique, and more specifically a SFC can be used for the efficient sorting of the points. A SFC has the ability to apply a linear ordering to a multi-dimensional domain. Many SFCs have been developed through the decades, all of which preserve a different degree of proximity in the data. Two very commonly used orderings are the Morton and the Hilbert curve. The curves are respectively shown in Figure 1a and 1b for the 2D case. Both curves have the characteristic of being Quadrant recursive, which indicates that the cells in any sub-quadrant have consecutive SFC values. SFCs, which can be extended to the nD space, have been proven to be very relevant for multidimensional storage (Lawder, 2000). A 3D implementation of the Morton curve is given in Figure 1c. One important advantage of utilising SFCs is that the derived one dimensional values can be indexed using a B-Tree.

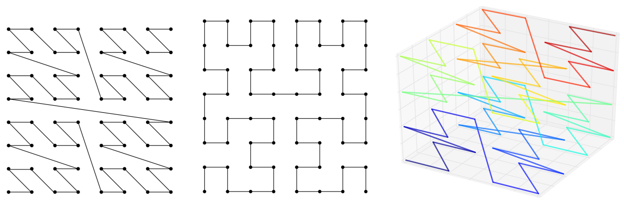

(a) The Morton curve (b) The Hilbert curve (c) The Morton curve in 3D

Figure 1: The Morton and Hilbert space filling curves.

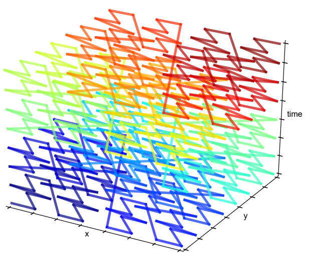
3.1 Storage Model

Diverging from the blocked models as commonly used by the DBMS, we explore the flat model with an improved spatio - temporal organisation. For this we use an integrated space and time approach and a SFC, the Morton curve, for the organisation of the points. Instead of using a (heap) table with a B-Tree index, we use an Index Organised Table (IOT) which avoids storing a large, separate index, thus not requiring to perform a join during query execution (between index and data).

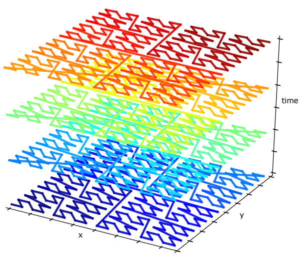
The Morton curve (also called Z-order or N-order curve) is based on interleaving the bits of the binary representation of the n-coordinates. For the SFC calculation, we define the curve for the full resolution of the point cloud domain. This allows us to avoid storing the x, y (,z) and t values, since those can be recovered back from the key. The above storage model leads to significantly less disk space but requires a decoding function that can recover the original dimensions.

One very important aspect when using SFCs is keeping in mind that they are based on hyper cubes and that all dimensions present in the curve should have the same cardinality i.e. be of the same size. For the spatial components that have the same nature, this assumption is not detrimental to assume. However, time has a different nature compared to the spatial components; it is measured in years, months, days, hours, minutes or seconds. Space, on the other hand, is measured in degrees, meters, centimetres or millimetres. The correspondence of those two (the relative scaling) can be considered as the factor of how much time is integrated with how much space. This integration should, nonetheless, be constrained by the fact that additional data need to be added into the database. Therefore, space on the hypercube should be reserved for new data to be added in the future.

Structuring space and time to support dynamic point clouds is not a trivial problem. The reason for this is that two contradictory requirements need to be taken into account; (1) points close in space and time should be stored close together (clustered) for fast spatio - temporal retrieval, but, (2) in a way that already organised data is not affected by new data (to achieve fast loading). The clash of these two requirements takes place when adding new data. The new points, in order to preserve space-time locality, will have to be inserted between the already stored points. As a result, the data already organised will have to be moved which is a very costly operation. For this reason, two integrations of space and time are explored: the **integrated** space and time approach, where space and time have an equal part in the Morton key calculation (Figure 2a) and, the **non - integrated** space and time approach, where time dominates over space and is stored as a separate column (Figure 2b). The two options actually represent two extremes of a continuum which is achieved by appropriately scaling the time dimension relative to space.



(a) Integrated



(b) Non-integrated

Figure 2: The two integrations of space and time.

3.2 Loading procedure

The loading procedure of our methodology is divided into two phases. The data are physically structured according to their position on the space filling curve and organised with a data containing B-Tree. The steps followed are:

1. *Preparation*: the data are read from the files and converted to the Morton keys. This conversion depends on the type of integration of space and time, the dimensions used in the Morton key calculation and the scaling of time. The data are bulk loaded into a normal heap table.
2. *Loading*: The data is read from the heap table, sorted based on the key and stored in the IOT. In this way, the index is created once, which will assure that the data are clustered. An incremental approach can replace this step. Incremental means that the data are added in batches and the index will be reorganised with every batch.

3.3 Query procedure

Since our use case (presented Section 5.1) comes from the coastal monitoring domain, we carried out a research of the most important queries used in those applications (de Boer et al., 2012; Lodder and van Geer, 2012), which are (Figure 3): (1) **Space only queries**, that request all the spatio - temporal objects located in a specific area. (2) **Space-time range queries**, that request all the spatio-temporal objects located in a specific area during a specific time range. (3) **Time only queries**, that request all the spatio-temporal objects existing during a specific time range.

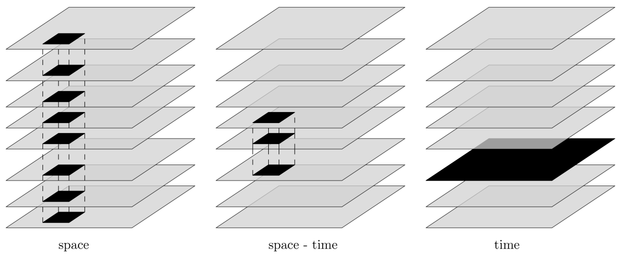
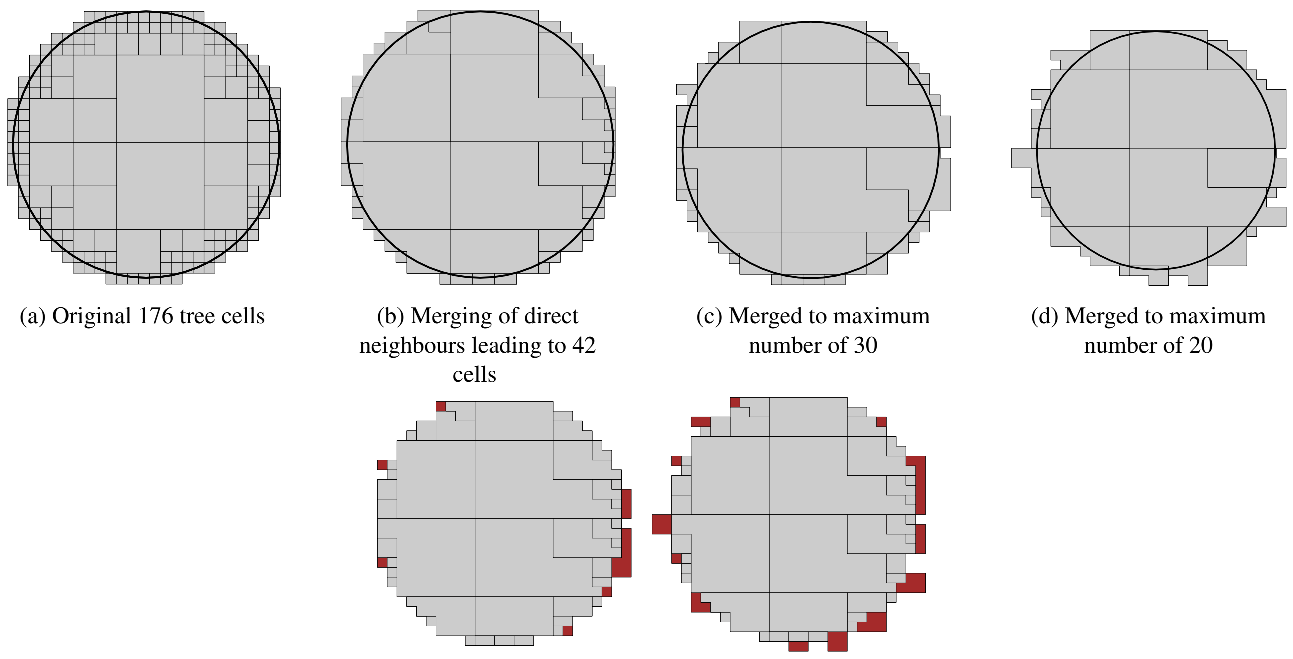


Figure 3: An overview of the important queries

Multidimensional selections using SFCs require a modified query algorithm that takes into account the space filling organisation. 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, within our method, we make use of the relationship between the Morton curve 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 Morton 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 used is as follows:

1. *Filtering*: The 2^n tree cells that intersect with the query region, up to a specific depth, are identified (Figure 4a). Note the mixture of big and small ranges returned, with the smaller ones located mostly near the boundary. The cells are then translated into the equivalent Morton ranges and the neighbouring ranges on the curve are merged (Figure 4b). Note that the direct neighbour merging can either have no effect on the cells or create non-rectangular ones. Unless differently defined, the ranges are further merged in case they exceed a specified maximum amount. 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. The result is an approximation of the query being asked because the ranges are not formed from the finest 2^n tree cells. Two examples with a different degree of merging (30 and 20) are present in Figure 4c and 4d respectively. The number of the tree cells after the direct neighbour merging is 42. With the application of the merging one can observe two things: First, that the 2^n tree approximation is very accurate, especially when compared to the commonly used Minimum Bounding Rectangle. Second, when applying a merging there is a certain loss in the accuracy of the approximation (Figure 4e and 4f) but, also a gain when the number of ranges is a bottleneck.
2. *Decoding and storing*: The previous result is decoded back to the original x, y, z and time dimensions and stored temporarily in a table. This is needed because there are not yet functions inside the database to perform the decoding on-the-fly.
3. *Refinement*: The final query result is obtained by performing a point-in-polygon operation or filtering out time and z.



(a) Original 176 tree cells

(b) Merging of direct neighbours leading to 42 cells

(c) Merged to maximum number of 30

(d) Merged to maximum number of 20

(e) The expansion of the original geometry (case (b)) in red after merging to 30 ranges

(f) The expansion of the original geometry (case (b)) in red after merging to 20 ranges

Figure 4: The different steps in the preparation of the filter step: Tree cell identification, direct neighbour merging and, merging to maximum number. Cases (c) and (d) depict different degrees of merging applied to the tree cells of case (b). The expansion of the area according to the two degrees of merging (30 and 20) is shown in cases (e) and (f) respectively

4. IMPLEMENTATION

4.1 Developed scripts

For the implementation of our methodology we have developed sets of Python scripts that perform the loading and querying procedures according to the specified parameters. The source code can be found at: `https://github.com/stpsomad/DynamicPCDMS`. Selected parts of the loading scripts along with some explanations can be found in the Appendix of this paper.

4.2 Database selection

The system chosen for the validation of the methodology is the Oracle Database. The version that was used for the tests is the Oracle Database 12c Enterprise Edition Release 12.1.0.1.2 - 64 bit Production. The system is chosen because of the availability of the IOT. With an IOT, the data itself is stored in a B-Tree index structure and physically clustered. This means that contrary to the usual way, IOTs do not store the table and the index separately. Another reason for choosing the Oracle database was that the full Morton keys can very easily become larger than 64 bit integers. The Oracle database includes the NUMBER type that can handle numbers up to 38 decimal digits, enough for 128 bit keys.

4.3 The representation of space and time

One of the issues faced when using time in general and inside SFCs specifically, is its unique nature. Time is usually represented with the date format inside the database. However, integers can be sorted more efficiently, a characteristic that is very important for our methodology. Furthermore, SFCs are implemented with integers. Therefore, both space and time need to be converted to and represented by integer values.

For space this issue is solved by applying a linear transformation to the spatial coordinates (translation and scale). In order to represent time as an integer many different ways can be identified that correspond to different time resolutions (seconds, days, years). We can express time simply as an integer of format yyyymmdd for day or yyyy for year resolution. This expression, however, as the resolution becomes finer, leads to time gaps. Another option is to use the Unix time that gives the the number of seconds since 00:00:00, 1/1/1970. This option can be very useful for datasets that are streamed every second, but is very verbose for day or year resolution. For day resolution we can chose to store the days since a specific epoch, e.g. 1/1/1990.

4.4 Hardware and Software

For the tests described in this document we have used a server with the following details: HP DL380p Gen8 server with 2 x 8-core Intel Xeon processors, at 2.9 GHz, 128 GB of main memory, and a RHEL 6 operating system. The disk storage which is directly attached consists of a 400 GB SSD, 5 TB SAS 15K rpm (\work) in RAID 5 configuration, and 2 x 41 TB SATA 7200 rpm in RAID 5 configuration (\pak1 and \pak2 respectively).

Purpose	Tablespace	File system
Data	-	\pak2
Heap table	USERS	\pak1
IOT	INDX	\pak2
DBMS Temporary storage	TEMP	\work
Query table	PCWORK	\work

Table 1: The distribution of files and tables in the available disks according to the specific purpose.

To minimise mixed read/ write operations on the same disk, especially during the loading procedure, we have distributed our data files and database tables over different disks. Within the DBMS this is achieved using different tablespaces. The chosen distribution is available in Table 1. The decision was made as follows: The data are stored in the /pak2 file system. To avoid disk contention, the /pak2 file system (INDX tablespace) should not be used for loading the data into the heap table. Therefore, the USERS tablespace is used (=/pak1). For the creation of the IOT data is read from USERS tablespace, and sorting will take place in the TEMP (=/work) tablespace. To avoid disk contention in the final stages of this phase (the writing of the resulting IOT) the INDX tablespace is used for storing the IOT.

5. BENCHMARK DESIGN

5.1 Loading

In order to test the performance of our storage model, we designed and executed a benchmark. The benchmark is designed to measure the performance in terms of storage space, loading times and query response times. In addition to that, it includes the test datasets used and the description of the queries both in geometry and time.

The data (provided by Deltares) originate from the Sand Engine use case, 21 million cubic metres of sand deposited at the coast of the province of South Holland in the Netherlands. The region has a size of 4.5 x 4.5 km. The purpose is to investigate how nature spreads this amount of sand along the coast as the years go by. For this reason, the area is measured at irregular periods (mostly after the occurrence of storms). So, although it is not measured every single day, the time resolution will be in days in order to offer the best possible organisation. In Table 2 we present the details of the benchmark stages. The data itself is available as a set of LAZ files, with approximately 100,000 points per file. It is important to mention that the spatial extent remains more or less the same, while the time extent is increasing. In addition to that, years 2000 to 2011 are artificially created from the subsequent measurements. With this type of dataset we aim to compare the scaling in size of the stored data and the effect of adding new temporal data in batches between the benchmarks, as well as, the query response times.

Benchmark	Points	Days	Size (MB)	Description	No. of files
Small	18M	230	346	2000 to 2002	230
Medium	44M	554	833	2000 to 2006	554
Large	74M	931	1409	2000 to 2015	931

Table 2: Benchmark stages of the Sand Engine use case. The size represents the size of the LAZ files in the filesystem.

In addition to comparing our storage model between the three benchmark sizes, the two integrations of space and time have to be compared with each other. In addition, for both the integrated and non-integrated approach, the z dimension can be part of the SFC value (z added) or not (z attribute). This leads to 4 different loading options, for each one of which the benchmark is repeated as mentioned previously. Finally, as mentioned in Section 3.1, SFCs are defined on hypercubes and thus the relative scaling between the space and time dimensions needs to be defined. Scaling, however, only makes sense for the integrated approach. In our implemented system, the user can choose to implement different degrees of integration, from a complete to a less deep integration.

All the tests are carried out in order to gain insight into which organisation is the most optimal. The notation used throughout the tables in the next subsection is: **xy** for the non-integrated with z as attribute, **xyz** for the non-integrated with z in key, **xyt** for the integrated with z as attribute, **xyzt** for the integrated with z in key.

5.2 Queries

The queries which are executed are described in detail in Table 3. *Type* is the type of query as defined in Section 3.3. *Start* and *End* are respectively the start and end date requested for retrieval. The *time type*, indicates whether the previously mentioned start and end date are continuous (i.e. between start date and end date) or discrete (i.e. only start date and end date). Finally, the *area* gives an indication of the space covered. The spatial representation of the queries used within our benchmark is shown in Figure 5. For testing purposes we query areas of different geometry like rectangle, polygon, line with buffer and point with buffer. We also test different sizes of the those geometries. This will give us an insight as to whether certain geometries behave differently using the same storage model.

ID	Type	Start	End	Time type	Description	Area (km^2)
ST-A	s-t	03/01/00	28/01/00	d	Large axis - aligned rectangle	0.44
ST-B	s-t	10/11/01	-	d	Large Polygon	0.46
ST-C	s-t	01/11/00	15/11/00	c	Medium, complex polygon	0.16
ST-D	s-t	01/08/01	31/08/01	c	Medium polygon	0.04
ST-E	s-t	01/08/01	31/08/01	c	Line with buffer 5 m	0.015
ST-F	s-t	01/01/02	15/01/02	c	Large Line with buffer 5 m	0.02
T-A	t	25/10/02	26/10/02	c	-	20.25
T-B	t	02/09/02	05/09/02	c	-	20.25
S-A	s	-	-	-	Small Polygon	0.04
S-B	s	-	-	-	Small polygon	0.002
S-C	s	-	-	-	Point buffer of 50 m	0.008
S-D	s	-	-	-	Diagonal line with buffer 5 m	0.009

Table 3: The description of the queries.
In *type*: *s-t* stands for space - time, *t* for time and *s* for space. In *time type*: *c* stands for continuous and *d* for discrete.

6. BENCHMARK RESULTS

In this section we present the results of loading the datasets and performing the above-mentioned queries according to the three benchmark stages, two integrations of space and time and two treatments of z. Space is encoded in mm and time in days since 1/1/1990. It must be noted that, when testing the performance of the various scalings of time for the integrated approach, we realised that the scaling of 10,000 gave the best performance for this specific use case. Such a scaling means that for a certain day, the area grouped close in disk is 10m by 10m. In the following tests only the results of this scaling are presented.

6.1 Loading results

During the benchmark, the files are processed one by one both for their conversion to the Morton key and the loading to the heap. To take into consideration the growing nature of our scenario, the medium and large benchmarks do not include a fresh reloading of the previous stage, but the new data is added to the already stored points. For our benchmark execution each approach is tested separately from the others (until both loading and querying are completed).

In Table 4 the loading times are presented. No parallel processing is present in any of the steps. From the table it is easy to see that, the conversion to the Morton key (SFC prep.) is in general the most expensive step in the procedure. The reason for this is

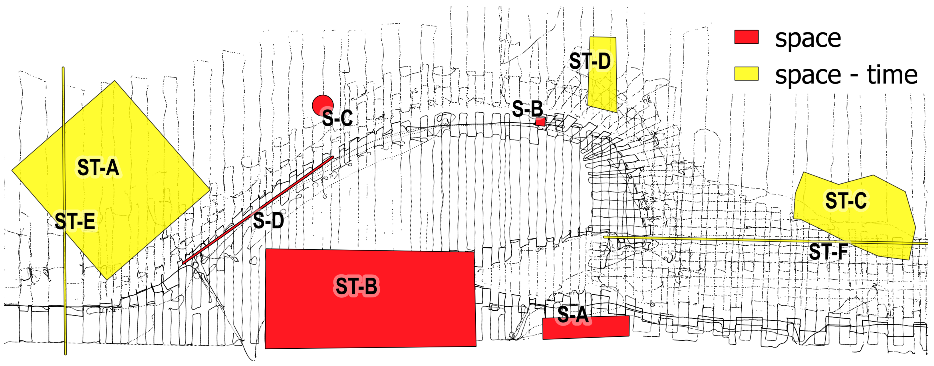


Figure 5: The spatial representation of the queries used within the benchmark

Approach	Time (s)			Size (GB)	Points		Points per sec.	
	SFC prep.	Load heap	Load IOT		Heap	IOT	Heap	IOT
xy - small	105.43	11.79	13.60	0.5	18,147,709	18,147,709	1,539,024	1,334,390
xy - medium	145.14	16.56	49.65	1,1	25,561,106	43,708,815	1,543,433	880,339
xy - large	167.75	19.72	78.00	1,9	30,205,111	73,913,926	1,531,699	947,614
xyz - small	352.37	9.91	10.5	0.4	18,147,709	18,147,709	1,830,384	1,728,353
xyz - medium	498.79	14.24	34.07	0.9	25,561,106	43,708,815	1,794,832	1,282,912
xyz - large	590.00	16.77	61.71	1,5	30,205,111	73,913,926	1,801,161	1,197,763
xyt - small	349.68	11.79	13.09	0.5	18,147,709	18,147,709	1,539,024	1,386,380
xyt - medium	492.29	16.56	40.39	1.1	25,561,106	43,708,815	1,543,433	1,082,169
xyt - large	594.10	19.72	74.11	1.9	30,205,111	73,913,926	1,531,699	997,354
xyzt - small	435.48	11.79	10.78	0.4	18,147,709	18,147,709	1,539,024	1,683,461
xyzt - medium	604.27	16.56	33.21	0.9	25,561,106	43,708,815	1,543,433	1,316,134
xyzt - large	722.08	19.72	57.96	1.5	30,205,111	73,913,926	1,531,699	1,275,258

Table 4: The incremental loading times for the two integrations of space and time and the two treatments of z.

that the code that we use so far is non-optimised Python code. Also, it is easy to see that from approach to approach (xy $\rightarrow$ xyz or xyt $\rightarrow$ xyzt) the conversion gets more costly. This can be explained because the complexity of the algorithm increases with the addition of more dimensions. The loading inside the heap tables is more or less in the same magnitude for the four approaches. As for the loading in the IOT, the treatment of z as an attribute seems to be more expensive in terms of time. This may be because one more column needs to be organised (compared to the treatment of z as part of the Morton key). Finally, comparing the storage requirements of the different approaches, it is easy to see that the treatment of z as an attribute requires in general more space. However, the differences are not big. Also, using a separate attribute for the time or integrating it in the key appears not to influence the storage.

6.2 Query results

The queries introduced in Section 5.2 are executed for each of the four storage organisations for all three data sizes (12 combinations). For all the queries we run both cold and hot runs. In contrast to the cold run, the hot run means that the query has already been executed and caching takes place. Within our benchmark each query is repeated 6 times before moving to the next one. The execution order of the queries (as presented in Table 3) is as follows: ST-A, ST-B, S-A, ST-C, T-A, ST-D, S-B, S-C, ST-E, T-B, S-D, ST-F. For the results presented here we ignore the most and least expensive response and calculate the average from the rest. As a result, the presented number correspond to hot runs.

Only the first fetching (filtering) of the data (along with the number of points in the refinement stage and the percentage of the extra points obtained between the two querying steps) is given in the following table. This step is the most crucial because it is directly related to the depth of the 2^n tree and the maximum number of ranges specified (degree of merging). The rest of the steps can be optimised further and are, therefore, of secondary importance. Finally, it is very important to mention that two different methods of posing the queries are used between the two integrations. In the integrated approach we load the ranges into a separate IOT and perform a join to fetch the data. This method, however, does not provide efficient response times for the non-integrated approach (where the index is composed out of 2 columns) and as a result the keys are specified in the WHERE predicate (see A.2.2). But because there is a limit to the number of ranges we can ask when using a SQL query, a limit of 200 ranges is applied. This is not the case for the integrated approach where in theory there are no limits when performing joins. Nevertheless, a number of 1 million ranges is set as maximum for practical reasons.

The query results (filter step) for all integrations of space and time, treatments of z and benchmark stages can be found in Table 5. All queries are executed using only one process (no parallel). This is based upon the observation that, in certain test cases, although parallelisation in Oracle was enabled, during query execution one core was actually being used. Only queries that do not use the primary key ([time,]Morton) to fetch the data i.e. space queries in the non-integrated approach seem to enable the parallel option. However, for consistency within our results, no parallelisation is enabled.

By first observing space - time and time queries, we can conclude that:

Approach	id	Filter step (s)			% extra points			Final Points		
		small	medium	large	small	medium	large	small	medium	large
xy	ST-A	0.33	0.33	0.33	12	12	12	3927	3927	3927
	ST-B	0.22	0.22	0.21	14	14	14	4237	4237	4237
	ST-C	0.92	0.91	0.93	12	12	12	2812	2812	2812
	ST-D	3.86	3.92	4.03	3	3	3	2185	2185	2185
	ST-E	2.94	3.09	3.38	38	38	38	380	380	380
	ST-F	1.34	1.57	1.40	137	137	137	327	327	327
	T-A	0.52	0.51	0.51	0	0	0	78902	78902	78902
	T-B	1.01	1.01	1.04	0	0	0	157806	157806	157806
	S-A	30.66	74.11	128.02	30	29	28	86017	231283	509964
	S-B	50.98	122.38	209.89	7	9	8	2788	8934	23949
	S-C	24.72	64.30	108.58	11	11	11	11501	32983	54111
	S-D	109.10	260.34	444.13	62	62	61	11933	33494	90670
xyz	ST-A	1.07	1.08	1.12	19	19	19	3927	3927	3927
	ST-B	0.65	0.64	0.66	17	17	17	4237	4237	4237
	ST-C	1.75	1.86	1.83	40	40	40	2812	2812	2812
	ST-D	3.93	3.97	4.30	24	24	24	2185	2185	2185
	ST-E	3.39	3.42	3.41	251	251	251	380	380	380
	ST-F	1.75	1.94	1.93	497	497	497	327	327	327
	T-A	0.33	0.32	0.31	0	0	0	78902	78902	78902
	T-B	0.62	0.63	0.62	0	0	0	157806	157806	157806
	S-A	122.95	292.07	503.75	30	29	28	86017	231283	509964
	S-B	114.90	291.73	481.49	32	37	33	2788	8934	23949
	S-C	105.02	260.18	422.42	22	22	22	11501	32983	54111
	S-D	103.44	244.45	417.43	237	237	218	11933	33494	90670
xyt	ST-A	0.05	0.05	0.06	2	2	2	3927	3927	3927
	ST-B	0.13	0.12	0.13	1	1	1	4237	4237	4237
	ST-C	0.04	0.04	0.04	12	12	12	2812	2812	2812
	ST-D	0.03	0.03	0.03	6	6	6	2185	2185	2185
	ST-E	0.04	0.04	0.04	27	27	27	380	380	380
	ST-F	0.04	0.04	0.04	35	35	35	327	327	327
	T-A	0.59	0.60	0.59	0	0	0	78902	78902	78902
	T-B	0.96	0.96	0.95	0	0	0	157806	157806	157806
	S-A	0.68	1.73	3.76	15	15	28	86017	231283	509964
	S-B	0.10	0.26	0.36	14	16	26	2788	8934	23949
	S-C	0.16	0.39	0.53	11	11	22	11501	32983	54111
	S-D	0.42	1.09	1.75	46	47	97	11933	33494	90670
xyzt	ST-A	0.11	0.11	0.11	2	2	2	3927	3927	3927
	ST-B	0.08	0.08	0.08	4	4	4	4237	4237	4237
	ST-C	0.06	0.06	0.06	12	12	12	2812	2812	2812
	ST-D	0.12	0.13	0.12	6	6	6	2185	2185	2185
	ST-E	0.13	0.13	0.13	50	50	50	380	380	380
	ST-F	0.11	0.11	0.11	67	67	67	327	327	327
	T-A	0.42	0.42	0.42	0	0	0	78902	78902	78902
	T-B	0.57	0.57	0.58	0	0	0	157806	157806	157806
	S-A	0.47	1.17	2.61	30	29	58	86017	231283	509964
	S-B	0.08	0.21	0.48	64	72	68	2788	8934	23949
	S-C	0.17	0.23	0.41	22	45	46	11501	32983	54111
	S-D	0.63	0.59	1.41	100	190	184	11933	33494	90670

Table 5: Query response times, the percentage of false hits compared to the actual number of points and the points returned by the queries.

- In general rectangles and polygons have faster response times in the non-integrated approach.
- In the non-integrated approach there are some differences in the response times between the two treatments of z, especially for rectangles and polygons. Adding z in the key without using it slows down the query execution.
- In the integrated approach having z as an attribute or as part of the key does not have a big effect on the execution time of the data.
- Response times of time queries are of the same magnitude per treatment of z.
- Clearly the results present good scalability (constant response times) for the benchmark stages used, since doubling the size of the data does not affect the query execution time. However, we must keep in mind that the specific use case is not massive and therefore not a good indicator for scalability. For this reason, we also tested our proposed methodology with a dataset of 2 billion points. The results (not presented here, but in a soon to be published MSc thesis) confirm the constant scalability of the method.
- Comparing the % of extra points received from the filtering step between the four approaches, we can observe that line queries receive the most extra points. Further, adding z in the key comes at the cost of more extra points. This can be solved by moving deeper in the 2^n tree, which requires a more dynamic algorithm for identifying the maximum depth. Clearly the non-integrated approach presents the largest amount of extra points mostly because of the way that the query is posed (WHERE clause with 200 maximum ranges).

Moving on to space queries, it is important to specify that they are different from space - time and time queries in that the number of points is increasing between the benchmarks, simply because more data is added. From the results we can observe that space queries perform better in the integrated approach while the worst case is for the non-integrated when z is part of the key. Comparing the percentage of extra points we can realise that there is not a distinct pattern (i.e. line buffers do not necessarily give more extra points). However, in the integrated approach and specifically the large benchmark the number of extra points doubles. Although not present in this table, the number of ranges with which the join is performed is less than in the small and medium benchmark. Again, this can be resolved by developing an algorithm that identifies the maximum depth in a more dynamic way.

During our tests we have also investigated the effect of the parameters: (1) depth of the tree and (2) the maximum number of ranges used. For this we present Figure 6 which shows the effect of going deeper in the 2^n tree on the number of extra points received for the three different types of queries in the integrated approach. It is clear that using more ranges, significantly decreases the number of extra points.

Finally, to show the reason why we limit to 200 ranges in the WHERE clause of the non-integrated approach, we present Table 6. For this, we use the same ranges in order to derive a maximum of 10, 100, 1000 and 10,000 for posing the SQL statement. There it is clear that adding more ranges in the SQL statement significantly slows down the fetching of the data, especially for space queries. However, increasing the number of ranges decreases the percentage of extra points received during the filtering phase. To have a balance between time and extra points, the maximum number of 200 is chosen.

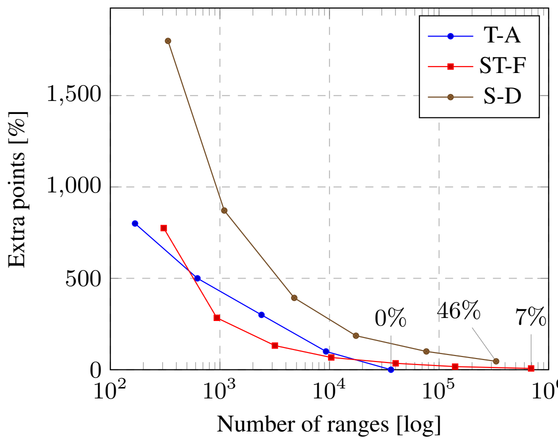


Figure 6: The effect of increasing the depth of the tree on the percentage (%) of extra points obtained during the filtering phase of the integrated approach. The x axis is logarithmic.

Query	Maximum ranges	Actual ranges	fetching [s]	extra [%]
Space - time (ST-D)	10	10306	0.08	4636%
	100	10306	0.10	384%
	1000	10306	20.30	18%
	10000	10306	69.38	9%
Space (S-B)	10	7458	3.65	4943%
	100	7458	34.37	458%
	1000	7458	390.25	54%
	10000	7458	1131.95	28%

Table 6: Using different magnitudes of maximum ranges in the WHERE statement of the non-integrated approach.

6.3 Validation and comparison

To have a kind of validation that our implemented prototype returns the right amount of points and for comparison purposes, the *out-of-the-box approach* of using Oracle spatial and date data types is implemented. For this we use a 3-D SDO_POINT and a 2D R-Tree for fast spatial access. To, also, be able to have fast access in the time dimension, a B-Tree index is built on the time column. The benchmark defined before, is executed for this case as well. An overview of the SQL codes used can be found in A.3 and A.4.

The results of the loading procedure are presented in Table 7 in terms of time and, Table 8 in terms of storage requirements. The same incremental loading as before is used. By comparing them with any of the proposed storage models, we can see that the total execution time of the loading procedure in the validation case is 3 to 6 time more expensive. The process is mostly affected by the R-Tree generation, while building the B-Tree index is the least expensive operation. Moving to the storage requirements, the Oracle spatial approach requires 5 times more space when compared to the proposed storage model with the highest storage requirements.

The query procedure is executed with the same configurations as the proposed methodology. Each query is executed 6 times and the results presented in Table 9 are calculated by excluding the most and least expensive response times from the average. Because of the existence of the R-Tree index, the Oracle database internally follows a two step approach during query execution. However, contrary to the implemented methodology, this two step

Stage	Time (s)				Points in table
	Preparation	Load	R-Tree	B-Tree	
small	31.04	244.27	891.99	15.37	18,147,709
medium	45.07	337.68	2273.38	31.69	43,708,815
large	51.48	400.27	4099.36	73.72	73,913,926

Table 7: The incremental loading times of the validation. Preparation refers to the reading of the LAZ files and their transformation to suitable representations for loading.

Stage	Size (GB)				Points in table
	Table	R-Tree	B-Tree	Total	
small	0.9	1	0.4	2.4	18,147,709
medium	2.3	2.5	0.9	5.8	43,708,815
large	4.0	4.3	1.5	9.8	73,913,926

Table 8: The storage requirements of the validation.

process is transparent to the user. For this reason, the results presented in this Table 9 represent the **total** query execution time (both filter and refinement step). Note that our implemented prototype studies only the fetching of the filter step. With a close inspection we can confirm that our implemented prototype is indeed retuning the correct amount of points. In addition to that, our implemented prototype gives more expensive response times, when considering all steps in the query procedure (not shown here). This has to do with the type of programming language used (Python) for such intensive operations and the movement of data between the application and the database. Both parts can be considerably be improved (see Future work). However, this "naive" approach does not provide constant execution times between the benchmark stages, mostly because of the presence of two indexes and the nature of the 2D R-Tree. This is a crucial observation that makes the method not suitable for managing dynamic point clouds.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this paper we have presented the design and execution of a benchmark appropriate for the data management of dynamic point clouds. We have investigated two integrations of space and time, that are essentially two extremes of the space - time continuum. Within this, we have also tested two treatments of the z dimension: using it as an attribute, or as part of the SFC calculation. We have, also, validated and compared our method with the out-of-the-box approach of using POINT and date data types. The ultimate goal is to investigate the most appropriate structure for managing time evolving point clouds. For this we have considered a use case coming from coastal monitoring domain. All the developed code can be found at `https://github.com/stpsomad/DynamicPCDMS`.

The main findings from our work is that the integrated approach has in general better query response times compared to the non-integrated. Both treatments of z are also appropriate for the specific use case. Further, in the non-integrated approach having z as part of the key significantly slows down query execution and increases the number of extra points received. A key aspect in our solution is the use of the IOT. This storage structure significantly reduces I/O operations as the data are contained in the index (compact and no effort/time to combine index and data from table when querying). Our final conclusion is that as the flat table model is indeed a very flexible solution for managing dynamic point clouds.

id	Total time (s)			Final Points		
	small	medium	large	small	medium	large
ST-A	0.52	1.28	1.89	3927	3927	3927
ST-B	0.83	2.08	3.85	4237	4237	4237
ST-C	0.36	0.72	2.52	2812	2812	2812
ST-D	0.33	0.42	1.00	2185	2185	2185
ST-E	0.29	0.32	0.41	380	380	380
ST-F	0.21	0.28	0.79	327	327	327
T-A	0.19	1.97	0.19	78902	78902	78902
T-B	0.31	1.97	0.31	157806	157806	157806
S-A	0.52	1.19	2.03	86017	231283	509964
S-B	0.11	0.16	0.24	2788	8934	23949
S-C	0.16	0.28	0.39	11501	32983	54111
S-D	0.17	0.32	0.69	11933	33494	90670

Table 9: The query response times and the number of points returned by the queries of the validation benchmark.

7.2 Future work

For the future, several issues need to be investigated and addressed. These include:

- implementing functionality inside the database (encoding, decoding SFC, range generation) or using a compiled language (e.g. `https://github.com/kwan2004/SFCLib`). The former would minimise the data movement now taking place between the database and the application during query execution. With the latter C++ library our preliminary results for the loading phase, show significant improvement (6 times faster) during the SFC preparation phase of the xyzt case (see Table 10).

Approach	SFC prep. [s]
mini	68.69
medium	95.27
full	110.63

Table 10: Using C++ code for the SFC transformation

- investigating parallel processing in all of the steps.
- using even more massive point cloud data for executing benchmarks.
- investigating the value of higher dimensional SFC keys, meaning what is the benefit of including more dimensions in the key e.g. Level of Detail, colour etc.
- investigating delta (change detection) queries. Delta queries are very important for coastal applications monitoring change.
- using a different SFC. For this, comparisons between the Hilbert and Morton curve are the most appropriate, since the former is considered to have higher clustering capabilities. Investigating the number of ranges during the query process would give insight into this, as well as to whether there is a price to pay for the better clustering during encoding or decoding.
- improving the refinement step. Although the refinement stage is not examined within this paper, we believe that it would be significantly aided by following a different approach after the filter stage. The approach includes having two sets of ranges: 1. completely inside the query geometry, that are

for sure part of the final points and do not need to be further refined (Figure 7, white cells) and, 2. ranges partly inside the query geometry that need to go to the refinement stage (Figure 7, grey cells).

- investigating the creation of blocks using the same space and time integrations will allow more efficient storage and compression. Within this, researching the degree of overlapping/ non-overlapping blocks and the percentage of full/ underfull blocks when dealing with time evolving point clouds would add insight to to the current point cloud data management storage models.

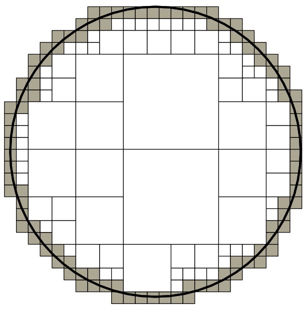


Figure 7: Separating internal ranges and ranges on boundary. White cells are inside the query region. Grey cells are partially inside the query region.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their constructive comments. Also, Guan Xuefeng for providing us his developed C++ code to perform preliminary tests.

References

Cura, R., Perret, J. and Paparoditis, N., 2015. Point Cloud Server (PCS) : Point cloud in-base management and processing. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci. II-3/W5, pp. 531–539.

de Boer, G. J., Baart, F., Bruens, A., Damsma, T., van Geer, P., Grasmeijer, B., den Heijer, K. and van Koningsveld, M., 2012. OpenEarth: using google earth as outreach for NCK's data. In: NCK-days 2012 : Crossing borders in coastal research : jubilee conference proceedings, University Library/University of Twente.

Fox, A., Eichelberger, C., Hughes, J. and Lyon, S., 2013. Spatio-temporal indexing in non-relational distributed databases. In: Big Data, 2013 IEEE International Conference on, IEEE, pp. 291–299.

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

Höfle, B., Rutzinger, M., Geist, T. and Stötter, J., 2006. Using airborne laser scanning data in urban data management-set up of a flexible information system with open source components. In: Proceedings of UDMS, Vol. 2006, p. 25th.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A. et al., 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In: Proceedings of the 24th annual ACM symposium on User interface software and technology, ACM, pp. 559–568.

Lawder, J., 2000. The application of space-filling curves to the storage and retrieval of multi-dimensional data.

Lodder, Q. and van Geer, P., 2012. MorphAn: A new software tool to assess sandy coasts. In: NCK-days 2012 : Crossing borders in coastal research : jubilee conference proceedings, University Library/University of Twente.

Martinez-Rubi, O., Kersten, M., Goncalves, R. and Ivanova, M., 2014. A column-store meets the point clouds. FOSS4G-Europe Academic Track.

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

Otepka, J., Ghuffar, S., Waldhauser, C., Hochreiter, R. and Pfeifer, N., 2013. Georeferenced point clouds: A survey of features and point cloud management. ISPRS International Journal of Geo-Information 2(4), pp. 1038–1065.

Ott, M., 2012. Towards storing point clouds in PostgreSQL. HSR Hochschule für Technik Rapperswil, Rapperswil, Switzerland.

Ramsey, P., 2014. A PostgreSQL extension for storing point cloud (LIDAR) data.

Ravada, S., Horhammer, M. and Baris, M. K., 2010. Point cloud: Storage, loading, and visualization.

Richter, R. and Döllner, J., 2014. Concepts and techniques for integration, analysis and visualization of massive 3d point clouds. Computers, Environment and Urban Systems 45, pp. 114–124.

Schöps, T., Sattler, T., Häne, C. and Pollefeys, M., 2015. 3D modeling on the go: Interactive 3D reconstruction of large-scale scenes on mobile devices.

Tian, Y., Ji, Y. and Scholer, J., 2015. A prototype spatio-temporal database built on top of relational database. In: 2015 12th International Conference on Information Technology - New Generations, Institute of Electrical & Electronics Engineers (IEEE).

van Oosterom, P. and Vijlbrief, T., 1996. The spatial location code. In: Proceedings of the 7th international symposium on spatial data handling, Delft, The Netherlands.

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

Westoby, M., Brasington, J., Glasser, N., Hambrey, M. and Reynolds, J., 2012. structure-from-motionphotogrammetry: A low-cost, effective tool for geoscience applications. Geomorphology 179, pp. 300–314.

Wijga-Hoefsloot, M., 2012. Point clouds in a database: Data management within an engineering company. Master's thesis, TU Delft, Delft University of Technology.

A. APPENDIX

A.1 Loading scripts

The example SQL statements below are given for the non - integrated and integrated approaches (respectively) for the treatment of z as an attribute. For the full code, the reader is referred to our on-line code. The setup for the treatment of z as key is different only in the absence of the z column. Starting with the *preparation* phase, we create the heap table where the data will be stored temporarily.

```
CREATE TABLE temp_xy (
time NUMBER, morton NUMBER, z NUMBER);

CREATE TABLE temp_xyt (morton NUMBER, z NUMBER);
```

For the bulk loading of the data we use the SQLLDR utility of the Oracle database. SQLLDR requires the creation of a set of files, including the control file which specifies how to load the data.

```
load data append into table temp_xy
fields terminated by ','
(time integer external, morton integer external,
z float external)

load data append into table temp_xyt
fields terminated by ','
(morton integer external, z float external)
```

The conversion from the files to the Morton keys is initialised by our developed Morton converter. The conversion is then pipelined with the SQLLDR utility (command line).

```
mortonconverter |
sqlldr [user]/[password]@//[host]:[port]/[DB]
direct=true control=con.ctl data= \"-\"
bad=bd.bad log=lg.log
```

After the data have been inserted, unsorted, into the table, it is possible to proceed to the *loading* phase. The heap table is used to populate the IOT using the following command:

```
CREATE TABLE IOT_xy
(time, morton z, CONSTRAINT IOT_xy_PK
PRIMARY KEY (time, morton)) ORGANIZATION INDEX
AS SELECT time, morton, z FROM temp_xy;

CREATE TABLE IOT_xyt
(morton z, CONSTRAINT IOT_xyt_PK
PRIMARY KEY (morton)) ORGANIZATION INDEX
AS SELECT morton, z FROM temp_xyt;
```

In case more data need to be added into the database, the new data are inserted into a heap table and then the two sources (old and new data) are combined together:

```
CREATE TABLE IOT_new_xy
(time, morton z, CONSTRAINT IOT_new_xy_PK
PRIMARY KEY (time, morton)) ORGANIZATION INDEX
AS SELECT time, morton, z FROM IOT_xy
UNION ALL
SELECT time, morton, z FROM temp_xy;

CREATE TABLE IOT_new_xyt
(morton z, CONSTRAINT IOT_new_xyt_PK
PRIMARY KEY (morton)) ORGANIZATION INDEX
AS SELECT morton, z FROM IOT_xyt
UNION ALL
SELECT morton, z FROM temp_xyt;
```

A.2 Query scripts

Within our python scripts we have implemented a work-flow that maps the spatio-temporal queries to Morton ranges, time or height predicates. The procedure depends on the specified integration and type of query. Again the SQL codes are given for the non-integrated and integrated approaches (respectively) for the treatment of z as attribute. The setup for the treatment of z as a key is sightly different.

The queries are loaded into a table called QUERIES. When a certain query needs to be executed the implemented scripts read the required information from the QUERIES table. The table contains the following information: 1. query ID, 2. dataset, 3. type of query, 4. geometry, 5. start and end date, 6. minimum and maximum height (if any).

A.2.1 Time only queries in the non-integrated approach do not require the identification of Morton keys. This is because time is stored as a separate column. The first filtering gives directly the refined points. This query requests points between two moments in time. These are then decoded back to their original dimensions and stored in a table.

```
SELECT time, morton, z FROM IOT_xy
WHERE (time BETWEEN 4681 AND 4682);
```

In the integrated approach time queries follow the two step query process. First, the Morton ranges are loaded into an IOT named RANGES. The data table and the RANGES table are joined. This concludes the filtering step. The data are decoded to their original coordinates and stored into a table. This tables is then used to proceed to the refinement step, by imposing predicates on the time dimension.

```
--RANGES table definition
CREATE TABLE RANGES(low NUMBER, upper NUMBER,
CONSTRAINT RANGES_iot_idx PRIMARY KEY (low))
ORGANIZATION INDEX;

-- Join operation
SELECT /*+ USE_NL (t r)*/ t.morton, t.z
FROM IOT_xyt t, RANGES r
WHERE (t.morton BETWEEN r.low AND r.upper);

-- Table storing the decoded points
CREATE TABLE decoded (
time DATE, X NUMBER, Y NUMBER, Z NUMBER);

-- Refinement Stage
CREATE TABLE result AS SELECT *
FROM (SELECT X, Y, Z, time FROM decoded)
WHERE (TIME BETWEEN
TO_DATE('2002/10/25', 'YYYY/MM/DD') AND
TO_DATE('2002/10/26', 'YYYY/MM/DD'));
```

A.2.2 Space only queries in the non-integrated approach filter on the Morton keys during the filtering step. Here, to avoid congestion in the paper we represent only 2 Morton ranges out of the 87 that were needed for this space query. This particular query requests all the time information for a polygonal geometry.

```
SELECT time, morton, z FROM IOT_xy
WHERE ((morton BETWEEN 181664219136000 AND
181664231718911) OR [..]  OR (morton BETWEEN
181675304681472 AND 181675313070079));
```

In the integrated approach the same JOIN procedure as with time queries is used.

Because no functions are available inside the database to perform the decoding of the Morton keys, this procedure is performed with Python. After that, follows the temporary storage in the database in order to finally perform the second filtering step. In this case it is a point in polygon operation. This step is the same independently from the integration and treatment of z used.

```
CREATE TABLE query AS (SELECT *
FROM TABLE(sdo_PointInPolygon(CURSOR(
SELECT X, Y, Z, time FROM decoded),
SDO_GEOMETRY('POLYGON ((72466 453045,
72498 453076, 72526 453048, 72493 453017,
72466 453045))', 28992), 0.001)));
```

A.2.3 Space - time queries in the non-integrated approach are performed similar to the space query. However, in the WHERE predicate a time range is also specified. The query represents a rectangle, where two moments in time are being asked.

```
SELECT time, morton, z FROM IOT_xy
WHERE (time IN (3655, 3680)) AND ((morton
BETWEEN 151177480110080 AND 151178553851903)
OR [..]  OR (morton BETWEEN 156420561436672
AND 156421635178495));
```

In the integrated approach the SQL code used is exactly the same as with space and, time queries.

The next step proceeds as with space queries: a point in polygon operator is used to retrieve the points inside the requested geometry. In the integrated approach we will also filter on time (or z) depending on which treatment is used.

```
-- Non-integrated
CREATE TABLE query AS (SELECT *
FROM TABLE(sdo_PointInPolygon(CURSOR(
SELECT X, Y, Z, time FROM decoded),
SDO_GEOMETRY('POLYGON ((71028 451007,
71027 451654, 71715 451656, 71716 451006,
71028 451007))', 28992), 0.001)));

-- Integrated
CREATE TABLE query AS (SELECT *
FROM TABLE(sdo_PointInPolygon(CURSOR(
SELECT X, Y, Z, time FROM decoded),
SDO_GEOMETRY('POLYGON ((71028 451007,
71027 451654, 71715 451656, 71716 451006,
71028 451007))', 28992), 0.001))
WHERE (TIME IN
(TO_DATE('2000/01/03', 'YYYY/MM/DD'),
TO_DATE('2000/01/28', 'YYYY/MM/DD'))));
```

A.3 Validation loading scripts

The loading follows the same logic as with the proposed methodology. In the **preparation phase** the data are transformed to the correct format and loaded into the database. The table where the spatial data will be inserted is initialised as follows:

```
CREATE TABLE valid (time DATE, geom SDO_GEOMETRY);
```

The data are then read from the LAZ files and formatted according to the rules required by the SQLLDR. Then, in the **loading phase**, the data are bulk loaded into the table.

```
load data append into table valid
fields terminated by ',' TRAILING NULLCOLS (
time DATE 'YYYY_MM_DD',
geom COLUMN OBJECT (
SDO_GTYPE integer external,
SDO_SRID integer external,
SDO_POINT column object (X float external,
Y float external, Z float external)));
```

```
las2txyz |
sqlldr [user]/[password]@//[host]:[port]/[DB]
direct=true control=con.ctl data= \"-\"
bad=bd.bad log=lg.log
```

Then, in order to be able to build spatial indexes, Oracle requires the insertion of certain spatial metadata.

```
INSERT INTO user_sdo_geom_metadata
(table_name, column_name, srid, diminfo)
VALUES ('valid', 'geom', 28992, SDO_DIM_ARRAY (
SDO_DIM_ELEMENT ('X',69000,80000,0.001),
SDO_DIM_ELEMENT ('Y',449000,460000,0.001),
SDO_DIM_ELEMENT ('Z',-100,100,0.001)));
```

The spatial index is created using an R-Tree. To be able to also ask efficient questions in the time dimension, a B-Tree is built on the relevant column. These two actions are achieved using the following SQL statements.

```
CREATE INDEX valid_rtree_IDX ON valid(geom)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS('sdo_indx_dims=2 tablespace=INDX
layer_gtype=POINT sdo_rtr_pctfree=0
work_tablespace=PCWORK sdo_fanout=48');

CREATE INDEX valid_btree_IDX ON valid(time);
```

A.4 Validation query scripts

The same queries are executed using the same QUERIES table. However, this time the query process is more straightforward and the filter and refinement step are performed in an automated way by the database system. The following examples present the same queries as presented previously for the three types of queries.

A.4.1 Time queries like in the non-integrated approach, require only refinement based on the time column. This is performed in the following way:

```
CREATE TABLE result AS
(SELECT t.geom, t.time FROM valid t, queries q
WHERE (q.ID = 5 AND
(t.TIME BETWEEN q.START_DATE AND q.END_DATE)));
```

A.4.2 Space queries require the use of a spatial operator. In this case we require to obtain all the points that intersect the geometry. This is performed as follows:

```
CREATE TABLE result AS
(SELECT t.geom, t.time FROM valid t, queries q
WHERE (q.ID = 3 AND
SDO_ANYINTERACT(t.geom, q.geom) = 'TRUE'));
```

A.4.3 Space - time queries require both a spatial and a time predicate. The query is performed as follows:

```
CREATE TABLE result AS
(SELECT t.geom, t.time FROM valid t, queries q
WHERE (q.ID = 1 AND
(t.TIME IN (q.START_DATE, q.END_DATE))
AND SDO_ANYINTERACT(t.geom, q.geom) = 'TRUE'));
```