

MSc thesis

Geomatics for the Built Environment

Using a Space Filling Curve
for the Management of Dynamic
Point Cloud Data in a Relational
DBMS

Styliani Psomadaki

ON COVER

The 3D Morton Space Filling Curve coloured according to the proximity of the points on the curve.

USING A SPACE FILLING CURVE FOR THE
MANAGEMENT OF DYNAMIC POINT CLOUD
DATA IN A RELATIONAL DBMS

A thesis submitted to the Delft University of Technology in
partial fulfilment
of the requirements for the degree of

Master of Science in
Geomatics for the Built Environment

by

Styliani Psomadaki

November 2016

Styliani Psomadaki: *Using a space filling curve for the management of dynamic point cloud data in a Relational DBMS* (2016)

© This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit

<http://creativecommons.org/licenses/by/4.0/>.

An electronic version of this thesis is available at

<http://repository.tudelft.nl/>

The work in this thesis was supported by:



Geo-Database Management Centre
Department of the OTB
Faculty of Architecture & the Built Environment
Delft University of Technology



Deltares
Independent institute for applied research in the field
of water and subsurface
Delft

Supervisors:	Drs. T.P.M. Tijssen Prof. dr. ir. P.J.M. van Oosterom
Co-reader:	Dr. R.C. Lindenberg
Ext. Supervisor:	Fedor Baart

ABSTRACT

The rapid developments in the field of point cloud acquisition technologies have allowed point clouds to become an important source of information for many applications. One of the newest applications of point clouds concerns the monitoring of the coast. Many countries, among which the Netherlands, use this source of data in order to determine the changes in coastal elevations. This means that point clouds are collected every hour, day, month, year; ultimately talking about dynamic point clouds. To be able to efficiently use this plethora of data, the management of those point clouds, dynamic or not, is proven to be crucial.

Point clouds, like the majority of geodata, have been traditionally managed using file-based solutions. Nevertheless, the last years database solutions have emerged. Typical examples are the point cloud extensions for PostgreSQL and the Oracle Database. Both options use a similar block-based organisation. In addition to the block based organisations, point clouds can also be managed using a flat table where each point is stored in a separate row. While the first approach is very scalable and efficient, the second is easier to implement and to update. To make the flat model scalable, a Space Filling Curve (SFC) can be used to cluster the data. Nonetheless, both approaches in their current forms, are not suited for the management of dynamic points. The reason for this is the fact that they do not consider the time dimension as part of the organisation and further insertions for the block-based approaches are not straightforward.

Within this thesis a SFC approach for managing dynamic point clouds is investigated. For this, the flat model approach using an Index Organised Table (IOT) within a Relational Database Management System (RDBMS) is used. Two variants coming from two extremes of the space - time continuum are then taken into account. In the first approach, space and time are both used within the SFC (integrated approach), while in the second one, time dominates over space (non-integrated approach). Along these two approaches, two treatments of the z dimension are, also, studied: as attribute or as part of the SFC. In addition to that, building on the coastal monitoring applications, the most important queries are identified: space - time, only time, only space.

The efficiency of the implemented methodology is tested through the execution of a benchmark. Using two use cases coming from coastal applications, the benchmark is executed once for daily and once for yearly data. The results show that the SFC approach is an appropriate method for managing dynamic point clouds. Furthermore, the integrated approach is the most suitable way to proceed. Achieving scalability, time efficiency and dynamic insertions can be achieved for various use cases.

ACKNOWLEDGEMENTS

Within this section I would like to take the opportunity to thank a few people that had an impact during the execution of this research.

First, I would like to thank *Peter van Oosterom* for his invaluable conversations and advice during this one year journey called Master thesis, but also for the amount of knowledge he gave us during the whole duration of the Master program. Next, I would like to thank *Theo Tijssen* who provided me with very important technical support, especially concerning the Oracle database. He was always kind enough to help me when things were going wrong. A great thanks goes to *Fedor Baart* and the people at Deltares. This opportunity to work at a company like Deltares taught me a lot of great things that go beyond the execution of a thesis. Finally, I would like to thank *Roderik Lindenbergh* for his valuable comments as the external reader of the thesis.

I would also like to thank my *parents*, and my *sister* for their infinite love and support. Finally, I want to thank my closest friends for always being there for me.

CONTENTS

1	INTRODUCTION	1
1.1	Motivation and Problem Statement	1
1.2	Use case	3
1.3	Objectives & Research Question	3
1.3.1	Scope	4
1.3.2	Significant findings	4
1.4	Thesis Outline	5
2	THEORETICAL BACKGROUND	7
2.1	Modelling the world	7
2.2	Point clouds	7
2.2.1	Relevant aspects for the management of point clouds .	8
2.2.2	Management of point clouds using files	9
2.2.3	Management of point clouds using databases	10
2.3	Point Access Methods	10
2.3.1	Indexing techniques	10
2.3.2	Subdivision of space: Quadtrees and Octrees	12
2.3.3	Space filling curves	13
3	RELATED WORK	17
3.1	Management of point clouds in DBMS	17
3.1.1	Oracle SDO_PC	18
3.1.2	Oracle Flat	18
3.1.3	Oracle Hybrid	19
3.1.4	PostgreSQL PC_Patch	19
3.1.5	PostgreSQL flat	20
3.1.6	GeoHashTree in PostgreSQL	20
3.1.7	MonetDB	20
3.1.8	Comparing block and flat based organisation	21
3.2	Spatio - temporal data management	22
3.2.1	Time in Geographic Information Systems	22
3.2.2	Storage and retrieval of multi-dimensional data	24
3.2.3	Point cloud implementations	25
3.3	Conclusions	26
4	SPACE FILLING CURVE APPROACH	27
4.1	Motivation	27
4.2	Storage model	28
4.2.1	Query requirements	29
4.2.2	Integrated approach	29
4.2.3	Non-integrated approach	30
4.2.4	The encoding of space and time	32
4.3	Loading procedure	33
4.3.1	Options for new data loading	33
4.3.2	Loading phases	34
4.4	Query procedure	35
4.4.1	Querying space filling curves	36
4.4.2	Decomposition and evaluation of the query geometry	38

4.4.3	Merging of the ranges	41
4.4.4	Refinement	46
4.4.5	Summarising the query procedure	46
5	IMPLEMENTATION AND EXPERIMENTS	49
5.1	Tools and datasets used	49
5.1.1	Software	49
5.1.2	Hardware	50
5.1.3	Datasets used	50
5.2	Metrics of performance	51
5.2.1	Fetching time	51
5.2.2	Percentage of extra points	51
5.2.3	Depth of the tree	52
5.2.4	Degree of merging	52
5.3	Implemented prototype	52
5.3.1	The encoding of space and time	52
5.3.2	Loading procedure	55
5.3.3	Query procedure	56
5.4	Experiments	57
5.4.1	Depth of the tree - integrated approach	57
5.4.2	The degree of merging - non-integrated approach	60
5.4.3	Depth of the tree with merging - non-integrated	62
6	BENCHMARKS	63
6.1	Sand Engine	64
6.1.1	Loading procedure	64
6.1.2	Query procedure	66
6.2	Coastline	73
6.2.1	Loading procedure	73
6.2.2	Query procedure	74
6.3	Validation and comparison	75
6.4	Summary	77
7	CONCLUSION AND FUTURE WORK	81
7.1	Conclusions	81
7.1.1	Research Questions	81
7.1.2	Contribution	84
7.1.3	Reflection and discussion	84
7.2	Future Work	85
7.2.1	Native database functionality	85
7.2.2	Investigating a different SFC	86
7.2.3	Investigation of parallel processing	86
7.2.4	Up-scaled benchmark of trillion points	87
7.2.5	Higher dimensional keys	87
7.2.6	Investigating delta queries	87
7.2.7	Two- set refinement stage	87
7.2.8	Investigating the generation of blocks	88
7.2.9	Even more dynamic data	88
7.2.10	Moving objects point cloud	89
7.2.11	True 4D query	89
A	DATASETS	91
A.1	Overview	91
A.2	Characteristics	92

A.2.1	Sand Engine	92
A.2.2	Coastline	93
A.2.3	Comparison	94
B	QUERY GEOMETRIES AND TIME RANGES	95
B.1	Sand Engine	95
B.2	Coastline	95
C	DETAILED RESULTS	99
C.1	Sand Engine	99
C.1.1	Depth of tree	99
C.1.2	The degree of merging	101
C.1.3	Depth of tree with merging	103
C.1.4	Benchmark (All stages)	105
C.2	Coastline	109
C.2.1	Small benchmark	109
C.2.2	Medium benchmark	110
C.2.3	Large benchmark	112
C.2.4	Full benchmark (All stages)	113
D	CODE DESCRIPTION AND SQL STATEMENTS	115
D.1	Loading scripts	115
D.2	Query scripts	118
D.2.1	Filter step	119
D.2.2	Refinement step	121
D.3	Validation scripts	123
D.4	Loading scripts	123
D.5	Query scripts	124
E	REFLECTION	133

LIST OF FIGURES

Figure 2.1	A schematic representation of the B-Tree data structure	11
Figure 2.2	A schematic representation of the B*-Tree data structure (Adapted from Comer [1979])	11
Figure 2.3	The R-Tree data structure of several objects with branching factor of 4	12
Figure 2.4	The two most commonly found quadtrees in literature: the region quadtree and the point quadtree.	13
Figure 2.5	The octree data structure of level 0, level 1 and level 2	13
Figure 2.6	The row order curve for 2, 4 and 8 bit representation	14
Figure 2.7	The most commonly used Space Filling Curves	15
Figure 2.8	A comparison of the clustering capabilities of three Space Filling Curves (from left to right): Row order, Hilbert, Morton	16
Figure 3.1	The major storage models for managing point clouds in DBMS environment and their implementations in well-known systems.	18
Figure 3.2	The effect of using bigger or smaller block sizes on the selected area obtained. Bigger blocks can even double the amount of the selected area, contrary to smaller blocks that can approximate the area more closely.	21
Figure 4.1	A schematic representation of the most important queries when managing dynamic point clouds. From left to right: only space queries, space and time queries and, only time queries.	30
Figure 4.2	A schematic representation of the integrated approach using 8 bits in the three dimensions	31
Figure 4.3	A schematic representation of the non-integrated approach using 8 bits in the three dimensions	31
Figure 4.4	A schematic representation of the loading procedure for the SFC approach.	35
Figure 4.5	Two step query processing approach.	36
Figure 4.6	A naive way to perform range queries when using space filling curves.	37
Figure 4.7	Decomposition strategy based on the relation between the morton curve and the Quadtree. Adapted from: van Oosterom and Vrijlbrief [1996]	39
Figure 4.8	Quadtree decomposition of a complex polygon at different levels and in comparison to the Minimum Bounding Rectangle approximation.	40
Figure 4.9	The search algorithm applied to three candidate tree cells. The grey Tree Cell is disjoint from the Query Region (hatched), while the red Tree Cell is completely within the Query Region. Finally the blue Tree Cell partially overlaps the Query Region and therefore, is split into 4 smaller tree cells and the algorithm is repeated.	41

Figure 4.10	An example of merging consecutive ranges. Left: the original situation with three morton ranges. Right: the merging results in one big morton range.	43
Figure 4.11	First example of merging ranges to obtain a maximum number. (a) The original situation with 6 morton ranges. (b) The maximum number is set to 3 and the regions are expanded accordingly until the three ranges are obtained. (c) The maximum number is set to 2 and the regions are expanded accordingly until the two ranges are obtained. The grey area represents the additional space added due to the merging of TCs.	44
Figure 4.12	Second example of merging ranges to obtain a maximum number. (a) The original situation with 11 morton ranges. (b) The maximum number is set to 3 and the regions are expanded accordingly until the three ranges are obtained. (c) The maximum number is set to 2 and the regions are expanded accordingly until the two ranges are obtained. The grey area represents the additional space added due to the merging of TCs.	44
Figure 4.13	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. Source: Pso-madaki et al. [2016]	45
Figure 4.14	A schematic representation of the query procedure .	47
Figure 5.1	The effect of applying no time scaling and a time scaling of 2 on the space - time proximity of the points. .	54
Figure 5.2	The effect of different degree of scaling on the space - time proximity of the points.	54
Figure 5.3	Effect of using deeper 2^n - Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space - time queries in the integrated approach (treatment of z as an attribute). ST-F is a line with buffer of 5 metres (776 points).	58
Figure 5.4	Effect of retrieving data from deeper 2^n - Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space only queries in the integrated approach (treatment of z as an attribute). S-A represents a rectangular geometry (112144 points). .	58
Figure 5.5	Effect of retrieving data from deeper 2^n - Tree Cells on the percentage of extra points obtained and fetching time. Case: Time only queries in the integrated approach (treatment of z as an attribute). T-A represents one day (78902 points).	59

Figure 5.6	Effect of retrieving data using deeper 2^n -Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space - time queries in the integrated approach (treatment of z added in the key). ST-F is a line with buffer of 5 metres (776 points).	59
Figure 5.7	The ratio of ranges returned by the z in the key treatment to the number of ranges for the z as attribute treatment when moving deeper in the 2^n -tree	60
Figure 5.8	Effect of imposing a different degree of merging in the same original Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space - time queries in the non-integrated approach (treatment of z as an attribute). ST-E represents line with buffer of 5 metres (380 points).	61
Figure 5.9	Effect of imposing a different degree of merging in the same original Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space only queries in the non-integrated approach (treatment of z as an attribute). S-D represents line with buffer of 5 metres (11933 points).	61
Figure 6.1	The storage requirements of the 4 storage models for the 3 benchmark stages	65
Figure 6.2	Schematic representation of the results found in Table 6.4 and Table 6.6 . The x axis represents the points available in the Index Organised Table (IOT), while the y axis represents the query response times. Each line also contains the % of false hits compared to the actual number of points. Case: Space - time queries	71
Figure 6.3	Schematic representation of the results found in Table 6.4 and Table 6.6 . The x axis represents the points available in the IOT, while the y axis represents the query response times. Each line also contains the % of false hits compared to the actual number of points. Case: Space only and time only queries	72
Figure 6.4	Morton ranges decomposition of polygonal geometries	78
Figure 6.5	Morton ranges decomposition of circular and line buffer geometries	79
Figure 7.1	Separating internal ranges and ranges on boundary. White cells are completely inside the query region and do not need refinement. Grey cells are partially inside the query region and have to proceed to the refinement stage.	88
Figure A.1	The location of the datasets on the map of the Netherlands	91
Figure A.2	A point cloud of the Sand Engine use case from the year 2011	92
Figure A.3	A point cloud of the Sand Engine use case from the year 2015	92
Figure A.4	The coastline point cloud dataset for the year of 2012	94
Figure B.1	The Sand Engine queries (spatial extent)	96
Figure B.2	The spatial extent of the Coastline queries	97

LIST OF TABLES

Table 5.1	The distribution of the available tablespaces according to the purpose.	50
Table 5.2	The used encodings of space and time for the two datasets (Sand Engine, Coastline)	53
Table 6.1	The benchmark cases and the used notation.	63
Table 6.2	Benchmark stages description of the Sand Engine use case. The size column corresponds to the size of the LAS files. More information about the dataset can be found in Section A.2.1	64
Table 6.3	The loading times for the two integrations of space and time and the two treatments of z for the Sand Engine use case. For more insight concerning the <i>Load heap</i> and <i>Load IOT</i> columns refer to Listing D.1 and Listing D.3 respectively.	64
Table 6.4	Query response times, the percentage of false hits compared to the actual number of points and, the points returned by the queries for the non-integrated approach with maximum number of ranges set to 200 . For more insight about how space - time, space and time queries are performed refer to Listing D.10 , Listing D.9 , and Listing D.6 respectively.	68
Table 6.5	Query response times, the percentage of false hits compared to the actual number of points and, the points returned by the queries for the integrated approach with maximum number of ranges set to 200 . For more insight about how space - time, space and time queries are performed refer to Listing D.8	69
Table 6.6	Query response times, the percentage of false hits compared to the actual number of points and, the points returned by the queries for the integrated approach with maximum number of ranges set to 1,000,000 . For more insight about how space - time, space and time queries are performed refer to Listing D.8	70
Table 6.7	Benchmark stages description of the Coastline use case for the Full benchmark . The size column corresponds to the size of the LAS files. More information about the dataset can be found in Section A.2.2	73
Table 6.8	The loading times of the integrated approach for the two treatments of z in the coastline use case (Full benchmark). For more insight concerning the <i>Load heap</i> and <i>Load IOT</i> columns refer to Listing D.1 and Listing D.3 respectively.	73
Table 6.9	The query response times, the percentage of false hits compared to the actual number of points and, the number of points returned by the queries in the integrated approach of the coastline dataset (Full benchmark).	75

Table 6.10	The loading response times for the validation of the Sand Engine use case. For more insight concerning how the the data are loaded, refer to Section D.4 . . .	76
Table 6.11	The storage requirements for the validation of the Sand Engine use case.	76
Table 6.12	The query response times, the percentage of false hits of the filter step compared to the actual number of points and, the number of points returned by the queries in the validation of the Sand Engine use case.	76
Table 7.1	Using C++ code for SFC calculation	86
Table A.1	Characteristics of the original Sand Engine data. . . .	93
Table A.2	Characteristics of the artificially created Sand Engine data.	93
Table A.3	Dataset description of the Coastline use case	94
Table A.4	Comparison of the two datasets	94
Table B.1	The description of the Zandmotor queries in time. In <i>type</i> : <i>s-t</i> stands for space - time, <i>t</i> for time and <i>s</i> for space. In <i>time type</i> : <i>c</i> stands for continuous and <i>d</i> for discrete.	95
Table B.2	The description of the Coastline queries in time. In <i>type</i> : <i>s-t</i> stands for space - time, <i>t</i> for time and <i>s</i> for space. In <i>time type</i> : <i>c</i> stands for continuous and <i>d</i> for discrete.	96
Table C.1	Depth of the tree experiment. Case: Space - time queries of integrated approach with <i>z</i> as an attribute.	99
Table C.2	Depth of the tree experiment. Case: Space only and time only queries of integrated approach with <i>z</i> as an attribute.	99
Table C.3	Depth of the tree experiment. Case: Space - time queries of integrated approach with <i>z</i> as part of the key.	100
Table C.4	Depth of the tree experiment. Case: Space only and time only queries of integrated approach with <i>z</i> as part of the key.	100
Table C.5	The degree of merging experiment. Case: Space - time and space only queries of non-integrated approach with <i>z</i> as attribute.	101
Table C.6	The degree of merging experiment. Case: Space - time and space only queries of non-integrated approach with <i>z</i> as part of the key.	102
Table C.7	Depth of the tree with merging experiment. Case: Space - time queries of non-integrated approach with <i>z</i> as an attribute.	103
Table C.8	Depth of the tree with merging experiment. Case: Space only queries of non-integrated approach with <i>z</i> as an attribute.	103
Table C.9	Depth of the tree with merging experiment. Case: Space - time queries of non-integrated approach with <i>z</i> as part of the key.	104
Table C.10	Depth of the tree with merging experiment. Case: Space only queries of non-integrated approach with <i>z</i> as part of the key.	104

Table C.11	Benchmark results for all query stages. Case: integrated approach with z as an attribute.	105
Table C.12	Benchmark results for all query stages. Case: integrated approach with z as part of the key.	106
Table C.13	Benchmark results for all query stages. Case: non - integrated approach with z as an attribute.	107
Table C.14	Benchmark results for all query stages. Case: non - integrated approach with z as part of the key.	108
Table C.15	The loading times for the two integrations of space and time and the two treatments of z in the coastline use case (Small benchmark).	109
Table C.16	The query response times, the percentage of false hits compared to the actual number of points and the number of points returned by the queries for the two integrations of space and time and the two treatments of z in the coastline use case (Small benchmark).	110
Table C.17	The loading times for the two integrations of space and time and the two treatments of z in the coastline use case (Medium benchmark).	110
Table C.18	The query response times, the percentage of false hits compared to the actual number of points and the number of points returned by the queries for the two integrations of space and time and the two treatments of z in the coastline use case (Medium benchmark).	111
Table C.19	The loading times for the two integrations of space and time and the two treatments of z in the coastline use case (Large benchmark).	112
Table C.20	The query response times, the percentage of false hits compared to the actual number of points and the number of points returned by the queries for the two integrations of space and time and the two treatments of z in the coastline use case (Large Benchmark).	112
Table C.21	Benchmark results for all query stages. Case: integrated approach with z as an attribute of the full benchmark.	113
Table C.22	Benchmark results for all query stages. Case: integrated approach with z as a key of the full benchmark.	114

ACRONYMS

DBMS Database Management System.....	2
GIS Geographic Information Systems	22
I/O Input/ Output.....	21
IOT Index Organised Table.....	xv
LAS LASer	9
LIDAR Light Detection And Ranging.....	51
LOD Level of Detail.....	9
MBR Minimum Bounding Rectangle.....	12
NOSQL Not only SQL.....	20
PAM Point Access Methods.....	10
PIP Point in Polygon.....	46
PDAL Point Cloud Abstraction Library.....	1
QR Query Region.....	38
RDBMS Relational Database Management System.....	20
SAM Spatial Access Methods	10
SFC Space Filling Curve.....	2
SQL Structured Query Language	5
STDBMS Spatio-Temporal Databases	22
TC Tree Cell.....	38

Over the last years, point cloud usage has seen a rapid growth and it is expected that this growth will become even bigger in the years to come. This is mainly due to the developments in point cloud acquisition technologies; namely terrestrial and airborne laser scanning, mobile mapping, image matching and multi-beam echo-sound techniques etc. Pulse frequencies have increased significantly and so has the acquired point density. These technologies have allowed large-scale acquisition projects, a typical example of which are the Actueel Hoogtebestand Nederland (AHN) datasets [AHN, 2016] for the country of the Netherlands.

But apart from these large-scale projects, recent advances have produced many low-cost and generally easy-to-use devices and techniques, such as the Microsoft Kinect¹, Google's Project Tango², depth cameras and structure from motion techniques combined with Unmanned Aerial Systems (UAS). These devices enable the acquisition of billions points in a short period of time [Schops et al., 2015; Rusu and Cousins, 2011; Westoby et al., 2012; Khoshelham, 2011] and have allowed repeated scans of the same area on a frequent basis, resulting in even more available spatial information and the addition of the fourth dimension; the temporal component. This increasing availability, spatio - temporal density and acquisition frequency of point cloud datasets has consequently made them suitable in a number of applications such as: 3D urban modelling [Haala and Kada, 2010], indoor modelling [Previtali et al., 2014], flood modelling [Abdullah et al., 2009], line of sight analysis [Peters et al., 2015], forest mapping [White et al., 2013] and many other. In the previously mentioned applications point clouds are used either directly or they form the basis in generating other representations (grid, vector) of the world.

1.1 MOTIVATION AND PROBLEM STATEMENT

Although technological developments have made it feasible to acquire a large amount of information, the management and storage of those massive point clouds is a challenge [van Oosterom et al., 2015; Cura et al., 2015; Richter and Döllner, 2014]. In the majority of today's applications file-centric approaches are chosen, meaning that point cloud data are stored and processed as a collection of files. A typical workflow includes using desktop applications or command line executables like Rapidlasso's LAStools (mixed - source) [Isenburg, 2012a] or the Point Cloud Abstraction Library (PDAL) (an open - source project), reading one or more files, processing the data and writing files back to the user. In the case of LAStools, all this functionality is integrated with a project management and quality control framework making it a stand-alone application [Hug et al., 2004]. Therefore, if those appli-

¹ <http://www.xbox.com/en-US/xbox-360/accessories/kinect>

² <https://www.google.com/atap/project-tango/>

cations perform well why look into a different direction, that of a Database Management System (DBMS)?

Despite the fact that file-based systems are continuously improving and gaining more functionality [Otepka et al., 2012], they generally present deficiencies with the increasing acquisition of point clouds and have certain limitations in the allowed file formats [Sabo et al., 2014]. The absence of multi-user functionalities is another important deficiency. The situation becomes even more complicated when taking into account the multiple dimensions of point clouds, as these are far from fixed and can be present in any order and combination [ASPRS, 2011]. Finally, integration with other (spatial) data is not a trivial functionality.

DBMS have for many decades already been dealing with the data storage, indexing techniques, scalability and availability requirements needed by the majority of spatial and non-spatial applications. DBMS provide concurrency control, transactions characterised by atomicity and isolation, security and version control [Elmasri and Navathe, 2010]. The database community, commercial and open-source, identifying the need for point cloud data management already provides native point cloud support. Both Oracle (Spatial and Graph) and PostgreSQL (PostGIS) use a rather similar approach for point cloud data storage. Their storage model is based on the physical reorganisation of the points into blocks [Ravada et al., 2010; Ramsey, 2014].

Although the existing point cloud data management solutions available in the DBMS perform well for specific cases, they present certain limitations. First, they consider point clouds as rather static objects, not taking into account the time dimension in the organisation of the points. This means that time is either stored as an attribute within the file system or database, or is only present within the file- or table name. Therefore, time does not directly influence the organisation of the data. This is in contrast with the growing number of point clouds continuously generated from low-cost sensors, which have shifted the nature of this spatial representation from static to dynamic. In specific applications, even, the time dimension becomes as equally selective as the planimetric coordinates. We are then talking about *dynamic point clouds*. This implies that storing time as an attribute no longer provides efficient searching, as the query plan will first select based on the spatial components and then on time. The disadvantages become even more obvious as time-dependent information is accumulated. Finally, the current solutions do not always provide support for further insertion of new data without having to rebuild the blocks from scratch. This hinders further development of applications that keep track of changes in an area using point clouds.

The ultimate goal when managing dynamic point clouds is to design a method to organise the data in a compact way that will support the efficient retrieval of the points. And since, even today, one-dimensional access methods (like the B-Tree) continue to be superior than multidimensional one's, the ideal solution thus would be to map the multi-dimensional space into a one-dimensional value and then use this value to efficiently store the points. Therefore we are in need of a structure that clusters points in space and time. However, there exists no mapping technique that completely preserves spatial proximity. A very popular approach is using a Space Filling Curve (SFC). A SFC has the ability to apply a linear ordering to a multi-dimensional domain. Several SFCs have been developed through the years with the Morton and the Hilbert curve being the most prominently used [Abel and Mark, 1990].

1.2 USE CASE

Coastal areas are highly dynamic environments where diverse populations (humans or animals) live. Changes around the coastal area can take place in very short periods of time (e.g. after a storm) or in the long-term (e.g. by constant erosion by waves and wind). As a result frequent monitoring of the coastal morphology is of great importance in order to understand how the changes occur and in order to efficiently plan flood defence mechanisms that will protect the hinterland.

The Netherlands is a country situated at the North Sea and its southwestern part is the delta of the three large rivers: Rhine, Meuse and the Scheldt. Over the years, protection of the coast has played a very important role for the development of the country, since almost one quarter of the land lies below sea level. Typical flood protection structures include dams, dikes and dunes. The Dutch coastal policy was introduced in 1990. There it was stated that the country should "hold the line" [Sisternans and Nieuwenhuis, 2004], meaning that the coastline should be prevented from moving towards the mainland. For this reason, the Ministry of Transport, Public Works, and Water Management (Rijkswaterstaat) as part of the coastal monitoring guidelines performs a yearly survey of the Dutch coast in order to determine changes in coastal elevations. These surveys are a combination of depth surveys from echo sounding equipments mounted on vessels and height surveys coming from Laser altimetry technology [Pot, 2011].

Point clouds have evolved to be a very important source of information for coastal applications [Carter et al., 2012]. The usefulness of point cloud data lies in the fact that point cloud acquisition techniques have become highly accurate and quick, allowing daily or even hourly collection of data. This comes in contrast to the until recently major sources of information, namely satellite imagery or topographic maps acquired with classical surveying methods, which either have low spatial or temporal resolution. Some common coastal applications where point clouds are widely used are [Carter et al., 2012]: (a) coastal change detection, (b) shoreline delineation, (c) coastal inundation prediction etc. However, as mentioned before the management of these spatio-temporal point cloud datasets still lacks support by the current management systems, thus leading to the underutilisation of point clouds in their raw format. In fact, the lack of dynamic point cloud data management and handling is a real world problem in the domain of coastal monitoring. The Deltares institute for applied research in the field of water and subsurface was the initiator of this research.

1.3 OBJECTIVES & RESEARCH QUESTION

The aim of this thesis is to explore a methodology that offers efficient storage and data handling of dynamic point clouds. The methodology should efficiently handle a range of spatio-temporal queries, should be scalable, and provide quick responses. Storage requirements are, also, a relevant requirement but not a priority.

The main research question of this thesis is:

Is a Space Filling Curve (SFC) approach an appropriate method for integrating the space and time components of point clouds in order to support efficient management and querying (use) in a DBMS?

In order to answer the main question, the following sub-questions are relevant:

1. What are dynamic point clouds and what are relevant use cases and requirements for their querying?
2. What are the relevant parameters that need to be taken into account for the management of dynamic point clouds when using a [SFC](#) approach?
3. What kind of [SFC](#) approaches can be used to support the integration of space and time, taking into account the continuous insertions of new points and efficient querying?
4. How do the different [SFC](#) approaches compare to each other according to the use cases?

1.3.1 Scope

In order to define a clear research scope, the following remarks are made:

1. The prototype implemented in this research is not aimed at being ready for commercial and non-commercial use. It only serves as a proof of concept.
2. The implemented prototype does not aim to compare different [DBMSs](#).
3. The use cases will focus on coastal monitoring applications due to the applications available at Deltares. However, the same methodology should be possible to be applied to datasets coming from a different application domain.
4. This thesis handles the management part of dynamic point clouds. This means that acquisition, post-processing and georeferencing, analysis, dissemination and visualisation of the dynamic point clouds is out of scope for this research.
5. Parallel computing although very relevant the last years, will not be investigated. Exceptions apply only when using the parallelisation available in the chosen [DBMS](#).

1.3.2 Significant findings

The [SFC](#) approach for managing dynamic point clouds is a method that is essentially different from the approaches used in the state-of-the-art [DBMSs](#). Rather than organising points in blocks, the method uses the flat table model that stores one point per row. In addition to that, instead of keeping the original dimensions in separate attributes, the method replaces them using a full resolution [SFC](#), from where the original dimensions can later be recovered. Two major integrations of space and time are implemented, which fundamentally correspond to two extremes of a space - time continuum. The first one, deals with the complete integration of space and time and is called the *integrated approach*. In the second one, time is more important and is called the *non-integrated approach*. In order to query both approaches, the query algorithm needs to be modified: first, the query region is approximated using a higher dimensional quadtree or 2^n -tree. Then, the returned ranges are fetched, the points are decoded back to their original dimensions and

pipelined into a refinement step. The storage model that structures space and time together in the curve is proven here to provide the best results for two major use cases (nationwide coastal monitoring and Sand Engine).

1.4 THESIS OUTLINE

The rest of this thesis document is organised as follows:

- [Chapter 2](#) provides a small introduction to the [THEORETICAL BACKGROUND](#) that is needed for the reader to understand the subject. A lot of elaboration takes place about point clouds and their management. Relevant point access methods existing in the literature are, also, briefly summarised.
- [Chapter 3](#), with the [RELATED WORK](#), starts by describing the current approaches used for the management of point clouds in general, and proceeds to approaches used for managing spatio - temporal data.
- [Chapter 4](#) introduces the [SPACE FILLING CURVE APPROACH](#) for managing dynamic point clouds. The storage model, as well as all the steps needed to load and query the data are described with diagrams and algorithms.
- [Chapter 5](#) covers the description of the [IMPLEMENTATION AND EXPERIMENTS](#) used for testing the proposed storage models.
- [Chapter 6](#) describes and analyses the [BENCHMARKS](#) used as a proof of concept.
- [Chapter 7](#), finally, gives the answer to the [Research Questions](#), discusses the [Contribution](#) to the field and criticises the drawbacks of the prototype. Finally, relevant [Future Work](#) is given.

In addition to the previously described chapters, this thesis document includes the following Appendices:

- [Appendix A](#) describes the use cases used for testing the methodology.
- [Appendix B](#) describes and depicts the queries used during the experiment and benchmark execution.
- [Appendix C](#) contains tables with the results obtained from the experiments and benchmarks.
- [Appendix D](#) gives an overview of the Structured Query Language (SQL) code generated from the implemented Python scripts.

2

THEORETICAL BACKGROUND

The [THEORETICAL BACKGROUND](#) chapter aims to provide the relevant theoretical knowledge for the subjects that will follow in the rest of the thesis document. More specifically, [Section 2.1](#) introduces the need for modelling the world around us. [Section 2.2](#) is all about point clouds and their management using files or [DBMS](#). Finally, [Section 2.3](#) introduces the topics of indexing and clustering points using a [DBMS](#) approach.

2.1 MODELLING THE WORLD

Understanding the world and its processes is a rather difficult subject. For this reason, scientists in the majority of their applications tend to capture and represent a part of the real world, rather than trying to represent the world in all its detail. This is exactly how the International Organization for Standardization - Technical Committee 211 (*ISO/TC211*) defines a model; an abstraction of some aspects of reality [[ISO, 2014](#)]. Two types of models can be identified: static and dynamic. The former study the state of the abstracted world in a specific moment in time, while the latter are used to identify the changes that occur over time (and maintain history).

Models are also very important for representing spatial information in a computer environment. In general, two types of models can be identified: the *raster* or *vector* representation. The first uses equally sized and contiguous cells (pixels) in order to represent a real world phenomenon. For example, in case elevation is represented, the value of each cell represents the height in that particular area covered. The second approach uses points, lines and polygons to capture reality. For the same application, elevation, its vector representation can be achieved by using either Triangulated Irregular Networks (TIN) or contour lines.

2.2 POINT CLOUDS

Another way of modelling the world around us that has seen a rapid growth in use the last years is point cloud data. From a mathematical point of view, a point cloud is a collection of points $P_i, i = 1, 2, \dots, n$ embedded in the three dimensional Cartesian space and which, as a whole, describe the surface of an object or many objects. Because of the nature of the way that they are acquired, point clouds are considered unstructured data (not found on a regular grid), an attribute which is reflected in the "cloud" part of the term. This "behaviour" makes point clouds a rather different spatial representation. On the one hand, point clouds can be considered as vector-based structures (since they are a collection of points), but on the other hand they present many similarities with raster data (sampling nature), given that the majority of the raster transformations and processes can be applied to

point clouds as well. However, point clouds are not a regular grid and therefore, it is proposed that point clouds should be considered as a third type of spatial representation [van Oosterom et al., 2015].

Each point P_i that is part of a point cloud has three coordinates, namely X_i, Y_i, Z_i , and attached several attributes. These attributes are highly correlated to the way the point cloud was acquired in the first place and in case of laser scanning they can be:

INTENSITY VALUE The value of reflected light that was received from a specific point.

RETURN NUMBER Since one laser pulse can meet more than one reflective surface it can be split into many return values.

NUMBER OF RETURNS The total number of returns for a specific point.

TIME STAMP The acquisition time of the point (usually in GPS time).

RGB (Red, Green, Blue) values of the specific point usually extracted from aerial images.

CLASSIFICATION INFORMATION which corresponds to the type of real world object the point belongs to e.g. ground, building etc.

2.2.1 Relevant aspects for the management of point clouds

Point clouds have evolved to establish a major source of information for many topographic applications. Nonetheless, their large volume, nature and complexity of their attributes has made the management of massive point clouds a rather challenging topic. In many applications even point clouds from different sources, with different resolutions and time coverages can be found and have to be managed. For many years now point clouds have been managed using file-based solutions. However, today new ways to manage point clouds can be achieved using [DBMS](#).

Whatever management medium is used for organising massive point cloud data, [van Oosterom et al. \[2015\]](#) present the relevant aspects that should be taken into account during the design phase:

- The storage of the X, Y, Z coordinates and the support for Coordinate Reference Systems.
- The attributes attached to the points like, intensity, return number, number of returns, classification, colour (Red, Green, Blue) in any order and combination.
- The spatial organisation of the points including efficient blocking, clustering (space filling curves) and indexing techniques in the multidimensional space. A parameter relevant to the clustering aspect is the choice of dimensions used for the clustering key calculation, for example 2D (X,Y) or 3D (X,Y,Z).
- The time component and its importance in the organisation of the point cloud. Time can either have no role in the organisation (stored as an attribute), or play a crucial role in the organisation by taking place in the key calculation of the clustering technique, e.g. in the form (X,Y,t) or (X,Y,Z,t).

- Compression techniques in order to reduce storage and enable dissemination via wireless networks.
- Level of Detail (LoD) strategies in order to significantly reduce the amount of data that are sent from the server to the client and to support efficient computations and visualisations. The latter involves decreasing the detail of the scene as the viewer moves away (zooms out). In other words, the further the viewer moves from an area, the lesser the points that should be presented from that area. Two types of approaches can be identified for LoD structuring: the multi-scale and the vario-scale (or continuous) approach. In the former, the levels are predefined while in the latter the level of detail is another dimension, called importance. In contrast to the discrete levels, the importance value can be of floating point data type. This extra dimension can then be used as a parameter for the organisation itself by sorting and clustering using space filling curves e.g. Hilbert or Morton curve. The spatial clustering could then be in the form (X,Y, imp) or (X,Y,t, imp) or (X,Y,Z, imp) or (X,Y,Z,t, imp)
- Operations like loading, selecting, and algorithms that operate on point clouds e.g. normal vector estimation, nearest neighbour finding etc.
- Parallel processing techniques in order to fully exploit the computing capabilities of the modern computers. Parallel processing should, in general, offer better performance than a single process.

2.2.2 Management of point clouds using files

Same as with the management of other geographic data, point clouds have for many years now been managed in the traditional way of using files and with many different data structures. These files can then be stored in a hierarchy of folders in the operating system and accessed by the software when needed.

There is no one best way to store a point cloud. Point clouds can either be stored in their original form (as points) or they can be simplified and stored using rasters. The latter is often the form in which point cloud information is given to the end users for analysis. Storing the raw point cloud information, on the other hand, is mostly achieved using ASCII or the LASer (LAS) file format [ASPRS, 2011]. ASCII formats have been utilised mostly because they are human readable but can become very large in size, while the LAS format being a binary format, stores points and their related metadata information using predefined data types. This significantly saves space and makes the data easier to be handled in a computer environment. According to the nature of the device capturing the point data, and as a result the number of attributes available, the LAS format provides different point data record formats. A lossless compression technique suitable for the LAS format is introduced in Isenburg [2012b] and is called the *LASzip* compressor. The output files from this compressor are called *LAZ* files. *LAZ* files require only 7 to 25 percent of the storage of the original file size.

Files have several advantages over database systems; specifically, available software, libraries and general purpose programming languages that can be used for processing, ease of use, compression schemes etc. However, since the amount of points easily reaches millions, on-the-fly indexing is

required in order to achieve efficiency while processing. This, together with the increasing availability of point cloud information, makes files an inefficient method of organisation, as being aware of which datasets are where and when is not simple. In fact, it requires extensive meta-data maintenance that can be complex due to the lack of standard formats. As a result, most of the time users resort to simplified raster transformations of the rich point cloud. This consequently leads to loss of information at the cost of easier management and analysis.

2.2.3 Management of point clouds using databases

Using a database for the management of point cloud information has been in the centre of the research for many years now. Especially the last years a lot of effort can be seen both in open-source and proprietary [DBMS](#), relational or not. Databases present many advantages compared to file-based organisations. First, with the increasing availability of low-cost sensors, available point cloud information for a specific area can increase dramatically. This means that updates (inserts or deletes) take place more often these days, which for a file-system organisation means continuous documentation of metadata. For a database, this is a generally supported function. Second, databases offer benefits like multi-user access, scalability and easier integration with other spatial data (vector or raster). These types of functionalities require (re-)development for file-based solutions [[van Oosterom et al., 2015](#)]. Instead of reinventing the wheel, [DBMS](#) can offer native support. Finally, file solutions do not support ad hoc queries. The development of a declarative language like the [SQL](#) available in the [DBMS](#) is thus required. Again, rather than reinventing the wheel and developing a new language to support queries, reusing [DBMS](#) solutions is more straightforward. The current methods for managing point cloud data inside a [DBMS](#) are described in [Chapter 3](#).

According to [Dobos et al. \[2014\]](#) some basic requirements that a relational point cloud database should hold are: spatial and non-spatial filtering, primary key searches, nearest neighbour determination, interactive visualisations and efficient loading, insertions and deletions that are supported by efficient indexing schemes.

2.3 POINT ACCESS METHODS

A lot has been said about the difficulty of managing point clouds. This difficulty, however, should not come at the cost of inefficient queries. For this reason, for many decades now, Spatial Access Methods ([SAM](#)) have been a very important topic in the research community. The term [SAM](#) refers to both *spatial indexing* and *clustering techniques*. Since in this thesis the main object managed is point clouds, only [SAM](#) relevant to points are described, namely Point Access Methods ([PAM](#)).

2.3.1 Indexing techniques

The purpose of spatial indexing, or indexing in general, is to support efficient data retrieval. In other words, indexes are important otherwise all rows within the table must be scanned during the execution of a query (full

table scan). This should be avoided as much as possible for the reason that read/ write operations on the disk are very expensive, especially for large tables. Depending on the type of disk used, the time needed to transfer a disk block from the disk to memory (access time) is hindered by certain types of delays. As a result, the number of blocks that need to be accessed during a database query should be the least possible.

The most well-known one-dimensional indexing technique is the B-Tree invented by [Bayer and McCreight \[1972\]](#). The B-Tree is a tree data structure and a generalisation of the binary search tree. What differentiates a B-Tree from a binary search tree is that it can store more than 2 keys per node and that it always stays balanced after random insertions or deletions. Assuming a B-Tree of order d , each internal (non-leaf) node contains maximum $2d$ keys and $2d + 1$ pointers. The number of keys within each node varies from node to node, but it is required that each node includes at least d keys and $d + 1$ pointers. This means, that a B-Tree node is always at least half full. An exemplary B-Tree is shown in [Figure 2.1](#). Leaf nodes contain the pointers to the actual data, usually stored elsewhere for a normal index.

Because of the balancing operations that take place during insertion and deletion, the B-Tree and its variants are the most widely used techniques for data organisation. The variant of the B-Tree found in the majority of databases today (like Oracle and PostgreSQL) is the B*-Tree. In the B*-Tree all the keys are stored in the leaf nodes while the rest of the levels are used as a guidance for fast search ([Figure 2.2](#)). In addition to that the leaf nodes are doubly linked which makes the B*-Tree very efficient for sequential accessing. A detailed overview of the B-Tree, its variances and description on insertion deletion operations is given in [Comer \[1979\]](#).

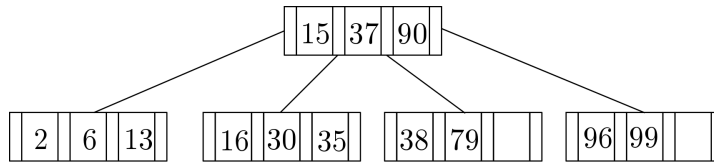


Figure 2.1: A schematic representation of the B-Tree data structure

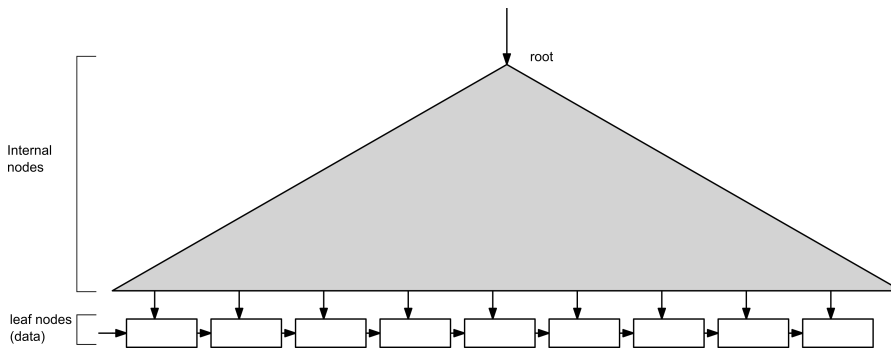


Figure 2.2: A schematic representation of the B*-Tree data structure (Adapted from [Comer \[1979\]](#))

Efficient access to spatial data is also very crucial. This is mainly due to their large volume, unstructured nature, and dynamic character. A naive, but yet popular, approach is the application of consecutive one-dimensional index structures, like the B-Tree. The efficiency of this approach is, however, not acceptable since each index has to be traversed independently from the

others, thus a high selectivity in one dimension does not reduce the rows having to be traversed at the other dimensions. This has resulted in the development of many spatial indexing techniques during the years [Gaede and Günther, 1998], the most important being the R-Tree where a grouping of spatially close objects takes place and spatial objects are represented with their minimum bounding rectangle. The R-Tree was developed by Guttman [1984] and similar to the B-Tree it is a balanced and dynamic tree. The tree is composed of internal nodes with entries in the form $(I, child - pointer)$, where $child - pointer$ is the pointer to the lower node (child) and I is the Minimum Bounding Rectangle (MBR) of all the rectangles of the children. The leaf nodes have the format (I, oid) , where oid is the address of the spatial object stored in the database and I is its MBR. Similar to the B-Tree, the R-Tree can contain M maximum number of entries (the branching factor) and at least $M/2$ entries (half-full). An R-Tree with branching factor of 4 is depicted in Figure 2.3. R-Trees can not only be used in 2 dimensions but also in 3, 4, etc.

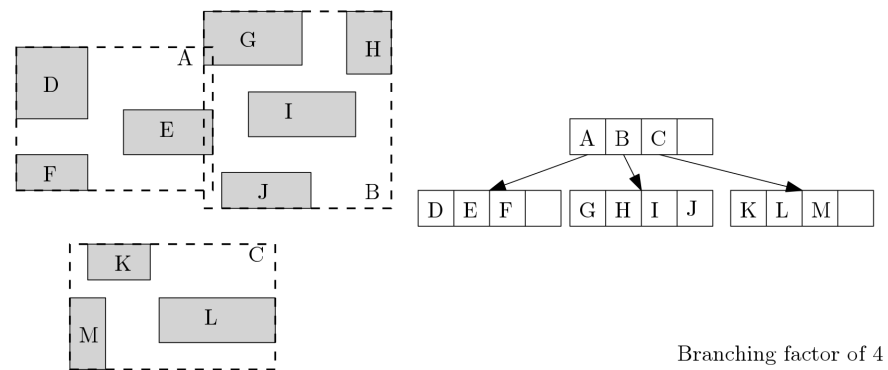


Figure 2.3: The R-Tree data structure of several objects with branching factor of 4

2.3.2 Subdivision of space: Quadtrees and Octrees

Quadtrees and Octrees are both hierarchical data structures that recursively decompose the space (2D or 3D respectively). Both structures are main memory data structures and thus directly not suited for disk storage (at least in their original format). A quadtree (2D) partitions a $(2^n, 2^n)$ array into four quadrants, while the octree (3D) partitions a $(2^n, 2^n, 2^n)$ array into eight octants. The types of quadtrees and octrees can be distinguished by [Samet, 1990]:

- the type of data they represent
- the decomposition process
- the resolution of the structure which can be variable or not.

The most commonly found variations of quadtrees in the literature are: the region quadtree and the point quadtree. The first is mostly used for approximating raster representations of polygons. For this, the raster is first approximated by a square. The square is then recursively subdivided into four smaller squares until each quadrant is inside or outside the polygon or it reaches the maximum resolution defined (Figure 2.4a). As the square gets decomposed, the quadrants receive a number usually in the order NW,

NE, SW, SE. The second quadtree variation is used to model point data by subdividing the space into four rectangles as in Figure 2.4b. The same variations can also be found in octrees. The decomposition of space using octrees is shown in Figure 2.5.

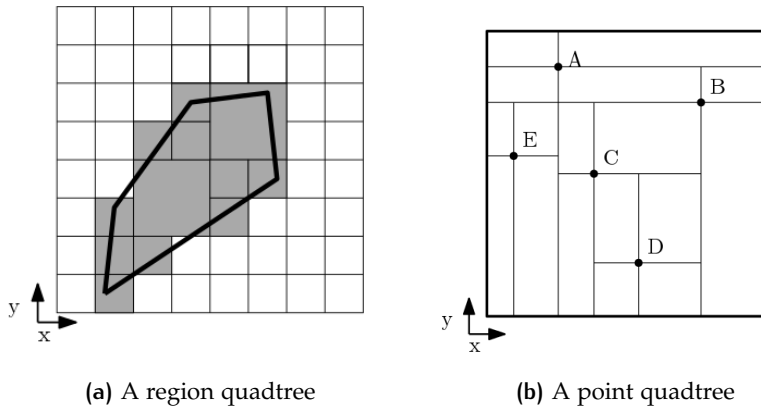


Figure 2.4: The two most commonly found quadrees in literature: the region quadtree and the point quadtree.

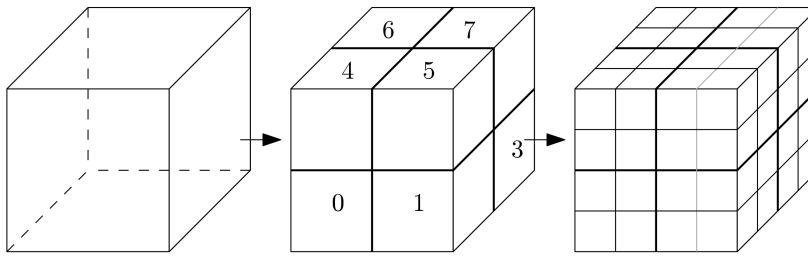


Figure 2.5: The octree data structure of level 0, level 1 and level 2

2.3.3 Space filling curves

One of the characteristics of spatial data in general, and point clouds in particular, is that there is no straightforward way to preserve the spatial proximity of the points in the storage medium. In other words, it would be ideal to store data which are close in space, also, close in the storage medium (cluster the data). This is very significant because it is very common that while requesting some data from the disk, several consecutive physical blocks are transferred as well (bulk transfer). If related information is stored on these contiguous physical blocks, then locating data on disk is no longer a bottleneck since the number of blocks that needs to be transferred is minimised. In a non-clustered situation blocks need to be accessed in a random order thus making access time significantly longer.

In order to take this bulk transfer of data into advantage, a primary key value that preserves locality is needed. A mapping from the higher dimensional space to a one-dimensional key can be performed using SFCs. Simply put, a SFC applies a linear ordering to a multi-dimensional domain. It

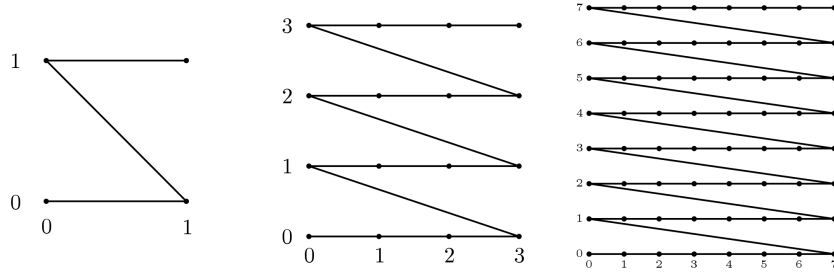


Figure 2.6: The row order curve for 2, 4 and 8 bit representation

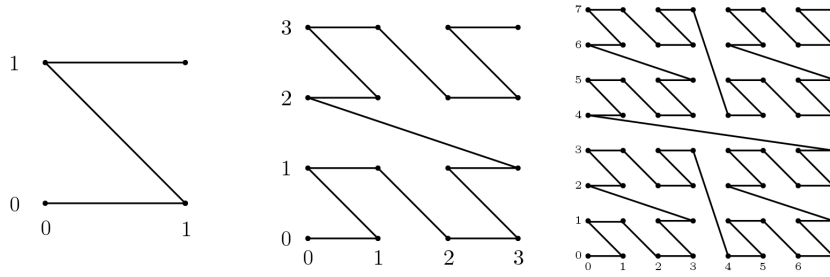
must be noted that [SFCs](#) are originally defined for discrete (raster-like) space. However, they can be applied to points by virtually overlaying a grid on top of the points such that each cell contains at most one point [[Abel and Mark, 1990](#)]. Therefore, [SFCs](#) are applied to integers.

The simplest [SFC](#) that can be identified is to traverse the points row by row ([Figure 2.6](#)). This type of space filling curve is called row order. This simplicity results in less clustering capabilities of the curve. As it is easy to imagine, as the number of bits increases, points in different columns but very close in reality, are found far away in the curve. Over the decades many space filling curves, each one of them preserving a different degree of proximity in the data, have been developed.

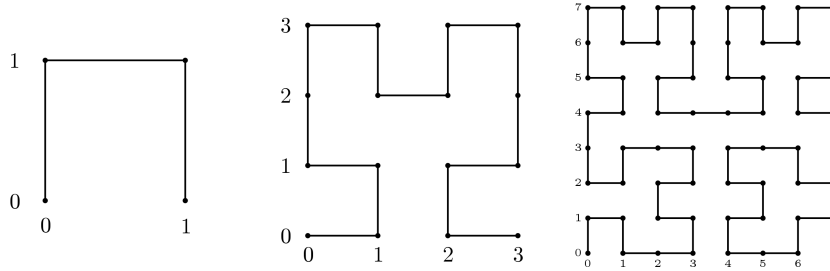
In [[Abel and Mark, 1990](#)] three properties of spatial orderings have been identified. These are:

- An ordering is *continuous* if, and only if, the cells in every pair with consecutive keys are four-connected neighbours.
- An ordering is *quadrant-recursive* if the cells in any valid sub-quadrant of the matrix are assigned a set of consecutive integers as keys.
- An ordering is *monotonic* if, and only if, for every fixed x , the keys vary monotonically with y in some particular way, and the other way around.

Two very commonly used orderings are the Morton [[Peano, 1890](#)] and the hilbert [[Hilbert, 1891](#)] curve. The morton curve (also called z-order or N-order curve) is based on interleaving the bits from the coordinates. For example, assuming a point with coordinates (9,12) its binary representation is (1001, 1100). By interleaving these bits we get the following binary number 11100001 which represents number 225. This number is the morton key of point (9,12). In mathematical terms, assuming a point P with binary coordinates $(X, Y) = (x_i x_{i-1} \dots x_0, y_i y_{i-1} \dots y_0)$ interleaving the bits is represented as $x_i y_i x_{i-1} y_{i-1} \dots x_0 y_0$. Same concept can be followed for higher dimensional points. The performance of the algorithm depends on the number of bits present in the used dimensions. The Morton curve for a 2-bit, 4-bit and 8-bit representation is shown in [Figure 2.7a](#). The hilbert curve maps multidimensional data into one dimension by following a recursive procedure. For the generation in 2D, first, the square domain is subdivided into four. Then for each one of these sub-squares, the first step is repeated and the previous version of the curve is rotated or reflected such that the pieces are connected to each other, see [Figure 2.7b](#). Similar concept is followed in higher dimensions. In 3D, for example, a cube is recursively subdivided and the curves are connected if they share a face.



(a) The Morton curve for 2, 4 and 8 bit representation



(b) The Hilbert curve for 2, 4 and 8 bit representation

Figure 2.7: The most commonly used Space Filling Curves

From the above description it is obvious that the Morton curve is generally easier to be constructed in comparison to the Hilbert curve and extended in higher dimensions. The ease of the construction, however, comes at the cost of preserving less proximity compared to the Hilbert curve. This characteristic is apparent by the presence of “jumps” in the Morton curve, which in other words means that two consecutive keys are not always neighbouring in space. On the other hand, the Hilbert curve has always a uniform length from point to point, thus containing no jumps. [Faloutsos and Roseman \[1989\]](#) showed that the Hilbert curve provides better results for range and nearest neighbour queries, thus significantly decreasing the cost of data retrieval from the database. Examples, of the different clustering capabilities of the above mentioned [SFC](#) are presented in [Figure 2.8](#). As it is visible from the figure, the row-order and Morton curve require on average a bigger amount of continuous ranges to be retrieved. The Hilbert curve, on the other hand, requires on average the lowest amount of ranges. The figure also corroborates that no ordering exists that preserves the total proximity.

The advantages of [SFCs](#) for the indexing of multi - dimensional data are discussed in [Lawder and King \[2000\]](#). The most important advantage is that scalable one-dimensional access methods (like a B-Tree) can be applied to the multidimensional point data. These structures make it possible to dynamically balance n-dimensional data in a [DBMS](#). This together with the clustering capabilities of the curves will eventually result in points clustered according to their spatial proximity in the n-dimensional space. In addition to that, since space filling curves are reversible -meaning, it is possible from the full key to find the original coordinates-, storing the actual coordinates is not necessary, which significantly reduces the storage requirements.

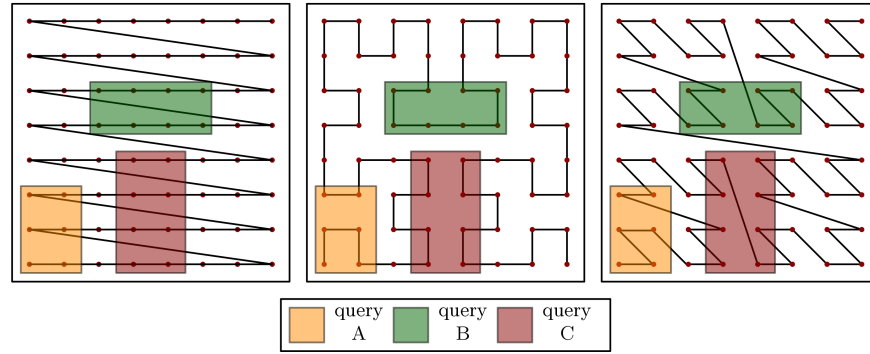


Figure 2.8: A comparison of the clustering capabilities of three Space Filling Curves (from left to right): Row order, Hilbert, Morton

Quadrees and space filling curves

The decomposition of space using quadrees or octrees (or in general 2^n -trees) can also have another use; they can be combined with a SFC to map the n -dimensional space into a one-dimensional one. In this kind of format, a 2^n -tree can be used together with a B-Tree to index a collection of points stored in the disk.

Morton keys in particular have a very strong connection with the quadrees. Their connection was established by Gargantini [1982] resulting in the linear Quadtree and Abel and Smith [1983]. Gargantini [1982] presents the mapping from x, y coordinates to quadrants, whose code is represented using base 4 numbers. The property is extensible to higher dimensions because of the quadrant recursive properties of space filling curves. The same characteristic is also valid for the Hilbert curve.

3

RELATED WORK

The [RELATED WORK](#) chapter aims to provide the available methods currently used in contemporary databases or in the academia to manage point clouds, dynamic or not. Therefore, the chapter is organised as follows: [Section 3.1](#) provides a short overview of existing point cloud data management implementations found in several [DBMSs](#). Then [Section 3.2](#) focuses on spatio-temporal data management used in Geographic Information Systems, multidimensional databases and of course the available point cloud implementations mostly found in the academia. The chapter ends with a short conclusion ([Section 3.3](#)) based on the aspects examined in the chapter.

3.1 MANAGEMENT OF POINT CLOUDS IN DBMS

The management of point clouds in [DBMS](#) has been in the centre of the research for many years already. The first research efforts started by investigating the re-use of the existing simple feature geometry. For example, [Zlatanova \[2006\]](#) argued that point clouds could be either stored with already existing data types, *POINT* or *MULTIPOINT*, or with user defined types. Storing point clouds using *POINT* data types was, also, proposed in [Höfle et al. \[2006\]](#), while [Wijga-Hoefsloot \[2012\]](#) proposed using the *POINTCLUSTER* data type, which essentially is a multipoint collection. When using the *POINT* data type each point is stored in one row. On the other hand, in the *MULTIPOINT* approach a number of points (most of the times the whole file) are stored as a group of points. The former can introduce a significant storage overhead. The latter has technical boundaries, i.e. 1,000,000 points in Oracle [[Kothuri et al., 2007](#), p. 744], and makes searching and accessing a subset of the group impossible, as the whole *MULTIPOINT* object has to be loaded into memory [[Ott, 2012](#)]. In addition to that, geometry and attributes need to be separated in the *MULTIPOINT* approach thus making updates, insertions and deletes complicated.

Currently, the database community provides several approaches for point cloud data management. In their majority three storage models can be distinguished ([Figure 3.1](#)). The first model is based upon the organisation of points in *blocks*, meaning in groups of spatially close points. The second organisation is the *flat* table model, where each point is stored separately in one row. The third organisation is a hierarchical model, where points are organised in a tree structure.

In this section the existing implementations of several well-known systems like Oracle¹, PostgreSQL², MonetDB³, etc. are presented shortly. In all of the following systems, the time dimension can be considered only as an attribute and is not part of the main organisation.

¹ <https://www.oracle.com/database/>

² <https://www.postgresql.org>

³ <https://www.monetdb.org/>

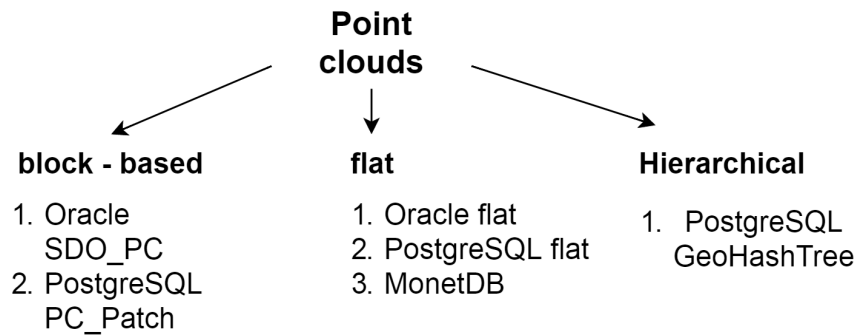


Figure 3.1: The major storage models for managing point clouds in DBMS environment and their implementations in well-known systems.

3.1.1 Oracle SDO_PC

For blocked organisation, Oracle Spatial and Graph⁴ provides the *SDO_PC* and *SDO_PC_BLK* data types which respectively represent the logical object and physical storage. *SDO_PC* stores the metadata information of a point cloud like the name, the spatial extent, the number of dimensions (spatial and non spatial), the resolution, the name of the *SDO_PC_BLK* table that contains blocks, the parameters for partitioning etc. The *SDO_PC_BLK* is the physical storage table with the block information and includes the spatial extent of the block, the resolution, the number of points, the *Binary Large Objects* (BLOB) that include the points, etc.

The process for creating storing a point cloud with the *SDO_PC* data type first requires the use of a staging table. This can be a normal table or an external table. After all the required points are loaded, the blocks are being generated using the *SDO_PC_PKG.CREATE_PC* procedure. The default blocking method is performed using a R-Tree. It is, however, also possible to use a Hilbert spatial partitioning, meaning to group points that are closer in the Hilbert curve [Godfrind and Horhammer, 2015]. The former blocking method is in general slow for large datasets. In case groups of LAS files are available, the previous steps can be replaced with the blocking scheme from PDAL⁵ and the available Oracle drivers.

So far, the Oracle *SDO_PC* does not offer any update mechanism. This means that updating a point or inserting new points is not available in the current versions. Consequently, the user has to generate the blocks from scratch. This is not optimal in a dynamic point cloud data management environment where the user will ultimately want to stream time varying point clouds and organise them in an optimal way very often.

3.1.2 Oracle Flat

The second option for storing point clouds in an Oracle database is by using a flat (normal) table approach. Each point in the point cloud is stored in a single row using common data types, like *NUMBER*. As this method simply uses a normal (heap) table, updates are easily performed.

Database specific hardware like the Oracle Exadata Database machine [Oracle, 2015] are more optimal for this type of storage. This hardware is

⁴ <http://www.oracle.com/technetwork/database-options/spatialandgraph/>

⁵ <http://www.pdal.io/>

designed to offer maximum performance for running the Oracle Database. Its performance for point cloud management was tested in [van Oosterom et al. \[2015\]](#).

3.1.3 Oracle Hybrid

A third option for storing point clouds can be achieved by using the new Hybrid solution of Oracle. This solution involves using an [IOT](#) and blocks that are spatially partitioned using a space filling curve approach [[Godfrind and Horhammer, 2015](#)].

Within this solution, the points are stored in an [IOT](#) same as with the Oracle flat solution, however, each point also contains the Hilbert key, pyramid information and a reference to a block. The key is based on the Oracle implementation of the Hilbert curve. So far (November 2016) the key can be calculated only in 2D using the function `sdo_pc_pkg.hilbert_xy2d` or `sdo_pc_pkg.generate_hilbert_vals`. The blocks are stored in a `SDO_GEOMETRY` table, which as a whole represent the Hilbert R-Tree of the points.

This method is undocumented and therefore it is not known if it provides update mechanisms. Updating and inserting, however, seem possible if the new and old points are stored also in a staging table and the generation of the Hilbert keys and blocks is repeated from scratch in regular moments in time. Nevertheless, the performance of this method in terms of time is not known. One important limitation is that only 2D Hilbert keys can be generated so far. Using higher dimensional space filling curve would allow the inclusion of the time dimension in the key.

3.1.4 PostgreSQL PC_Patch

PostgreSQL provides point cloud support through the `pgpointcloud` extension developed by [Ramsey \[2014\]](#). The idea behind the development was that points should not be stored as PostGIS `POINTS`, but rather organised into patches (the equivalent of blocks in Oracle). For this reason, the `PC_Point` and `PC_Patch` data types were developed. The data types were designed to support integration with PostGIS⁶ using the `pointcloud_postgis` extension.

Since there is no fixed way of how data should be stored inside the database or how many dimensions each point cloud object has, a description of the contents it represents described in an XML schema document [[Ramsey, 2014](#)] called `pointcloud_formats`. The actual loading of the point cloud into the database can take place either by making use of well-known binary (WKB) objects or with the [PDAL](#) driver for `pgpointcloud`.

This organisation was used and to some parts extended by [Cura et al. \[2015\]](#). In their paper the authors describe a complete point cloud management system supporting metadata, compression, filtering, fast loading and processing, as well as integration with vectors and raster objects.

Space filling curve approaches are not available in the current implementation of the point cloud support of PostgreSQL. The only available spatial clustering functionality in PostGIS is the `GeoHash`⁷ function that is based on the Z-order curve. Furthermore, it is not documented how new insertions of data can be handled without creating overlapping blocks.

⁶ <http://postgis.net/>

⁷ <https://en.wikipedia.org/wiki/Geohash>

3.1.5 PostgreSQL flat

The implementation of the PostgreSQL flat storage model follows the same approach as in the case of Oracle. The data are loaded in a table and indexed using a B-tree index in the X and Y coordinates respectively. Although loading can be remarkably faster than the block-based approach, the major drawbacks become evident during querying. This happens because a B-tree index on X and Y is not an appropriate indexing method [van Oosterom et al., 2015].

An improvement in the flat table organisation is included in van Oosterom et al. [2015] where the authors used a space filling curve approach to spatially cluster points. Since space filling transformation functions are not offered in PostgreSQL, the transformation to the 64bit Morton key takes place outside the database. Inside the database, the X, Y and Z coordinates and the Morton key are stored in a flat table. During the querying step, two different phases take place; first, using the relationship between the Quadtree and the Morton curve (Figure 2.3.3) an approximation of the Morton ranges belonging to the query region is given. As a second step, this approximation is refined using the standard PostgreSQL functionality (i.e. point in polygon operation). Compared to the pure flat table approach, the query response times become more constant meaning that they are independent from the size of the database. They only depend on the size and complexity of the query region, as well as the size of the output.

3.1.6 GeoHashTree in PostgreSQL

The GeoHashTree [Sabo et al., 2014] is a generic data structure that was developed and prototyped with PostgreSQL in order to support point clouds in various types (unstructured points and rasters) and data of various resolutions. The GeoHashTree depends on the geocoding system geohash that encodes coordinates into a character string.

The GeoHashTree structure as the name implies is a tree structure where all nodes that have the same parent share the same geohash prefix. In this way, points close in reality share the same parent. Each parent, apart from the geohash value, also includes some statistical information related to the attributes. Each new point that is added to the tree is first checked for its resolution and stored in the appropriate level. In a second step, two optimisations can take place according to the attributes stored in the tree: migration of attributes and change of type. In the first method, points that share the same attribute value, will have the attribute migrated to the parent level. The second method is a compression mechanism applied to numerical attributes. Depending on the variability of the attribute, its data type might change in order to reduce storage. The GeoHashTree is now part of the *pointcloud* extension of PostgreSQL and is used as a compression mechanism.

3.1.7 MonetDB

MonetDB deviates from the Relational Database Management System (RDBMS) approaches presented previously in that it is part of the Not only SQL (NoSQL) databases that emerged with the development of the Internet. These databases have the characteristic that they do not organise data using relational tables. There are four types of NoSQL databases: key-value stores, doc-



Figure 3.2: The effect of using bigger or smaller block sizes on the selected area obtained. Bigger blocks can even double the amount of the selected area, contrary to smaller blocks that can approximate the area more closely.

ument, column and graph databases. MonetDB is a column-store database meaning that it stores data in using columns (rather than rows in [RDBMS](#)). In the case of the spatial data i.e. X, Y, Z coordinates, MonetDB stores each coordinate using a separate column. Same goes for the attributes attached to each point. This approach is an equivalent of the flat model, since each point is stored as a separate object.

MonetDB has been tested together with some of the previously mentioned approaches in [Martinez-Rubi et al. \[2014\]](#) and [van Oosterom et al. \[2015\]](#), where it presented some scalability issues compared to the block approaches. However, in [Martinez-Rubi et al. \[2015\]](#) this organisation was significantly improved by using a space filling curve. Two different approaches were followed: 1. the same approach followed in the Morton-added approach of PostgreSQL flat ([Section 3.1.5](#)) where the data are clustered using the 2D Morton key but the original X, Y, Z coordinates are also stored and used for the querying, and 2. the Morton-replaced approach where only the Morton key and the Z are stored and the X, Y are decoded from the code. The Morton-replaced approach showed a decrease in storage, while the Morton-added showed an increase due to the extra column used to store the Morton key. This organisation significantly improved the scalability of the queries, although the responses are not completely constant as the size of the database increases i.e. the number of points increases.

3.1.8 Comparing block and flat based organisation

As a conclusion, a comparison between the flat- and block- based storage organisations is provided. In general, the block - based approach presents better scalability of query response times, less overhead and potentially better compression than the flat table approach [[van Oosterom et al., 2015](#)]. The reason for this is the reduced amount of rows stored in the table that consequently reduce the size of the table and the index. Nonetheless, the scalability depends on the block capacity. If the block size is relatively big, although the size of the table is reduced, at the query stage more points will have to be decompressed and checked for actually belonging to the query area. This happens because the amount of Input/ Output (I/O) for a query in the blocked organisation is directly proportional to the amount of blocks that the query region intersects with, which depends on the area of query region. An example is presented in [Figure 3.2](#). For the same query area,

the left case will return more points, compared to the case on the right. In either way, blocks offer notably faster access times than searching through every single point. However, as shown before flat tables can be remarkably improved using a [SFC](#) approach.

Another parameter that worsen the quality of the block-based approach are overlapping blocks. Overlapping blocks affect the amount blocks that will have to be unpacked and examined during the query process. Finally, adding new data in the block-based approach is not as straight-forward as insertions in normal flat tables.

3.2 SPATIO – TEMPORAL DATA MANAGEMENT

The three geometrical dimensions (X, Y, Z) have been well studied and researched when it comes to the management of point clouds. Nonetheless, it is very often the case that repeated scans of the same area take place in regular periods of time. This amount of acquired data has made these datasets rather spatio-temporal datasets that need to be optimally managed in a [DBMS](#) environment. Spatio-Temporal Databases ([STDBMS](#)) are database systems that manage both the spatial and temporal component of datasets using an integrated approach. This integration of space and time is currently not available in the previously mentioned approaches. On the contrary, the majority of the structures are focused more on creating an inventory of point cloud datasets and do not support updates or further insertions for a specific point cloud object.

In the domain of managing spatio-temporal or dynamic point clouds not a lot of research exists so far. In current implementations time is considered as an attribute and not as another dimension. In this way, time does not have an effect in the organisation of the data. Since the topic of managing spatio-temporal point clouds is in general new, approaches from other domains are also being considered. These are the domain of Geographic Information Systems ([GIS](#)) ([Section 3.2.1](#)) and multi-dimensional databases ([Section 3.2.2](#)).

3.2.1 Time in Geographic Information Systems

Time has received a lot of attention within the fields of [GIS](#) and [STDBMS](#) for many decades. The first steps towards the integration of space and time started with the separate research on temporal and spatial databases. However, as the amount of literature shows, the integration of spatial and temporal data types is far from trivial. A more comprehensive overview of spatio-temporal models can be found in [Pelekis et al. \[2004\]](#). For many applications today, time is modelled according to the ISO 19108 standard [[ISO, 2008](#)].

One of the challenges faced with spatio-temporal [GIS](#) and databases is the different nature and semantics of the two concepts that need to be taken into account. The data models that have emerged through the years vary, among others, in two aspects:

1. the temporal resolution, meaning the way of partitioning the line of time, and
2. the granularity of data related to time, which denotes in which level of the spatial data (here points) the time dimension is added, i.e. to the whole or subset of a dataset or the attribute level (of the points).

These two aspects should, also, be considered for the problem of managing spatio-temporal point clouds. Depending on the density of the measurements the temporal resolution can be in decades, years, hours, seconds. Also, the time component could be part of the whole dataset -if the frequency of the measurements is quite discrete i.e. every year- or part of each point when the measurements take place in seconds, hours etc.

Spatio-temporal data models

The simplest and most fundamental model for representing time in a GIS is the snapshot model by [Langran and Chrisman \[1988\]](#). The modelling of space takes place in time layers, in other words, it represents the state of the objects at different times. The different time layers are independent from one another. The drawbacks of this model are duplication of the data, difficulty in detecting the changes from one moment to the next one and difficulty in applying integrity rules [[Langran, 1992](#)].

Other commonly used data models are based on time stamping every object of the database with two time stamps, one for the creation (tmin) and one for the termination of the object (tmax). This approach is also known as state based modelling. To differentiate the objects that are still valid, a special value is given to their tmax attribute e.g. *MAX.TIME*, *CURRENT* etc. This type of model is easy to be implemented and to obtain the state of an object at a specific moment in time. However, at the same time, it is not possible to query what changes occurred and why [[Pelekis et al., 2004](#)].

The deficiencies of the time stamping models is dealt within the event-based models [Peuquet and Duan \[1995\]](#). These spatio-temporal models are based on the explicit management of the events that lead to changes. Within these models, an inventory of the transactions is kept. To be able to find out the historical states of a dataset, a traversal of the events needs to take place.

When applying the previously mentioned data models to the concept of point clouds, certain remarks can be made. The snapshot model is easily applied to point cloud datasets that have a discrete time resolution (e.g. year). One of the problems that arise from this model, as with the GIS data, is the redundancy and the difficulty to find out the changes between the two states. Moving to the state based models, it is easy to realise that they are difficult to be applied to point clouds. This model requires change detection techniques to identify which parts of the point cloud are unchanged or not. Such a method is described in [Section 3.2.3](#). Finally, event based models may not fit for point cloud storage as some times it is not known a priori what type of events occurs due to their continuous representation of space.

Spatio-temporal indexing and query

Another challenge in the spatio-temporal research is the efficient querying both in time and space (spatio-temporal query). The spatial component is usually specified by a polygonal region (in 2D) or a polyhedron (in 3D). The time component is similarly specified by a time range. Optimised searching in the three dimensions (2D space and time) or four dimensions (3D space and time) is possible only if the 3D or 4D data types are combined with indexing and clustering techniques. An overview of spatio-temporal access methods can be found in [Mokbel et al. \[2003\]](#) and [Nguyen-Dinh et al. \[2010\]](#).

A simple solution to the spatio-temporal indexing is applying a R-Tree extended to multidimensional indexing. This is called a 3D RTree [[Vazirgiannis et al., 1998](#)] and it considers time as another dimension together

with the X, Y coordinates. A drawback of this technique is that time queries are dependent on the number of historical objects stored in the database [Mokbel et al., 2003]. Another possibility is the RT-Tree [Xu et al., 1990], where each node of the R-Tree is associated with time ranges. Because time has a secondary role, time queries can be inefficient within this structure. Xu et al. [1990] also introduce the Multiple R-Tree (MR-Tree). This structure combines the R-Trees of each time stamp in order to avoid redundancy for the unchanged regions. A similar idea is implemented with the Historical R-Tree (HR-Tree) [Nascimento and Silva, 1998]. Both structures perform efficiently for time slice queries (data for a specific time), while time range queries are inefficient because of this multi- R-Tree structure.

3.2.2 Storage and retrieval of multi-dimensional data

Multidimensional databases exist in many different forms and shapes, consisting of logical entities that contain a set of attributes. These attributes are required to be stored in an efficient way that will enable fast retrieval of the data based on values (or ranges of values) of one or more of the attributes available.

Multidimensional data can be easily compared to multidimensional points in the nD space, where each attribute corresponds to one dimension. The bottleneck with multidimensional data, as with multidimensional points, is their large size and inefficient access methods that require the maintenance of primary and secondary indexes in order to support retrieval according to values other than the primary key. The ultimate goal within multidimensional databases is to reach the superior performance of one-dimensional access methods like the B-tree and its variants. In such systems it is also required that primary key values determine the placement of the records in the disk storage of the computer, thus lead to clustering of the records that share similar characteristics.

The majority of the organisation methods found in the literature perform a kind of partitioning to the data (like the blocks used for point clouds). Those partitions are, usually, in hyper-rectangular form with the index operating on those partitions created. The partitions are significantly less than the original data, thus enabling fast retrieval. The downside of such a structure lies in the update or insertion process that might lead to rectangles containing too much data which need to be further split. Another issue are overlapping hyper-rectangles leading to the retrieval of significantly more data that need to be further refined in a second step. Multidimensional indexing structures widely used are the:

KD-TREE implemented by Bentley [1975]. A special case of the Binary Search Tree that recursively subdivides the n-dimensional space into sub-spaces (or half-spaces) by using $(d - 1)$ -dimensional hyperplanes. The kD-tree is a main memory indexing structure. The data are then stored as leaf nodes in the tree. Searching and inserting new points is a straightforward action. However, the index is dependent on the order of insertion of the point data.

KD-B-TREE implemented by Robinson [1981]. The kD-B-tree combines certain aspects of the kD-tree and the B-Tree (Section 2.3.1), making this indexing structure suitable for disk storage. In particular, space is subdivided as in the kD-tree, however, keeps the index balanced (like the B-tree) with each sub-space associated with tree nodes. The points are

stored in the leaf nodes. A disadvantage of the kD-B-tree concerns the significant reorganisation when inserting new data.

GRID FILE implemented by [Nievergelt et al. \[1984\]](#). This access method divides the space using a non-regular grid. The major advantage of the Grid File is data points can be found with no more than two disk accesses. A disadvantage is the exponential directory growth rate during the insertion operation.

R-TREE The R-Tree is described in [Section 2.3.1](#). A disadvantage of the R-Tree concerns the overlapping rectangles that are to a certain extent tackled in some of its variations.

Multidimensional organisation is also possible following a different approach. This approach is used in [Lawder \[2000\]](#) who uses the Hilbert curve to map multidimensional points into one dimension. His implementation covers data up to 16 dimensions, which can very well relate to the number of dimensions in a point cloud. In contrast to the previous approach, the curve is divided into sections and each one of them is assigned a section in the disk. Another space-filling related approach is followed by [Terry et al. \[2011\]](#), who use a structure of variable granularity depending on the density of the points in the multidimensional space. Space with less density is represented with larger regions (partitions) while highly dense areas can reach up to the finest resolution defined.

3.2.3 Point cloud implementations

Spatio-temporal point cloud implementations are present in the literature, although limited in number. A [NoSQL](#) spatio-temporal implementation using the Apache Accumulo software is presented in [Fox et al. \[2013\]](#). The spatio-temporal index structure developed interleaves the parts geohash of the (x,y) point with parts of the string representation of the date. A geohash is an implementation of the Morton space filling curve, ultimately generating a recursive quadtree of the world. As an example of their used structure we assume the geohash *u01mtw0* and the date *May 7, 2012 at 10:17pm*, the final interleaved structure will be *u201205 – 01m – tw00722*. The key is a string and it is indexed lexicographically. This type of structure is however, platform specific, as it accommodates the key structure of the Accumulo system which is a key value store. The results show that the spatial and time complexity of the queries increase the query response times. Nevertheless their system provides efficient insert and update operations.

Another spatio-temporal implementation closely related to point clouds can be found in [Tian et al. \[2015\]](#) where the authors apply a [SFC](#) (Morton) to efficiently organise 3D points (lat, lon, t) in a 3D (Morton) R-tree which is stored in a relational database. The methodology followed by the authors is the following: First, the coordinates (latitude, longitude, time) are linearly transformed to obtain integers and the Morton key is calculated. Second, the spatial objects are sorted based on their Morton key and, based on the fanout and degree, the Morton R-Tree is constructed. Third, the [MBRs](#) of the 3D points are stored in the [RDBMS](#). To query this database, the [MBR](#) of the query is calculated. The minimum and maximum of the [MBR](#) are then translated into two Morton ranges which are used to traverse the tree. While traversing the tree the relevant points are returned to the user. Their prototype was compared with a spatial database and their organisation gave

faster queries and scalability with concurrent queries. One drawback is that this approach lacks the ability to insert new data.

Finally, a different approach is described in [Richter and Döllner \[2014\]](#). The authors also driven by the increasing availability of point cloud data describe the architecture of a system that deals with data coming from varying sensors and devices, handling also database updates. In their proposed system the time dimension is handled by an "incremental database update process". This means that the unchanged parts between consequent timestamps are not repeated in the database. To achieve this, change detection techniques need to take place before the data are loaded into the database. Two change detection approaches are proposed to be used, a point-to-point for large scale changes or point-to-splat for small scale change detection. After change detection has been completed all points changed are integrated into the database. Points already present have their time stamps and frequency information updated. This implementation achieves reduced storage requirements and allows efficient change detection from one moment to another. However, this makes it hard to restore what happened at a specific moment as a number of change entries have to be applied to the initial state.

3.3 CONCLUSIONS

The most straightforward conclusion that can be made from this chapter is that contemporary [DBMSs](#) do not provide methods for managing dynamic point clouds. They are, as a result, suitable for static applications. Spatio-temporal implementations can, nonetheless, be found in the academia. The [NoSQL](#) implementation although very relevant, might not be as suitable for a relational database that does not follow the key structure of the used platform. The 3D Morton R-tree method, does not allow further insertions of data and the incremental database update process is mostly efficient for change detection queries, but not queries that request points from a specific time moment. However, we can see a trend: [SFCs](#) are a very good starting point for managing dynamic point clouds.

4 | A SPACE FILLING CURVE APPROACH

The [SPACE FILLING CURVE APPROACH](#) chapter provides the details of the proposed methodology used within this thesis document. [Section 4.1](#) gives the motivation for using a [SFC](#) for the management of dynamic point clouds. [Section 4.2](#) provides the description of the [SFC](#) approach and its variances. [Section 4.3](#) gives an overview of the possible alternatives that can be followed when loading the data. Finally, [Section 4.4](#) provides the methodology used for querying the data using this improved organisation.

4.1 MOTIVATION

The main incentive behind using a [SFC](#) approach for the management of dynamic point clouds stems from the limitations present in the current point cloud data management solutions. These data structures, although provide efficient storage and query response times, they are either not designed for applications that have a dynamic character, are not very flexible in answering certain questions or do not allow further insertion of new data.

More specifically, the methods presented in [Section 3.1](#) consider point clouds as static objects, with the time component taking no part in the organisation of the points. Geographic information, however, is highly correlated to the moment in time in which the data was captured and, in many applications time is as selective as space or needed in integrated space - time selections (change detection). This suggests that efficient space-time selections require a different approach than the available ones. The flat model in its raw format presents deficiencies that have to do with the scalability of the model. Nevertheless, it is a very flexible solution. It can be either used as a final storage model or as an intermediate stage in order to efficiently create blocks (staging table) for example by sorting the points based on their position on the [SFC](#) [[van Oosterom et al., 2015](#)].

The methods in [Section 3.2](#), coming from the spatio - temporal domain, provide more insight into what a suitable solution should look like. There it becomes clear that a [SFC](#) approach is a logical way to proceed for the problem at hand. An improved organisation using [SFCs](#) was presented in the work of [Martinez-Rubi et al. \[2015\]](#) and [van Oosterom et al. \[2015\]](#). This organisation achieved constant queries between the different sizes of the datasets used. Therefore, building on the work of the previous, an improved spatio-temporal data management approach is investigated within this thesis.

The implemented spatio-temporal data management approach should fulfil certain requirements and characteristics [[Gaede and Günther, 1998](#)]:

- It should be *dynamic*. This means that the chosen data structure should support insertions, updates and deletions that do not degrade in time.
- It should efficiently support *operations* other than retrieval of the data, for example insertions, computation of normal vectors etc.

- It should present *scalability*. With the increasing availability of growing point clouds, the data structure should be able to adapt well to database growth.
- It should be *efficient* in terms of time and space. The former means that spatio-temporal searches should be fast. Because fast is a subjective term, the approach should minimise as much as possible the number of disk accesses.

Why space filling curves?

SFCs have a special role in this thesis. The motivation for using clustering techniques, in general, and SFCs, in particular, is threefold. First of all, they lead to *dimensionality reduction*. This is an important characteristic since a multi-dimensional point can be described with a single value, making it feasible to apply classical B-Tree indexing techniques. SFCs are of great value for multidimensional indexing [Lawder, 2000]. Second of all, a full resolution space filling curve can *eliminate the need to store the actual dimensions* (X, Y, Z, t coordinates) inside the database. Consequently this means that the sizes of the tables are reduced, which is very important for large datasets. The only requirement is that a function that decodes the keys into the original dimensions is needed. Third of all, SFCs preserve to a certain extent the *proximity* of the points. This characteristic can be used as a clustering mechanism on the physical level, minimising thus the I/O cost for fetching the data.

Within this thesis the Morton curve is used. The reasons for doing so are the ease of generating Morton codes (just bitwise interleaving) and the ease of extending it to higher dimensions. This, however, will come at the cost of less spatial proximity, e.g. compared to the Hilbert curve.

4.2 STORAGE MODEL

Databases reside in computerised environments and are physically stored on a computer storage medium. Computer storage is classified into *memory* and *disk storage*. The tasks of the DBMS software is to fetch, update and analyse the data whenever this is requested. The database data, which resides in the disk storage, needs to be located on the disk, read into the memory in order to be processed and if needed written back to the disk. This mandates a storage model that can efficiently find the data when it is needed.

However, the movement from the disk to the memory suffers from certain delays. In case a Hard Disk Drive is used the delays include: 1. the *seek time*, which corresponds to the time it takes for the disk controller to position the read/write head on the correct track. 2. the *rotational delay*, which is the time it takes for the requested block to rotate in the right position under the read/write head. 3. the *block transfer time*, which is the time required to transfer the requested block. From those delays, the two first are the most expensive. For this reason, many manufacturers offer bulk transfers, in which consecutive blocks -apart from the needed one- are transferred to the memory. Therefore, intentionally placing data that are typically requested together on contiguous blocks can take advantage of bulk transfers and thus provide better query response times.

Structuring space and time in an integrated approach is, nonetheless, not a trivial problem as two **contradictory requirements** need to be considered during the conceptual design. From the paragraph above it becomes clear that points close in space and time should be stored (to some extent) in contiguous blocks in disk storage for fast spatio - temporal retrieval. However, because our chosen organisation should be dynamic and support further insertions, we also require that the already organised points are not reorganised (at least not massively or too often) when inserting new data to achieve fast loading. This is where the clash of requirements occurs. To preserve space - time proximity between the new and old data, the new points will have to be inserted between the already organised points. This, however, is a very costly operation and it should be avoided as much as possible, especially in a highly dynamic environment in which data are streamed every minute or hour.

Therefore, our method suggests to explore to organisations of space and time. The two options represent two extremes of the space - time continuum that can be achieved by appropriately scaling the time dimension relative to space. But before introducing the two approaches used in this thesis, it is important to specify the most important queries that need to be answered when managing these dynamic point clouds.

4.2.1 Query requirements

To be able to design an organisation that is suitable for dynamic point clouds, it is important to define the most prominent search conditions that are expected to be asked by the user. Having these clearly identified can aid the database designer to choose the most suitable organisation that speeds up their query execution. This facilitation, however, may come at the cost of less efficiency when query predicates different than these prominent queries are asked.

Our use case, as presented in [Chapter 1](#) and in [Section 5.1.3](#), comes from the coastal monitoring domain. To identify the most prominent queries used a literature research was conducted and the user (Deltares) was interviewed. This process gave three important queries:

ONLY SPACE QUERIES These types of queries request all spatio-temporal objects located in a specific area (over a complete time range).

SPACE - TIME QUERIES These types of queries request all spatio-temporal objects located in a specific area during a specific time range.

ONLY TIME QUERIES These types of queries request all spatio-temporal objects located in a specific moment in time or during a specific time range (for the whole spatial domain).

The former queries are schematically presented in [Figure 4.1](#). Each grey plane represents point cloud data at a specific moment in time.

4.2.2 Integrated approach

The first approach, the **integrated approach**, covers the complete integration of space and time. This is achieved by giving space and time an equal part within the [SFC](#). In terms of Morton keys, space (x,y[z]) and time are interleaved, resulting in a 3D or 4D key. Depending on the treatment of z

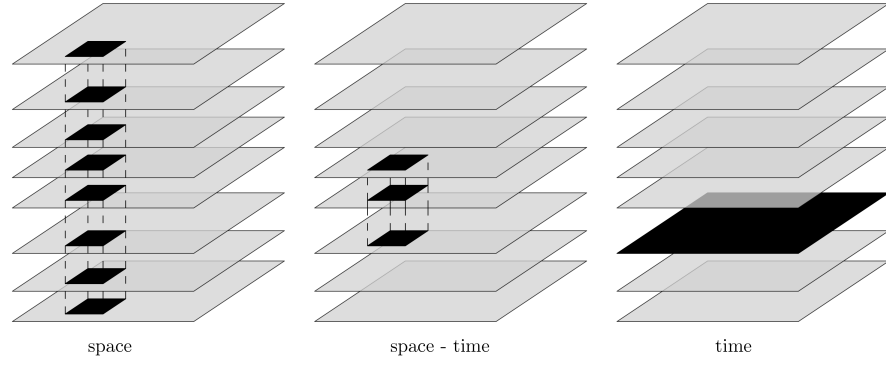


Figure 4.1: A schematic representation of the most important queries when managing dynamic point clouds. From left to right: only space queries, space and time queries and, only time queries.

the integrated approach can be divided into two storage models: one where z is treated as an attribute and one where is part of the [SFC](#) key. Inside the database only the full resolution key is stored. Spatio-temporal locality is preserved to the extent that the used [SFC](#) preserves locality.

In theory, this solution will not completely fulfil the second requirement presented at the beginning of the section. When inserting new data, the space - time points will have to be inserted between the already organised points in order to remain physically ordered. An alternative that to some extent reduces the effects of reorganisation during new data insertion, is appropriately scaling the time dimension. This parameter is of great significance because of the inherently different nature of the spatial and temporal dimensions. Scaling, actually, is the factor of how much time is integrated with how much space within the [SFC](#), leading to a different degree of proximity in the data. The scaling is defined such that the most important query still remains or becomes more efficient after its application. In our case, the scaling should offer a compromise between the three types of queries considered.

Concerning the queries mentioned in the previous [Section 4.2.1](#), space and space - time queries are expected to present efficient response times. On the other hand, time queries will be slightly more inefficient given that points coming from the same time will be stored in many different blocks which will need to be fetched and unpacked. How efficient or inefficient each query is is directly related to the scaling of space and time.

For a schematic representation of the integrated approach [Figure 4.2](#) is provided showing a 3D [SFC](#) where z is an attribute. Observe that areas with similar colours will be stored closer in the disk than areas with different colours (i.e. blue, red).

4.2.3 Non-integrated approach

The second approach, the **non-integrated approach**, resembles the snapshot model of [Langran and Chrisman \[1988\]](#). Within this approach space and time are loosely integrated, with time dominating over space. Inside the database this is achieved by storing time in a separate column and including only space $(x,y[z])$ in the Morton calculation¹. As with the previous

¹ An alternative is to concatenate time and [SFC](#) key in one value. However, with this type of organisation we should be more careful when decoding the key.

approach, the treatment of z can be divided into two storage models: one where z is treated as an attribute and one where it is part of the [SFC](#) key. Same, the original dimensions are not explicitly stored in the database. In theory this structuring of space and time should not affect the already organised points. However, space - time locality is limited since points close in space but with consecutive time moments will be stored very far apart in the disk. Ultimately we are imposing a row order curve between points of different time and a Morton curve between points coming from the same time moment.

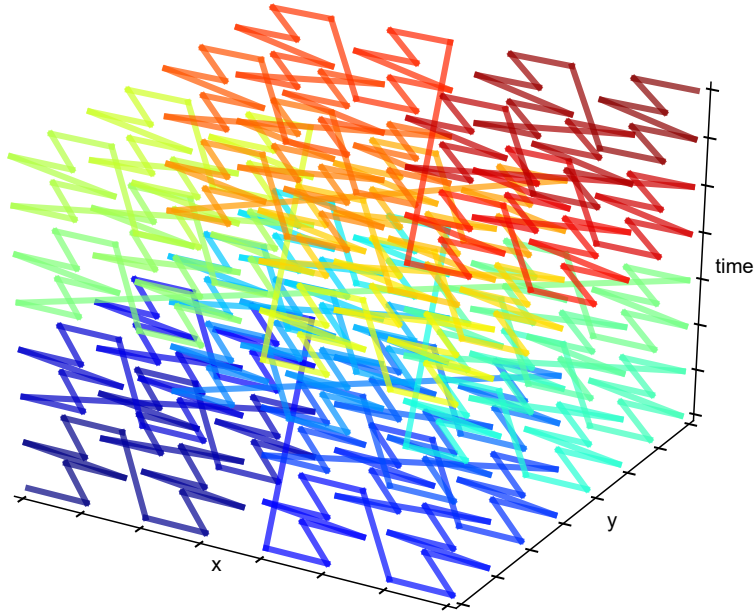


Figure 4.2: A schematic representation of the integrated approach using 8 bits in the three dimensions

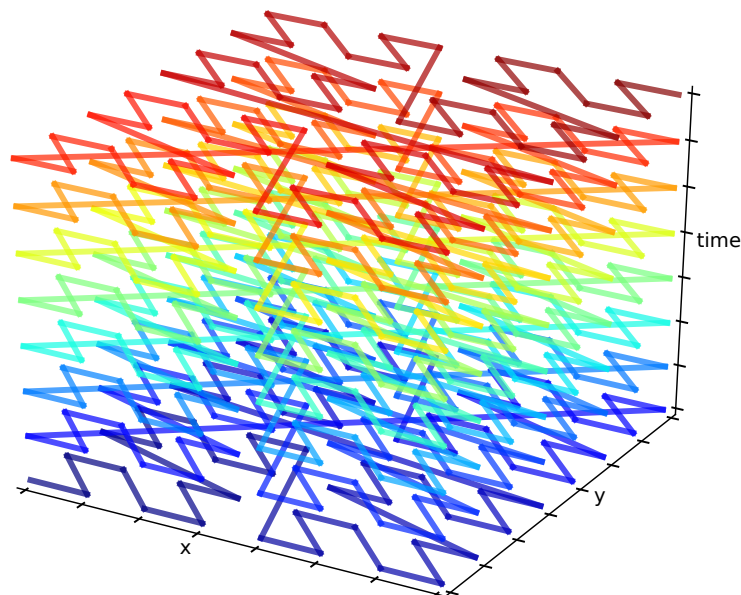


Figure 4.3: A schematic representation of the non-integrated approach using 8 bits in the three dimensions

Concerning the most important queries, time and space - time, should be the most efficient. However, since there is limited preservation of locality, space queries are expected to be less efficient. The reason for this is that the needed data is further apart.

The non-integrated approach can be represented schematically in [Figure 4.3](#). Observe the loose integration between point clouds of different time moments.

4.2.4 The encoding of space and time

The encoding of space and time is a crucial preprocessing step due to the fact that a [SFC](#) is implemented using integers. For this reason we need to express the normally used data types with integers.

Space encoding

Space in the majority of the applications is measured and expressed in kilometres, meters, centimetres, millimetres or in degrees. In all of the above units space (x, y, z) can be transformed into an integer by appropriately translating and scaling the specified coordinate. This linear transformation is expressed as follows:

$$\begin{aligned} x' &= (x - x_{off}) \times s_x \\ y' &= (y - y_{off}) \times s_y \\ z' &= (z - z_{off}) \times s_z \end{aligned} \tag{4.1}$$

where

$x_{off}, y_{off}, z_{off}$ are the offsets in the x, y and z dimensions, and s_x, s_y, s_z are the scale factors in the x, y and z dimensions respectively.

Time encoding

The encoding of time is somehow more complicated than that of space because of its different nature and semantics. Time is measured in centuries, years, months, days, hours, minutes, seconds etc. Representing time can be performed in many different ways some of which are only applicable to specific time resolutions:

1. The most naive way to represent time is by putting its building parts (year=yyyy, month=MM, day=dd, hour=hh, minute=mm, second=ss) next to each other creating an integer of the format *yyyyMMddhhmmss* for second resolution, *yyyyMMddhh* for hour resolution, *yyyyMMdd* for day resolution, *yyyy* for year resolution etc. This expression of time, however, introduces time gaps as the resolution gets finer. As an example, date 2016/12/31 with a day resolution is expressed as 20161231. The next valid date integer is of course 20170101. Therefore, we end up not using 8870 time units of the [SFC](#).
2. *Julian Date* is a system for measuring time as a number of days since some fixed day. In this case, the zero point represents the beginning of the Julian period which is 12:00 UTC on 1 January 4713 BC. As an example considering the date and time Thursday, 16 June 2016 12:00:00, the Julian date time receives the value 2457556. The disadvantage of this representation is that its starting point is not representative for

geomatics applications and thus can be receive large values. For this reason, scientists have developed the Truncated Julian date that has a starting point on midnight May 24, 1968. The previous date now corresponds to 17555. The truncated Julian date is suitable for datasets with day resolution, although the starting point may still not be representative for the majority of geomatics applications.

3. *Unix time* is another system for representing time with an integer value. It counts time in seconds since 00:00:00 UTC on 1 January 1970. As an example considering the date and time Thursday, 16 Jun 2016 14:28:13, the Unix time receives the value 1466087293. This option is very useful for datasets that are streamed every second, but leads to very detailed representations when the time resolution is in days or years. A way to compensate for the large values is to set a time offset.

A decisive factor when choosing a specific time encoding is taking into account that data will be added in the future and that the system should remain equally efficient. *SFCs* naturally are based on hyper-cubes and all dimensions present in it should, more or less, be of similar significance [van Oosterom et al., 2015]. Simply put, the several dimensions should fill equally the hyper-cube for the most efficient querying. However, since most of the time dynamic point clouds will cover a specific area of the earth, the time dimension should be chosen to be structured and scaled such that space on the hyper-cube is reserved for the new data to be added in the future without overflowing the cube.

4.3 LOADING PROCEDURE

The purpose of the loading procedure is to physically order the point cloud data based on the values of an ordering key. Depending on the integration of space and time, the ordering key is either the combination of time and the space filling value (non-integrated approach) or the space filling value (integrated approach). For physical clustering an *IOT* is used; a data contained within B-Tree.

4.3.1 Options for new data loading

In a situation where the nature of the new insertions is less dynamic, i.e. small-scale insertions every year, then the easiest way for additional data loading is by inserting the new data directly into the already organised points. Because of the dynamic nature of B-Trees, these changes should be automatically reflected in the shape of the tree. However, in the case that the new data have to be continuously inserted, this methodology has serious drawbacks and will result in an enormous performance overhead.

To account for dynamic insertions, a design choice of the loading procedure is to use two tables: an *unordered table* which is called the **heap table** and a *sorted table* which is called the **main table**. The transaction table can either contain only the new data that need to be inserted or it can be an unsorted copy of the main table. According to what function the transaction table plays, the loading procedure when new data are added can be as follows:

- a The main table is already filled with some data and new data need to be inserted. The heap table is empty and the new data are added unsorted there. When all the required insertions are finished the new and old data are combined together. The heap table is emptied again. This solution requires less duplication of the data. For only a short moment in time there are two copies of the new data present in both tables.

The re-organisation can be adapted to take place at specific moments in time e.g. every week for data with daily resolution. This will avoid continuous re-organisations of the main table which can become very expensive. However, it also means that the search algorithm gets more complicated. If the needed data are not found in the main table, the heap table needs to be scanned linearly. Use cases that do not require up-to-date information at all times can avoid this type of searching. Another way of querying is by defining a view to logically 'merge' both tables in one 'virtual' table.

- b The point cloud data are present in both the heap and the main table while a view is used for querying the data from the latter. The new point cloud data are inserted at the end of the heap table. Periodically, a new main table is created from its contents. The view used for the querying is now pointed to the new main table and the old main table is dropped. This alternative requires a lot of disk storage since at some moment in time there are three copies of the point cloud data.

Same as with the previous alternative, the data do not have to be reorganised every time new data are added. It is possible to periodically reorganise the points e.g. every week or every month. However, this means that the newest data cannot be queried. If we are talking about data added every year or month this is the best alternative, as the re-organisation will take place only once without affecting the availability of the older data. The data will be unavailable only for as long as it takes for the query view to be pointed at the new table.

- c For highly dynamic data the previous methods present some deficiencies. In the majority of the applications the old data are rarely used and therefore their maintenance is not a priority. On the other hand, the most up-to-date data should be available at all times. Therefore, a different scheme than that of an *archive* and a *current* table is needed. The archive table contains the historical data and is maintained at few moments in time, while the current table contains the most recent insertions and is maintained very often. Of course the same problem as before arises again, the current table cannot be reorganised every minute of the day. This mandates again an ordered current table that is not fully up-to-date but contains the data sorted and, an unsorted current table where the new data are added. Only at specific moments in time e.g. every hour the new data are combined with the sorted current table. When the data in the current table are considered as historic, they are moved to the archive and both current tables are emptied and receive the newest data.

4.3.2 Loading phases

The loading procedure is divided into two obligatory phases (the *preparation*, the *loading*) and one optional (the *post-processing*). Schematically, the obligatory phases are depicted in [Figure 4.4](#).

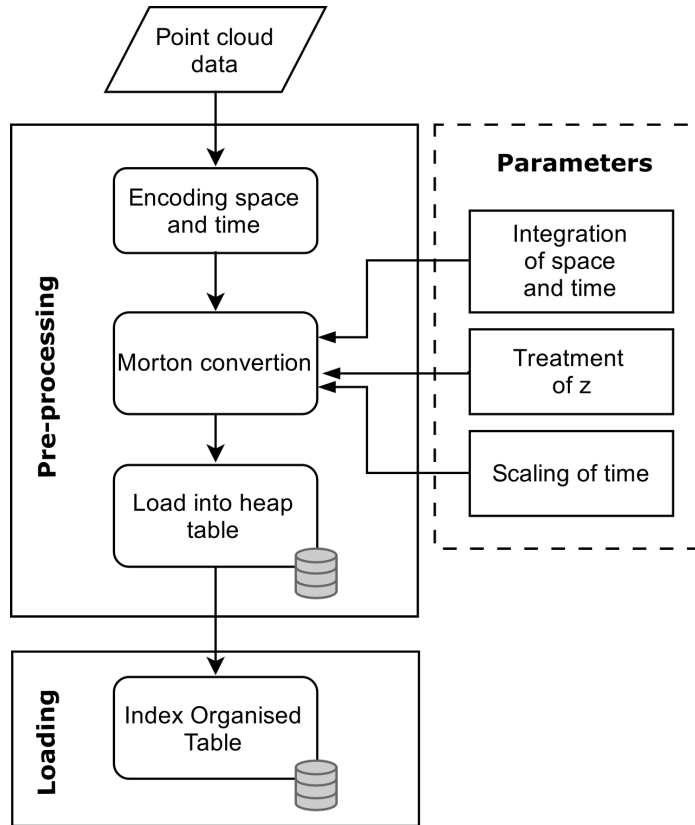


Figure 4.4: A schematic representation of the loading procedure for the SFC approach.

1. *Preparation*: The preparation phase is itself composed by two sub-phases. The first phase includes reading the point cloud data, encoding space and time into the chosen format and calculating the Morton key. This conversion varies depending on the integration of space and time, which treatment of z is used (z as an attribute or as part of the Morton key) and the scaling of time. The second phase includes the bulk loading of the converted data into a normal heap table (unsorted).
2. *Loading*: This phase corresponds to the storage of the points in an [IOT](#). This is achieved by reading the data from the heap table and physically sorting them based on the specified key. When new data need to be added in the table one of the above-mentioned alternatives can be used.
3. *Post-processing* (optional): This last and optional loading phase gathers the required optimiser statistics, which are important so that the query optimiser can choose the best execution plan for each [SQL](#) statement posed to the table.

4.4 QUERY PROCEDURE

A point cloud database, and especially a dynamic one, should be able to not only efficiently store data but also support efficient retrieval of required spatial objects. Because the number of point cloud data can very easily escalate to billions of points it is not ideal to compare a query geometry

with each one of the points present in the database. Same requirement applies for general-purpose spatial databases. A widely used approach is that of an "approximation - based query processing" [Brinkhoff et al., 1993]. This approach (as described in Orenstein [1989] and Brinkhoff et al. [1993]) contains two steps namely:

FILTER STEP Within this step the spatial index built on the objects eliminates objects that definitely do not satisfy the query. However, because geometries are most of the time approximated by their bounding boxes, this elimination leads to sets of *candidates*. In other words, the result of this step is a superset of the actual answer containing also objects that actually do not belong to the response set, called *false hits*.

REFINEMENT STEP In the refinement step the objects are tested one by one and finally the false hits are detected and removed from the answer (*response set*). Usually this step requires CPU-intensive operations, like point-in-polygon operations.

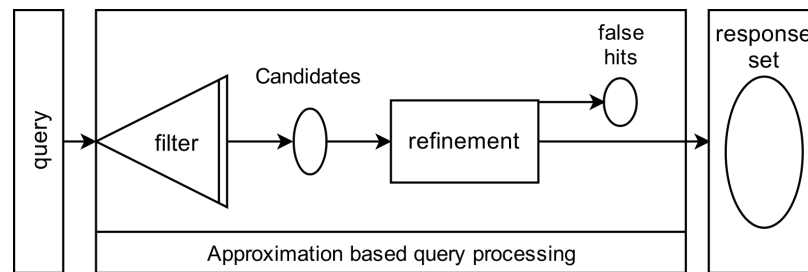


Figure 4.5: Two step query processing approach.

Source: Brinkhoff et al. [1993]

Approximations are important because it is often sufficient to retrieve even the approximated response of the filter step in order to reduce processing time. However, it becomes obvious that the performance is related to the algorithm that approximates either the objects in the database or the query geometry. The result of the filter step becomes more accurate when the approximation deviates from the actual object as less as possible [Brinkhoff et al., 1993]. If the approximation algorithm is very fine-tuned then the false hits in the filtering step are so low in number that the refinement step can be ignored. The ultimate goal is thus having an approximation algorithm that reduces the number of points in the candidate set as much as possible.

4.4.1 Querying space filling curves

In order to be able to perform multidimensional selections using *SFCs*, the query algorithm needs to be modified so that the *SFC* organisation is taken into account. The reason behind this requirement is that the transformation into a linear space results in a loss of spatial relationships between objects. In other words, a mapping from the multidimensional space to a one dimensional, can never be done in such a way that all neighbouring objects in the multidimensional space are also close in the one dimensional one. Therefore it is up to the query algorithm to efficiently handle this loss. When querying *SFC*, the *n*-dimensional query region needs to be translated into a number of continuous runs on the (Morton) curve, that is to say a search problem in 1 dimension.

In the literature several ways to perform a range query using the Morton curve can be identified. The most naive way to perform range queries is to obtain the Morton codes of the coordinates of the lower - left and upper - right corner and then use this code range in order to obtain the candidate data during the filter step. An example is given in Figure 4.6, from which it is easy to realise that this method is not very efficient and effective. A lot of records that are within the code range (12 to 50) are not part of the range query. Actually, in this specific example only 30% of the candidate points actually belong to the range query. As a result querying space filling curves require a different approach than this one.

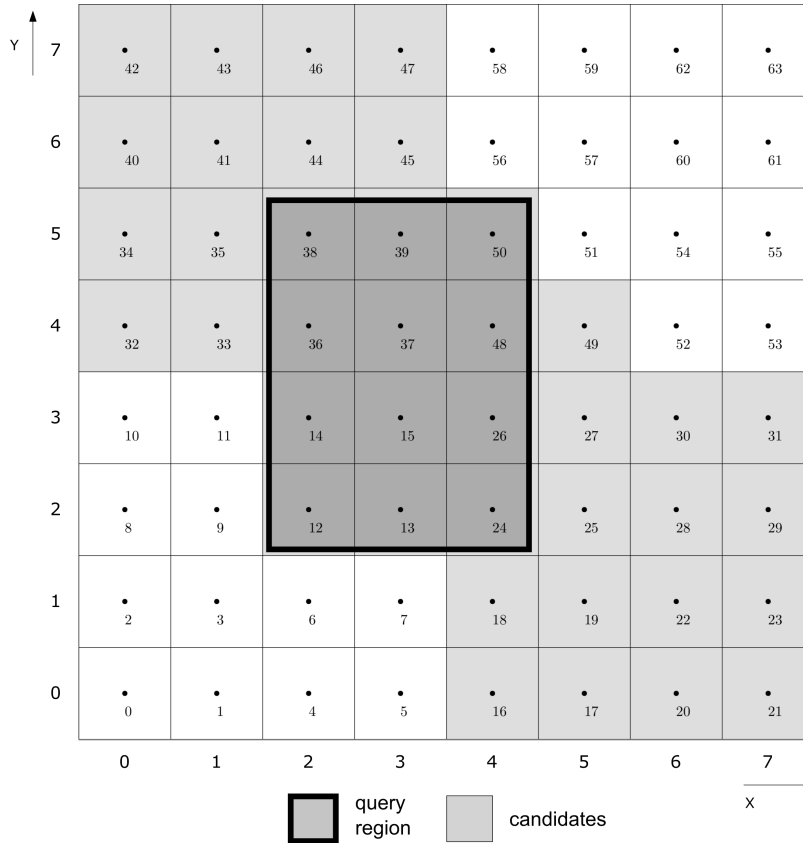


Figure 4.6: A naive way to perform range queries when using space filling curves.

A more efficient way is the one presented in Tropf and Herzog [1981]. The authors have implemented a methodology that takes the previous naive way of querying SFC and speeds up the search by calculating the *BIGMIN* and *LITMAX* values. In their paper, the authors describe the algorithm for the calculation of those values. Within this thesis, I do not make use of the previous implementation but rather follow a different approach, that of the interrelationship between Quadtree structures and the Morton curve (see also Figure 2.3.3).

Each quad-code is directly related into specific ranges of Morton codes. As an example observe Figure 4.7. This example is first presented in van Oosterom and Vijlbrief [1996] and is appropriately adapted. In order to proceed to the filter step we need to identify for the specific polygon (Figure 4.7a) the number of continuous runs on the Morton curve. The grid on the back represents the QuadTree of the whole area for which we have point cloud information up to a specific level (here level 3). In the second step

(Figure 4.7b), the decomposition algorithm identifies the quadcodes strings that approximate the spatial object. This is achieved by checking whether the quadcode intersects with the geometry (see Section 4.4.2). Following a Z-order, the SW quadrant takes the quadcode string of 0, the SE string of 1, the NW string of 2 and NE string of 3 (base 4). Each successive digit on the string represents the quadrant of a deeper level. This means, that the shorter the length of the quadcode, the more the SFC values contained in the returned range. In this specific example, the polygon is approximated by quadcode strings '0', '20', '21', '300'. The mapping from each quadcode string to morton ranges is performed in step 3 (Figure 4.7c). As it can be visually seen, quadcode '0' corresponds to morton range 0-15, quadcode '20' to 32-35, quadcode '21' to morton range 36-39 and quadcode '300' to morton code 48. Further, since quadcodes '20' and '21' have neighbouring morton ranges those can be merged to one, going from 32 to 39.

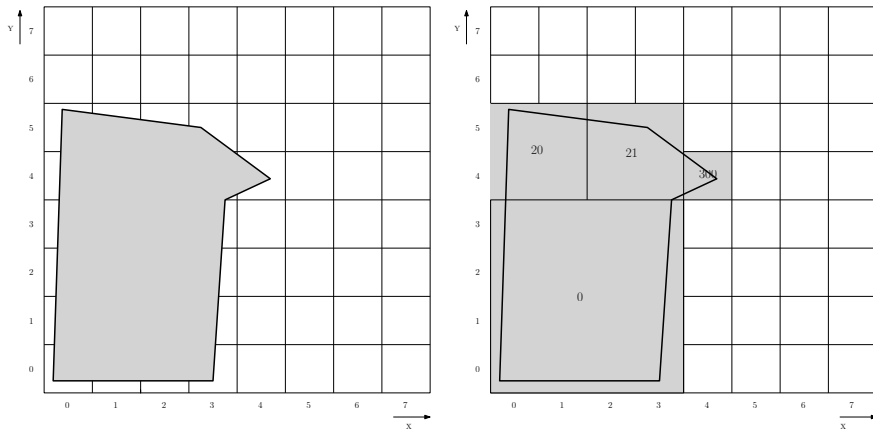
The advantage of utilising the relationship between morton curve and the quadtree is that it generally provides better approximation of the query geometry than using a single MBR. An example is shown in Figure 4.8. Cases 4.8a to 4.8d show the approximation of the polygon using finer quadtree cells. It is obvious that the deeper the quadtree level used, the better the cells approximate the actual geometry. This means, that during the filter step less false hits will be yielded and less redundant data will have to be processed in the refinement step. In addition to that, compared to the use of the MBR, as depicted in Figure 4.8, levels 7 and 8 of the Quadtree describe this complex spatial object with more accuracy. Finally, this whole process of identifying the quadtree cells that approximate the query geometry is completely independent of the data stored inside the database and can be extended to n-dimensional data very easily. From this point, no more references to quadtree cells will be made but rather to 2^n -tree ones since the methodology used in this thesis extends up to 4 dimensions.

4.4.2 Decomposition and evaluation of the query geometry

As mentioned in the previous section, in order to query SFCs, the query object in the n-dimensional space needs to be decomposed to a number of 1-dimensional intervals. For this the relationship between the morton curve and the 2^n -tree is used. The decomposing elements of the query object are acquired by reporting the 2^n -Tree Cell (TC) that lie completely within or partially intersect with the object.

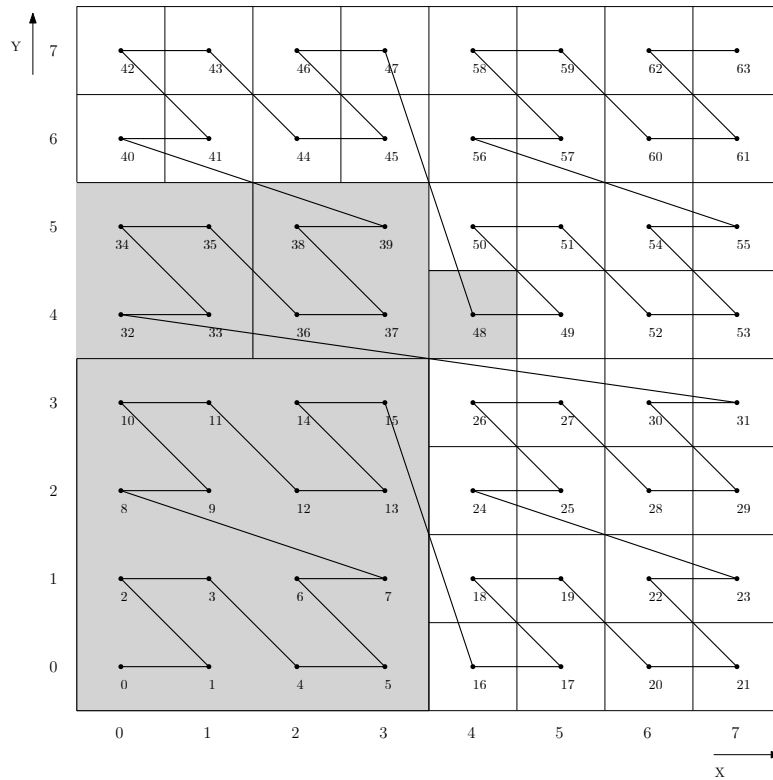
The *search algorithm* used in this thesis is the one presented in Orenstein and Merrett [1984], but adapted to account for 2^n -trees. The algorithm identifies the 2^n -TCs that best approximate the Query Region (QR). Starting from 2^n -TCs that approximate the whole point cloud region, the n-dimensional space is subdivided recursively until either the full resolution of the grid or the maximum recursion level is reached. For each TC one of the following cases can happen:

- The TC is disjoint from the QR. The TC does not contain any points that satisfy the query. Nothing further happens in this case.
- The TC is completely within the QR. All of the points inside the TC are contained within the QR. No further split takes place.
- The TC overlaps with the QR. In this case, the TC is split into 2^n smaller TCs (where n the dimensions used) which are handled recursively until the maximum depth is reached.



(a) The query polygon

(b) The decomposition of the polygon based on the Quadtree



(c) The relationship between quadcodes and the morton curve

Figure 4.7: Decomposition strategy based on the relation between the morton curve and the Quadtree. Adapted from: [van Oosterom and Vijlbrief \[1996\]](#)

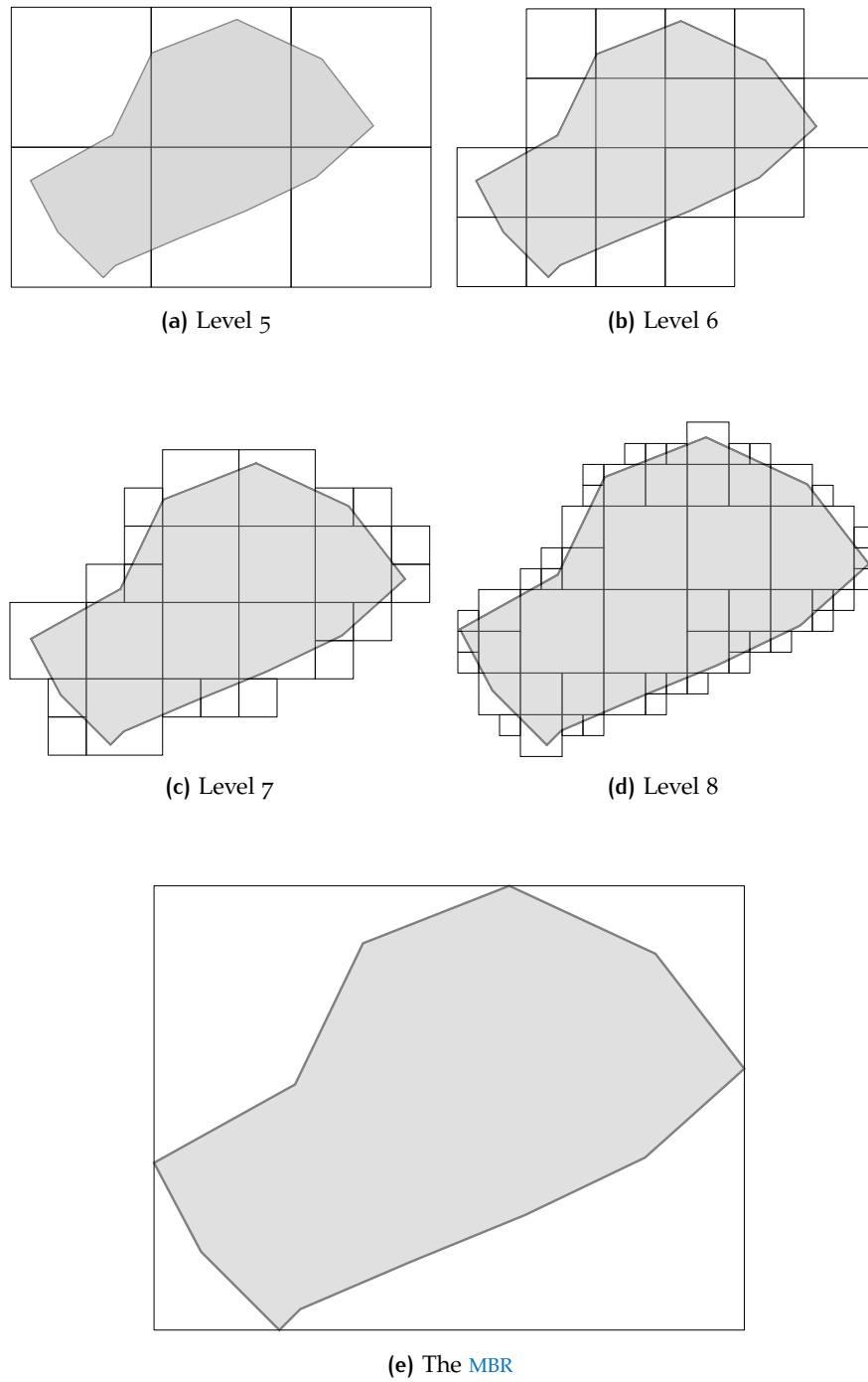


Figure 4.8: Quadtree decomposition of a complex polygon at different levels and in comparison to the Minimum Bounding Rectangle approximation.

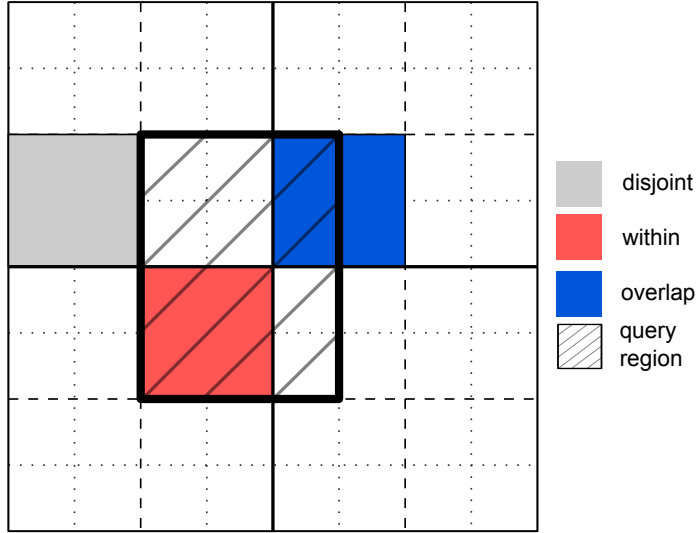


Figure 4.9: The search algorithm applied to three candidate tree cells. The grey Tree Cell is disjoint from the Query Region (hatched), while the red Tree Cell is completely within the Query Region. Finally the blue Tree Cell partially overlaps the Query Region and therefore, is split into 4 smaller tree cells and the algorithm is repeated.

An example of the use of the search algorithm is depicted in [Figure 4.9](#). All three cases can be identified in the given example. The adapted algorithm used in 2D is available in [Algorithm 4.1](#). Note that in higher dimensions (higher than 2) instead of the SW, SE, NW, NE quadrants, space is subdivided in $2^3 = 8$ octans for 3D, or $2^4 = 16$ hexadectans in 4D. To account for the fact that recursive calls of the function can lead to a large amount of TCs, a maximum depth of recursion is used. In case that this maximum level is reached, the TC being processed is added to the list of morton ranges and not further split. The output of the algorithm is a list of the morton ranges that are either used to retrieve the data or are further processed if certain conditions apply.

4.4.3 Merging of the ranges

One of the issues that may arise from using the previous *RangeSearch* algorithm, is that a vast amount of morton ranges may be generated. This is most of the times not ideal because it will require a large join operation or there are limitations to length of the SQL string that can be used. Also, near the boundary of the query region a lot of small morton ranges may be generated and as a result the amount of work may be large.

In the literature the same problem is dealt by adjusting the decomposing algorithm itself. More specifically, in [Orenstein \[1989\]](#) two adapted decomposition strategies are defined:

SIZE-BOUND This decomposition strategy puts a limit to the number of TC that can be generated. To achieve fair sizes, a breadth-first search is used. The splitting ends when the maximum number of ranges is reached. However, the algorithm still continues to refine in one dimension as in this way the accuracy of the approximation is increased without increasing the number of ranges.

Algorithm 4.1: Range Search Algorithm used to retrieve the morton ranges in a 2D case.

Data: Query Region (QR), Tree Cell (TC), level, maxlevel
Result: Morton ranges

```

1 RangeSearch(QR, TC, level, maxlevel)
2 ranges ← emptyList
3 if TC disjoint QR then
4   do nothing
5 else
6   /* partial or complete overlap of TC with QR */
7   quadCode = FindQuadCode(TC)
8   if TC within QR or level = maxlevel then
9     mortonRange = (quadCode, level)
10    ranges.insert(mortonRange)
11  else
12    /* partial overlap of TC with QR */
13    foreach quad in (SW, SE, NW, NE) do
14      RangeSearch(quad, SR, level + 1, maxlevel)
15  end
16 end
17 return ranges

```

ERROR-BOUND This decomposition strategy puts an accuracy requirement on the approximation of the geometry instead of looking at the number of ranges. The algorithm continues to decompose the geometry until the distance between the boundary of the QR and the boundary of the TC is above the error threshold.

In this thesis to deal with the previously mentioned limitations two actions can be taken: first, a merging of consecutive morton ranges takes place (introducing no additional space/ overhead) and second, ranges are merged to a maximum number specified. These two actions differ from the two decomposition strategies in that they correspond to post-processing actions, rather than directly affecting the ranges generated.

Merging consecutive ranges

Merging consecutive ranges is one fundamental way to reduce the number of morton ranges. This step will result in less ranges during the fetching process, although sequential accesses will be increased (space expanded). An example of the application of the consecutive range merging is shown in Figure 4.10. From the previous stage, three morton ranges have been calculated as approximating the QR. However, these three morton ranges are consecutive (8-11, 12-15, 16-31) and can be merged into one big one (8-31). This significantly aids the query procedure as the query optimiser will have to parse only one range, instead of the original three.

The algorithm to perform this action is described in Algorithm 4.2. The process requires a sorted list (ascending) of the morton ranges. The merging is performed by iterating over the sorted ranges and checking whether the upper boundary of a current range and the lower boundary of its next range are consecutive integers. If this is the case, then the process is continued until there are no more consecutive ranges to merge.

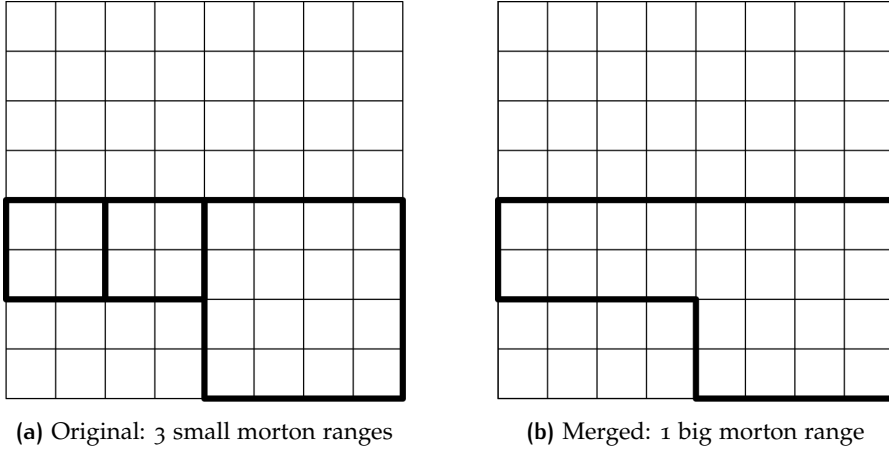


Figure 4.10: An example of merging consecutive ranges. Left: the original situation with three morton ranges. Right: the merging results in one big morton range.

Algorithm 4.2: Algorithm for merging consecutive morton ranges.

Data: Sorted list of morton ranges (ranges)
Result: Merged list of morton ranges

```

1 mergedList  $\leftarrow$  emptyList
2 min, max  $\leftarrow$  ranges[0]
3 foreach number in 1..size(ranges) do
4   if max - ranges[number][lower] = 1 then
5     | max  $\leftarrow$  ranges[number][upper]
6   else
7     | mergedList.insert(min, max)
8     | min, max  $\leftarrow$  ranges[number]
9   end
10
11 mergedList.insert(min, max)
12 return mergedList

```

Merging to maximum number

Having closed the gaps between consecutive ranges, it might be the case that the ranges are still too many to be parsed by the query optimiser. For this reason, similar to the *SIZE-BOUND* decomposition strategy but in a post-processing manner, a merging of ranges in order to reach a maximum number takes place. The basic idea behind this method is that the *QRs* are expanded into larger but fewer *TCS*. The critical part during this process is identifying the right number of ranges that are not too many or too low, and the data fetching is not becoming too expensive. Therefore, a balance between fetching time and number of ranges is important to be found. Also, the cost to be paid by this method is that more false hits might be retrieved during the filter step.

The merging process used in this thesis proceeds in the following way. The distances of the neighbouring ranges (differences) are calculated. The resulting list of differences is sorted in ascending order. The number of gaps (n) to be closed is the difference between the current number of ranges and

the specified maximum. A threshold that corresponds to the n_{th} distance is set. Then the procedure starts closing the smallest gaps using the threshold defined².

Two examples of the application of the merging algorithm are shown in Figure 4.11 and Figure 4.12. In both of the examples, case (a) represents the original un-merged case. In cases (b) the maximum number of ranges used is 3 and in cases (c), it is 2. It is obvious that the smaller the maximum number, the bigger the expansion of the TCs and more false hits will be fetched during the filter step. The jumps of the morton curve are, also, noticeable and affect to a great extent the final result.

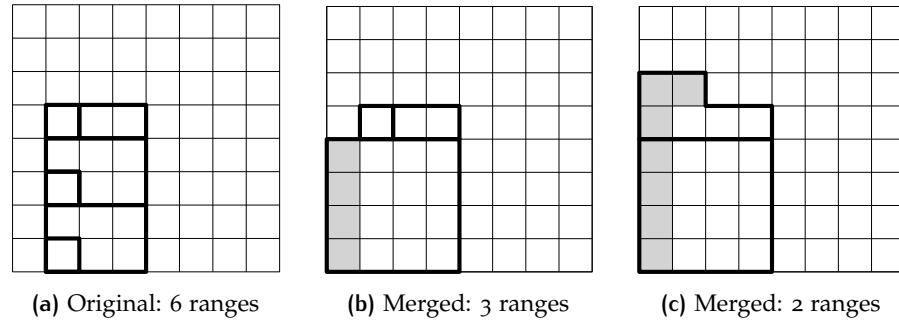


Figure 4.11: First example of merging ranges to obtain a maximum number. (a) The original situation with 6 morton ranges. (b) The maximum number is set to 3 and the regions are expanded accordingly until the three ranges are obtained. (c) The maximum number is set to 2 and the regions are expanded accordingly until the two ranges are obtained. The grey area represents the additional space added due to the merging of TCs.

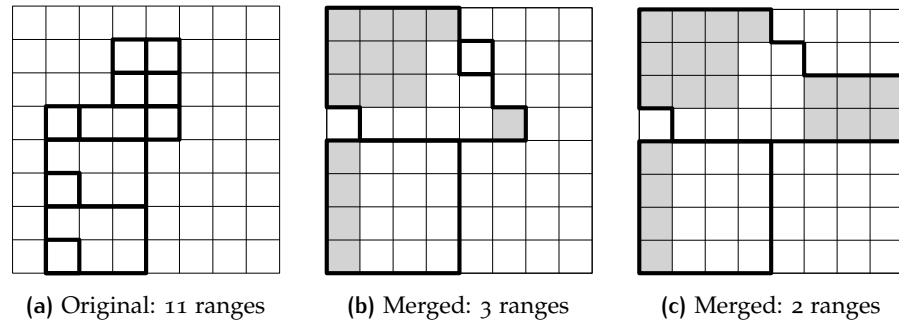
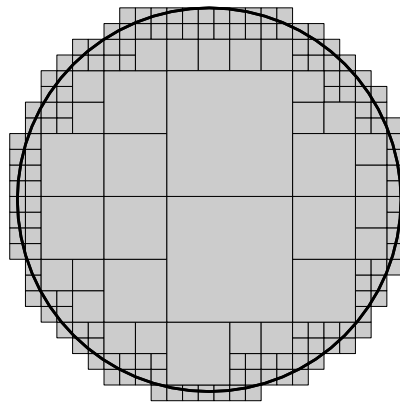


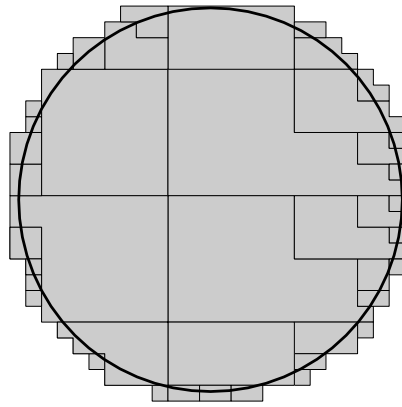
Figure 4.12: Second example of merging ranges to obtain a maximum number. (a) The original situation with 11 morton ranges. (b) The maximum number is set to 3 and the regions are expanded accordingly until the three ranges are obtained. (c) The maximum number is set to 2 and the regions are expanded accordingly until the two ranges are obtained. The grey area represents the additional space added due to the merging of TCs.

The whole filter step is depicted for a circular geometry in Figure 4.13.

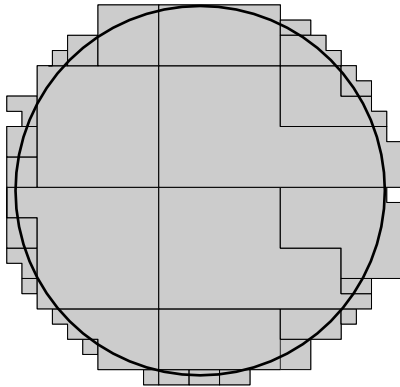
² For more information see the code on-line: <https://github.com/stpsomad/DynamicPCDMS/>



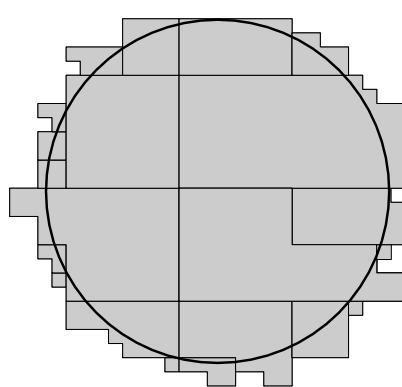
(a) Original 176 tree cells



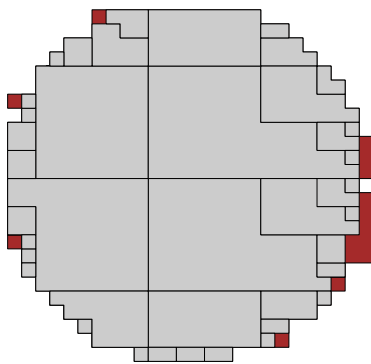
(b) Merging of direct neighbours leading to 42 cells



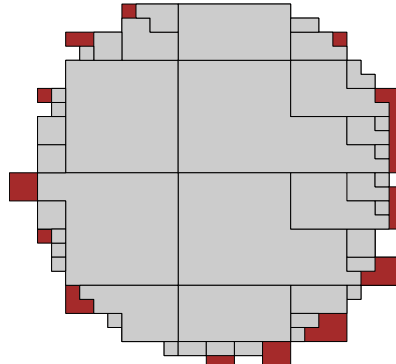
(c) Merged to maximum number of 30



(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 4.13: 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. Source: Psomadaki et al. [2016]

4.4.4 Refinement

Having identified all the candidate points (false and true hits), a way to further refine the selection is needed. Since this thesis deals with point clouds, the best option to perform the refinement step is by using a Point in Polygon (PIP) operator. As the name implies, the operator determines for a set of points which one of them lies inside an (arbitrary) polygon and which not. A well known way to solve a PIP problem is by drawing a line from each candidate point to a point outside the polygon and counting the number of intersections that take place. If the number is odd, then this point is inside the polygon. This rule is known as the *even-odd* or *parity rule*. PIP operators are found in the major spatial databases e.g. *ST.Contains* in PostGIS³, *SDO.Contains* in Oracle Spatial⁴ or the *SDO.PointInPolygon* in Oracle⁵.

4.4.5 Summarising the query procedure

Having described all the relevant aspects of the querying procedure, the section concludes by putting them together into one complete workflow. As mentioned in the beginning of the section, the two-step procedure of the filter and refinement step is used. The original methodology is now adapted according to the problem of retrieving dynamic point clouds.

STEP 1 The workflow starts by the transforming the n-dimensional query (here $n \leq 4$) to a number of continuous runs on the morton curve. For this a 2^n -tree structure is used and the cells that overlap with the given geometry are selected. The decomposition stops when the maximum recursion depth is reached. The selected cells are then associated with the corresponding morton ranges. Before the ranges are returned to the next step, the neighbouring ranges are merged. Then, either the ranges are returned or, if specified, they pass through another processing step. Within this step the k morton ranges are scaled down to m morton ranges, where $m < k$.

STEP 2 Moving to the filter step, the specified morton ranges are utilised for the retrieval of the points. For this the IOT is used. A point is fetched if and only if it falls in one of the corresponding morton ranges. The result of this step is a superset of the needed points.

STEP 3 The retrieved points in the form of morton values are decoded back to the original X, Y, Z, t dimensions depending on which integration is used, which treatment of z is currently taking place (z in the key or not) and scaling of time. The result of this step is a set of candidate points.

STEP 4 The candidate points are inserted into the refinement stage. The points are tested one by one using a PIP operator and, if needed, are further refined using time and Z predicates. The output is returned to the user.

Schematically, the query process is depicted in Figure 4.14.

³ see: <http://postgis.net/docs/manual-1.4/ST.Contains.html>

⁴ see: <http://docs.oracle.com/database/121/SPATL/GUID-CD3E09BC-4533-4D3F-BB5D-8D6DCDF5C5FF.htm#SPATL1024>

⁵ see: <http://docs.oracle.com/database/121/SPATL/GUID-959C15D2-509C-4511-ADEB-1ADA6FBA8D0A.htm#SPATL1507>

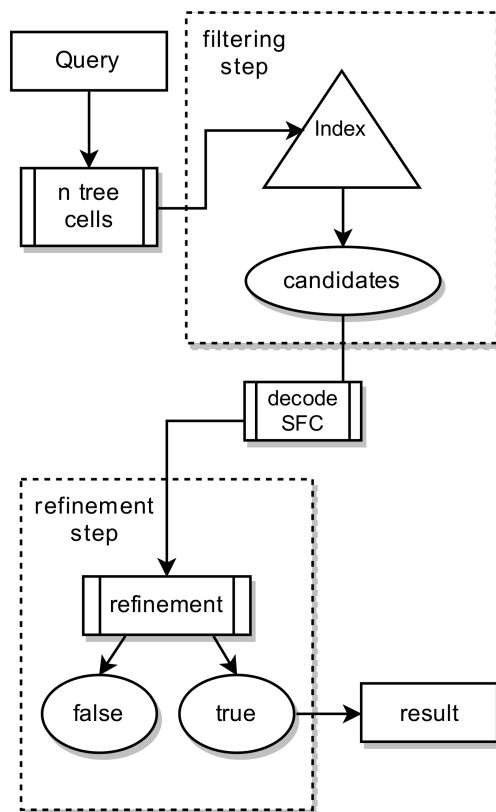


Figure 4.14: A schematic representation of the query procedure

5 | IMPLEMENTATION AND EXPERIMENTS

This chapter contains a description of the prototype implemented to show whether the [SFC](#) approach provides an effective solution to the management of dynamic point clouds. The theory has already been introduced in [Chapter 4](#). A number of experiments and certain benchmarks have been executed for this reason.

The chapter is organised as follows: In [Section 5.1](#) the tools and datasets used to perform the experiments and benchmarks are presented. After that, in [Section 5.2](#) the metrics which are examined in the experiments section are introduced and explained. In [Section 5.3](#) certain important details about the implemented prototype are given. Finally, in [Section 5.4](#) follow the experiments performed in order to study the previously presented metrics.

5.1 TOOLS AND DATASETS USED

For the implementation of the methodology presented, a set of Python scripts have been developed. The source code can be found at: <https://github.com/stpsomad/DynamicPCDMS>. The scripts have the ability to take as input: the type of integration of space and time, the scaling of time and the treatment of z (z as part of the Morton key or not) and implement the methodology of [Chapter 4](#).

5.1.1 Software

The software used for the prototyping of the proposed methodology is:

PYTHON The Python programming language was used for its ability for easy and fast prototyping. However, at the same time this means that the time efficiency of the code is limited. The version of Python used is the 2.7.

ORACLE The Oracle Database is a proprietary [RDBMS](#). 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 choice of Oracle instead of an open-source alternative is justified because of two advantages that it offers to this thesis:

- The existence of the Index Organised Table ([IOT](#)). In an [IOT](#) the data are stored in the leaf-nodes of the index. Therefore, it avoids storing a large (separate) index in addition to the data table (heap), thus not requiring to perform a join during query execution (between index and data).

- The existence of the *NUMBER* data type. Full Morton keys can very easily become larger than 64 bit integers¹. The *NUMBER* data type can handle the 128 bit keys.

POINTCLOUD PYTHON PACKAGE This python package was developed as part of the project "Massive Point Clouds for eSciences"². The source code of the package is available at <https://github.com/NLeSC/pointcloud-benchmark>. This package was taken as a starting point for the development of the code.

5.1.2 Hardware

For the tests and benchmarks 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 7 operating system. The disk storage which is directly attached consists of a 400 GB SSD, 5 TB SAS 15K rpm in RAID 5 configuration, and 2 x 41 TB SATA 7200 rpm in RAID 5 configuration.

To make sure that we minimise mixed read/ write operations on the same disk, especially during the loading procedure, we have distributed our files and database over different disks. Within the *DBMS* this is achieved using different tablespaces. The available tablespaces for the Oracle database on the server are distributed as follows (Table 5.1):

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

Table 5.1: The distribution of the available tablespaces according to the purpose.

5.1.3 Datasets used

Due to the fact that the Deltares research institute was the initiator of this thesis, both use cases come from the coastal monitoring domain. Two datasets of different sizes and characteristics were used. The differences are relevant in order to evaluate how the proposed structures behave according to them. For more detail, the user is referred to the [Appendix A](#). Within this section a short description is given for the two use cases.

¹ This was actually one very important issue when designing the system. With a 64 bit code we can only allow 21 bits per dimension in 3D. In the spatial dimensions this means that with a millimetre accuracy only 2 kilometres of data can be stored. Therefore the need arises to use $N \times 32$ bit codes, where N is the number of dimensions. The challenge now is how to store these larger codes. Some options that can be considered are:

- Encode in character strings e.g. base 64, base32, HEX.
- Breaking up the larger code into multiple 64 bit integers.
- Use the Oracle *NUMBER* data type.
- Use a different system that supports arbitrary length bit strings.

² The website of the project can be found at: <http://www.gdmc.nl:8080/mpc>

SAND ENGINE Dataset with small spatial extent but day resolution in the time dimension. The point cloud is acquired using jet skis and all-terrain vehicles.

COASTLINE OF THE NETHERLANDS Dataset with big spatial extent. The time resolution is, however, limited to every year, with only 4 years being available at this moment in time. The dataset is acquired using Light Detection And Ranging (**LIDAR**) technology and is provided by Rijkswaterstaat.

For both of the datasets, points having the same planar (x,y) coordinates are removed. This was performed because of the unique constraint that an **IOT** needs to fulfil. Although this uniqueness constrain is only violated for the approaches where z is an attribute, the cleaned datasets are used throughout the experiments and benchmarks.

5.2 METRICS OF PERFORMANCE

In order to be able to assess how well or not the proposed methodology performs during query execution, it is important to define a set of metrics. These metrics are different from the ones defined in [Section 4.1](#) because they are quantitative measures. The following four (4) metrics are defined: 1. Fetching time, 2. Percentage of extra points 3. Depth of the tree, 4. Degree of merging. The main challenge is to find the balance between the four of them.

5.2.1 Fetching time

The fetching time, and especially the fetching time of the filter step, should be relatively fast according to the number of points belonging in the query region. Nevertheless the user should not wait too much to get an answer. It is also important that the fetching time is constant even when the size of the table grows (and the size of response is equal).

Note that the reason why the fetching time of the filter step is the most important originates from the fact that the current form of the query procedure requires many steps, some of which (with the current implementation) have to take place outside the database (see [Section 5.3.3](#)). In the future this might be optimised by moving certain functionality inside the database. Nevertheless, the filter step is directly related to the data structure that is used.

5.2.2 Percentage of extra points

Another very important factor is the percentage (%) of false hits retrieved during the filter step compared to the true hits from the refinement step. It is good that the number of false hits does not outnumber the number of true hits, otherwise the database spends a lot of time retrieving unwanted information which will also increase execution time of the refinement step. In addition to that, if the percentage of false hits is low (e.g. 1 - 10%) then the refinement step could be even ignored, thus saving processing time at the cost of some extra data.

5.2.3 Depth of the tree

The depth of the 2^n -tree is of great importance because it directly affects the number of ranges that are used to approximate the [QR](#). The finer the [TCs](#) that are used for the computation of the ranges, the more accurate is the result during the filter step (less false hits). On the other hand, the finer the [TC](#), the more ranges that need to be used for the fetching of the data. Using a large amount of ranges during query execution might result in a decrease of the performance of the query. However, this needs to be corroborated with experiments.

5.2.4 Degree of merging

As presented in [Section 4.4.3](#), the merging of the data also affects query execution. If the number of ranges is too high, then fetching is expected to become expensive and merging is the way to limit the number of ranges obtained. Merging effects are the same as the previous metric: introduce more false hits (by having less ranges) since more space is added outside the [QR](#). The higher the degree of merging, the more chance that during the filter step a higher amount of false hits will be fetched. At the opposite side, a low degree of merging reduces the number of ranges only by a small amount and thus query execution remains expensive.

Nevertheless, it is also interesting to investigate whether there is a connection between the depth of the tree and the degree of merging. By that it is meant, if finer 2^n -[TCs](#) affect the quality of the merging and thus the percentage of extra points obtained.

A very important factor of the merging is the choice of good algorithm that is near linear $O(n \log n)$. Having a merging algorithm that is too expensive makes the whole query process also slow. However, developing the best algorithm is out of scope for this thesis, which mostly deals with the efficiency of the data structure.

5.3 IMPLEMENTATION

This section describes the most important implementation details of the method which was theoretically introduced in [Chapter 4](#). For the full implementation the user is referred to the source code available on-line, and some scripts and [SQL](#) code on [Appendix D](#).

5.3.1 The encoding of space and time

The encoding of space and time is perhaps the most influencing factor in the integration of space and time. One of the implementation decisions was to encode full resolution Morton keys, thus making the storage of the original x , y , z and time dimensions not a requirement (they are contained in the key). This significantly reduces the storage size of the table but imposes one extra step during the query procedure (decoding). Since [SFCs](#) are implemented from integers alone, the following encoding of space and time takes place for the two datasets ([Table 5.2](#)):

The choice of the above mentioned offsets in the x , y and z dimensions (columns 2-4) is justified from the fact that positive numbers need to be

Dataset	x_{off} [m]	y_{off} [m]	z_{off} [m]	s_x, s_y, s_z	Time encoding	Scaling of time
Sand Engine	69,000	440,000	-100	1,000	days since 1/1/1990	10,000
Coastline	4,000	374,000	-100	100	year (yyyy)	10,000

Table 5.2: The used encodings of space and time for the two datasets (Sand Engine, Coastline)

derived before applying the needed dimensional scaling. In the case of the z dimension, the offset imposed is negative since the majority of the heights are below sea level. Offsetting is a very important transformation, as it significantly reduces the size of the Morton key and as a result the size of the [IOT](#). Next, the dimensional scaling is defined in column 5. These specific dimensional scalings correspond to the spatial resolution of the datasets: mm for the Sand Engine (scaling of 1000) and cm for the Coastline (scaling of 100).

Although the parameters for the spatial dimensions are straightforward to be identified, the ideal encoding of time is more complicated. Different approaches to derive integers from dates are introduced in [Section 4.2.4](#). Because of the different nature of the time dimension in the two use cases, also, different encodings of time take place:

- For the Sand Engine use case with a day resolution, it is chosen to encode time as days passed since 1/1/1990. This specific epoch is used in order to make it possible to store data from the past. If this is not needed then the epoch can be changed to a more suitable one.
- For the coastline use case with a year resolution, it is chosen to encode time in the format yyyy. Because the resolution is so coarse, this type of formatting does not generate gaps, making it very appropriate and straightforward method.

The used encodings of time, although very appropriate for the given use cases, do not necessarily lead to a useful integration of space and time when time is part of the [SFC](#) key (integrated approach) and later on characteristic space - time queries are performed. In the current form of the encoding, space and time are deeply integrated. The correspondence of the units is consequently $1\text{ mm} = 1\text{ day}$ in the Sand Engine use case and $1\text{ cm} = 1\text{ year}$ for the Coastline. This type of organisation makes some of the important queries (only time and space-time) very expensive, thus violating the first metric of performance (fast fetching time). A schematic explanation for the inefficiency of these queries is given in [Figure 5.1](#) (left). In this part of the figure the correspondence is 1 cm to 1 year. This results in data coming from year 1 (blue) to be interleaved with data coming from year 2 (red). A way to avoid this type of integration is to scale time appropriately in order certain number of points from one year to be stored close to each other ([Figure 5.1](#), right).

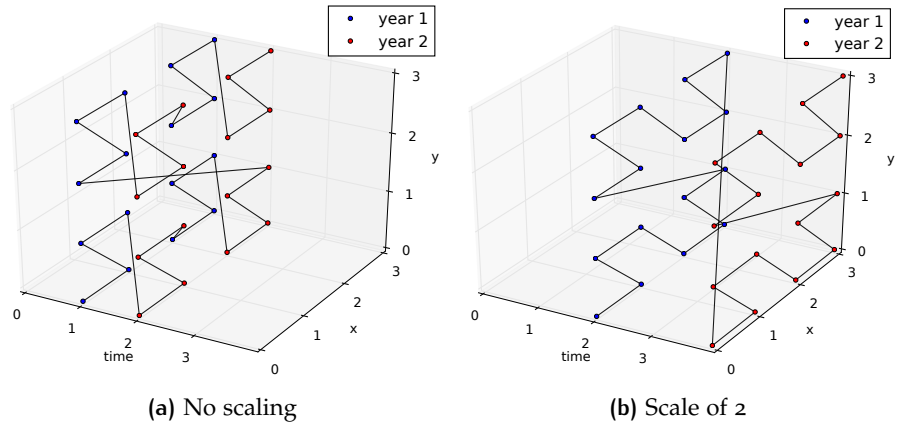


Figure 5.1: The effect of applying no time scaling and a time scaling of 2 on the space - time proximity of the points.

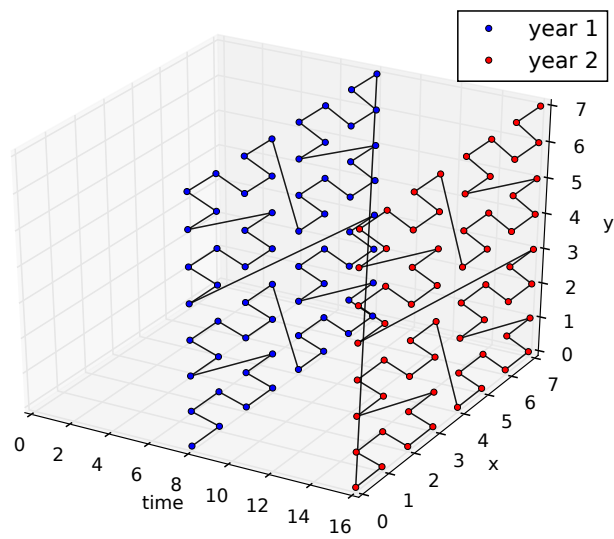
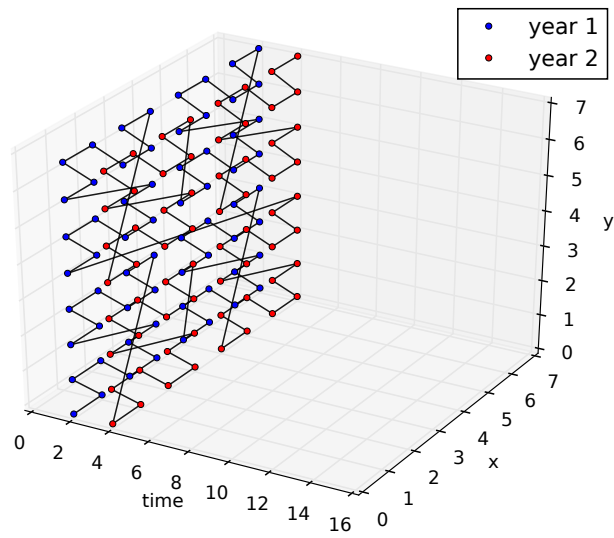


Figure 5.2: The effect of different degree of scaling on the space - time proximity of the points.

Of course, it cannot be avoided that points from different years are stored close together, when they are spatially close. This is actually required, otherwise we end up having another non-integrated approach. An example of how scaling time affects the space-time proximity is shown in [Figure 5.2](#). In the two examples two different scales of time are applied (2 and 8 respectively). In the original situation (not shown here) every four points of year 1 are interleaved with 4 points of year 2. By applying a scale of 2, the gap between the two years is increased and now the interleaving happens every 16 points on the curve. Using a scale of 8, increases the number of points to 256. The ultimate challenge for the use cases is to identify what degree of space - time proximity balances the metrics of performance for the three important queries of [Section 4.2.1](#).

In order to find out which scaling of time is the most suitable for the two use cases, certain tests took place. For the Sand Engine case specifically, scales 1, 100, 1000 and 10000 were tested. The experiments showed that applying no scaling makes time queries very inefficient (approximately 50 seconds for 79000 points, one day), but space queries perform very efficient (sub-second). Time only queries are, however, more important for this specific use case than space ones. For this reason the scale of 10000 was chosen. Within this configuration the same time query is performed in 0.6 seconds. Inevitably, with such a scale space queries become less efficient. Nevertheless, the gain in time queries is bigger than the loss in space ones. The same work flow is also followed in the coastline use case, where again a scale of 10000 offers better results than no scaling. In the Sand Engine use case a scale of 10000 means that for a certain day the area grouped is 10m by 10m, while the coastline use case the grouped area is 100m by 100m per year. The chosen scale has also the property that it allows future data to be stored in the same efficient way. More specifically the Sand Engine use cases allows the storage of hundreds of years in the future, without overflowing the curve. In coastline use case thousands of future years are possible to store.

The above explained parameters for the encoding of space and time introduce the need to maintain a metadata table during the loading procedure. These parameters are very important to be stored otherwise the query procedure cannot be executed. How often this metadata table is updated depends on how often new data are inserted into the database.

5.3.2 Loading procedure

In [Section 4.3](#) several different solutions for handling the loading procedure are presented. These solutions depend on the dynamic nature of the use cases used. The use cases used within this thesis can be characterised as having a medium dynamic nature. This means, that time is relevant, however, new data are not streamed very often (highest frequency is once every day). For this reason, the loading procedure follows the transaction and main table approach. The transaction table acts as a container for the new data, while the "old" data are found sorted in the main table, which is an [IOT](#). In order to combine the new and old data, a *UNION ALL* operator is used. For an overview of the [SQL](#) code used to perform the loading, the reader is referred to [Section D.1](#) of [Appendix D](#).

5.3.3 Query procedure

The query procedure follows the methodology defined in [Section 4.4](#). The way in which the query statement is composed depends on the integration of space and time used, as well as, the type of query asked. An overview of the query scripts used when posing a query is described in [Section D.2](#) of the Appendix. Of high importance here is to further explain what steps take place inside or outside the database (and why) and the two different ways developed to pose the queries. The query geometries and time ranges that were actually used throughout the next sections are described in [Appendix B](#).

The current state of the query procedure includes a mixture of inside and outside the database functionalities. The reason for this type of implementation comes from the fact that there are no functions available inside the database that can perform the needed actions. This is also true for the loading procedure, where the Morton conversion is not native to the database and therefore the data have to be shipped from Python to Oracle. The steps that take place outside the database within the query procedure are: the 2^n -tree decomposition of the query geometry and time ranges and decoding of the key of the filtered points. Inside the database, the filter step and the refinement step take place. This shipment of data from inside the database to outside and vice versa, makes the query procedure slightly inefficient. Also, because Python is not a compiled language, certain processes e.g. the 2^n -tree decomposition are a bottleneck in the query execution time.

One of the challenges when storing high resolution [SFC](#) keys, is that the query shape may result in a WHERE clause that has to include a large number of continuous ranges when a high resolution [TCs](#) are used. This might be even the case when a merging of ranges takes place. For this reason, two different ways to pose the query have been developed; the usual, WHERE clause execution and the alternative, where the query is a join between two tables, the data table and [QR](#) range table. The WHERE clause query execution is self-explanatory, so no further descriptions will be given. On the other hand, the join of the two tables requires further explanation.

The join methodology starts with the usual 2^n -tree decomposition. However, instead of composing a WHERE clause from the returned ranges, these are inserted in a separate [IOT](#) with columns *LOW*, *UPPER*. Because no overlap is present between the ranges, the index is built on the *LOW* column of the table. In the next step, the data table and the range table are joined and the result is the filter step of the query procedure. An exemplary [SQL](#) statement using the join method is:

```
SELECT /*+ USE_NL (t r)*/ t.morton[, t.z]
FROM DATA_TABLE t, RANGES r
WHERE (t.morton BETWEEN r.low AND r.upper);
```

This method is a very efficient and smart way to retrieve the data as, opposite to the size limits of a WHERE clause string, joins have no theoretical limits. In practice, however, the number of ranges should not be unlimited because at some point the insertion of the ranges inside the database will become a bottleneck. Two remarks need to be made for the join method of posing a query. First of all, there is a need to use an optimiser hint to make this method work efficiently. Without the hint the optimiser proceeds to sort the table (although it is already sorted given it is an [IOT](#)) and the execution time is costly. Using hints is an acceptable way of executing a

statement, although it does not represent the ideal situation. Second of all, the join method does not provide a relative fast execution time for the non-integrated approach. The reasons are not obvious but probably having time as a separate column complicates the join that takes place. As a result, the query execution of the non-integrated approach takes place using a WHERE clause, while for the integrated approach the join method is used.

5.4 EXPERIMENTS

The metrics introduced in [Section 5.2](#) were defined to assess the performance of the proposed methodology. As mentioned in the specific section, the goal is to realise which combination of those metrics balances the fetching time of the filter step. Within this section, the results from a number of tests performed are analysed. All the tests are executed for the queries of the Sand Engine use case and using a small table size (20 million points).

5.4.1 Depth of the tree - integrated approach

The *depth of the tree* experiment studies the effect of using deeper 2^n -TCs for the decomposition of the [QR](#) on both the number of false hits obtained and the fetching time. Because WHERE clauses have practical limits on the length of the string, this experiment is only performed for the integrated approach. The join method allows us to use a large number of Morton ranges, although a limit of 1,000,000 ranges is set for practical reasons. The tests executed start from level 9 and proceed up to level 15 of the 2^n -tree³. For each depth, the query is repeated 3 times (directly after each other). In that manner we obtain both cold and hot runs. A hot run means that the query has been executed once and therefore caching effects take place. In [Section C.1.1](#) of the Appendix, the tables with the results for all queries can be found. The results represent the average of two hot runs⁴. Within this section, graphs from one query per category are represented. The conclusions are, however, derived by combining the two sources of information.

[Figure 5.3](#) presents a representative example of how space - time queries (that treat *z* as an attribute) behave when using finer TCs. The red line represents how the % of extra points (number of false hits compared to number of the refined points) is affected, while the blue line represents the effect on the fetching time. From the graph it is clear that the number of extra points decreases reciprocally to the number of ranges. At level 15 of this query the % of extra points is 67%. The fetching time, following a rather similar trend, decreases up to a certain point and then fluctuates around 0.04 seconds. The fluctuation is, however, very small (around 0.01 seconds) and it can be ignored. [Figure 5.4](#) is a representative example of how space only queries are affected by deeper TCs. From the graph is clear to see that both curves decrease reciprocally to the number of ranges. Actually, the fastest execution time is achieving less than 10% of extra points. Finally, the effect on time only queries is depicted in [Figure 5.5](#). Following the same pattern as before, time only queries are significantly improved by deeper

³ Level 15 is used for practical reasons. The Python script identifying the Morton ranges can result in very expensive execution times which makes testing very time consuming.

⁴ Although the cold runs are not presented in this document, it must be noted that their response time is very similar to the hot runs. For some cases only, the cold run is 0.1 to 0.2 more expensive than the hot runs.

TCs. The relationship between the two variables and the number of ranges is inverse linear, while at level 12 the % of extra points reaches the value of 0 (due to dataset distribution and scaling of time). Time only queries give very interesting results as they converge the fastest to receiving no false hits. This reaction is very much caused by the degree of scaling that is used.

To conclude, in the integrated approach that treats z as an attribute, all types of queries are significantly aided by the finer **TCs**. In fact, the improvement in the % of extra points has no significant negative effect on the execution time.

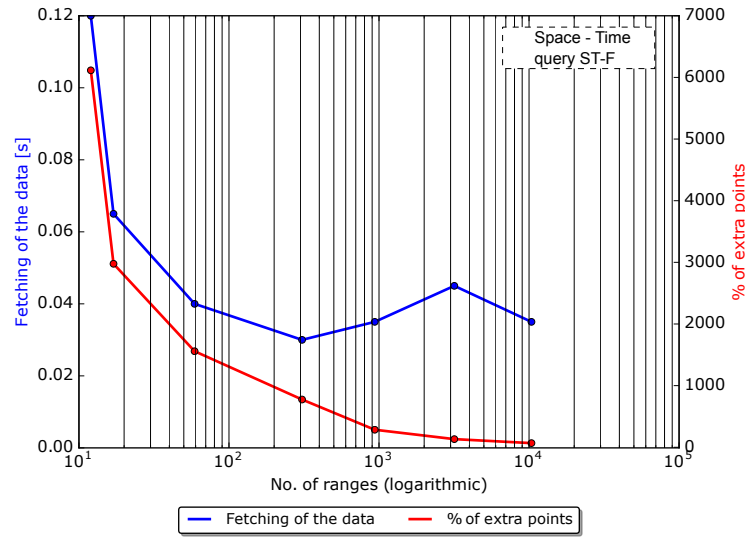


Figure 5.3: Effect of using deeper 2^n -Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space - time queries in the integrated approach (treatment of z as an attribute). ST-F is a line with buffer of 5 metres (776 points).

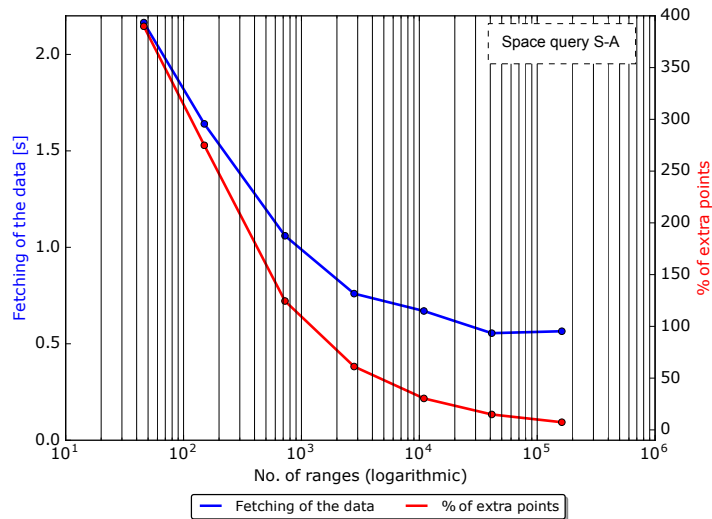


Figure 5.4: Effect of retrieving data from deeper 2^n -Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space only queries in the integrated approach (treatment of z as an attribute). S-A represents a rectangular geometry (112144 points).

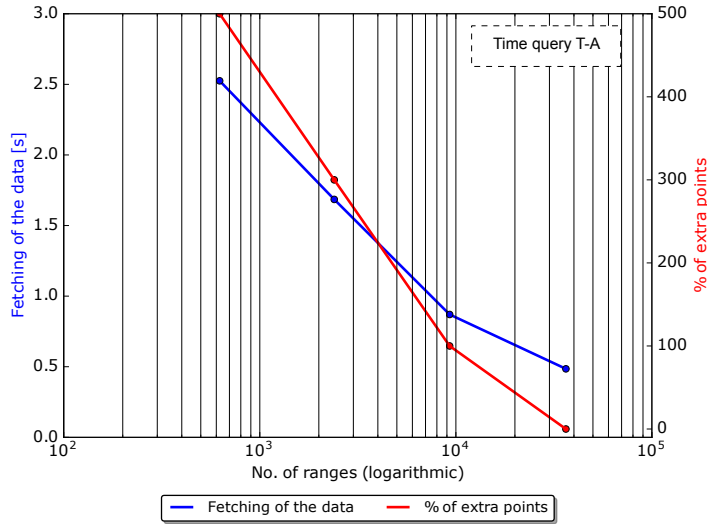


Figure 5.5: Effect of retrieving data from deeper 2^n –Tree Cells on the percentage of extra points obtained and fetching time. Case: Time only queries in the integrated approach (treatment of z as an attribute). T-A represents one day (78902 points).

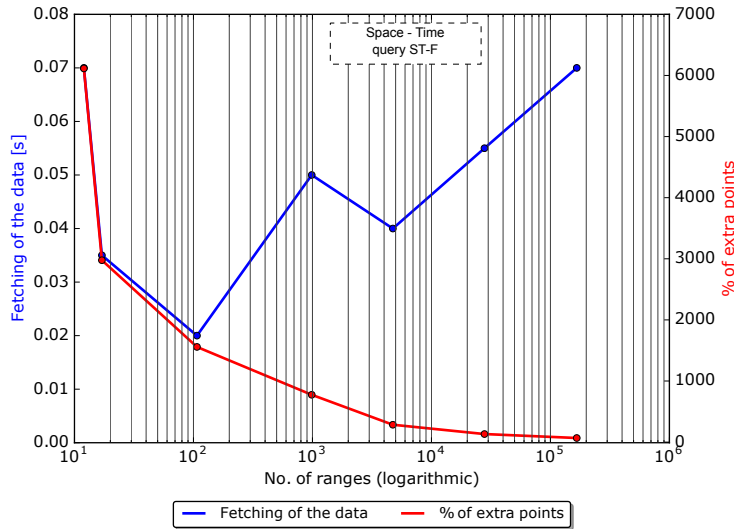


Figure 5.6: Effect of retrieving data using deeper 2^n –Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space - time queries in the integrated approach (treatment of z added in the key). ST-F is a line with buffer of 5 metres (776 points).

So far only the treatment of z as an attribute has been considered. The question therefore arises, whether adding the z dimension in the Morton key affects the previously observed trends that treat it separately. For this reason, the same experiment is repeated for the treatment of z as part of the key. Figure 5.6 represents the effect that deeper TCs have on the same space - time query. As with the previous treatment of z , the % of extra points follows the same pattern of decrease. This is not, however, the case for the fetching time that sees a slight increase after approximately 100 ranges. Of course, the fetching time is still below 0.1 seconds for a much lower amount

of extra points, thus making deeper TCs more desirable in this treatment of z as well. For space only type of queries, the trend is very similar to the treatment as an attribute, although at level 15 there is a slight increase in the fetching time. Nevertheless, the response is still sub-second. Time only queries present no difference between the two treatments, continuing to converge at the same level of the 2^n -tree.

Apart from analysing the patterns between fetching time and the extra points, it is interesting to also investigate the differences in the amount of ranges required between the two treatments of z . For this, the ratio: ranges when z is in the key to ranges when z is attribute is calculated for all levels available. For a schematic investigation of the effect Figure 5.7 is prepared. There we can observe that up to level 10 the number of ranges between the two treatments of z is the same. Then, levels 11 and 12 show the same amount of increase in the ranges for the three types of queries. After this level, space - time and space queries follow a different but rather similar trend. The conclusion is, nonetheless, that the deeper the level of the 2^n -tree, the bigger the ratio, which increases exponentially. This effect is most probably caused due to the jumps in the Morton curve, that decrease the proximity of the points as the number of dimensions interleaved is increased.

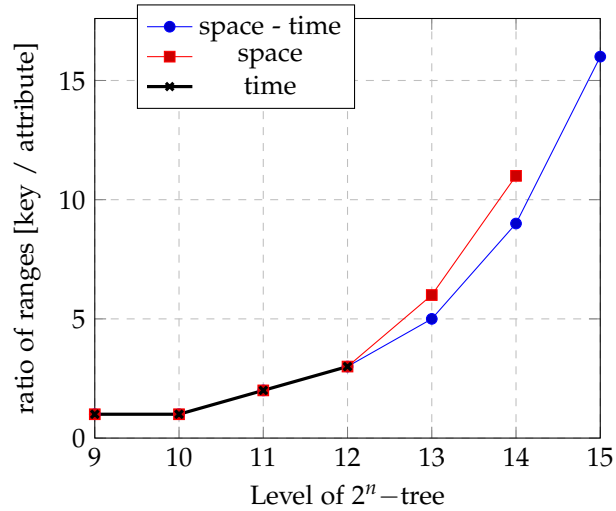


Figure 5.7: The ratio of ranges returned by the z in the key treatment to the number of ranges for the z as attribute treatment when moving deeper in the 2^n -tree

5.4.2 The degree of merging - non-integrated approach

The degree of merging experiment studies the effect of imposing a different degree of merging in the same original TCs on both the % of extra points obtained and the filter fetching time. This test is specially relevant for the non-integrated approach because of potential limitations on the length of WHERE clauses. Studying it will allow us to find out which configuration can balance the filter fetching execution time and the % of extra points. As with the previous test, each query configuration is repeated 3 times, from which the average of the two hot runs is used for making conclusions. The result tables of this experiment for both treatments of z can be found at Tables C.5 and C.6 of the Appendix. Within this section, graphs from one

query per category (space -time and space only queries) are represented⁵. The conclusions are, however, derived by also generalising the results available in the tables.

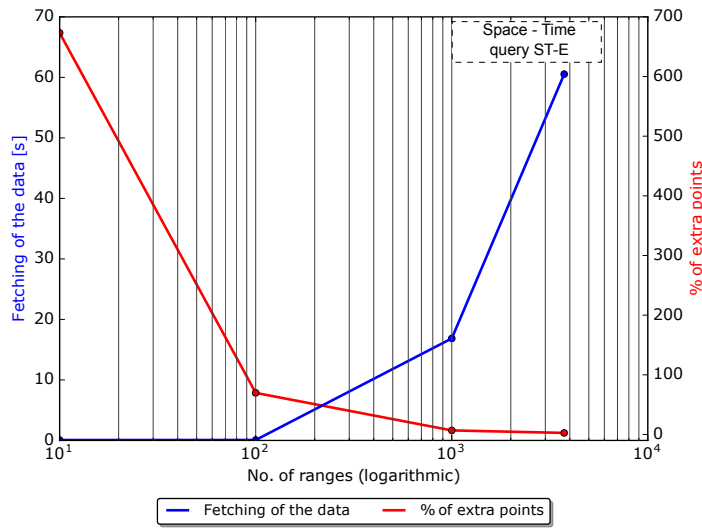


Figure 5.8: Effect of imposing a different degree of merging in the same original Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space - time queries in the non-integrated approach (treatment of z as an attribute). ST-E represents line with buffer of 5 metres (380 points).

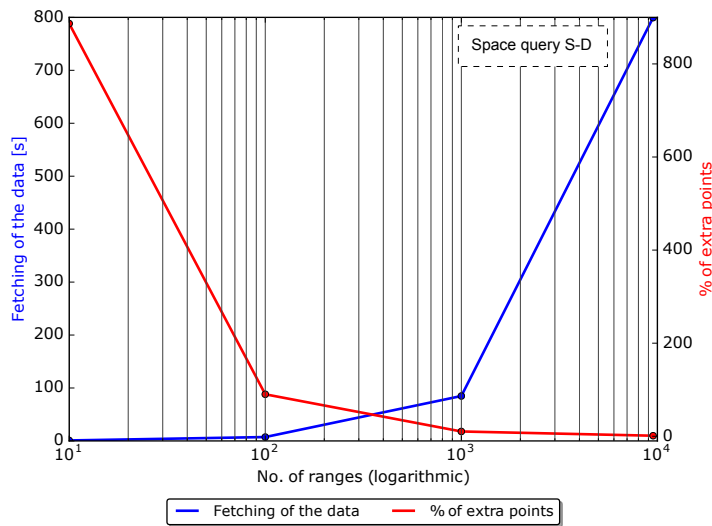


Figure 5.9: Effect of imposing a different degree of merging in the same original Tree Cells on the percentage of extra points obtained and the fetching time. Case: Space only queries in the non-integrated approach (treatment of z as an attribute). S-D represents line with buffer of 5 metres (11933 points).

⁵ Keep in mind that time queries are executed without having to identify Morton keys.

Figure 5.8 presents a representative example of a space - time query while, Figure 5.9 an example of a space only query. Both types of queries in the non - integrated approach present the same trends. With the increase of the number of ranges in the WHERE clause, as expected, the % of extra points decreases reciprocally. However, opposite to the integrated approach, a larger number of ranges leads to an exponential increase in the filter fetching time. For space only queries, even, the increase is so sharp that it results in fetching times of hundreds of seconds. This behaviour is not suitable for any of the two use cases. By closely observing the graphs one can realise that a merging of 200 offers a rather good balance between the % extra points and the fetching time. For this reason, the merging of 200 is used throughout the rest of the experiments and the benchmarks of the next section.

5.4.3 Depth of the tree with merging - non-integrated

Having identified the degree of merging that balances queries in the non-integrated approach, the depth of tree experiment is performed. Contrary to the integrated case, a merging to a maximum of 200 ranges takes place at the same time. This experiment is performed for two reasons: first, to explore how the non-integrated approach performs when moving deeper in the tree and second, to find out whether composing the 200 ranges from deeper TCs has an effect on the % of extra points. The result tables of this experiment for both treatments of z can be found in Tables C.7, C.8 and C.9, C.10 of the appendix.

Starting with this experiment, the previous conclusions that fetching time is increased with a higher number of ranges, is corroborated here as well. The faster the execution time, the more extra points are returned during the filter step. This is, however, not ideal as receiving a large amount of extra points makes the CPU-intensive refinement step a very expensive operation. The second observation from the results is that composing 200 ranges from deeper TC results in no significant gain in the % of extra points obtained. We can therefore already conclude that the non-integrated approach will not be the best solution when managing dynamic point clouds.

6 | BENCHMARK DESIGN AND RESULTS

Apart from the specific tests performed in the previous chapter, a complete benchmark was designed and executed. Its purpose is to test the performance of the proposed storage model in terms of storage space, loading time and query response times. The detailed description of the used datasets is available in [Appendix A](#). Three benchmark stages have been designed for each use case, each one of them being two times bigger than the previous one. With the three benchmark stages (small, medium, large) the aim is to compare: how the size of the stored data scales between benchmarks, what is the effect of adding new spatio-temporal data in batches and what can we conclude about the scalability of the queries. The final goal is to gain insight into which storage model is the most optimal for each use case.

Since both use cases originate from the same coastal monitoring domain, there are some characteristics of the benchmarks which are shared between them. First of all, each alternative storage model is run separately from the rest, until both loading and querying are completed. Second of all, during the loading procedure the medium and large benchmarks do not include a fresh reloading of the previous stage. This is done in order to take into consideration the growing nature of the scenarios. As a result, the new data are added to the already stored points. Finally, the queries ([Appendix B](#)) are executed for each of the four storage models and all three benchmark stages (12 combinations in total). Each query is executed both in a cold and hot system (repeated 6 times) before moving to the next query. The results that are presented in this document are processed as follows: the most and least expensive response time are ignored and an average is calculated from the remaining 4. As a result, the presented number correspond to hot runs. Only the fetching from the filter step (along with the number of points and the percentage of the extra points obtained between the two querying steps) is given in the following tables. 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, currently of secondary importance.

approach	treatment	notation
non-integrated	z attribute	xy
non-integrated	z added	xyz
integrated	z attribute	xyt
integrated	z added	xyzt

Table 6.1: The benchmark cases and the used notation.

Within the rest of this section, the notation in [Table 6.1](#) is used for the four storage models executed. For each one of these, the notation is complemented with the characters: S (for small), M (for medium) and L (for large),

in order to show which benchmark size is being considered for the specific results.

The rest of the chapter is organised as follows: In [Chapter 6](#) the benchmark design and results are analysed. In [Section 6.1](#) and [Section 6.2](#) the methodology is validated using a naive Oracle spatial approach. Finally, in [Section 6.4](#) a summary of the results presented is given.

6.1 SAND ENGINE

The Sand Engine benchmark stages have approximately the same spatial extent but different extent on the temporal dimension. The details of the benchmark stages are available in [Table 6.2](#). As is it is easy to see, the dataset is not massive in size but has a higher dynamic nature (compared to the coastline dataset presented in [Section 6.2](#)). Some of the results in this section are also included in [Psomadaki et al. \[2016\]](#).

Stage	Points	Days	Size (MB)	Description	No. of files
Small	19M	230	347	from 2000 to 2002	230
Medium	44M	554	836	from 2000 to 2006	554
Large	74M	931	1414	from 2000 to 2015	931

Table 6.2: Benchmark stages description of the Sand Engine use case. The size column corresponds to the size of the LAS files. More information about the dataset can be found in [Section A.2.1](#).

6.1.1 Loading procedure

[Table 6.3](#) presents the results of the loading procedure for the four storage models and the three benchmark stages. No parallel processing is used for any of these steps.

Approach	Time (s)			Size (MB)	Points		Points per sec.	
	conversion	Load heap	Load IOT		Heap	IOT	Heap	IOT
xy - S	105.43	11.79	13.60	471	18,147,709	18,147,709	1,539,024	1,334,390
xy - M	145.14	16.56	49.65	1130	25,561,106	43,708,815	1,543,433	880,339
xy - L	167.75	19.72	78.00	1897	30,205,111	73,913,926	1,531,699	947,614
xyz - S	352.37	9.91	10.5	368	18,147,709	18,147,709	1,830,384	1,728,353
xyz - M	498.79	14.24	34.07	885	25,561,106	43,708,815	1,794,832	1,282,912
xyz - L	590.00	16.77	61.71	1495	30,205,111	73,913,926	1,801,161	1,197,763
xyt - S	349.68	11.79	13.09	471	18,147,709	18,147,709	1,539,024	1,386,380
xyt - M	492.29	16.56	40.39	1130	25,561,106	43,708,815	1,543,433	1,082,169
xyt - L	594.10	19.72	74.11	1897	30,205,111	73,913,926	1,531,699	997,354
xyzt - S	435.48	11.79	10.78	386	18,147,709	18,147,709	1,539,024	1,683,461
xyzt - M	604.27	16.56	33.21	927	25,561,106	43,708,815	1,543,433	1,316,134
xyzt - L	722.08	19.72	57.96	1566	30,205,111	73,913,926	1,531,699	1,275,258

Table 6.3: The loading times for the two integrations of space and time and the two treatments of z for the Sand Engine use case. For more insight concerning the *Load heap* and *Load IOT* columns refer to [Listing D.1](#) and [Listing D.3](#) respectively.

From the results it is easy to realise that the most expensive phase in the loading procedure is currently the Morton conversion. The reason behind

this is that the code being used is non-optimised Python code. In addition to that, adding one extra dimension in the key ($xy \rightarrow xyz$, $xyt \rightarrow xyzt$), makes the conversion a more expensive operation. This is an expected behaviour, given that the complexity of the algorithm increases. Concerning the bulk loading of the data into the heap table, it can be observed that the process costs approximately the same amount of time for all four storage models. This observation can be explained since the same loading utility (SQLDR) is being used for the same amount of points. Finally, for the loading into the **IOT** it can be distinguished that, between the two integrations of space and time, the same treatment of z costs approximately the same amount of time. Having z in the key, makes the creation of the **IOT** faster. One explanation for this behaviour could be the use of the NUMBER data type in all columns being created.

The storage requirements for all four storage models are schematically presented in Figure 6.1. Apparently, the storage models requiring the most space are the cases where z is treated as an attribute. Both cases actually have the same storage requirements. Nonetheless, adding z in the key for the non-integrated approach (xyz), requires less space than the xyt case (of the integrated approach). This means that obtaining a 3D key with only the spatial components requires less space than having a 3D key with the x , y and time dimensions. Finally, we can observe that the **IOT** requires slightly more storage than the corresponding LAS files.

To conclude, among all storage models, the non-integrated approach with z in the key offers the fastest loading and the least storage requirements. This could be considered as an expected behaviour since adding new points in the **IOT** should not affect the already organised points. However, the $xyzt$ case of the integrated approach, also, shows very close results and this suggests that having all four dimensions in one key is a compact and appropriate solution for storing dynamic point clouds.

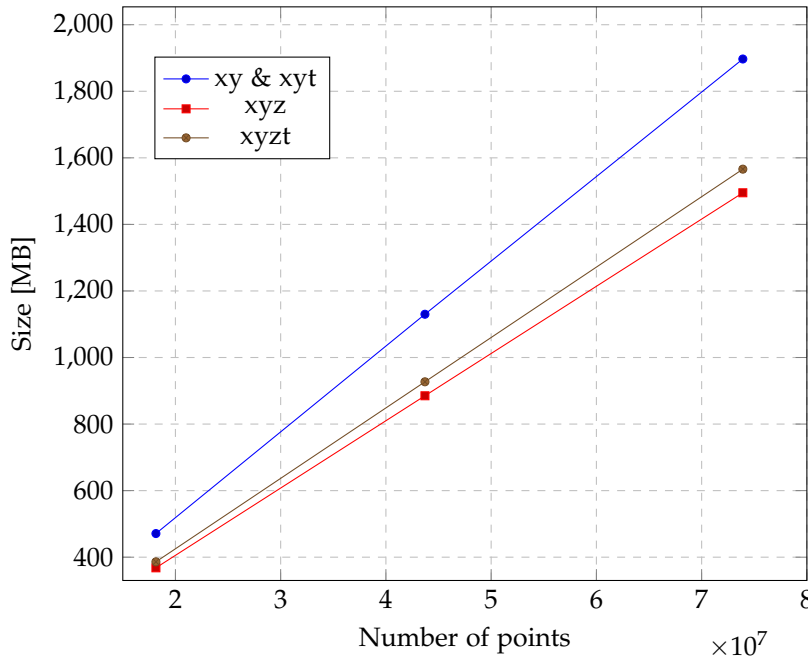


Figure 6.1: The storage requirements of the 4 storage models for the 3 benchmark stages

6.1.2 Query procedure

The scalability of the queries is tested within this section. The queries have been chosen in a way that allows them to be executed in all three benchmark stages. Like the loading procedure, no parallel processing is used. The decision was based upon the fact that although parallelisation was enabled for the queries in certain test cases, only one core was actually being used during query execution¹. Parallelisation, however, seems to work in the non-integrated approach and during space queries. In space queries the primary key (time, Morton) is not used for fetching of the data and that seems to enable parallelisation. To have consistency between our results, no parallel processing is used.

Table 6.4 presents the results of the hot run of the 12 queries in the non-integrated approach. The execution order of the queries 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. To have a fair comparison between the two integrations (due to the different query methods and the merging of 200 taking place in the non-integrated approach), in Table 6.5, results from the integrated approach with maximum number of ranges set also to 200 are presented. In addition to that, the results from the integrated approach with maximum allowed ranges set to 1,000,000 are presented in Table 6.6.

From the query results we observe that the integrated approach with 1,000,000 maximum number of ranges offers the fastest response times for the majority of the queries and the lowest % of extra points. Other more specific observations are (see also Figure 6.2 and Figure 6.3):

- In all of the three tables the response times for space - time and time queries are constant between the benchmark stages. This is a desired characteristic which means that the system adapts well, even when doubling the size of the database. For space queries, the number of points increases with each benchmark because more data (days) are present in the database and thus the output of the queries is larger (observe the *final points* columns of the space queries).
- Comparing the two integrations using the same amount of maximum ranges, we can see that the integrated approach is affected with a lot extra points. Even so, the response time of the integrated approach continues to be better for space - time and space queries. Again, the non-integrated approach is proven to be not the most optimal solution for querying spatio-temporal point clouds.
- Comparing the two treatments of *z* in the non-integrated approach, we see an overall deterioration both in the query response times and the % of extra points of space - time and space queries when *z* is added to the key. *z* in the key is more expensive as the whole extent of the *z* dimension has to be used during the range identification. Time queries, on the other hand, see an improvement, firstly because time is dominant and secondly because one less column has to be fetched from disk.
- Comparing the two treatments of *z* in the integrated approach we can observe that: 1. For the case when the maximum number of ranges is 200, there is an improvement in the response times of all types of queries. However, the % of extra points for space - time and

¹ Possibly a bug in the Oracle database

some space queries seems to almost double when treating z as part of the key. 2. When the maximum number of ranges is 1,000,000 we can see a slight increase (0.1 seconds) in the response time of space - time queries, and in certain cases double % of extra points. This is caused because the two storage models do not reach the same maximum depth in the tree. The solution is, of course, to fetch the same amount of extra points (go one level deeper). Space queries, have an overall improvement when z is part of the key, but more % of extra points are fetched. Time queries are improved when z is in the key and there is no effect in the % of extra points.

- By comparing the time queries, one can see that the slowest response times are found in the integrated approach with the merging of 200. The reason is the large amount of extra points being fetched due to the high degree of merging. Contrary to the previous case, the rest of the cases examined have time queries of similar response times.
- By comparing the % of extra points in all three of the tables for both treatments of z , we can observe that line buffer queries (ST-E, ST-F, S-D) receive the largest amount of extra points, when compared to polygons and circles. In general, such diagonal geometries with a small width are difficult to be approximated. Nevertheless, the decomposition used here is going to be better than a bounding box approximation (see also Figures 6.4 and 6.5).
- One undesired effect can be found in the space queries of the integrated approach: when moving from the medium to the large benchmark the number of extra points seems to double. Although not shown here, during the large benchmark the decomposition algorithm reaches one level less compared to the small and medium benchmark. This can be resolved by developing an algorithm that identifies the maximum depth of the tree in a more dynamic way.

	id	fetching (s)			% extra points			Final Points		
		S	M	L	S	M	L	S	M	L
z attribute	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
	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
	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
z in key	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
	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
	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

Table 6.4: Query response times, the percentage of false hits compared to the actual number of points and, the points returned by the queries for the **non-integrated approach** with **maximum number of ranges set to 200**. For more insight about how space - time, space and time queries are performed refer to [Listing D.10](#), [Listing D.9](#), and [Listing D.6](#) respectively.

	id	fetching (s)			% extra points			Final Points		
		S	M	L	S	M	L	S	M	L
z attribute	ST-A	0.05	0.04	0.04	36	36	36	3927	3927	3927
	ST-B	0.09	0.08	0.09	206	206	206	4237	4237	4237
	ST-C	0.05	0.05	0.05	111	111	111	2812	2812	2812
	ST-D	0.03	0.03	0.03	55	55	55	2185	2185	2185
	ST-E	0.02	0.02	0.02	402	402	402	380	380	380
	ST-F	0.03	0.03	0.03	798	798	798	327	327	327
	S-A	1.50	5.05	14.09	222	309	433	86017	231283	509964
	S-B	0.12	0.48	1.32	597	849	925	2788	8934	23949
	S-C	0.30	1.04	3.83	338	474	1243	11501	32983	54111
	S-D	1.58	5.61	26.08	2339	3095	5392	11933	33494	90670
	T-A	3.37	3.39	3.41	710	710	710	78902	78902	78902
	T-B	4.84	4.86	4.86	483	483	483	157806	157806	157806
z in key	ST-A	0.04	0.04	0.04	136	136	136	3927	3927	3927
	ST-B	0.06	0.06	0.06	267	267	267	4237	4237	4237
	ST-C	0.04	0.04	0.04	262	262	262	2812	2812	2812
	ST-D	0.02	0.02	0.02	130	130	130	2185	2185	2185
	ST-E	0.02	0.02	0.02	962	962	962	380	380	380
	ST-F	0.02	0.02	0.02	1512	1512	1512	327	327	327
	S-A	0.88	2.91	7.80	254	346	439	86017	231283	509964
	S-B	0.10	0.31	0.76	900	994	974	2788	8934	23949
	S-C	0.25	0.65	2.14	565	568	1281	11501	32983	54111
	S-D	0.85	3.11	14.06	2374	3119	5403	11933	33494	90670
	T-A	1.84	1.83	1.83	710	710	710	78902	78902	78902
	T-B	2.60	2.61	2.63	483	483	483	157806	157806	157806

Table 6.5: Query response times, the percentage of false hits compared to the actual number of points and, the points returned by the queries for the **integrated approach** with **maximum number of ranges set to 200**. For more insight about how space - time, space and time queries are performed refer to [Listing D.8](#).

	id	fetching (s)			% extra points			Final Points		
		S	M	L	S	M	L	S	M	L
z attribute	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
	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
	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
z in key	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
	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
	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

Table 6.6: Query response times, the percentage of false hits compared to the actual number of points and, the points returned by the queries for the **integrated approach** with **maximum number of ranges set to 1,000,000**. For more insight about how space - time, space and time queries are performed refer to [Listing D.8](#).

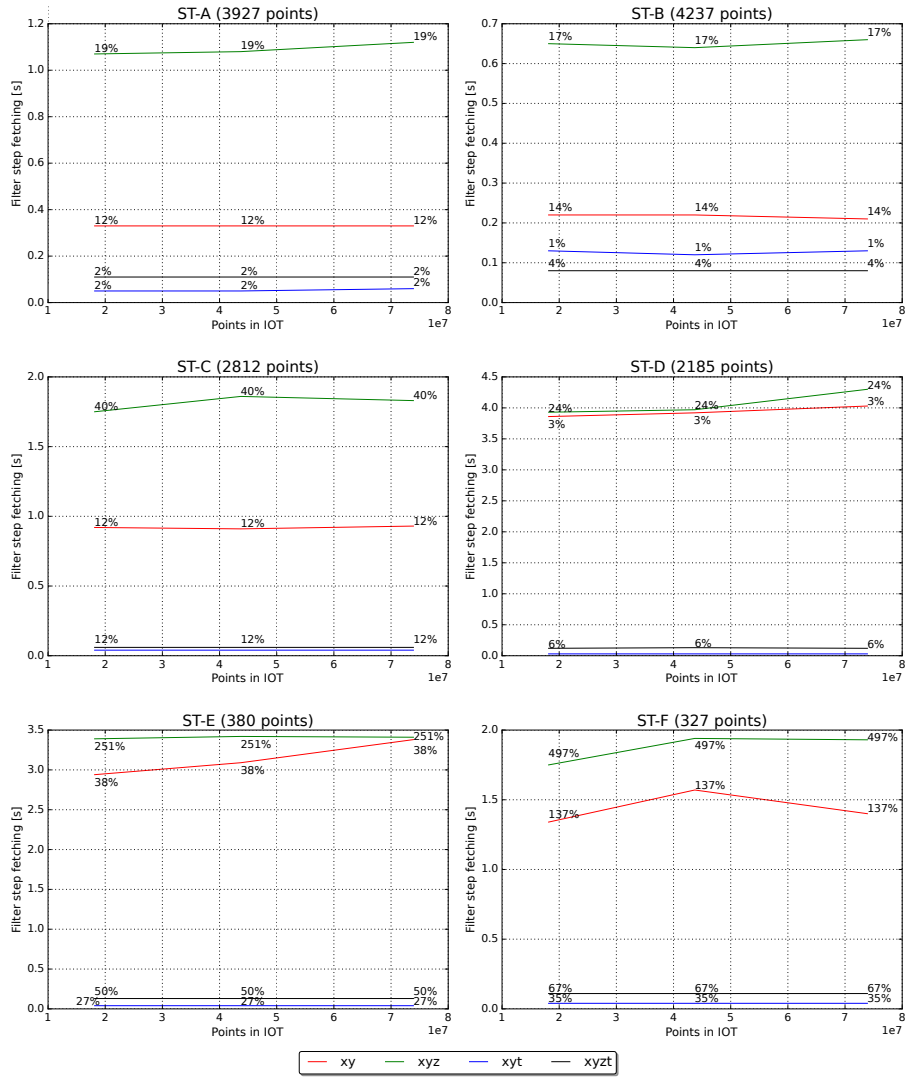


Figure 6.2: Schematic representation of the results found in Table 6.4 and Table 6.6. The x axis represents the points available in the IOT, while the y axis represents the query response times. Each line also contains the % of false hits compared to the actual number of points. Case: **Space - time queries**

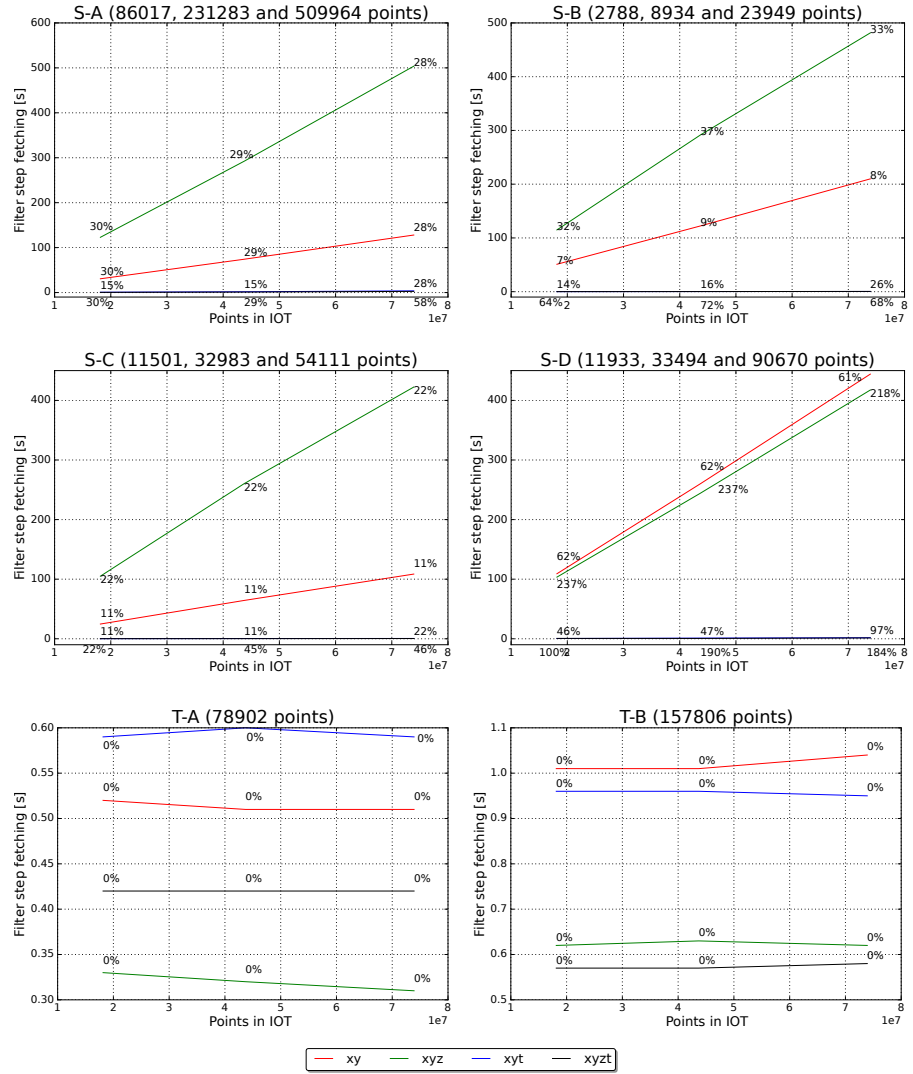


Figure 6.3: Schematic representation of the results found in Table 6.4 and Table 6.6. The x axis represents the points available in the IOT, while the y axis represents the query response times. Each line also contains the % of false hits compared to the actual number of points. Case: **Space only and time only queries**

6.2 COASTLINE

The Coastline benchmark stages have approximately the same spatial extent but different extent on the temporal dimension. The details of the benchmark stages are available in [Table 6.7](#). Contrary to the Sand Engine dataset, the Coastline one is more massive in size but lacks dynamic nature (only 4 years are available).

Stage	Points	years	Size (GB)	Description	No. of files
Small	500M	1	9.4	2012	13
Medium	995M	2	18.7	2012 to 2013	120
Large	2020M	4	37.9	2012 to 2015	338

Table 6.7: Benchmark stages description of the Coastline use case for the **Full benchmark**. The size column corresponds to the size of the LAS files. More information about the dataset can be found in [Section A.2.2](#).

Because the dataset is massive, first, a simplified *small benchmark* with about 140 million points was executed. This small benchmark (same temporal configuration as [Table 6.7](#)) covers the area around the Sand Engine. The main purpose of this benchmark was to choose the right scaling of the time dimension (10,000 see also [Section 5.3.1](#)). Having configured the use case the *medium and large benchmark* were performed. The medium benchmark covers a big part of the province of South Holland, and the large benchmark covers the whole coastline of the province of South Holland. The results of the above mentioned benchmarks can be found in [Section C.2](#) of the Appendix. Finally, the *full benchmark* of the whole coastline is executed and used within this section to study the scaling behaviour of the storage model. This stepwise benchmark approach with the different spatial extents allowed more control over the process. From the results of the small and medium benchmarks I came to the conclusion that the non-integrated approach was not a good solution for the time being and therefore, was not executed for the rest of the benchmarks.

6.2.1 Loading procedure

[Table 6.8](#) presents the results of the loading procedure for the integrated approach. For the reasons explained in previous sections, no parallel processing is taking place in any of the following results.

Approach	Time (hr)			Size (GB)	Points		Points per sec.	
	conver.	Load heap	Load IOT		Heap	IOT	Heap	IOT
xyt - S	2:48	0:05	0:33	11.4	500,212,673	500,212,673	1,452,146	246,034
xyt - M	2:36	0:05	0:47	22.7	494,909,041	995,121,714	1,562,439	352,724
xyt - L	5:38	0:11	0:56	46.2	1,025,476,822	2,020,598,536	1,492,237	593,290
xyzt - S	3:21	0:05	0:11	9.9	500,212,673	500,212,673	1,659,871	746,965
xyzt - M	3:12	0:04	0:19	19.6	494,909,041	995,121,714	1,749,987	853,625
xyzt - L	6:42	0:09	0:40	39.8	1,025,476,822	2,020,598,536	1,737,570	822,512

Table 6.8: The loading times of the integrated approach for the two treatments of z in the coastline use case (**Full benchmark**). For more insight concerning the *Load heap* and *Load IOT* columns refer to [Listing D.1](#) and [Listing D.3](#) respectively.

The observations made for the Sand Engine use case are corroborated for this use case as well. This time, however, it is easy to see how more expensive the Morton conversion in Python is, especially when compared to the loading on the heap table and the [IOT](#). The loading into the heap is a very fast operation considering the amount of points and is approximately of the same magnitude for both treatments of z . Contrary to that, adding z in the key makes the creation of the [IOT](#) considerably faster. Actually, the loading in the [IOT](#) when using z as an attribute is unexpectedly slow; approximately 300,000 points per second when loading 500 million points. Concerning the storage requirements, when treating z as an attribute, the size of the database is 20% bigger than the size of the LAS files. On the other hand, for the z in the key treatment, the storage is closer (5%) to that of the LAS files. Finally, when comparing the storage sizes of the two treatments of z , having z as an attribute requires 15% more storage space than when z is in the key.

To conclude, among the two treatments of z the fastest loading and the least storage requirements are provided when z is part of the key. This is the most compact solution as the table only consists of one column containing all four dimensions ($x,y,z,time$).

6.2.2 Query procedure

This section examines the scalability of the queries. The queries were picked such that they could be executed in the small and medium benchmarks discussed before. This allows me to test whether there are significant differences when using a different spatial extent. The execution order of the queries is as follows: ST-A, S-A, ST-B, ST-C, ST-D, ST-E, ST-F, ST-G, S-B, S-C. Parallisation is not enabled in any of the queries executed. [Table 6.9](#) presents the query results of the 10 queries executed. Queries ST-F and ST-G are outside domain in the time dimension for the small benchmark stage and therefore no data are presented. For both treatments of z , the maximum number of allowed ranges is set to 1,000,000.

In general, we can conclude that treating z as part of the key offers, in general, better response times for both space and space - time queries, even if the z dimension is not used in the selection². However, the faster response comes at the cost of more extra points being fetched. This can be easily resolved by moving deeper into the 2^n -tree. For practical reasons this was not applied because the Python code developed for the range identification becomes very expensive, something that would have made the benchmarks very timely to be executed. Other more specific remarks are:

- Both storage models present constant response times for space - time queries that only depend on the number of points that need to be fetched and not on the size of the table.
- Space queries, in this case as well, present a doubling of the percentage of extra points when moving either from the small benchmark stage to the medium or from the medium to the large. As proposed before, this can be resolved by having a more dynamic algorithm for identifying the maximum depth of the tree.

² If z was to be used in the selection, then the result is expected to be much better.

- When treating z as part of the key, line buffer queries receive the largest amount of extra points. For the treatment of z as an attribute this seems to only apply during space queries.
- When comparing the *small* (Table C.16), *medium* (Table C.18), *large* (Table C.20) and *full benchmarks* (Table 6.9) we can see that the query response times remain the same. As opposed to that, the percentage of extra points is not completely unaffected by the change in the spatial extents. When moving from the medium to the large benchmark, the percentage of extra points when z is part of the key seems to increase. When looking closely at the raw results, one can immediately observe that the depth of the tree in the large and full benchmark is one level less than the one used in the medium and small. Again this has to do with the implemented algorithm for approximating the query geometry, which is not dynamic.

	id	fetching (s)			% extra points			Final Points		
		S	M	L	S	M	L	S	M	L
xyt	ST-A	2.29	2.20	2.20	3	3	3	389554	389554	389554
	ST-B	1.98	1.95	1.95	4	4	4	342239	342239	342239
	ST-C	0.48	0.49	0.48	4	4	4	69339	69339	69339
	ST-D	0.46	0.50	0.47	2	2	2	69967	69967	69967
	ST-E	0.86	0.88	0.83	4	4	4	134998	134998	134998
	ST-F	-	2.59	2.48	-	5	5	-	461432	461432
	ST-G	-	3.15	3.06	-	3	3	-	518702	518702
	S-A	0.45	0.96	1.73	2	2	4	71999	151687	301429
	S-B	0.14	0.29	0.49	9	9	17	20135	38523	71823
	S-C	0.57	1.12	2.27	5	10	10	98382	174746	382500
xyzt	ST-A	1.25	1.35	1.34	7	7	7	389554	389554	389554
	ST-B	1.11	1.22	1.23	8	8	8	342239	342239	342239
	ST-C	0.32	0.32	0.33	17	17	17	69339	69339	69339
	ST-D	0.41	0.40	0.41	9	9	9	69967	69967	69967
	ST-E	0.57	0.60	0.58	17	17	17	134998	134998	134998
	ST-F	-	1.53	1.53	-	5	5	-	461432	461432
	ST-G	-	1.97	1.97	-	7	7	-	518702	518702
	S-A	0.28	0.58	1.10	8	16	16	71999	151687	301429
	S-B	0.10	0.19	0.39	32	32	63	20135	38523	71823
	S-C	0.39	0.77	1.65	20	39	39	98382	174746	382500

Table 6.9: The query response times, the percentage of false hits compared to the actual number of points and, the number of points returned by the queries in the integrated approach of the coastline dataset (**Full benchmark**).

6.3 VALIDATION AND COMPARISON

To validate that our implemented prototype returns the right amount of points and to have some sort of comparison, the naive approach of using Oracle spatial and date data types is implemented. For this a 3 dimensional SDO.POINT is used together with 2D R-Tree for fast spatial access. To also achieve fast access in the time dimension, a B-Tree is built on the time column. For the SQL scripts used, the reader is referred to Section D.3.

To perform the validation, the Sand Engine dataset is used. For this the same benchmark stages are executed, with the same query geometries. The results of the loading procedure are presented in [Table 6.10](#). When compared to any of the storage models of the proposed methodology ([Table 6.3](#)), certain observations can be made. The total execution time of the loading procedure is 3 to 6 time more expensive, mostly due to the R-Tree index generation. The least expensive operation is the B-Tree index. Note, that both R-Trees and B-Trees are built from scratch when moving from one benchmark stage to the next. The storage requirements of the three benchmark stages are presented in [Table 6.11](#). Again the differences are quite noticeable when compared to the proposed methodology. In general, the Oracle spatial approach requires 5 times more storage when compared to the storage model with the highest storage requirements.

Approach	Time (s)				Points
	preparation	Load	R-Tree	B-Tree	
S	31.04	244.27	891.99	15.37	18,147,709
M	45.07	337.68	2273.38	31.69	43,708,815
L	51.48	400.27	4099.36	73.72	73,913,926

Table 6.10: The loading response times for the validation of the Sand Engine use case. For more insight concerning how the data are loaded, refer to [Section D.4](#).

Approach	Size (MB)				Points
	Table	R-Tree	B-Tree	total	
S	1007	1097	371	2475	18,147,709
M	2414	2643	891	5948	43,708,815
L	4065	4468	1505	10038	73,913,926

Table 6.11: The storage requirements for the validation of the Sand Engine use case.

id	Total time (s)			% extra points			Final Points		
	S	M	L	S	M	L	S	M	L
ST-A	0.52	1.28	1.89	0	0	0	3927	3927	3927
ST-B	0.83	2.08	3.85	106	106	106	4237	4237	4237
ST-C	0.36	0.72	2.52	116	116	116	2812	2812	2812
ST-D	0.33	0.42	1.00	163	163	163	2185	2185	2185
ST-E	0.29	0.32	0.41	7752	7752	7752	380	380	380
ST-F	0.21	0.28	0.79	11961	11961	11961	327	327	327
S-A	0.52	1.19	2.03	203	213	207	86017	231283	509964
S-B	0.11	0.16	0.24	93	120	105	2788	8934	23949
S-C	0.16	0.28	0.39	27	25	27	11501	32983	54111
S-D	0.17	0.32	0.69	1011	940	860	11933	33494	90670
T-A	0.19	1.97	0.19	0	0	0	78902	78902	78902
T-B	0.31	1.97	0.31	0	0	0	157806	157806	157806

Table 6.12: The query response times, the percentage of false hits of the filter step compared to the actual number of points and, the number of points returned by the queries in the validation of the Sand Engine use case.

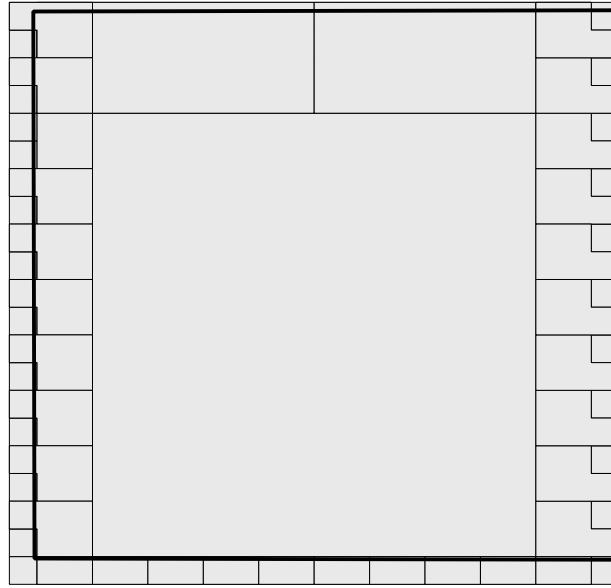
When benchmarking the query procedure, the same configuration as defined before is used. The queries are repeated 6 times and the presented results are the average of four left when excluding the least and the most expensive response. Since an R-Tree index is built for fast spatial access, the Oracle database also follows a two step approach during query execution. However, contrary to the implemented methodology, this two step process is transparent to the user. To have an idea how well the filter step (using the SDO.FILTER operator) approximates the query geometry, the operator was executed separately and the number of points was counted. This allows me to have the same percentage (%) of extra points comparison but this time using an MBR approximation.

The results from the query execution are available in Table 6.12. Note that contrary to the proposed storage models (Table 6.6), the times presented here represent the total query execution time (both filter and refinement step) and are for most of the cases better than the total time from the implemented methodology. However, this naive approach for managing spatio-temporal point clouds does not provide constant execution times between the benchmark stages of space - time queries. This suggests that the combination of an R-Tree and B-Tree index does not scale good. In addition to that, time queries show a fluctuating behaviour that cannot be explained at this moment in time. These are crucial characteristics that make this method not suitable. For the validation part of this benchmark, we can indeed observe that the implemented methodology does return the correct amount of points. Finally, when comparing the MBR approximation during the filter step with the 2^n -tree decomposition used within the proposed methodology we observe that:

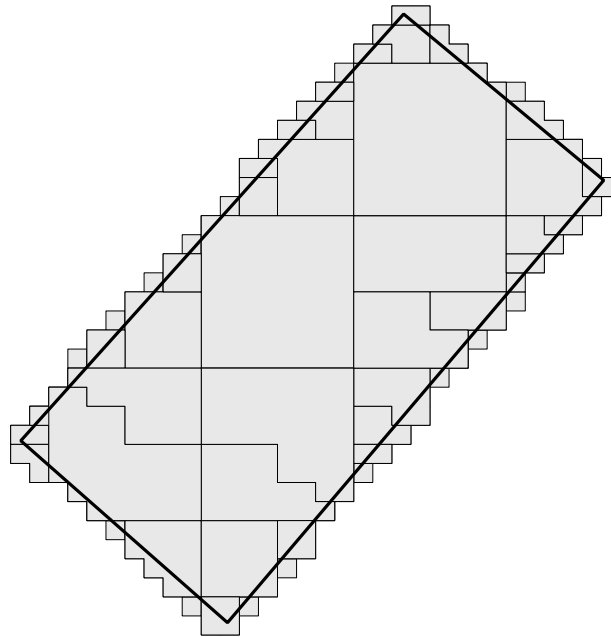
- Axis aligned rectangles (ST-A) are fully approximated using the MBR solution. The quadtree decomposition (2D) is nevertheless not worse since it returns 2 % extra points. The 2^n -tree decomposition of query ST-A is depicted in Figure 6.4a.
- Polygons and circles in the validation approach seem to return approximately twice the amount of the actual points during the filter step. The 2^n -tree decomposition, on the other hand, returns less than 15 % extra points for space - time queries, while space queries have at the worst case 70% extra points. The quadtree decomposition of a polygon (ST-B) is depicted in Figure 6.4b and of a circle (S-C) in Figure 6.5a.
- Line buffers are the worst approximated geometries when using MBRs. In the case of the ST-F query, even, the filter step returns 120 times more points than the actual number. The 2^n -tree decomposition, however, when being at the right depth can achieve to describe such geometries very precisely. A focused view of the quadtree decomposition for a line buffer query (ST-E) before any merging takes place is depicted in Figure 6.5b.

6.4 SUMMARY

Within this chapter a benchmark was designed and executed. From the presented results (as a whole) we can conclude that the integrated approach is the best way to manage and query dynamic point clouds. From the managing point of view, this approach offers a very compact storage model, which

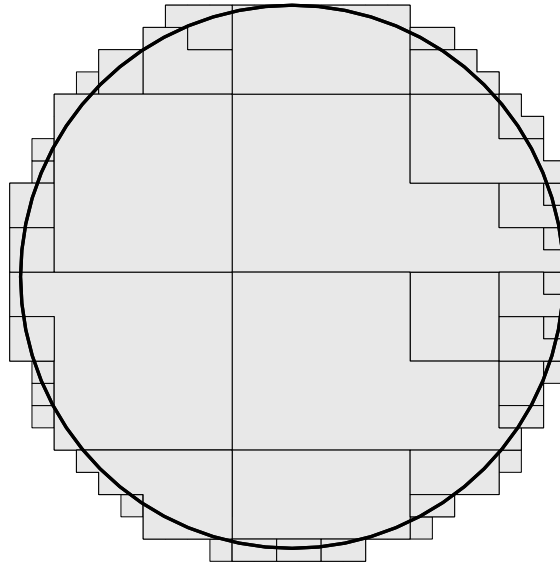


(a) The decomposition of an axis aligned polygon (Query ST-A) using quadtree cells.

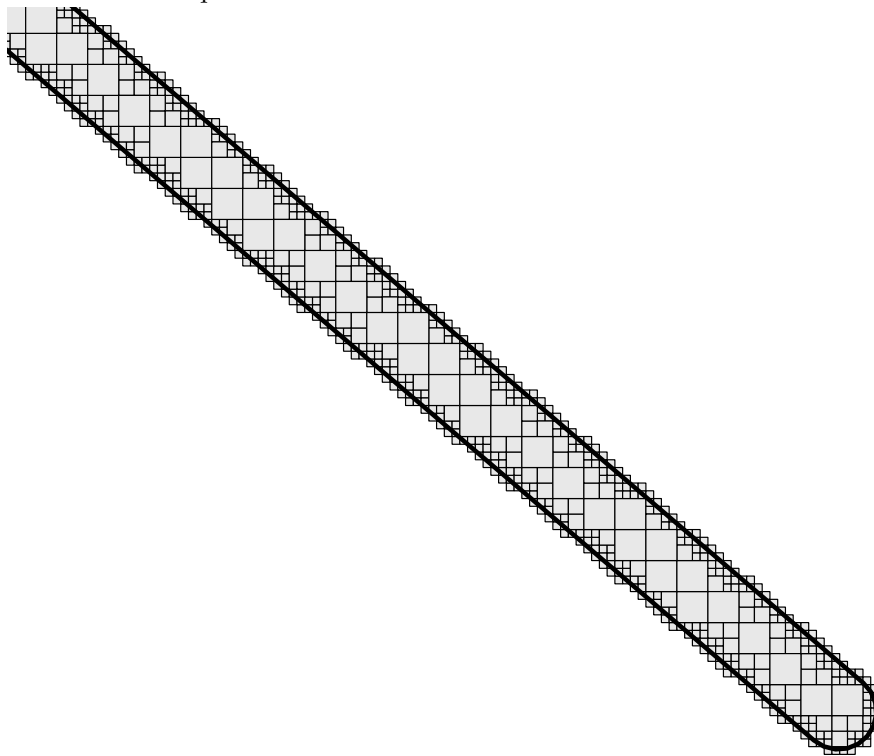


(b) The decomposition of a polygon (Query ST-B) using quadtree cells.

Figure 6.4: Morton ranges decomposition of polygonal geometries



(a) The decomposition of a circle (Query S-C) using quadtree cells.



(b) The decomposition of a line buffer geometry (Query ST-E) using quadtree cells.

Figure 6.5: Morton ranges decomposition of circular and line buffer geometries

in the case of z *in the key*, includes an [IOT](#) with only one column. From the query point of view, this approach offers scalable query response times and significantly less false hits than a [MBR](#) approximation, usually used in today's spatial databases. Nevertheless, the approach still requires optimisation in certain parts, something that is discussed in [Section 7.2](#).

7

CONCLUSION AND FUTURE WORK

This chapter concludes the research performed in this document. As the title suggests, in the next pages the conclusions are described ([Section 7.1](#)) and the future work is discussed ([Section 7.2](#)).

7.1 CONCLUSIONS

The conclusions section is composed out the following relevant subsections: In [Section 7.1.1](#), the research questions introduced in [Section 1.3](#) are being answered. [Section 7.1.2](#) gives a summary of the overall contribution of this thesis. [Section 7.1.3](#) presents my reflections about the proposed methodology.

7.1.1 Research Questions

This subsection contains the answer to the research question. In doing so, the sub-questions are being answered first. Finally, the subsection concludes by answering the main research question.

1. What are dynamic point clouds and what are relevant use cases and requirements for their querying?

As with all models of the real world, two types of point clouds can be identified. More specifically point clouds can be considered as static objects, where it is sufficient to study a phenomenon in a specific moment in time. In addition to that, point clouds are lately used for identifying changes occurring throughout time or take place in spatio-temporal analysis, thus being considered as dynamic objects. The latter spatial representation is referred in this thesis as a dynamic point cloud.

A very relevant use case for dynamic point clouds can be found in the context of the coast. Coastal areas, being highly dynamic, require yearly or daily or even hourly monitoring. One way to monitor the changes occurring around the coast is by acquiring point cloud data. Many researchers, among which employees of the Deltares institute, make use of point clouds to perform applications like, coastal change detection, shoreline delineation, coastal inundation prediction, spatio-temporal visualisations etc.

The most prominent requirements for querying dynamic point clouds are:

- **Only space queries** This type of query returns all the spatio - temporal points that are found in a specific area, and is considered as a special type of query. The special part emerges upon the fact that, as more spatio-temporal datasets are accumulated over time, the number of points returned by the same space query increases.

- **Space - time queries** This type of query returns all spatio-temporal objects found in a specific area during a specific time range or moment in time.
- **Only time queries** This type of query returns all spatio-temporal objects found in a specific moment in time or during a specific time range.

2. *What are the relevant parameters that need to be taken into account for the management of dynamic point clouds when using a [SFC](#) approach?*

When referring to the overall process of managing and querying dynamic point clouds, the relevant parameters that need to be taken into account are:

VARIOUS OPERATIONS The approach should be able to support more than just retrieval of the data, i.e. loading, updates, simple analysis (normal vector calculation).

DYNAMIC INSERTIONS The approach used should be able to support the insertion of new spatio - temporal points without having to rebuild the dataset from scratch. An example includes adding the coastline of a neighbouring country e.g. Belgium or adding new yearly data.

SCALABLE OPERATIONS The data retrieval using the data structure should not be affected by the size of the database. Querying from a dataset that has 10 times more points, should be as fast.

TIME AND SPACE EFFICIENCY The data structure should lead to a minimised number of disk accesses and efficient storage.

When querying dynamic point clouds, four more parameters need to be taken into consideration and be balanced:

FETCHING TIME The fetching time of the filter step should be minimised and require the least possible disk accesses. In addition to that, referring to the scalable operations, the fetching time should be independent of the database size.

PERCENTAGE OF EXTRA POINTS The number of false hits retrieved during the filter step should be minimised. In case the number is small compared to the final points, then the refinement step which is a CPU intensive operation, can be ignored.

DEPTH OF THE TREE The depth of the tree used should lead to an accurate decomposition of the Query Region ([QR](#)) into ranges, while not generating an extreme amount of them.

THE DEGREE OF MERGING Whenever the number of ranges exceeds a specified maximum, a merging should take place. The degree of merging should be defined based on the use case.

3. *What kind of [SFC](#) approaches can be used to support the integration of space and time, taking into account the continuous insertions of new points and efficient querying?*

Given that the management and querying of dynamic point clouds should fulfil two contradictory requirements (space - time proximity for fast retrieval and, loading that does not reorganise too much already stored data,

for fast insertions), two integrations of space and time are used. Each one of them fulfils one requirement more than the other. The first approach is the **integrated approach** that equally uses all dimensions for the generation of the [SFC](#). A different degree of proximity can be achieved by appropriately scaling the time dimension, relative to the spatial ones. The second approach is the **non-integrated approach**, where time dominates over the spatial components, that are organised internally using a [SFC](#). The former, achieves better space - time proximity, while the latter can support batch loadings of new data without having to reorganise the old data. Concerning the latter, this will not be true if we were to add new temporal data of a completely different area (e.g. add temporal information for Belgium and Germany). Reorganisation will not take place if the new data are added in an ascending order for the same spatial extent.

The two options represent two extremes of a continuum. Depending on whether the z dimension is part of [SFC](#) or not, two treatments can be defined per each type of integration.

4. *How do the different [SFC](#) approaches compare to each other according to the use cases?*

Through the execution of experiments and benchmarks using two use cases, one with daily temporal dimension and another with yearly, it was proven that the best approach to follow is the integrated approach. Because the methodology is generic one can generalise this result and conclude that use cases of similar characteristics will also show the same behaviour. When looking at the two methods separately, one can realise that:

- The non-integrated approach is easy to be implemented and possibly more portable to other databases that do not have a similar NUMBER data type like the Oracle database. However, the method appears to have many disadvantages when it comes to the query procedure. First, space queries have relatively slow response times. Second, a higher number of Morton ranges during the query execution seems to have a negative effect on the fetching time, thus a larger number of extra points needs to be fetched to achieve a relative fast filter step. Third, certain discrepancies in query response time can be found for bigger datasets concerning the scalability requirement. Finally, having the z dimension in the key significantly worsens query response times of space - time and space queries.
- The integrated approach is a more compact way to structure dynamic point clouds. Unlike the non-integrated approach, this method can be significantly aided by a larger number of Morton ranges during query execution which is performed as a join between the data table and a table containing the ranges. Nonetheless, a limit should be set for practical reasons.

Having z as part of the key, also, does not seem to have a significantly negative effect, although some more extra points might need to be fetched. Concerning some disadvantages, this method is very much dependent on the scaling of the time dimension that needs to be configured using a subset of the whole dataset.

Having given answers to the sub-question above, the main research question can also be answered.

Is a Space Filling Curve (SFC) approach an appropriate method for integrating the space and time components of point clouds in order to support efficient management and querying (use) in a DBMS?

Yes, as it is corroborated from the experiments studying the metrics of performance and the executed benchmarks, the SFC approach is an appropriate method for managing dynamic point clouds. Actually the best approach, given the defined use cases, concerns an equal treatment of the spatial and time dimensions in the SFC. At this point in time, the approach is platform dependent (Oracle database) due to the use of the IOT and NUMBER data type. However, the method is scalable and can be used for a wide variety of use cases. The user only needs to define the appropriate encoding of space and time and, scaling of time for the needed use cases.

7.1.2 Contribution

With this thesis I have introduced a new method for managing dynamic point clouds i.e. point clouds where the time dimension affects the organisation of the points. This method includes using a Space Filling Curve (SFC). I have investigated two integrations of space and time and have shown that the integrated approach is the best way to follow for daily and yearly data. Within this I have also tested two treatments of z (z as an attribute and z as part of the key), and have validated and compared the proposed method with a naive Oracle spatial approach.

The proposed methodology fulfils important characteristics such as: wide variety of operations, dynamic insertions, scalable and efficient queries, together with a fairly good approximation of the QR that results in less extra points compared to the usually used MBR approximation.

7.1.3 Reflection and discussion

Six major criticisms need to be made about the developed prototype.

PROGRAMMING LANGUAGE AND CODE QUALITY The first has to do with the efficiency of the programming language used (Python) and the quality of the code. These two components make the implemented prototype not very efficient in its current state. For example, the morton range generation is an extremely expensive operation and for this reason was ignored from the benchmark results. The same accounts for the encoding and decoding of the Morton ranges.

DATA MOVEMENT The second has to do with the movement of data from the database to the Python application and back. The method would considerably be aided by providing native database functions that can perform those actions (see [Future Work](#)).

FOCUSING ON THE FILTER STEP The third criticism is primarily the result of the previous two. Due to the chosen programming language and because of the data movement, I have only considered the filter step within the tests. This part of the methodology directly uses the data structure and it is independent of the programming language. The rest of the steps can be markedly improved with a different approach (see [Future Work](#)). Nevertheless, even when considering expensive steps, the method remains scalable.

FOCUSING ON HOT RUNS OF THE QUERIES The fourth criticism goes to the studying of the hot runs of the query response times. Because hot runs are used, it means that it is actually not possible to observe if the least possible disk accesses take place. Such parameter should have been studied using the cold runs of the queries. However, cold runs cannot always be trusted as they might include other operations that the **DBMS** performs that are not related to the fetching itself.

SPACE FILLING CURVE USED The fifth has to do with the **SFC** used for clustering the points. The one used was chosen for its fast prototyping capabilities. However, this specific **SFC** is not optimal at clustering, compared to other curves like the Hilbert curve, thus affecting the quality of the clustered points.

COMBINING WITH GIS DATA Finally, it has to be noted that in its current form, the prototype does not provide easy integration with other **GIS** data, like vector or raster. This is important to mention since one of the arguments for storing point clouds inside the database is their full integration with other spatial models. To be able to solve this limitation a lot of functionality has to be transferred to the database itself (see **Future Work**)

7.2 FUTURE WORK

Within this section several recommendations for future work are provided. The future work has to do with areas that were not investigated due to time limitations and possibilities for improving the method. Within the following list, the first two items are the most crucial that should be considered first.

7.2.1 Native database functionality

One of the limitations of my implemented prototype is the movement of data between the Python application and the **DBMS** used. Input/ Output operations are always a bottleneck, so the method would benefit from having native database functionality. This functionality includes the **SFC** calculation during loading, the range generation and the decoding of the selected ranges. This would also allow the two - step filter step to become transparent to the user, as in the case of the normal spatial operators in mainstream **DBMS**. Finally, it is important to consider that such functionality would most probably be developed using compiled languages (Java or C for the case of Oracle) and all operations would become much faster when compared to response times using Python.

Some preliminary work has already been done with the use of C++ code instead of Python during the loading procedure. The **SFC** transformation is still performed outside the database. The code used is developed by Xuefeng Guan and can be found at: <https://github.com/kwan2004/SFCLib>. Both the Morton and Hilbert curve are used in this case. The results (column 2) show significant improvement (6 times faster) during the **SFC conversion** phase of the xyz case **Table 6.3**, when using the Morton curve (see **Table 7.1**). However, these numbers are only preliminary results and, as a result, more research should take place. As part of the future work is studying all possibilities (both integrations of space and time and both treatments of z).

Approach	Morton [s]		Hilbert [s]	
	SFC conversion	IOT	SFC conversion	IOT
Small	68.69	10.33	75.81	11.13
Medium	95.27	32.1	104.55	32.2
Full	110.63	57.61	119.99	56.13

Table 7.1: Using C++ code for SFC calculation

7.2.2 Investigating a different SFC

The Morton curve is a SFC that is very easily programmed and extended in higher dimensions. Nevertheless, a lot of research has shown that its clustering capabilities are not as optimal as other curves [Faloutsos and Roseman, 1989]. For this reason, the use of a different SFC, and more specifically the Hilbert curve, is very relevant for future work. For this, one should compare loading times, query response times, the number of ranges used when decomposing the QR and decoding times. Another thing to investigate is the scaling of the time dimension, since different curves might require a different configuration in this area. The researcher should then be able to conclude which is the best curve to use when managing dynamic point clouds using SFCs.

From the previously presented Table 7.1, one can easily conclude that during the loading procedure the two curves compare as follows:

- The SFC conversion is slightly faster for the Morton curve (10%).
- The generation of the IOT is almost the same for the two curves. The size of the dataset is, however, not massive and specific patterns are hard to be noticed.

Preliminary tests have also been performed for the query procedure using the same C++ Library. Also here, patterns are not noticeable when comparing the two curves. This might be affected 1. by the size of the used dataset, 2. the fact that within the tests the z dimension is not selective. Nonetheless, this needs more evaluation in the future.

7.2.3 Investigation of parallel processing

An important characteristic of today's computers is the use of multi-core CPUs that significantly increase performance. This suggests that parallel processing should be incorporated within our method. Actually this method should be combined with the first recommendation of native database functionality. Parallel processing can easily take place in the SFC calculation, the decoding and range generation. Another area that should be considered for parallelisation is the fetching of the data from the IOT. So far, this does not seem to work for the system tested (this is already reported to the developers in a previous project). With parallel processing, though, one can never be sure about its positive effects, so experiments should be performed. For example in van Oosterom et al. [2015] there were cases where the out-of-core parallelisation actually lead to worst response times during query execution.

7.2.4 Up-scaled benchmark of trillion points

Larger point clouds are appearing every day. Within this thesis I have tested a 2 billion point dataset. Such a size is very relevant but an up-scaled benchmark of trillion points should also take place to prove that the method is indeed scalable. Candidate dynamic datasets (also for coastal management) can be found in the data viewer of the National Oceanic and Atmospheric Administration (NOAA)¹. With a quick search, the data that can be found start from 1996 and are updated yearly. Concerning the number of points available these are definitely trillions, given the number of datasets available.

7.2.5 Higher dimensional keys

Since point clouds are characterised by many attributes (see [Section 2.2](#)), an investigation on whether those attributes or, information about Level of Detail and quality can also be incorporated into the [SFC](#) is meaningful. Advantages for adding more dimensions inside the [SFC](#) are: the compact storage that solves conflicts between the important dimensions [[Lawder and King, 2001](#)], multidimensional indexing using B-Trees and [IOTs](#). Possible disadvantages arise when not all the dimensions included in the [SFC](#) are selective, when the [SFC](#) keys get too big, when the dimensions included do not compose a hypercube etc.

7.2.6 Investigating delta queries

One of the most widely used application in the field of coastal monitoring is the delta query. This query is intended to answer what changes occur between two time moments. A way to answer this query by immediately making use of the [SFC](#) structure is needed. This is not trivial since the nearest neighbour between two different scans needs to be identified.

7.2.7 Two- set refinement stage

Although the procedures taking place after the filter step are not considered within this document, the refinement step would considerably improve by incorporating knowledge from the filter step. The idea is to use the information about the position of a [TC](#) relative to the [QR](#). Based on this information, the algorithm can already decide what objects already belong to the response set ([Figure 7.1](#), white cells) and thus ignore them from the [PIP](#) operation. Only the ranges that are partly inside the query geometry ([Figure 7.1](#), grey cells) need to proceed to the refinement stage. This solution, although sounding easy to implement, can be very hard in reality. The problem arises when a merging to a maximum number needs to take place for the white cells. As explained before, this step adds additional space to the selection, thus cancelling out the whole idea. In addition to that, merging should now be done with two sets and not one.

¹ <https://coast.noaa.gov/dataviewer/>

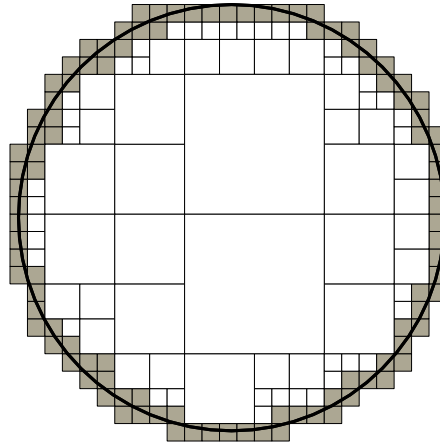


Figure 7.1: Separating internal ranges and ranges on boundary. White cells are completely inside the query region and do not need refinement. Grey cells are partially inside the query region and have to proceed to the refinement stage.

7.2.8 Investigating the generation of blocks

Although the flat model is flexible, it requires a lot of storage. Investigating the generation of blocks using the same integrations of space - time will allow more efficient storage and compression. Investigating their size, the degree of overlapping and non - overlapping and the percentage of full/under-full blocks when dealing with time evolving point clouds would add insight to the current point cloud data management storage models.

A possible way to compress the blocks is the following: Starting with a 128 bit [SFC](#), we can use the highest 64 bits as the block-id. The rest of the 64 bits will then be used within each 4D blocks to define the points. The gain in storage from this step will be approximately a factor of 2, assuming that there is a reasonable number of points in the block, e.g. 10,000. Then, inside the block the points are sorted along the [SFC](#) and only the delta-[SFC](#) key value to the previous point in block is stored. This step should also reduce the storage by a factor of 4. Finally, we can compress the delta-point numbers (explore using [gzip](#)). This should then result in a much more compact storage. Of course, the drawback with compression is the need to "de-compress" the points again when querying. Another thing is that insertions and deletions require a lot of attention.

Other things that need to be considered are: should all blocks be of equal size or of equal number of points contained? This will certainly affect the factor of compression as introduced before. Also, if the latter is chosen, then a tuning related to the right amount of points in the block should take place.

7.2.9 Even more dynamic data

Having experimented with daily and yearly data the question arises whether the same structure is suitable for datasets streamed every hour and minute. Part of the research is to explore loading mechanisms, suitable encoding of space and time and time scalings.

7.2.10 Moving objects point cloud

With the development of mobile devices, wireless networks, etc., we are more than ever faced with objects that move in the 4D space (space and time). This suggests that not only point clouds representing surfaces should be considered. Investigating whether moving objects can be managed using the same integrations of space and time is thus very relevant.

7.2.11 True 4D query

Within this thesis a 4D [SFC](#) was tested and has been proven to be a promising solution for managing dynamic point clouds. However, although z is included in the key, it was not selective in the queries tested. This part of the future work will investigate queries that are also selective in the z dimension, thus showing the real value of including the z dimension in the key.

A | DATASETS

This chapter describes in more detail the characteristics of the two datasets used within this thesis. The datasets were provided by Deltares and are both open data. This means that all the tests performed within this thesis can be replicated by interested parties. In general, the two data sets are very different from each other in the spatial and the temporal component. However, since they both refer to the country of the Netherlands, they use the same reference system: the Amersfoort/RD New with EPSG code of 28992.

A.1 OVERVIEW

The spatial extent of the two datasets is depicted in [Figure A.1](#). The first data set used is the Sand Engine (red box), while the second is the coastline of the country of the Netherlands (yellow polygon). As it can be observed the Sand Engine area is present in both use cases.

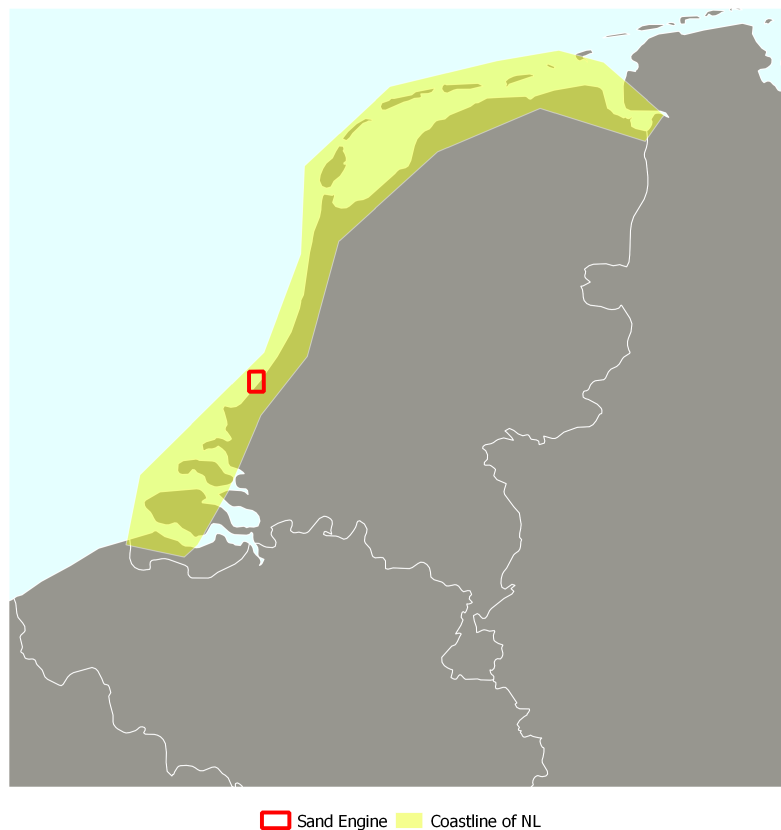


Figure A.1: The location of the datasets on the map of the Netherlands

A.2 CHARACTERISTICS

Within this section more detailed characteristics of the two datasets are presented. The characteristics have to do with the number of files, the spatial extent and resolution, the temporal extent and resolution, and finally the sizes of the data. Both of the datasets are stored in the format LAS¹.

A.2.1 Sand Engine

The Sand Engine dataset (or Zandmotor in Dutch) represents measurements obtained for an experiment taking place in the province South Holland of the Netherlands. The Sand Engine was created by depositing 21 million cubic meters of sand between the areas Ter Heijde and Kijkduin. The purpose of this pilot program is to investigate how nature spreads this amount of sand along the coast as the years go by. In order to see if the experiment is developing as thought, a monitoring of the area at irregular moments in time (after storms) takes place. The point clouds are acquired using jet skis and all-terrain vehicles. Two examples of the Sand Engine point clouds are shown in [Figure A.2](#) and [Figure A.3](#).

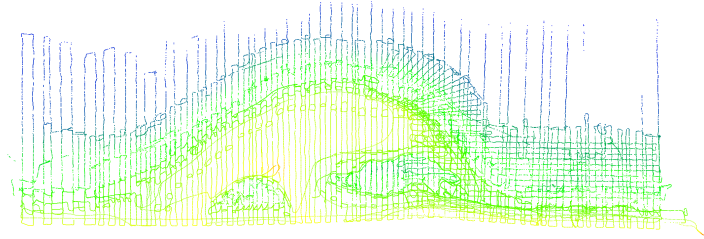


Figure A.2: A point cloud of the Sand Engine use case from the year 2011

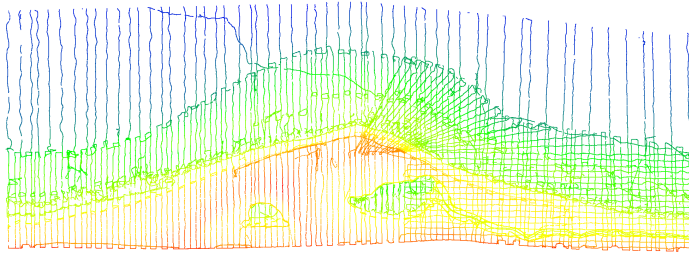


Figure A.3: A point cloud of the Sand Engine use case from the year 2015

The characteristics of the Sand Engine dataset are present in [Table A.1](#). Available data can be found from the year 2011 and 2015. The number of files represents the different days of the year when the area was monitored, and on average 7 files per year are available. The size of the total files per year is also not big, making this use case very small for being a dynamic point cloud. For this reason, artificial data are created from the original ones. The characteristics of the artificial data are present in [Table A.2](#). With

¹ One remark needs to be made. The files used for the execution of the benchmark were compressed LAS files (LAZ). However, the LAZ files were actually not compressed at all, which means that although they had the .laz extension, they no compression was applied. This observation does not affect the results presented in this thesis.

the artificial data, the number of points for the whole use cases reaches the 73,913,926.

Concerning the spatial and temporal component of the dataset, the monitored area is approximately $4.5 \times 4.5 \text{ km}^2$ and the spatial resolution is in millilitre (mm). The time dimension spans from 2000 till 2015 (original and artificial data) and the resolution is in days. Another interesting characteristic is that each file contains a full point cloud of the area for a given day. Finally, as with the majority of the temporal data, time is only mentioned within the file name.

year	no of files	size (MB)
2011	5	7.8
2012	11	17.4
2013	6	11.6
2014	5	10.7
2015	6	13.2

Table A.1: Characteristics of the original Sand Engine data.

year	no of files	size (MB)
2000	76	121.6
2001	70	112.0
2002	84	134.4
2003	83	132.8
2004	84	134.4
2005	77	123.2
2006	80	128.0
2007	87	139.2
2008	87	139.2
2009	86	137.6
2010	76	121.6
2011	8	12.8

Table A.2: Characteristics of the artificially created Sand Engine data.

A.2.2 Coastline

The coastline dataset, as the name implies is a series of point cloud data acquired yearly by Rijkswaterstaat (Ministry of Infrastructure and the Environment) for the monitoring of the area around the coast. In the introduction of the thesis, the reasons why the coast is so important for the Netherlands are described shortly. The point cloud of the coastline for the year of 2012 is depicted in Figure [Figure A.4](#).

The characteristics of the point cloud dataset are presented in Table [Table A.3](#). Available data can be found from the year 2012 and 2015. Only one full point cloud of the coastline is produced every year. The number of files per year represent the number of tiles that are used. As it can be observed, the 2012 dataset is different from the rest. In fact years 2013 until 2015 have two different versions, the unfiltered (raw point cloud) and the filtered (buildings or trees are removed). The 2012 one only represents the filtered point cloud. For this reason, to be consistent, only the filtered parts of the rest are used for the benchmarks. In total approximately 2 billion points are present in all four years. Finally, the size of the data is in GB scale. Therefore, although the dataset is less dynamic, it is a good candidate to corroborate the scalability of the methodology.

Concerning the spatial extent and resolution, the point cloud covers the whole coastline of the Netherlands with a centimetre resolution. The time extent is 4 years with and the resolution yearly. Similar to the Sand Engine use case, time is only present in the folder or file name.

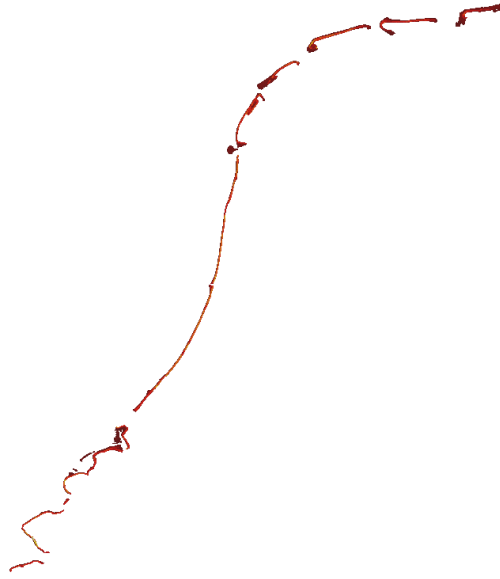


Figure A.4: The coastline point cloud dataset for the year of 2012

year	no of files	size (GB)
2012	13	9.4
2013	107	9.3
2014	108	9.3
2015	110	9.9

Table A.3: Dataset description of the Coastline use case

A.2.3 Comparison

The two datasets presented are different in nature, size, number of files and spatial and time resolution. Their different nature poses different requirements in their organisation. For example, the coastline dataset will be updated only once per year but the amount of data is significantly more than the Sand Engine. On the other hand, the Sand Engine datasets are updated more frequently (every day) and therefore, the methodology developed should not make the database unavailable for a long period of time with every update. Hence, it is important to see how the data structures react to the differences. A comparison of the different characteristics of the datasets is provided in [Table A.4](#).

Dataset	Avg. points per file	Time resolution	Spatial resolution
Sand Engine	100,000 – 200,000	Day	mm
Coastline	50,000 – 20,000,000	Year	cm

Table A.4: Comparison of the two datasets

B

QUERY GEOMETRIES AND TIME RANGES

This chapter provides a description of the query geometries that are used throughout the experiments and benchmarks of this thesis. The spatial component of the queries is also given in maps. For testing purposes the query geometries have different shapes; rectangle, polygon, line with buffer and circle. Different sizes of the those geometries are also tested. This will give us an insight as to whether certain geometries behave differently using the same storage model.

B.1 SAND ENGINE

The queries which are executed are described in detail in Table [Table B.1](#). *Type* is the type of query as defined in [Section 4.2.1](#). *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 is shown in [Figure B.1](#).

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	1 day within the range	20.25
T-B	t	02/09/02	05/09/02	c	2 days within the range	20.25
S-A	s	-	-	-	Small Polygon	0.04
S-B	s	-	-	-	Small polygon	0.002
S-C	s	-	-	-	Circle of 50 m radius	0.008
S-D	s	-	-	-	Diagonal line with buffer 5 m	0.009

Table B.1: The description of the Zandmotor queries in time. 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.

B.2 COASTLINE

The queries used for the coastline use case are described in [Table B.2](#) and have the same attributes as the Sand Engine ones. However, contrary to the Sand Engine, in this use case time queries are not performed. The reasons are practical. In the full benchmark, the coastline dataset contains

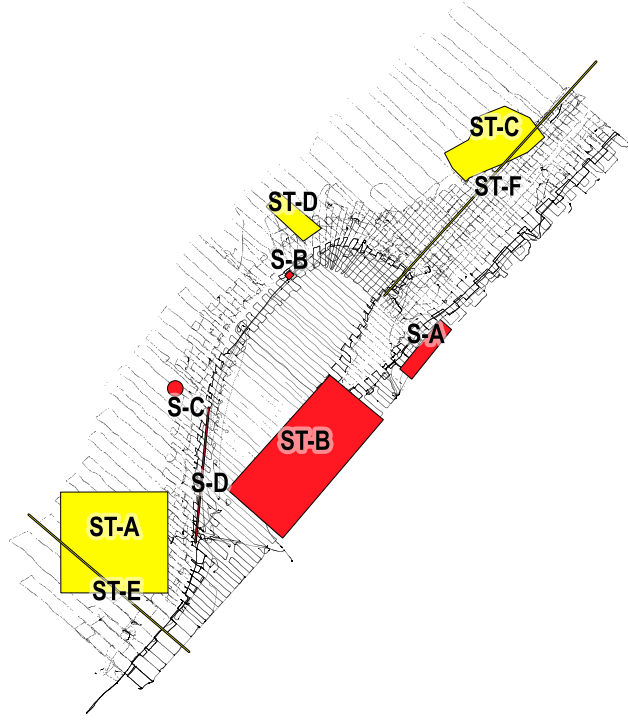


Figure B.1: The Sand Engine queries (spatial extent)

500,000,000 points per year. Such number of points is not easily processed with the processing power of today's computers. If such a query would take place, it would either require a [LoD](#) approach where the detail of the point cloud is decreased or the points would be requested as tiles, thus leading to space - time queries. The spatial representation of the queries is shown in [Figure B.2](#).

id	type	start	end	time type	Description	area (km ²)
ST-A	s-t	2012	-	d	Large axis - aligned rectangle	0.11
ST-B	s-t	2012	-	d	Medium, complex polygon	0.12
ST-C	s-t	2012	-	d	Large Line with buffer 20 m	0.05
ST-D	s-t	2012	-	d	Small Line with buffer 20 m	0.02
ST-E	s-t	2012	-	d	Medium Line with buffer 20 m	0.04
ST-F	s-t	2013	-	d	Large, complex polygon	0.50
ST-G	s-t	2012	2012	c	Large, polygon	0.20
S-A	s	-	-	-	Small Circle with 130 m radius	0.02
S-B	s	-	-	-	Large Line with buffer 20 m	0.04
S-C	s	-	-	-	Small Line with buffer 20 ms	0.02

Table B.2: The description of the Coastline queries in time. 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.

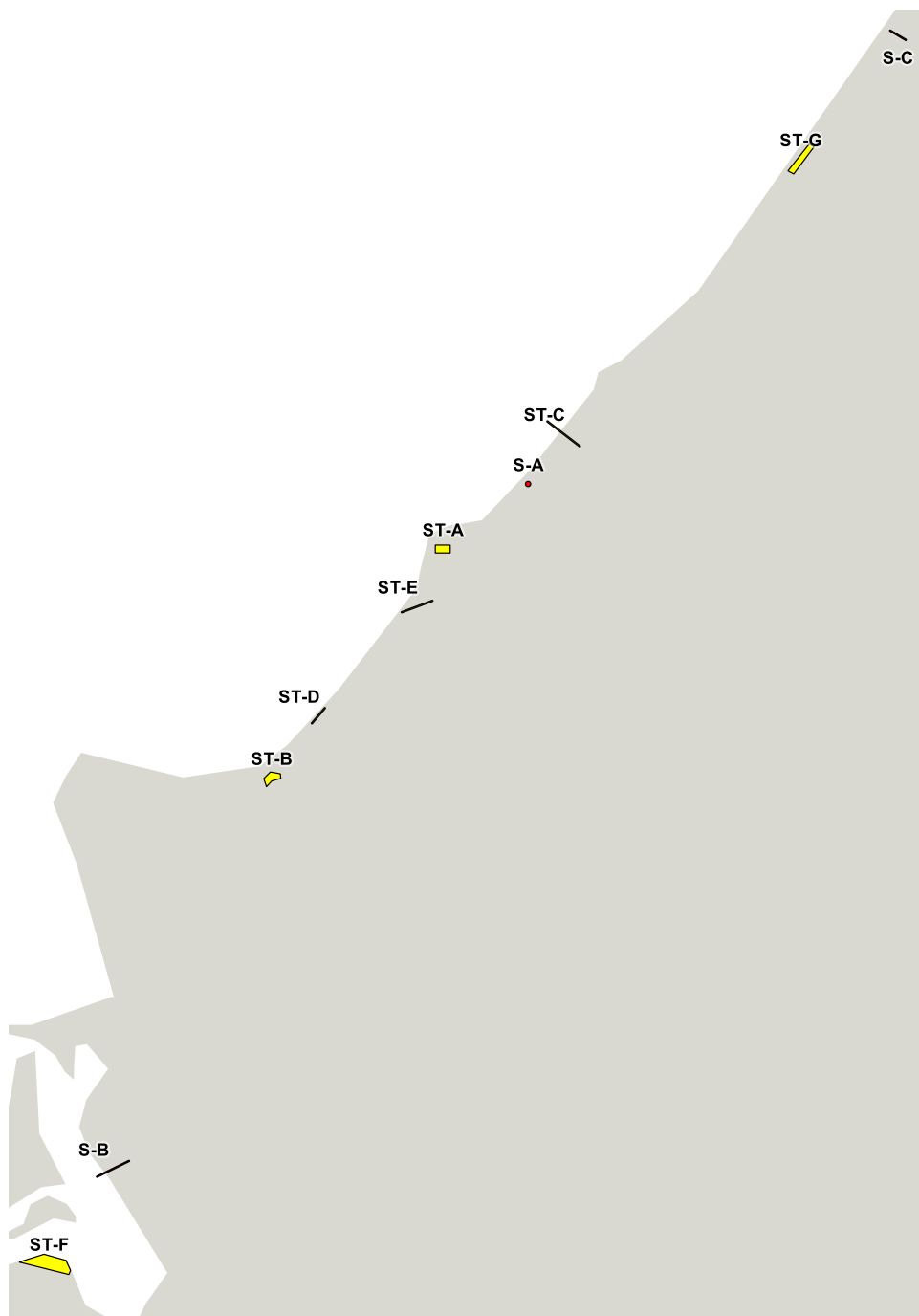


Figure B.2: The spatial extent of the Coastline queries

C | DETAILED RESULTS

C.1 SAND ENGINE

C.1.1 Depth of tree

The following Tables (C.1 to C.4) contain the results of the experiment described in Section 5.4.1 of the current document. The tables include only the average of the two hot runs.

ID	Depth	No. ranges	Fetching [s]	% extra	ID	Depth	No. ranges	Fetching [s]	% extra
ST-A	9	32	0.31	1287	S-A	9	46	2.17	389
	10	72	0.09	256		10	150	1.64	274
	11	264	0.05	100		11	725	1.06	124
	12	924	0.07	12		12	2798	0.76	61
	13	3526	0.05	7		13	10931	0.67	30
	14	13600	0.07	2		14	41163	0.55	14
	15	53726	0.07	1		15	161734	0.57	7
ST-B	9	14	0.33	1244	S-B	9	22	0.61	4110
	10	44	0.11	368		10	42	0.16	903
	11	139	0.10	284		11	45	0.18	903
	12	488	0.11	241		12	671	0.09	310
	13	1827	0.08	115		13	1842	0.07	126
	14	7053	0.06	3		14	7686	0.08	64
	15	27711	0.06	2		15	24370	0.06	25
ST-C	9	3	0.17	891	S-C	9	44	1.03	1641
	10	22	0.09	470		10	84	0.57	804
	11	121	0.07	240		11	336	0.17	126
	12	420	0.07	91		12	595	0.16	99
	13	1514	0.06	21		13	3119	0.13	44
	14	5805	0.07	11		14	13113	0.13	21
	15	15320	0.05	5		15	45855	0.10	10
ST-D	9	10	0.19	1329	S-D	9	88	2.43	3740
	10	12	0.08	502		10	336	1.25	1798
	11	56	0.04	159		11	1096	0.69	871
	12	174	0.03	82		12	4769	0.36	392
	13	459	0.05	27		13	17443	0.25	186
	14	1041	0.05	12		14	76552	0.18	99
	15	8954	0.05	5		15	331700	0.19	45
ST-E	9	18	0.19	7923	T-A	9	624	2.53	500
	10	18	0.09	3386		10	2397	1.69	300
	11	98	0.03	1257		11	9292	0.87	100
	12	260	0.02	514		12	36400	0.49	0
	13	678	0.04	207	T-B	9	624	3.74	350
	14	2200	0.03	110		10	2397	1.67	100
	15	10799	0.03	50		11	9292	0.84	0
ST-F	9	12	0.12	6115	Table C.2: Depth of the tree experiment. Case: Space only and time only queries of integrated approach with z as an attribute.				
	10	17	0.07	2976					
	11	59	0.04	1557					
	12	307	0.03	774					
	13	937	0.03	284					
	14	3176	0.05	132					
	15	10378	0.03	67					

Table C.1: Depth of the tree experiment.
Case: Space - time queries of integrated approach with z as an attribute.

ID	Depth	No. ranges	Fetching [s]	% extra
ST-A	9	32	0.21	1287
	10	72	0.07	256
	11	528	0.07	100
	12	2772	0.05	12
	13	17630	0.05	7
	14	122400	0.07	2
ST-B	9	14	0.21	1244
	10	44	0.07	368
	11	278	0.06	284
	12	1464	0.08	241
	13	9135	0.07	115
	14	63477	0.07	3
ST-C	9	3	0.10	891
	10	25	0.07	470
	11	244	0.04	240
	12	1334	0.05	91
	13	8173	0.07	21
	14	57159	0.05	11
ST-D	9	10	0.10	1329
	10	12	0.05	502
	11	112	0.03	159
	12	558	0.05	82
	13	2541	0.05	27
	14	11505	0.04	12
ST-E	9	18	0.12	7923
	10	18	0.06	3386
	11	196	0.03	1257
	12	816	0.05	514
	13	3616	0.03	207
	14	21412	0.04	110
ST-F	9	12	0.07	6115
	10	17	0.03	2976
	11	107	0.02	1557
	12	986	0.05	774
	13	4738	0.04	284
	14	27906	0.05	132
S-A	9	46	1.31	389
	10	210	0.97	274
	11	1327	0.65	124
	12	9476	0.48	61
	13	64700	0.42	30
	14	448392	0.47	14
S-B	9	22	0.38	4110
	10	42	0.11	903
	11	49	0.11	903
	12	2013	0.09	310
	13	9373	0.05	126
	14	73122	0.07	64
S-C	9	44	0.63	1641
	10	84	0.35	804
	11	672	0.12	126
	12	1785	0.16	99
	13	17653	0.10	44
	14	130182	0.11	21
S-D	9	88	1.30	3740
	10	336	0.73	1798
	11	2110	0.40	871
	12	14473	0.23	392
	13	87118	0.17	186
	14	683951	0.24	99
T-A	9	624	1.49	500
	10	2397	0.99	300
	11	18584	0.53	100
	12	109200	0.33	0
T-B	9	624	2.21	350
	10	2397	0.97	100
	11	18584	0.53	0

Table C.4: Depth of the tree experiment.
Case: Space only and time only queries of integrated approach with z as part of the key.

Table C.3: Depth of the tree experiment.
Case: Space - time queries of integrated approach with z as part of the key.

c.1.2 The degree of merging

The following tables (C.5 and C.5) contain the results of the experiment described in Section 5.4.2 of the current document. The tables include only the average of the two hot runs.

ID	No. ranges	Depth	Fetching [s]	% extra
ST-A	10	19	0.07	62.7
	100	19	0.49	3.8
	1000	19	4.85	0.4
	5151	19	27.51	0
ST-B	10	19	0.05	47.9
	100	19	0.33	5.7
	1000	19	3.11	0.7
	6864	19	23.78	0.1
ST-C	10	19	0.08	65.6
	100	19	0.64	14.1
	1000	19	8.17	1
	3567	19	27.63	0.4
ST-D	10	20	0.08	46.4
	100	20	0.11	3.8
	1000	20	21.33	0.2
	3277	20	68.23	0.1
ST-E	10	19	0.07	673.2
	100	19	0.07	69.7
	1000	19	16.86	6.8
	3744	19	60.54	2.6
ST-F	10	19	0.225	2868.8
	100	19	0.69	275.2
	1000	19	8.17	34.9
	10000	19	72.04	3.4
S-A	10	20	1.11	82.4
	100	20	8.41	8.8
	1000	20	93.24	0.9
	5191	20	475.05	0.2
S-B	10	22	0.86	43.9
	100	22	8.18	4.6
	1000	22	91.79	0.6
	2810	22	267.13	0.2
S-C	10	22	0.86	31.9
	100	22	7.35	2.9
	1000	22	92.20	0.4
	5265	22	458.60	0.1
S-D	10	20	0.91	886.7
	100	20	7.31	90.2
	1000	20	84.84	10.3
	9504	20	799.47	1.2

Table C.5: The degree of merging experiment. Case: Space - time and space only queries of non-integrated approach with z as attribute.

ID	No. ranges	Depth	Fetching [s]	% extra
ST-A	10	14	0.04	92.3
	100	14	0.57	29.5
	1000	14	5.67	6.2
	3516	14	18.41	2.4
ST-B	10	14	0.03	64.3
	100	14	0.35	25
	1000	14	3.33	6.6
	3531	14	11.07	3.7
ST-C	10	15	0.05	94.7
	100	15	0.78	45.7
	1000	15	8.41	20.7
	3877	15	28.80	5
ST-D	10	16	0.09	118.1
	100	16	0.13	44.7
	1000	16	20.87	8.1
	8897	16	186.71	2.6
ST-E	10	15	0.09	1347.1
	100	15	0.10	327.6
	1000	15	17.24	98.7
	4111	15	67.87	50
ST-F	10	15	0.09	3099.4
	100	15	0.85	784.7
	1000	15	8.16	193
	8652	15	61.78	67.3
S-A	10	16	1.02	149.4
	100	16	8.83	49.5
	1000	16	90.82	11.1
	10000	16	937.67	4
S-B	10	17	0.97	395.4
	100	17	7.89	53.5
	1000	17	91.88	14.3
	5206	17	484.66	7.5
S-C	10	17	0.89	108.9
	100	17	7.39	34.2
	1000	17	87.93	9.2
	10000	17	867.32	2.7
S-D	10	16	0.98	1349.6
	100	16	7.25	334.4
	1000	16	82.39	94.6
	10000	16	840.16	32.1

Table C.6: The degree of merging experiment. Case: Space - time and space only queries of non-integrated approach with z as part of the key.

c.1.3 Depth of tree with merging

The following tables (C.7 to C.10) contain the results of the experiment described in Section 5.4.3 of the current document. The tables include only the average of the two hot runs.

ID	Depth	No. ranges	Fetching [s]	% extra	ID	Depth	No. ranges	Fetching [s]	% extra
ST-A	9	10	0.07	151.3	S-A	9	2	2.45	389.9
	10	16	0.06	42.7		10	4	1.83	274.9
	11	18	0.05	33.2		11	10	1.20	124.5
	12	54	0.33	12.1		12	20	1.86	61.2
	13	84	0.49	7.7		13	41	3.42	30.4
	14	183	1.00	2.4		14	79	6.84	14.9
	15	200	1.11	1.8		15	161	13.32	7.4
	16	200	1.11	1.8		16	200	16.95	5.1
	17	200	1.05	1.7		17	200	16.86	4.6
	18	200	0.93	1.8		18	200	16.83	4.5
ST-B	9	6	0.06	91.6	S-B	9	1	0.74	4110.8
	10	13	0.05	55.9		10	1	0.31	903.3
	11	27	0.14	28.2		11	1	0.30	903.3
	12	53	0.22	13.7		12	4	0.48	310.2
	13	105	0.38	7.5		13	6	0.68	126.4
	14	200	0.69	3.9		14	12	1.14	64.4
	15	200	0.68	3.5		15	21	1.70	25.2
	16	200	0.68	3		16	39	3.05	14.4
	17	200	0.62	3		17	86	6.96	7.5
	18	200	0.66	3.1		18	170	13.77	3.3
ST-C	9	2	0.1	170.1	S-C	9	2	1.17	1641.7
	10	5	0.13	113.8		10	2	0.63	804.8
	11	13	0.10	70.4		11	4	0.51	126.4
	12	28	0.06	43.9		12	5	0.59	99.3
	13	53	0.07	21.9		13	10	0.90	44.5
	14	113	0.90	11.7		14	23	1.74	21.8
	15	200	1.55	5.5		15	42	3.03	10.6
	16	200	1.6	4.6		16	83	6.38	5.1
	17	200	1.54	4.3		17	155	11.41	2.6
	18	200	1.28	4.2		18	200	15.12	1.8
ST-D	9	2	0.28	732.8	S-D	9	4	2.73	3740.9
	10	5	0.21	251		10	8	1.34	1798.8
	11	8	0.11	102.2		11	14	1.25	871
	12	13	0.11	60.1		12	30	2.25	392.9
	13	25	0.10	27.3		13	56	3.92	186.4
	14	48	0.10	12.9		14	123	8.78	99.6
	15	101	0.13	5.9		15	200	15.30	61.7
	16	198	0.18	2.6		16	200	14.46	49.6
	17	200	0.19	2		17	200	14.35	47.8
	18	200	0.16	1.8		18	200	15.38	47.5
ST-E	9	4	0.41	4577.6	ST-F	9	10	0.39	4557.8
	10	7	0.26	1930.5		10	17	0.15	2204.9
	11	12	0.10	954.5		11	35	0.08	1143.1
	12	25	0.09	436.8		12	71	0.07	553.2
	13	48	0.09	207.4		13	141	1.18	284.4
	14	100	0.09	110.5		14	200	1.67	163.3
	15	200	0.12	52.1		15	200	1.68	146.2
	16	200	0.11	40.5		16	200	1.72	139.4
	17	200	2.99	37.9		17	200	1.30	137.3
	18	200	0.10	37.6		18	200	1.33	136.4

Table C.7: Depth of the tree with merging experiment. Case: Space - time queries of non-integrated approach with z as an attribute.

Table C.8: Depth of the tree with merging experiment. Case: Space only queries of non-integrated approach with z as an attribute.

ID	Depth	No. ranges	Fetching [s]	% extra
ST-A	9	11	0.05	151
	10	20	0.04	43
	11	28	0.18	33
	12	200	1.08	19
	13	200	1.11	20
	14	200	1.06	21
	15	200	1.10	21
	16	200	1.15	21
ST-B	9	7	0.04	92
	10	20	0.12	56
	11	51	0.20	28
	12	200	0.65	17
	13	200	0.66	17
	14	200	0.70	17
	15	200	0.66	17
	16	200	0.67	17
ST-C	9	3	0.08	170
	10	7	0.05	114
	11	24	0.06	70
	12	109	0.83	44
	13	200	1.51	40
	14	200	1.50	40
	15	200	1.48	40
	16	200	1.47	40
ST-D	9	3	0.16	733
	10	6	0.12	251
	11	16	0.10	102
	12	46	0.11	60
	13	164	0.16	27
	14	200	5.93	24
	15	200	5.81	24
	16	200	5.46	24
ST-E	9	5	0.17	4578
	10	9	0.11	1931
	11	28	0.10	954
	12	91	0.11	437
	13	200	4.02	279
	14	200	3.88	261
	15	200	5.22	251
	16	200	3.92	250
ST-F	9	10	0.12	4558
	10	17	0.10	2205
	11	59	0.51	1143
	12	189	1.61	553
	13	200	1.67	503
	14	200	1.65	497
	15	200	1.61	497
	16	200	1.68	496

Table C.9: Depth of the tree with merging experiment. Case: Space - time queries of non-integrated approach with z as part of the key.

ID	Depth	No. ranges	Fetching [s]	% extra
S-A	9	2	1.56	390
	10	5	1.33	275
	11	17	1.66	124
	12	58	5.04	61
	13	200	17.19	30
	14	200	18.20	27
	15	200	16.61	28
	16	200	17.69	27
S-B	9	1	0.51	4111
	10	1	0.18	903
	11	1	0.16	903
	12	12	1.10	310
	13	29	2.34	126
	14	112	9.14	64
	15	200	16.32	32
	16	200	16.41	32
S-C	9	2	0.79	1642
	10	2	0.56	805
	11	8	0.80	126
	12	12	1.05	99
	13	53	3.78	45
	14	200	14.83	22
	15	200	15.72	20
	16	200	14.85	20
S-D	9	2	0.79	1642
	10	2	0.56	805
	11	8	0.80	126
	12	12	1.05	99
	13	53	3.78	45
	14	200	14.83	22
	15	200	15.72	20
	16	200	14.85	20

Table C.10: Depth of the tree with merging experiment. Case: Space only queries of non-integrated approach with z as part of the key.

c.1.4 Benchmark (All stages)

The following four tables contain the benchmark results considering all query steps taking place in our implemented prototype.

Stage refers to the benchmark stage, *ID* to the query ID, *Range calc.* refers to the step that identifies the ranges approximating the *QR*, *No. of ranges* refers the total ranges used to fetch the data (after applying merging), *Depth* refers to the original depth used for identifying the ranges, *Fetch* refers to the time spent in order to fetch the data from the *IOT*, *Decode* refers to the time spent decoding the ranges in Python, *Store* refers to the time spent storing the decoded points into a temporary table, *Candidate pts.* refers to the points returned from the filter step, *Final pts.* refers to the points actually belonging to the *QR*, *Refinement* refers to the time spent in order to refine the candidate points.

Stage	ID	Range calc. (s)	Range IOT (s)	No. of ranges	Depth	Fetch (s)	Decode (s)	Store (s)	Candidate pts	Final pts	Refinement (s)
Small	ST-A	4.1	0.30	13600	14	0.05	0.08	0.07	4020	3927	0.21
	ST-B	29.6	0.43	109843	16	0.13	0.08	0.07	4275	4237	0.22
	ST-C	3.1	0.18	5805	14	0.04	0.06	0.07	3140	2812	0.20
	ST-D	8.7	0.20	8954	15	0.03	0.05	0.07	2314	2185	0.19
	ST-E	83.8	0.27	39329	16	0.04	0.01	0.07	483	380	0.17
	ST-F	61.5	0.29	40164	16	0.04	0.01	0.06	443	327	0.17
	S-A	66.0	0.29	41163	14	0.68	1.68	0.18	98801	86017	1.43
	S-B	176.5	0.42	88119	16	0.10	0.06	0.07	3189	2788	0.20
	S-C	77.1	0.29	45855	15	0.16	0.24	0.08	12724	11501	0.33
	S-D	357.4	1.09	331700	15	0.42	0.33	0.09	17374	11933	0.38
	T-A	10.1	0.28	36400	12	0.59	1.40	0.15	78902	78902	0.27
	T-B	3.0	0.20	9292	11	0.96	2.58	0.26	157806	157806	0.45
Medium	ST-A	4.1	0.34	13600	14	0.05	0.08	0.07	4020	3927	0.22
	ST-B	31.2	0.47	109843	16	0.12	0.08	0.08	4275	4237	0.22
	ST-C	3.1	0.20	5805	14	0.04	0.06	0.07	3140	2812	0.20
	ST-D	8.8	0.21	8954	15	0.03	0.05	0.07	2314	2185	0.19
	ST-E	82.4	0.28	39329	16	0.04	0.01	0.07	483	380	0.16
	ST-F	58.7	0.29	40164	16	0.04	0.01	0.07	443	327	0.17
	S-A	154.8	0.47	95420	14	1.73	4.32	0.38	265100	231283	3.14
	S-B	404.2	0.77	206773	16	0.26	0.20	0.08	10383	8934	0.30
	S-C	185.4	0.49	107388	15	0.39	0.69	0.12	36448	32983	0.67
	S-D	822.6	2.32	778217	15	1.09	0.93	0.13	49269	33494	0.79
	T-A	10.3	0.27	36400	12	0.60	1.44	0.16	78902	78902	0.27
	T-B	2.9	0.18	9292	11	0.96	2.57	0.23	157806	157806	0.42
Large	ST-A	4.2	0.38	13600	14	0.06	0.08	0.07	4020	3927	0.22
	ST-B	29.5	0.47	109843	16	0.13	0.08	0.07	4275	4237	0.22
	ST-C	3.7	0.21	5805	14	0.04	0.06	0.07	3140	2812	0.21
	ST-D	8.9	0.22	8954	15	0.03	0.05	0.07	2314	2185	0.19
	ST-E	82.6	0.29	39329	16	0.04	0.01	0.06	483	380	0.17
	ST-F	62.1	0.30	40164	16	0.04	0.01	0.06	443	327	0.17
	S-A	85.6	0.35	56727	13	3.76	10.48	0.86	653251	509964	7.19
	S-B	221.4	0.58	128030	15	0.36	0.58	0.10	30121	23949	0.55
	S-C	99.0	0.39	68493	14	0.53	1.19	0.15	66180	54111	1.02
	S-D	363.5	1.40	402797	14	1.75	2.80	0.23	178215	90670	1.95
	T-A	10.0	0.29	36400	12	0.59	1.37	0.16	78902	78902	0.25
	T-B	2.9	0.21	9292	11	0.95	2.50	0.23	157806	157806	0.41

Table C.11: Benchmark results for all query stages. Case: integrated approach with *z* as an attribute.

Stage	ID	Range calc. (s)	Range IOT (s)	No. of ranges	Depth	Fetch (s)	Decode (s)	Store (s)	Candidate pts	Final pts	Refinement (s)
Small	ST-A	32.9	0.72	122400	14	0.11	0.10	0.07	4020	3927	0.21
	ST-B	17.5	0.40	63477	14	0.08	0.11	0.07	4393	4237	0.22
	ST-C	65.5	0.37	57159	14	0.06	0.08	0.07	3140	2812	0.20
	ST-D	503.0	0.71	161983	15	0.12	0.06	0.07	2314	2185	0.19
	ST-E	307.1	0.74	198029	15	0.13	0.02	0.06	570	380	0.17
	ST-F	217.1	0.73	166417	15	0.11	0.02	0.06	547	327	0.17
	S-A	118.3	0.38	64700	13	0.47	2.42	0.20	112144	86017	1.48
	S-B	97.9	0.43	73122	14	0.08	0.12	0.07	4583	2788	0.21
	S-C	285.9	0.62	130182	14	0.17	0.35	0.09	14008	11501	0.34
	S-D	639.5	2.58	683951	14	0.63	0.61	0.09	23817	11933	0.43
	T-A	35.9	0.51	109200	12	0.42	1.77	0.15	78902	78902	0.26
	T-B	5.4	0.26	18584	11	0.57	3.29	0.25	157806	157806	0.44
Medium	ST-A	33.0	0.76	122400	14	0.11	0.10	0.07	4020	3927	0.21
	ST-B	18.4	0.38	63477	14	0.08	0.11	0.07	4393	4237	0.22
	ST-C	67.0	0.37	57159	14	0.06	0.08	0.07	3140	2812	0.20
	ST-D	488.0	0.72	161983	15	0.13	0.06	0.07	2314	2185	0.18
	ST-E	317.0	0.82	198029	15	0.13	0.02	0.06	570	380	0.16
	ST-F	222.6	0.66	166417	15	0.11	0.02	0.06	547	327	0.16
	S-A	276.7	0.68	149717	13	1.17	6.26	0.39	298813	231283	3.52
	S-B	226.5	0.67	171096	14	0.21	0.38	0.08	15331	8934	0.34
	S-C	69.2	0.30	40977	13	0.23	1.20	0.12	47722	32983	0.75
	S-D	192.6	0.82	203610	13	0.59	2.11	0.17	97172	33494	1.16
	T-A	35.8	0.54	109200	12	0.42	1.78	0.15	78902	78902	0.26
	T-B	5.5	0.25	18584	11	0.57	3.44	0.23	157806	157806	0.41
Large	ST-A	34.3	0.65	122400	14	0.11	0.10	0.07	4020	3927	0.21
	ST-B	18.3	0.36	63477	14	0.08	0.11	0.07	4393	4237	0.22
	ST-C	65.6	0.34	57159	14	0.06	0.08	0.07	3140	2812	0.20
	ST-D	490.6	0.67	161983	15	0.12	0.06	0.07	2314	2185	0.19
	ST-E	303.1	0.81	198029	15	0.13	0.02	0.07	570	380	0.17
	ST-F	216.9	0.69	166417	15	0.11	0.02	0.06	547	327	0.16
	S-A	73.7	0.32	49236	12	2.61	16.34	0.83	806032	509964	8.04
	S-B	531.4	1.35	384102	14	0.48	0.99	0.11	40198	23949	0.63
	S-C	146.8	0.48	92203	13	0.41	1.74	0.16	78887	54111	1.05
	S-D	416.4	1.58	457738	13	1.41	5.30	0.35	257766	90670	2.55
	T-A	34.0	0.51	109200	12	0.42	1.76	0.16	78902	78902	0.27
	T-B	5.4	0.24	18584	11	0.58	3.38	0.24	157806	157806	0.41

Table C.12: Benchmark results for all query stages. Case: integrated approach with z as part of the key.

Stage	ID	Range calc. (s)	Range IOT (s)	No. of ranges	Depth	Fetch (s)	Decode (s)	Store (s)	Candidate pts	Final pts	Refinement (s)
Small	ST-A	0.1	-	54	12	0.33	0.04	0.08	4404	3927	0.22
	ST-B	0.1	-	53	12	0.22	0.04	0.08	4818	4237	0.22
	S-A	0.1	-	41	13	30.66	0.84	0.21	112144	86017	1.49
	ST-C	0.2	-	113	14	0.92	0.03	0.07	3140	2812	0.20
	ST-D	0.4	-	198	16	3.86	0.02	0.07	2242	2185	0.19
	ST-E	1.9	-	200	17	2.94	0.01	0.06	524	380	0.16
	ST-F	2.7	-	200	17	1.34	0.01	0.07	776	327	0.16
	S-B	0.1	-	86	17	50.98	0.03	0.07	2996	2788	0.20
	S-C	0.1	-	42	15	24.72	0.10	0.09	12724	11501	0.33
	S-D	0.2	-	200	15	109.10	0.12	0.09	19299	11933	0.40
	T-A	0.0	-	0	0	0.52	0.59	0.17	78902	78902	-
	T-B	0.0	-	0	0	1.01	1.17	0.27	157806	157806	-
Medium	ST-A	0.1	-	54	12	0.33	0.04	0.07	4404	3927	0.21
	ST-B	0.1	-	53	12	0.22	0.04	0.08	4818	4237	0.23
	ST-C	0.2	-	113	14	0.91	0.03	0.08	3140	2812	0.20
	ST-D	0.4	-	198	16	3.92	0.02	0.07	2242	2185	0.20
	ST-E	1.8	-	200	17	3.09	0.01	0.07	524	380	0.17
	ST-F	2.6	-	200	17	1.57	0.01	0.06	776	327	0.17
	S-A	0.1	-	41	13	74.11	2.05	0.42	298813	231283	3.41
	S-B	0.1	-	86	17	122.38	0.06	0.08	9701	8934	0.29
	S-C	0.1	-	42	15	64.30	0.26	0.12	36448	32983	0.66
	S-D	0.2	-	200	15	260.34	0.32	0.13	54227	33494	0.85
	T-A	0.0	-	0	0	0.51	0.60	0.17	78902	78902	-
	T-B	0.0	-	0	0	1.01	1.13	0.25	157806	157806	-
Large	ST-A	0.1	-	54	12	0.33	0.04	0.08	4404	3927	0.22
	ST-B	0.1	-	53	12	0.21	0.04	0.07	4818	4237	0.23
	ST-C	0.2	-	113	14	0.93	0.03	0.07	3140	2812	0.20
	ST-D	0.4	-	198	16	4.03	0.02	0.07	2242	2185	0.19
	ST-E	1.9	-	200	17	3.38	0.01	0.06	524	380	0.16
	ST-F	2.7	-	200	17	1.40	0.01	0.06	776	327	0.17
	S-A	0.1	-	41	13	128.02	4.31	0.84	653251	509964	7.13
	S-B	0.1	-	86	17	209.89	0.15	0.10	25847	23949	0.51
	S-C	0.1	-	42	15	108.58	0.43	0.14	59899	54111	0.96
	S-D	0.2	-	200	15	444.13	0.88	0.24	146328	90670	1.79
	T-A	0.0	-	0	0	0.51	0.59	0.16	78902	78902	-
	T-B	0.0	-	0	0	1.04	1.20	0.24	157806	157806	-

Table C.13: Benchmark results for all query stages. Case: non - integrated approach with z as an attribute.

Stage	ID	Range calc. (s)	Range IOT (s)	No. of ranges	Depth	Fetch (s)	Decode (s)	Store (s)	Candidate pts	Final pts	Refinement (s)
Small	ST-A	0.4	-	200	12	1.07	0.09	0.08	4669	3927	0.22
	ST-B	0.5	-	200	12	0.65	0.10	0.07	4941	4237	0.23
	ST-C	1.0	-	200	13	1.75	0.07	0.08	3930	2812	0.21
	ST-D	1.8	-	200	14	3.93	0.05	0.07	2715	2185	0.20
	ST-E	7.1	-	200	15	3.39	0.02	0.07	1332	380	0.18
	ST-F	2.4	-	200	14	1.75	0.04	0.08	1953	327	0.18
	S-A	0.3	-	200	13	122.95	1.78	0.19	112145	86017	1.49
	S-B	0.7	-	200	15	114.90	0.07	0.07	3678	2788	0.21
	S-C	0.4	-	200	14	105.02	0.22	0.08	14009	11501	0.34
	S-D	0.9	-	200	14	103.44	0.62	0.12	40168	11933	0.59
	T-A	0.0	-	0	0	0.33	1.44	0.15	78902	78902	-
	T-B	0.0	-	0	0	0.62	2.68	0.25	157806	157806	-
Medium	ST-A	0.4	-	200	12	1.08	0.09	0.07	4669	3927	0.22
	ST-B	0.5	-	200	12	0.64	0.10	0.07	4941	4237	0.22
	ST-C	1.0	-	200	13	1.86	0.08	0.07	3930	2812	0.21
	ST-D	1.7	-	200	14	3.97	0.04	0.07	2715	2185	0.19
	ST-E	7.1	-	200	15	3.42	0.02	0.07	1332	380	0.17
	ST-F	2.4	-	200	14	1.94	0.04	0.07	1953	327	0.18
	S-A	0.3	-	200	13	292.07	4.57	0.39	298814	231283	3.48
	S-B	0.7	-	200	15	291.73	0.20	0.08	12218	8934	0.31
	S-C	0.4	-	200	14	260.18	0.62	0.11	40266	32983	0.69
	S-D	0.9	-	200	14	244.45	1.70	0.20	112885	33494	1.31
	T-A	0.0	-	0	0	0.32	1.44	0.16	78902	78902	-
	T-B	0.0	-	0	0	0.63	2.72	0.25	157806	157806	-
Large	ST-A	0.4	-	200	12	1.12	0.09	0.07	4669	3927	0.21
	ST-B	0.5	-	200	12	0.66	0.10	0.08	4941	4237	0.22
	ST-C	1.0	-	200	13	1.83	0.08	0.07	3930	2812	0.20
	ST-D	1.7	-	200	14	4.30	0.05	0.07	2715	2185	0.19
	ST-E	7.1	-	200	15	3.41	0.02	0.07	1332	380	0.18
	ST-F	2.4	-	200	14	1.93	0.04	0.07	1953	327	0.18
	S-A	0.3	-	200	13	503.75	10.16	0.79	653252	509964	7.10
	S-B	0.7	-	200	15	481.49	0.49	0.10	31745	23949	0.56
	S-C	0.4	-	200	14	422.42	0.99	0.15	66181	54111	1.01
	S-D	0.9	-	200	14	417.43	4.36	0.37	288090	90670	2.80
	T-A	0.0	-	0	0	0.31	1.42	0.16	78902	78902	-
	T-B	0.0	-	0	0	0.62	2.69	0.25	157806	157806	-

Table C.14: Benchmark results for all query stages. Case: non - integrated approach with z as part of the key.

C.2 COASTLINE

This section contains the three additional benchmarks executed as part of the coastline use case. The purpose of the additional benchmarks was to configure the scaling of the time dimension and to have more control over the process. In general it is easier to test specific parameters for a small benchmark than a full, as the user has to wait more time in the second case. Also, this type of executing benchmarks allowed us to eliminate a certain integration of space and time from the very beginning, saving thus a lot of effort from further benchmark executions.

C.2.1 Small benchmark

The query results of the small benchmark can be found in [Table C.16](#). There we can observe the query response times and percentages of extra points from all four storage models discussed within this thesis. It is important to note again the differences in the ways of posing the queries between the two integrations of space and time. For the non-integrated approach a WHERE clause with maximum number of ranges set to 200 is used. For the integrated approach, the ranges are inserted into an [IOT](#) and the query is executed by performing a join.

Moving on to interpreting the results, we can see that the 200 ranges in the WHERE clause do not so much affect the percentage of extra points obtained (with some exceptions). However, the query response times in the non-integrated approach are 2- 6 times more expensive than the integrated ones. The non-integrated approach is, also, tested in the medium benchmark ([Table C.18](#)), where another undesired effect takes place. Query ST-G in the medium benchmark stage becomes extremely expensive, only to become 40 times faster in the next stage. The previous two observations made me realise that the non-integrated approach is, for the time being, not a suitable approach. Nevertheless, the approach should have been further investigated, especially concerning the fluctuation. Because of lack of time, I excluded the non-integrated approach from any further benchmarks.

Approach	Time (s)			Size (GB)	Points		Points per sec.	
	morton prep.	Load heap	Load IOT		Heap	IOT	Heap	IOT
xy - S	174.80	28.22	34.12	0.8	35,715,834	35,715,834	1,265,516	1,046,771
xy - M	175.93	24.60	76.49	1.6	37,255,628	72,971,462	1,514,181	954,000
xy - L	336.45	38.58	163.57	3.2	68,522,972	141,494,434	1,775,999	865,039
xyz - S	702.98	25.46	24.78	0.7	35,715,834	35,715,834	1,402,632	1,441,317
xyz - M	753.86	23.38	62.90	1.4	37,255,628	72,971,462	1,593,408	1,160,119
xyz - L	1294.07	38.65	130.70	2.8	68,522,972	141,494,434	1,773,085	1,082,589
xyt - S	709.48	26.74	26.26	0.8	35,715,834	35,715,834	1,335,902	1,360,085
xyt - M	763.42	24.40	75.19	1.6	37,255,628	72,971,462	1,527,176	970,494
xyt - L	1366.19	43.93	136.90	3.2	68,522,972	141,494,434	1,559,965	1,033,561
xyzt - S	888.01	24.76	23.41	0.7	35,715,834	35,715,834	1,442,742	1,525,666
xyzt - M	892.89	22.60	62.45	1.4	37,255,628	72,971,462	1,648,147	1,168,478
xyzt - L	1695.54	39.39	110.57	2.8	68,522,972	141,494,434	1,739,567	1,279,682

Table C.15: The loading times for the two integrations of space and time and the two treatments of z in the coastline use case (Small benchmark).

Case	id	fetching (s)			% extra points			Final Points		
		S	M	L	S	M	L	S	M	L
xy	ST-A	12.31	11.97	12.14	2	2	2	389554	389554	389554
	ST-B	12.22	11.60	12.17	3	3	3	342239	342239	342239
	ST-C	2.35	2.50	2.16	10	10	10	69339	69339	69339
	ST-D	2.01	2.21	2.42	13	13	13	69967	69967	69967
	ST-E	4.84	4.41	4.67	14	14	14	134998	134998	134998
	S-A	0.80	2.12	4.02	8	8	8	71999	151687	301429
xyz	ST-A	12.47	12.77	12.58	11	11	11	389554	389554	389554
	ST-B	12.94	13.03	12.66	16	16	16	342239	342239	342239
	ST-C	2.97	3.01	2.93	48	48	48	69339	69339	69339
	ST-D	1.79	1.77	1.75	43	43	43	69967	69967	69967
	ST-E	6.15	6.19	5.97	56	56	56	134998	134998	134998
	S-A	2.52	5.32	10.30	14	14	13	71999	151687	301429
xyt	ST-A	2.17	2.18	2.15	3	3	3	389554	389554	389554
	ST-B	1.92	1.92	1.92	4	4	4	342239	342239	342239
	ST-C	0.50	0.51	0.49	4	4	4	69339	69339	69339
	ST-D	0.50	0.49	0.49	2	2	2	69967	69967	69967
	ST-E	0.88	0.88	0.87	4	4	4	134998	134998	134998
	S-A	0.48	0.94	1.72	2	2	4	71999	151687	301429
xyzt	ST-A	1.35	1.37	1.36	3	3	3	389554	389554	389554
	ST-B	1.21	1.21	1.22	4	4	4	342239	342239	342239
	ST-C	0.39	0.39	0.40	9	9	9	69339	69339	69339
	ST-D	0.37	0.36	0.37	9	9	9	69967	69967	69967
	ST-E	0.63	0.64	0.64	9	9	9	134998	134998	134998
	S-A	0.28	0.57	1.11	8	8	16	71999	151687	301429

Table C.16: The query response times, the percentage of false hits compared to the actual number of points and the number of points returned by the queries for the two integrations of space and time and the two treatments of z in the coastline use case (Small benchmark).

c.2.2 Medium benchmark

Approach	Time (s)			Size (GB)	Points		Points per sec.	
	morton prep.	Load heap	Load IOT		Heap	IOT	Heap	IOT
xy - S	435.74	68.30	67.70	2.0	87,764,427	87,764,427	1,284,995	1,296,373
xy - M	379.61	53.64	164.23	3.9	82,834,612	170,599,039	1,544,180	1,038,781
xy - L	707.39	98.35	313.53	7.4	154,540,077	325,139,116	1,571,281	1,037,027
xyz - S	1824.31	63.24	60.67	1.7	87,764,427	87,764,427	1,387,819	1,446,587
xyz - M	1654.04	47.32	143.92	3.4	82,834,612	170,599,039	1,750,476	1,185,374
xyz - L	3060.91	86.89	270.09	6.4	154,540,077	325,139,116	1,778,666	1,203,818
xyt - S	1846.18	64.65	63.06	2.0	87,764,427	87,764,427	1,357,582	1,391,761
xyt - M	1674.60	53.04	154.04	3.8	82,834,612	170,599,039	1,561,751	1,107,498
xyt - L	3114.51	96.83	345.01	7.3	154,540,077	325,139,116	1,596,072	942,405
xyzt - S	2238.40	59.19	65.23	1.7	87,764,427	87,764,427	1,482,634	1,345,461
xyzt - M	2029.13	47.79	147.83	3.4	82,834,612	170,599,039	1,733,242	1,154,022
xyzt - L	3769.96	87.51	263.20	6.4	154,540,077	325,139,116	1,765,871	1,235,331

Table C.17: The loading times for the two integrations of space and time and the two treatments of z in the coastline use case (Medium benchmark).

Case	id	fetching (s)			% extra points			Final Points		
		S	M	L	S	M	L	S	M	L
xy	ST-A	13.62	13.47	13.64	2	2	2	389554	389554	389554
	ST-B	11.48	11.42	11.50	3	3	3	342239	342239	342239
	ST-C	2.56	2.52	2.50	10	10	10	69339	69339	69339
	ST-D	2.49	2.52	2.47	13	13	13	69967	69967	69967
	ST-E	4.50	4.52	4.50	14	14	14	134998	134998	134998
	ST-F	-	7.11	7.02	-	2.3	2.3	-	330261	330261
	ST-G	-	830.72	18.42	-	8.3	8.3	-	518702	518702
	S-A	1.12	2.36	4.61	8	8	8	71999	151687	301429
	S-B	0.51	0.96	1.76	31.8	31.9	31.6	20135	38523	71823
	S-C	2.11	3.74	8.07	19.7	20.4	20.4	98382	174746	382500
xyz	ST-A	13.58	13.52	13.64	11	11	11	389554	389554	389554
	ST-B	11.87	11.87	12.04	16	16	16	342239	342239	342239
	ST-C	2.96	2.97	2.91	48	48	48	69339	69339	69339
	ST-D	1.75	1.75	1.75	43	43	43	69967	69967	69967
	ST-E	5.36	5.39	5.26	56	56	56	134998	134998	134998
	ST-F	-	8.92	8.86	-	4.1	4.1	-	330261	330261
	ST-G	-	820.33	21.04	-	40.4	40.4	-	518702	518702
	S-A	2.60	5.41	10.75	14	14	13	71999	151687	301429
	S-B	0.83	1.60	2.94	55.4	54.8	54.0	20135	38523	71823
	S-C	3.94	7.32	16.58	40.9	43.6	45.5	98382	174746	382500
xyt	ST-A	2.16	2.16	2.16	3	3	3	389554	389554	389554
	ST-B	1.93	1.92	1.92	4	4	4	342239	342239	342239
	ST-C	0.49	0.50	0.49	4	4	4	69339	69339	69339
	ST-D	0.49	0.49	0.49	2	2	2	69967	69967	69967
	ST-E	0.88	0.88	0.87	4	4	4	134998	134998	134998
	ST-F	-	1.84	1.82	-	3.7	3.7	-	330261	330261
	ST-G	-	3.10	3.02	-	3.2	3.2	-	518702	518702
	S-A	0.49	0.95	1.72	2	2	4	71999	151687	301429
	S-B	0.16	0.30	0.48	8.7	8.7	16.9	20135	38523	71823
	S-C	0.64	1.10	2.31	4.9	10.2	10.2	98382	174746	382500
xyzt	ST-A	1.36	1.35	1.37	3	3	3	389554	389554	389554
	ST-B	1.21	1.23	1.24	4	4	4	342239	342239	342239
	ST-C	0.39	0.39	0.40	9	9	9	69339	69339	69339
	ST-D	0.37	0.35	0.36	9	9	9	69967	69967	69967
	ST-E	0.64	0.63	0.65	9	9	9	134998	134998	134998
	ST-F	-	1.15	1.14	-	2.3	2.3	-	330261	330261
	ST-G	-	1.91	1.92	-	6.5	6.5	-	518702	518702
	S-A	0.28	0.57	1.13	8	8	16	71999	151687	301429
	S-B	0.11	0.18	0.34	17.0	31.9	31.6	20135	38523	71823
	S-C	0.41	0.71	1.67	19.7	20.4	38.7	98382	174746	382500

Table C.18: The query response times, the percentage of false hits compared to the actual number of points and the number of points returned by the queries for the two integrations of space and time and the two treatments of z in the coastline use case (Medium benchmark).

c.2.3 Large benchmark

Approach	Time (s)			Size (GB)	Points		Points per sec.	
	morton prep.	Load heap	Load IOT		Heap	IOT	Heap	IOT
xyt - S	5232.27	155.47	205.97	5.5	245,905,596	245,905,596	1,581,667	1,193,890
xyt - M	4440.71	136.58	439.40	10.4	219,547,749	465,453,345	1,607,426	1,059,293
xyt - L	8698.05	266.03	860.26	20.1	429,411,108	894,864,453	1,614,151	1,040,226
xyzt - S	6293.14	145.70	180.86	4.9	245,905,596	245,905,596	1,687,750	1,359,646
xyzt - M	5332.41	123.63	394.91	9.2	219,547,749	465,453,345	1,775,912	1,178,631
xyzt - L	10363.12	239.08	820.88	17.6	429,411,108	894,864,453	1,796,128	1,090,128

Table C.19: The loading times for the two integrations of space and time and the two treatments of z in the coastline use case (Large benchmark).

Case	id	fetching (s)			% extra points			Final Points		
		S	M	L	S	M	L	S	M	L
xyt	ST-A	2.20	2.18	2.20	3	3	3	389554	389554	389554
	ST-B	1.93	1.93	1.91	4	4	4	342239	342239	342239
	ST-C	0.51	0.50	0.50	4	4	4	69339	69339	69339
	ST-D	0.49	0.49	0.49	2	2	2	69967	69967	69967
	ST-E	0.90	0.89	0.89	4	4	4	134998	134998	134998
	ST-F	-	1.83	1.86	-	3.7	3.7	-	330261	330261
	ST-G	-	3.11	3.34	-	3.2	3.2	-	518702	518702
	S-A	0.49	0.95	1.74	2	2	4	71999	151687	301429
	S-B	0.16	0.30	0.53	8.7	8.7	16.9	20135	38523	71823
	S-C	0.64	1.13	2.48	4.9	10.2	10.2	98382	174746	382500
xyzt	ST-A	1.36	1.27	1.25	7	7	7	389554	389554	389554
	ST-B	1.21	1.11	1.13	8	8	8	342239	342239	342239
	ST-C	0.33	0.32	0.31	17	17	17	69339	69339	69339
	ST-D	0.40	0.39	0.40	9	9	9	69967	69967	69967
	ST-E	0.59	0.57	0.57	17	17	17	134998	134998	134998
	ST-F	-	1.03	1.03	-	3.7	3.7	-	330261	330261
	ST-G	-	1.86	1.84	-	6.5	6.5	-	518702	518702
	S-A	0.28	0.55	1.03	8	16	16	71999	151687	301429
	S-B	0.11	0.19	0.36	31.8	31.9	62.7	20135	38523	71823
	S-C	0.42	0.73	1.56	19.7	38.6	38.7	98382	174746	382500

Table C.20: The query response times, the percentage of false hits compared to the actual number of points and the number of points returned by the queries for the two integrations of space and time and the two treatments of z in the coastline use case (Large Benchmark).

c.2.4 Full benchmark (All stages)

Stage	ID	Range calc. (s)	Range IOT (s)	No. of ranges	Depth	Fetch (s)	Decode (s)	Store (s)	Candidate pts	Final pts	Refinement (s)
Small	ST-A	1.6	0.23	4508	19	2.29	4.50	0.43	400985	389554	3.90
	ST-B	1.9	0.17	4955	19	1.98	3.90	0.42	355382	342239	3.33
	ST-C	10.7	0.24	33327	21	0.48	0.90	0.23	72415	69339	0.96
	ST-D	21.9	0.17	17054	22	0.46	0.88	0.22	71600	69967	0.94
	ST-E	8.4	0.22	26679	21	0.86	1.59	0.18	140884	134998	1.55
	ST-F	-	-	-	-	-	-	-	-	-	-
	ST-G	-	-	-	-	-	-	-	-	-	-
	S-A	4.4	0.18	13067	21	0.45	0.92	0.23	73396	71999	0.94
	S-B	2.9	0.17	7595	20	0.14	0.27	0.27	21892	20135	0.38
	S-C	5.0	0.18	15147	21	0.57	1.27	0.26	103154	98382	1.29
Medium	ST-A	1.7	0.44	4508	19	2.20	4.88	0.52	400985	389554	4.39
	ST-B	1.8	0.19	4955	19	1.95	4.26	0.55	355382	342239	3.62
	ST-C	11.9	0.27	33327	21	0.49	0.92	0.15	72415	69339	1.00
	ST-D	22.9	0.24	17054	22	0.50	0.92	0.14	71600	69967	0.99
	ST-E	8.8	0.24	26679	21	0.88	1.83	0.22	140884	134998	1.77
	ST-F	0.6	0.17	1307	17	2.59	5.78	0.86	483083	461432	5.59
	ST-G	35.4	0.34	71675	20	3.15	6.71	0.72	535058	518702	5.78
	S-A	22.0	0.25	23471	21	0.96	1.96	0.23	154573	151687	1.80
	S-B	20.9	0.22	22209	20	0.29	0.53	0.12	41864	38523	0.63
	S-C	10.8	0.21	11165	20	1.12	2.22	0.26	192594	174746	2.27
Large	ST-A	1.7	0.89	4508	19	2.20	4.76	0.50	400985	389554	4.46
	ST-B	1.8	0.20	4955	19	1.95	4.23	0.50	355382	342239	4.21
	ST-C	11.8	0.29	33327	21	0.48	0.90	0.15	72415	69339	1.02
	ST-D	21.1	0.22	17054	22	0.47	0.89	0.14	71600	69967	1.02
	ST-E	8.9	0.24	26679	21	0.83	1.70	0.21	140884	134998	1.79
	ST-F	0.6	0.17	1307	17	2.48	5.51	0.62	483083	461432	4.94
	ST-G	34.9	0.40	71675	20	3.06	6.34	0.63	535058	518702	5.58
	S-A	11.5	0.20	12151	20	1.73	3.76	0.44	313246	301429	3.36
	S-B	12.2	0.19	6320	19	0.49	1.06	0.16	83968	71823	1.09
	S-C	27.6	0.24	17847	20	2.27	5.38	0.52	421396	382500	4.50

Table C.21: Benchmark results for all query stages. Case: integrated approach with z as an attribute of the full benchmark.

Stage	ID	Range calc. (s)	Range IOT (s)	No. of ranges	Depth	Fetch (s)	Decode (s)	Store (s)	Candidate pts	Final pts	Refinement (s)
Small	ST-A	3.1	0.26	8225	18	1.25	7.01	0.50	417417	389554	4.56
	ST-B	3.4	0.21	9044	18	1.11	6.35	0.43	370392	342239	3.93
	ST-C	11.7	0.27	30641	19	0.32	1.52	0.25	81210	69339	1.06
	ST-D	32.0	0.45	108775	20	0.41	1.41	0.33	76495	69967	1.04
	ST-E	8.9	0.24	24310	19	0.57	2.80	0.34	158435	134998	1.79
	ST-F	-	-	-	-	-	-	-	-	-	-
	ST-G	-	-	-	-	-	-	-	-	-	-
	S-A	4.2	0.20	11284	19	0.28	1.46	0.15	77889	71999	1.00
	S-B	2.1	0.18	4158	18	0.10	0.49	0.09	26530	20135	0.45
	S-C	5.4	0.22	13936	19	0.39	2.22	0.19	117753	98382	1.43
Medium	ST-A	2.9	0.58	8225	18	1.35	6.88	0.53	417417	389554	4.46
	ST-B	3.6	0.21	9044	18	1.22	6.23	0.42	370392	342239	3.85
	ST-C	10.6	0.27	30641	19	0.32	1.51	0.15	81210	69339	1.07
	ST-D	32.3	0.49	108775	20	0.40	1.38	0.14	76495	69967	1.03
	ST-E	8.3	0.25	24310	19	0.60	2.77	0.23	158435	134998	1.89
	ST-F	1.9	0.18	5228	17	1.53	7.98	0.64	483083	461432	4.90
	ST-G	316.3	0.61	153646	19	1.97	8.91	0.62	552408	518702	5.71
	S-A	3.5	0.18	2578	18	0.58	3.10	0.24	175738	151687	2.00
	S-B	10.0	0.22	7855	18	0.19	0.97	0.13	50806	38523	0.71
	S-C	5.4	0.22	3946	18	0.77	3.88	0.30	242166	174746	2.44
Large	ST-A	3.1	0.93	8225	18	1.34	7.13	0.48	417417	389554	4.39
	ST-B	3.5	0.21	9044	18	1.23	5.88	0.45	370392	342239	3.77
	ST-C	11.2	0.28	30641	19	0.33	1.50	0.15	81210	69339	1.08
	ST-D	32.6	0.55	108775	20	0.41	1.41	0.14	76495	69967	1.05
	ST-E	8.2	0.25	24310	19	0.58	2.87	0.23	158435	134998	1.90
	ST-F	1.9	0.20	5228	17	1.53	7.68	0.58	483083	461432	4.88
	ST-G	324.9	0.70	153646	19	1.97	9.23	0.61	552408	518702	5.82
	S-A	9.2	0.17	3000	18	1.10	5.77	0.43	348564	301429	3.45
	S-B	2.9	0.20	2150	17	0.39	2.07	0.19	116864	71823	1.33
	S-C	12.8	0.21	5568	18	1.65	8.73	0.56	530600	382500	4.70

Table C.22: Benchmark results for all query stages. Case: integrated approach with z as a key of the full benchmark.

D

CODE DESCRIPTION AND SQL STATEMENTS

This chapter contains a selection of the most important SQL statements that were used for the implementation of the methodology described in [Chapter 4](#). Apart from the used SQL statements, also other alternatives that were considered but not used are described. Since, this is a selection of statements, for the full code the reader is referred to the on-line code found at: <https://github.com/stpsomad/DynamicPCDMS>.

The chapter is divided into two sections, namely; SQL statements concerning the creation of the tables and the loading, and examples of query statements.

D.1 LOADING SCRIPTS

The implementation for all four cases examined thoroughly within this thesis begins with the **preparation phase** that concerns the creation of the heap table. The heap table is an unordered table where the point cloud data are bulk loaded. The existence of the heap table is temporary because after the data have been organised, it is dropped.

Listing D.1: Heap table creation for the four cases

```
-- CREATING HEAP TABLE

-- Case non-integrated (z as an attribute)
CREATE TABLE xy_temp (
time NUMBER,
morton NUMBER,
z NUMBER)
TABLESPACE USERS
PCTFREE 0 NOLOGGING;

-- Case non-integrated (z added in code)
CREATE TABLE xyz_temp (
time NUMBER,
morton NUMBER)
TABLESPACE USERS
PCTFREE 0 NOLOGGING;

-- Case integrated (z as an attribute)
CREATE TABLE xyt_temp (
morton NUMBER,
z NUMBER)
TABLESPACE USERS
PCTFREE 0 NOLOGGING;

-- Case integrated (z added in code)
CREATE TABLE xyzt_temp (
morton NUMBER)
TABLESPACE USERS
PCTFREE 0 NOLOGGING;
```

The bulk loading of the data begins by converting all the required data into the [SFC](#) of our choice. In the case examined the chosen [SFC](#) is the morton curve. For this, a conversion utility has been created, the *mortonConverter*. The converter in its current form takes as input a configuration file (*.ini) that includes parameters like: the format of the files, the integration used, the treatment of z, the scaling of time, the resolution of time etc. According to the parameters used, the converter proceeds to the transformation of the coordinates and the conversion to the morton curve.

This procedure is pipelined with the SQLLDR utility¹ of the Oracle Database that bulk loads external files into the database. The data are sent into the previously initialised heap table. A general example of use of the *mortonConverter* and the SQLLDR utility is:

Listing D.2: Morton conversion pipelined with the SQLLDR utility

```
python -m pointcloud.mortonConverter [config_file] |
sqlldr [user]/[password]@//[host]:[port]/[database]
direct=true control=control_file.ctl data=\"-\"
bad=bad_file.bad log=log_file.log
```

After the bulk loading is completed, the implementation moves to the second phase of the loading procedure; the **Loading**. Within this step the Index Organised Table ([IOT](#)) is generated from the previously created heap table. This step either finalises the loading procedure or the user proceeds to the phase of gathering the required optimiser statistics.

Listing D.3: [IOT](#) creation for the four cases

```
-- CREATING INDEX ORGANISED TABLE

-- Case non-integrated (z as an attribute)
CREATE TABLE xy
(time, morton, z,
CONSTRAINT xy_PK PRIMARY KEY (time, morton))
ORGANIZATION INDEX
TABLESPACE INDX
PCTFREE 0 NOLOGGING
AS
SELECT time, morton, z FROM xy_temp;

-- Case non-integrated (z added in code)
CREATE TABLE xyz
(time, morton,
CONSTRAINT xyz_PK PRIMARY KEY (time, morton))
ORGANIZATION INDEX
TABLESPACE INDX
PCTFREE 0 NOLOGGING
AS
SELECT time, morton FROM xyz_temp;

-- Case integrated (z as an attribute)
CREATE TABLE xyt
(morton, z,
CONSTRAINT xyt_PK PRIMARY KEY (morton))
ORGANIZATION INDEX
TABLESPACE INDX
PCTFREE 0 NOLOGGING
AS
SELECT morton, z FROM xyt_temp;

-- Case integrated (z added in code)
```

¹ <http://docs.oracle.com/database/121/SUTIL/GUID-DD843EE2-1FAB-4E72-A115-21D97A501ECC.htm#SUTIL003>

```

CREATE TABLE xyzt
(morton,
CONSTRAINT xyzt_PK PRIMARY KEY (morton))
ORGANIZATION INDEX
TABLESPACE INDX
PCTFREE 0 NOLOGGING
AS
SELECT morton FROM xyzt_temp;

```

In the case that new data need to be added into the database, the procedure is as follows:

- (a) The heap table is created as in [Listing D.1](#).
- (a) The new data are bulk loaded into the heap tables as in [Listing D.2](#).
- (a) The new and the old data are combined together as in [Listing D.4](#).

Listing D.4: Adding new data to the IOT for the four cases

```

-- LOADING NEW DATA TO THE INDEX ORGANISED TABLE

-- Case non-integrated (z as an attribute)
CREATE TABLE xy_new
(time, morton, z,
CONSTRAINT xy_new_PK PRIMARY KEY(time, morton))
ORGANIZATION INDEX TABLESPACE INDX
PCTFREE 0 NOLOGGING
AS
SELECT time, morton, z FROM xy_temp
UNION ALL
SELECT time, morton, z FROM xy;

-- Case non-integrated (z added in code)
CREATE TABLE xyz_new
(time, morton,
CONSTRAINT xyz_new_PK PRIMARY KEY(time, morton))
ORGANIZATION INDEX TABLESPACE INDX
PCTFREE 0 NOLOGGING
AS
SELECT time, morton FROM xyz_temp
UNION ALL
SELECT time, morton FROM xyz;

-- Case integrated (z as an attribute)
CREATE TABLE xyt_new
(morton, z,
CONSTRAINT xyt_new_PK PRIMARY KEY(morton))
ORGANIZATION INDEX TABLESPACE INDX
PCTFREE 0 NOLOGGING
AS
SELECT morton, z FROM xyt_temp
UNION ALL
SELECT morton, z FROM xyt;

-- Case integrated (z added in code)
CREATE TABLE xyzt_new
(morton,
CONSTRAINT xyzt_new_PK PRIMARY KEY(morton))
ORGANIZATION INDEX TABLESPACE INDX
PCTFREE 0 NOLOGGING
AS
SELECT morton FROM xyzt_temp
UNION ALL
SELECT morton FROM xyzt;

```

An alternative to the previous **preparation phase** is to use the external table² feature of Oracle. This feature allows to use data outside the database as if they were a table inside the database. To use external tables the following steps need to be followed (Listing D.5):

- (a) Oracle directories pointing to the data are created.
- (a) The connection between the directory and the database is established.

The reason why I do not use this method is that it requires writing binary or ASCII files and storing them somewhere on the system. This is not ideal as it ends up having extra duplication of the data.

Listing D.5: IOT creation for the four cases

```
CREATE DIRECTORY INPUT_DATA_DIR AS 'DIRECTORY';
GRANT READ ON DIRECTORY INPUT_DATA_DIR TO [user];
GRANT WRITE ON DIRECTORY INPUT_DATA_DIR TO [user];

CREATE TABLE XY_TEMP_EXT (
  time NUMBER,
  morton NUMBER,
  z NUMBER)
  ORGANIZATION EXTERNAL(
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY INPUT_DATA_DIR
    ACCESS PARAMETERS (
      RECORDS DELIMITED BY NEWLINE
      FIELDS TERMINATED BY ' ' )
    LOCATION ('*.txt'))
  REJECT LIMIT 0;
```

D.2 QUERY SCRIPTS

The queries (presented in Appendix B) are executed by the use of Python scripts. The queries are loaded into a database table called QUERIES. The spatial part of the query is loaded using Well Known Text (WKT). The QUERIES table contains the following information:

- (a) The query ID
- (a) The dataset in which it belongs (Sand Engine, Coastline)
- (a) The type of query (i.e. space - time, only space and only time)
- (a) The geometry (NULL if the type of query is time)
- (a) The start and end date (NULL if the type of query is space)
- (a) The minimum and maximum height (if any)

When a certain query needs to be executed the implemented scripts read the required information from the QUERIES table.

² <http://docs.oracle.com/database/121/SUTIL/GUID-44323E01-7D72-45EC-915A-99E596769D9E.htm#SUTIL011>

D.2.1 Filter step

According to the type of integration, treatment of z used and type of query the Python script uses the required 2^n -tree to identify the morton ranges that approximate the geometry. The SQL statement for the filter step can be specified in two different ways. Within the following examples, I specify the morton ranges in the *WHERE* statement for the non-integrated approach and I use a *join* statement for the integrated one.

Time queries

In the non-integrated approach time queries are straight forward and do not require a two step approach. This is because time is stored separate. A time query that requests the point between two moments in time for both treatments of z is specified as in [Listing D.6](#).

Listing D.6: Time query in the non-integrated approach

```
SELECT time, morton[, z]
FROM xy[z]
WHERE (time BETWEEN 4681 AND 4682)
```

In the integrated approach time queries follow the two step query process. The filter step is executed as follows:

- (a) The morton ranges are loaded into an *IOT* named RANGES.
- (a) The data table and the RANGES table are joined.

The ranges table has the following schema:

Listing D.7: The initialisation of the *IOT* containing the morton ranges

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

The ranges are bulk loaded with the sasme SQLLDR utility. The same query as in the non-integrated approach is now executed using the following SQL statement. Note that an optimiser hint (*USE_NL (t r)*) is being used as, without it, the proposed execution plan does not provide an efficient execution time.

Listing D.8: Time query in the integrated approach

```
SELECT /*+ USE_NL (t r)*/ t.morton[, t.z]
FROM xy[z]t t, RANGES r
WHERE (t.morton BETWEEN r.low AND r.upper);
```

Space queries

Space queries in the non-integrated approach only filter on the morton column. In this example the spatial component is a rectangle. The query is performed as follows:

Listing D.9: Space query in the non-integrated approach

```
SELECT time, morton, z
FROM xy[z]
```

```

WHERE ((morton between 170018964766720 and 170019501637631) OR
(morton between 170021917556736 and 170022722863103) OR
(morton between 170022991298560 and 170026212524031) OR
(morton between 170026480959488 and 170029165314047) OR
(morton between 170039902732288 and 170040171167743) OR
(morton between 170040439603200 and 170040708038655) OR
(morton between 170046345183232 and 170046882054143) OR
(morton between 170047150489600 and 170048492666879) OR
(morton between 170049566408704 and 170050103279615) OR
(morton between 170050371715072 and 170054935117823) OR
(morton between 170059230085120 and 170059766956031) OR
(morton between 170060035391488 and 170061377568767) OR
(morton between 170061646004224 and 170061914439679) OR
(morton between 170062451310592 and 170064598794239) OR
(morton between 170065135665152 and 170065404100607) OR
(morton between 170065672536064 and 170067820019711) OR
(morton between 170070504374272 and 170070772809727) OR
(morton between 170072114987008 and 170076678389759) OR
(morton between 170076946825216 and 170077483696127) OR
(morton between 170078557437952 and 170079899615231) OR
(morton between 170080168050688 and 170080704921599) OR
(morton between 193519683633152 and 193519952068607) OR
(morton between 193531226357760 and 193533373841407) OR
(morton between 193533642276864 and 193533910712319) OR
(morton between 193534447583232 and 193539816292351) OR
(morton between 193540890034176 and 193541426905087) OR
(morton between 193541695340544 and 193541963775999) OR
(morton between 193544111259648 and 193546795614207) OR
(morton between 193547064049664 and 193548406226943) OR
(morton between 193570954805248 and 193572028547071) OR
(morton between 193573370724352 and 193573639159807) OR
(morton between 193582765965312 and 193583034400767) OR
(morton between 193583302836224 and 193583571271679) OR
(morton between 193584913448960 and 193585987190783) OR
(morton between 193586524061696 and 193586792497151) OR
(morton between 193591355899904 and 193595650867199) OR
(morton between 193596187738112 and 193596456173567) OR
(morton between 193597798350848 and 193598066786303) OR
(morton between 193598335221760 and 193598603657215) OR
(morton between 193617125703680 and 193619004751871) OR
(morton between 193619273187328 and 193619810058239));

```

In the integrated approach the same methodology as in the time queries is followed. The SQL statement is also the same as in [Listing D.8](#).

Space - time queries

Space - time queries in the non-integrated approach could be considered a combination of time and space queries. By this is meant that the WHERE clause contains predicates both in the time and morton columns. In the following example the spatial component represents an axis aligned rectangle, while the temporal component requests two distinct moments in time.

Listing D.10: Space - time query in the non-integrated approach

```

SELECT time, morton[, z]
FROM xy[z]
WHERE ((time IN (3655, 3680)) AND
((morton between 151177480110080 and 151178553851903) OR
(morton between 151180701335552 and 151182848819199) OR
(morton between 151544699813888 and 151545773555711) OR
(morton between 151546847297536 and 151552216006655) OR
(morton between 151553289748480 and 151554363490303) OR
(morton between 151555437232128 and 151560805941247) OR
(morton between 151579059552256 and 151580133294079) OR

```

```
(morton between 151581207035904 and 151586575745023) OR
(morton between 151587649486848 and 151588723228671) OR
(morton between 151589796970496 and 151595165679615) OR
(morton between 151682138767360 and 151683212509183) OR
(morton between 151684286251008 and 151689654960127) OR
(morton between 151690728701952 and 151691802443775) OR
(morton between 151692876185600 and 151698244894719) OR
(morton between 151716498505728 and 151717572247551) OR
(morton between 151718645989376 and 151724014698495) OR
(morton between 151725088440320 and 151726162182143) OR
(morton between 151727235923968 and 151732604633087) OR
(morton between 153193967255552 and 153195040997375) OR
(morton between 153196114739200 and 153201483448319) OR
(morton between 153202557190144 and 153203630931967) OR
(morton between 153204704673792 and 153210073382911) OR
(morton between 154114163998720 and 154116311482367) OR
(morton between 154118458966016 and 154120606449663) OR
(morton between 154131343867904 and 154133491351551) OR
(morton between 154135638835200 and 154137786318847) OR
(morton between 154182883475456 and 154185030959103) OR
(morton between 154187178442752 and 154189325926399) OR
(morton between 154200063344640 and 154202210828287) OR
(morton between 154204358311936 and 154206505795583) OR
(morton between 154389041905664 and 154391189389311) OR
(morton between 154393336872960 and 154394410614783) OR
(morton between 154481383702528 and 154761630318591) OR
(morton between 154762704060416 and 154763777802239) OR
(morton between 154764851544064 and 154770220253183) OR
(morton between 154771293995008 and 154772367736831) OR
(morton between 154790621347840 and 154795990056959) OR
(morton between 154797063798784 and 154798137540607) OR
(morton between 154799211282432 and 154804579991551) OR
(morton between 154805653733376 and 154806727475199) OR
(morton between 154893700562944 and 154899069272063) OR
(morton between 154900143013888 and 154901216755711) OR
(morton between 154902290497536 and 154907659206655) OR
(morton between 154908732948480 and 154909806690303) OR
(morton between 154928060301312 and 154933429010431) OR
(morton between 154934502752256 and 154935576494079) OR
(morton between 154936650235904 and 154942018945023) OR
(morton between 154943092686848 and 154944166428671) OR
(morton between 156130651144192 and 156165010882559) OR
(morton between 156199370620928 and 156233730359295) OR
(morton between 156405529051136 and 156410897760255) OR
(morton between 156411971502080 and 156413045243903) OR
(morton between 156414118985728 and 156419487694847) OR
(morton between 156420561436672 and 156421635178495)))
```

In the integrated approach the query statement is, again, the same as in [Listing D.8](#).

D.2.2 Refinement step

Ideally the refinement step should be able to be performed together with the filter one. However, since only the morton keys are stored inside the database (and not the original dimensions) a decoding of the key needs to take place first. At this moment in time, there exists no function inside the database that can do this decoding. For this reason, the application is “forced” to perform this decoding outside the database. This is not ideal as a lot of unnecessary and expensive movement of data needs to take place. Nevertheless, the refinement step is performed as follows.

The result of the filter step is fetched inside the Python code and the decoding takes place. These data are then returned to the database for the

final refinement step. Because the *SDO_PointInPolygon* operator in Oracle is used, the table that is created has the following schema.

Listing D.11: Time query in the integrated approach

```
CREATE TABLE refinement (
time VARCHAR2(20),
X NUMBER,
Y NUMBER,
Z NUMBER)
```

Finally, the refinement step is performed. Between the two integrations, this step differs slightly in that the integrated approach requires an extra predicate in the time dimension for time and space - time queries. The refinement step of the previously presented queries are:

Time queries

As mentioned before in the non-integrated approach no further refinement is required. The query procedure ends with the creation of the previous table. On the other hand, in the integrated approach a refinement in the time dimension needs to take place.

Listing D.12: Time refinement query in the integrated approach

```
CREATE TABLE result
AS SELECT *
FROM (SELECT X, Y, Z, TO_DATE(TIME, 'yyyy/mm/dd') AS TIME
      FROM refinement)
WHERE (TIME BETWEEN TO_DATE('2002/10/25', 'YYYY/MM/DD') AND
TO_DATE('2002/10/26', 'YYYY/MM/DD'));
```

Space queries

The refinement step in the integrated and non-integrated approach is the same and includes the execution of a point in polygon operation.

Listing D.13: Space refinement query in the integrated and non-integrated approach

```
CREATE TABLE result
AS (SELECT *
FROM TABLE (mdsys.sdo_PointInPolygon (CURSOR (
SELECT X, Y, Z, TO_DATE(TIME, 'yyyy/mm/dd') AS TIME
FROM refinement),
MDSYS.SDO_GEOMETRY('POLYGON ((73205.024 452445.009,
73465.172 452762.02, 73537.673 452698.05,
73281.789 452379.617, 73205.024 452445.009))',
28992), 0.001)));
```

Space - time queries

The refinement step of space - time queries for the non-integrated approach is the same as the one used for space queries.

Listing D.14: Space - time refinement query in the non-integrated approach

```
CREATE TABLE result
AS (SELECT *
FROM TABLE(mdsys.sdo_PointInPolygon(CURSOR(
SELECT X, Y, Z, TO_DATE(TIME, 'yyyy/mm/dd') AS TIME
FROM refinement),
MDSYS.SDO_GEOMETRY('POLYGON ((71028.591 451007.796,
```

```
71027.169 451654.613, 71715.212 451656.034,
71716.634 451006.374, 71028.591 451007.796))',
28992), 0.001)))
```

For the integrated approach, an extra time predicate needs to be added compared to the non-integrated case.

Listing D.15: Space - time refinement query in the integrated approach

```
CREATE TABLE result
AS (SELECT *
FROM TABLE (mdsys.sdo_PointInPolygon (CURSOR (
SELECT X, Y, Z, TO_DATE(TIME, 'yyyy/mm/dd') AS TIME
FROM refinement),
MDSYS.SDO_GEOMETRY('POLYGON ((71028.591 451007.796,
71027.169 451654.613, 71715.212 451656.034,
71716.634 451006.374, 71028.591 451007.796))',
28992), 0.001))
WHERE (TIME IN (TO_DATE('2000/01/03', 'YYYY/MM/DD'),
TO_DATE('2000/01/28', 'YYYY/MM/DD'))));
```

D.3 VALIDATION SCRIPTS

To validate that the answers of the queries using the proposed methodology are correct, a solution using available spatial and date data types was developed. This solution is also a proof that using 3D-point/ date data types with normal R-Tree and B-tree indexes is not an efficient alternative.

D.4 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:

Listing D.16: Initialisation of the spatial table

```
CREATE TABLE validation (
GEOM SDO_GEOMETRY,
TIME DATE);
```

The data are then read from the LAZ files and formatted according to the rules required by the SQLLDR ([Listing D.17](#)). Then, in the **loading phase**, the data are bulk loaded into the table ([Listing D.18](#)).

Listing D.17: Space - time refinement query in the integrated approach

```
load data
append into table validation
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
));
```

Listing D.18: Space - time refinement query in the integrated approach

```
python -m las2txyz [config_file] |
sqlldr [user]/[password]@[host]:[port]/[database]
direct=true control=control_file.ctl data=\"-\"
bad=bad_file.bad log=log_file.log
```

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

Listing D.19: Inserting the spatial metadata into the user_sdo_geom_metadata view

```
INSERT INTO user_sdo_geom_metadata
(table_name, column_name, srid, diminfo)
VALUES ('validation', '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.

Listing D.20: Space - time refinement query in the integrated approach

```
CREATE INDEX valid_rtree_IDX ON validation(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 validation(TIME);
```

D.5 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 by the database system. The following examples present the same queries as presented previously for the three types of queries.

Time queries

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

Listing D.21: Space - time refinement query in the integrated approach

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

Space queries

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:

Listing D.22: Space - time refinement query in the integrated approach

```
CREATE TABLE result AS
(SELECT t.GEOM AS GEOMETRY, t.TIME AS TIME
FROM validation t, queries q
WHERE (q.ID = 3 AND
SDO_ANYINTERACT(t.GEOM, q.GEOMETRY) = 'TRUE'));
```

Space - time queries

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

Listing D.23: Space - time refinement query in the integrated approach

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


BIBLIOGRAPHY

- Abdullah, A., Rahman, A., and Vojinovic, Z. (2009). LiDAR filtering algorithms for urban flood application: Review on current algorithms and filters test. *Laserscanning* 9, 38:30–36.
- Abel, D. and Smith, J. (1983). A data structure and algorithm based on a linear key for a rectangle retrieval problem. *Computer Vision, Graphics, and Image Processing*, 24(1):1 – 13.
- Abel, D. J. and Mark, D. M. (1990). A comparative analysis of some two-dimensional orderings. *International Journal of Geographical Information System*, 4(1):21–31.
- AHN (2016). <http://www.ahn.nl/> Actueel Hoogtebestand Nederland (AHN).
- ASPRS (2011). LAS specification, version 1.4 R13. Technical report, The American Society for Photogrammetry & Remote Sensing. The LAS specification.
- Bayer, R. and McCreight, E. M. (1972). Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189.
- Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517.
- Brinkhoff, T., Kriegel, H.-P., and Schneider, R. (1993). Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 40–49, Washington, DC, USA. IEEE Computer Society.
- Carter, J., Schmid, K., Waters, K., Betzhold, L., Hadley, B., Mataosky, R., and Halleran, J. (2012). Lidar 101: An introduction to lidar technology, data, and applications. *National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center. Charleston, SC*.
- Comer, D. (1979). Ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137.
- Cura, R., Perret, J., and Paparoditis, N. (2015). Point cloud server (PCS): Point clouds in-base management and processing. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1:531–539.
- Dobos, L., Csabai, I., Szalai-Gindl, J. M., Budavári, T., and Szalay, A. S. (2014). Point cloud databases. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, page 33. ACM.
- Elmasri, R. and Navathe, S. (2010). *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 6th edition.
- Faloutsos, C. and Roseman, S. (1989). Fractals for secondary key retrieval. In *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 247–252. ACM.

- 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*, pages 291–299. IEEE.
- Gaede, V. and Günther, O. (1998). Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231.
- Gargantini, I. (1982). An effective way to represent quadrees. *Commun. ACM*, 25(12):905–910.
- Godfrind, A. and Horhammer, M. (2015). Oracle support options for point clouds. In *Management of massive point cloud data: wet and dry* (2).
- Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57.
- Haala, N. and Kada, M. (2010). An update on automatic 3D building reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(6):570–580.
- Hilbert, D. (1891). Ueber die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen*, 38(3):459–460.
- 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*, volume 2006, page 25th.
- Hug, C., Krzystek, P., and Fuchs, W. (2004). Advanced LIDAR data processing with LAStools. In *XXth ISPRS Congress*, pages 12–23.
- Isenburg, M. (2012a). LAStools: Efficient tools for LiDAR processing. Available at: <http://www.cs.unc.edu/~isenburg/lastools/> [Accessed October 9, 2012].
- Isenburg, M. (2012b). Laszip: lossless compression of lidar data. unc.[online] 2014.
- ISO (2008). *Geographic information — Temporal schema*. International Organization for Standardization.
- ISO (2014). *Geographic information – Reference model – Part 1: Fundamentals*. International Organization for Standardization.
- Khoshelham, K. (2011). Accuracy analysis of kinect depth data. In *ISPRS workshop laser scanning*, volume 38, page W12.
- Kothuri, R., Godfrind, A., and Beinat, E. (2007). *Pro Oracle Spatial for Oracle Database 11g*. Springer Science Business Media.
- Langran, G. (1992). Time in geographic information systems, technical issues in geographic information systems.
- Langran, G. and Chrisman, N. R. (1988). A framework for temporal geographic information. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 25(3):1–14.
- Lawder, J. (2000). The application of space-filling curves to the storage and retrieval of multi-dimensional data.

- Lawder, J. K. and King, P. J. (2000). Using space-filling curves for multi-dimensional indexing. In *Advances in Databases*, pages 20–35. Springer.
- Lawder, J. K. and King, P. J. H. (2001). Querying multi-dimensional data indexed using the hilbert space-filling curve. *ACM Sigmod Record*, 30(1):19–24.
- 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):11–18.
- Mokbel, M. F., Ghanem, T. M., and Aref, W. G. (2003). Spatio-temporal access methods. *IEEE Data Engineering*, 26(2):40–49.
- Nascimento, M. A. and Silva, J. R. O. (1998). Towards historical r-trees. In *Proceedings of the 1998 ACM symposium on Applied Computing SAC*. Association for Computing Machinery (ACM).
- Nguyen-Dinh, L.-V., Aref, W. G., and Mokbel, M. (2010). Spatio-temporal access methods: Part 2 (2003-2010).
- Nievergelt, J., Hinterberger, H., and Sevcik, K. C. (1984). The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71.
- Oracle (2015). *Oracle Exadata Database Machine X5-2*.
- Orenstein, J. A. (1989). Redundancy in spatial databases. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD '89, pages 295–305, New York, NY, USA. ACM.
- Orenstein, J. A. and Merrett, T. H. (1984). A class of data structures for associative searching. In *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '84, pages 181–190, New York, NY, USA. ACM.
- Otepka, J., Mandlbürger, G., and Karel, W. (2012). The OPALS data manager—efficient data management for processing large airborne laser scanning projects. *Proceedings of the ISPRS Annals of the Photogrammetry, Melbourne, Australia*, 25:153–159.
- Ott, M. (2012). Towards storing point clouds in PostgreSQL. *HSR Hochschule für Technik Rapperswil, Rapperswil, Switzerland*.
- Peano, G. (1890). Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160.
- Pelekis, N., Theodoulidis, B., Kopanakis, I., and Theodoridis, Y. (2004). Literature review of spatio-temporal database models. *The Knowledge Engineering Review*, 19(03):235–274.
- Peters, R., Ledoux, H., and Biljecki, F. (2015). Visibility analysis in a point cloud based on the medial axis transform. In *Europhysics Workshop on Urban Data Modelling and Visualisation, Delft (The Netherlands), Nov. 23th, authors version*.

- Peuquet, D. J. and Duan, N. (1995). An event-based spatiotemporal data model (estdm) for temporal analysis of geographical data. *International Journal of Geographical Information Systems*, 9(1):7–24.
- Pot, R. (2011). System description noord-holland coast. Technical report.
- Previtali, M., Barazzetti, L., Brumana, R., and Scaioni, M. (2014). Towards automatic indoor reconstruction of cluttered building rooms from point clouds. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1:281–288.
- Psomadaki, S., van Oosterom, P. J. M., Tijssen, T. P. M., and Baart, F. (2016). Using a space filling curve approach for the management of dynamic point clouds. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W1:107–118.
- 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 http://www.cigi.illinois.edu/cybergis/docs/Kazar_Position_Paper.pdf.
- Richter, R. and Döllner, J. (2014). Concepts and techniques for integration, analysis and visualization of massive 3D point clouds. *Computers, Environment and Urban Systems*, 45:114–124.
- Robinson, J. T. (1981). The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, SIGMOD '81*, pages 10–18, New York, NY, USA. ACM.
- Rusu, R. B. and Cousins, S. (2011). 3D is here: Point cloud library (PCL). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE.
- Sabo, N., Beaulieu, A., Bélanger, D., Belzile, Y., and Piché, B. (2014). *The GeoHashTree: a multi-resolution data structure for the management of point clouds*. Natural Resources Canada/Ressources naturelles Canada.
- Samet, H. (1990). *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Schops, T., Sattler, T., Hane, C., and Pollefeys, M. (2015). 3D modeling on the go: Interactive 3D reconstruction of large-scale scenes on mobile devices. In *3D Vision (3DV), 2015 International Conference on*, pages 291–299. IEEE.
- Sistermanns, P. and Nieuwenhuis, O. (2004). Holland Coast (the Netherlands). *EUROSION Case Study*. DHV group.
- Terry, J., Stantic, B., Terenziani, P., and Sattar, A. (2011). Variable granularity space filling curve for indexing multidimensional data. In *Proceedings of the 15th International Conference on Advances in Databases and Information Systems, ADBIS'11*, pages 111–124, Berlin, Heidelberg. Springer-Verlag.
- 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).

- Tropf, H. and Herzog, H. (1981). Multidimensional Range Search in Dynamically Balanced Trees. *Angewandte Informatik*, (2):71–77.
- 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*, 49:92–125.
- 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*.
- Vazirgiannis, M., Theodoridis, Y., and Sellis, T. (1998). Spatio-temporal composition and indexing for large multimedia applications. *Multimedia Systems*, 6(4):284–298.
- Westoby, M., Brasington, J., Glasser, N., Hambrey, M., and Reynolds, J. (2012). ‘Structure-from-Motion’ photogrammetry: A low-cost, effective tool for geoscience applications. *Geomorphology*, 179:300–314.
- White, J. C., Wulder, M. A., Vastaranta, M., Coops, N. C., Pitt, D., and Woods, M. (2013). The utility of image-based point clouds for forest inventory: A comparison with airborne laser scanning. *Forests*, 4(3):518–536.
- Wijga-Hoefsloot, M. (2012). Point clouds in a database: Data management within an engineering company. Master’s thesis, TU Delft, Delft University of Technology.
- Xu, X., Han, J., and Lu, W. (1990). R-tree: an improved r-tree index structure for spatiotemporal databases. In *Proc. 4th Int’l. Symp. on Spatial Data Handling*.
- Zlatanova, S. (2006). 3D geometries in spatial DBMS. In *Innovations in 3D geo information systems*, pages 1–14. Springer.

E | REFLECTION

This graduation research officially started in November 2015 and it took more than the usually required 8 months. The reason for this delay was the fact that I decided that the topic needed more extensive research that was not possible to complete in the official graduation period. I also took some time to submit a paper for the 3D Geoinfo conference which took place in October 2016 in Athens, Greece. The conference was a good opportunity to present this work to the academic community and I consider myself lucky that my supervisors (both at the TU Delft and the company Deltares that initiated the thesis) agreed with my given choice. In the course of the extended period, however, I did more experiments than the initially considered one's and those gave even more insight to the overall topic of managing dynamic point clouds. An example of this extended work was the cooperation with a visiting researcher Xuefeng Guan who was dealing with rather similar research. In his work, time was replaced by the [LoD](#). During the summer, I actually used his implemented code and run some preliminary tests for some parts of the future work.

In the beginning of the thesis, the research questions also included the studying of aspects (like blocking with [SFCs](#)) that in the course of time were proven to be more software oriented and less research-focused and, for this reason, those aspects were not studied at the end. The research questions were thus adapted.

The main aim of this Master thesis was to research whether a [SFC](#) approach is an appropriate method for integrating the space and time components of point cloud datasets. In other words, the research required the implementation of a prototype and the execution of experiments and benchmarks to be able to make an informed conclusion. From those it is corroborated that the [SFC](#) approach is an appropriate method for managing dynamic point clouds. Actually the best approach concerns an equal treatment of the spatial and time dimensions in the [SFC](#). The method is scalable and can be used for a wide variety of use cases. The user only needs to define the appropriate encoding of space and time and, scaling of time for the needed use cases.

From the website of the master program, one learns that: *"The science Geomatics is concerned with the acquisition, analysis, management and visualisation of geographic data with the aim of gaining knowledge and a better understanding of the built and natural environments."* This thesis deals with the management part of the program and uses a wide variety of terms introduced in many of the courses: space filling curves and spatial clustering, quadrees, octrees, spatial indexing techniques, databases, point clouds, programming etc. The thesis is as a result very technical and in line with the program. In addition to that, it is also pure research with the results discussed in [Chapter 5](#).

The research conducted in this document is directly applicable to the field of geomatics. It actually was a continuation of a project "Massive Point Clouds for eSciences" where many companies were involved. The problem of managing dynamic point clouds is a real-world problem and the

thesis was initiated by Deltares. The company possesses many point cloud datasets that are used for the monitoring of the Dutch coast and efficient management of those datasets is an active problem. Of course, the project being a Master thesis does not completely solve the problem, but a first step is taken and directions for future work are given in [Section 7.2](#).

The relationship between the project and the wider social context can be seen by the nature of the datasets used and the cooperation with Deltares. The Netherlands is a country where the protection of the coast has played a very important role for its development. Almost one quarter of the land lies below sea level. Typical flood protection structures include dams, dikes and dunes. The Dutch coastal policy states that the country should "hold the line" meaning that the coastline should be prevented from moving towards the mainland. For this reason, the Ministry of Transport, Public Works, and Water Management (Rijkswaterstaat) as part of the coastal monitoring guidelines performs a yearly survey of the Dutch coast in order to determine changes in coastal elevations. Another important project is the Sand Engine. The Sand Engine was created by depositing 21 million cubic meters of sand between the areas Ter Heijde and Kijkduin. The purpose of this pilot program is to investigate how nature spreads this amount of sand along the coast as the years go by. In order to see if the experiment is developing as thought, a monitoring of the area at irregular moments in time (after storms) takes place. Point clouds have evolved to be a very important source of information for coastal applications. The usefulness of point cloud data lies in the fact that point cloud acquisition techniques have become highly accurate and quick, allowing daily or even hourly collection of data. This plethora of data leads to dynamic point clouds that need to be efficiently managed. By studying the available solutions, I realised that current solutions offered by commercial and open-source databases are not enough for handling this sort of information. Therefore, I implemented a different solution that provides contribution both in the academic community and the society.

COLOPHON

This document was typeset using \LaTeX . The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.

