Point Clouds in a Database

Data Management within an Engineering Company

Master Thesis - M.E. Wijga-Hoefsloot

Professor:
- prof.dr.ir. P.J.M. van Oosterom
Supervisors:
- dr.dipl.-ing. S. Zlatanova
- M.P. Kodde MSc
Co-reader:
- dr.ir. B.G.H. Gorte

Master Geomatics
Department GIS-Technology
OTB Research Institute for the Build Environment
Faculty of Architecture
Delft University of Technology

Geo-Information Services
Leidschendam
Fugro GeoServices B.V.

27 September 2012

MSc Geomatics
Faculty of Architecture
Delft University of Technology

M.E. Wijga-Hoefsloot
1007157

Weidedreef 121
2727 EC Zoetermeer
The Netherlands

Preface

This thesis is the result of my graduation assignment at the Delft University of Technology. The research associated with this thesis was initiated by the research department GIS-Technology (GISt, GDMC) of the OTB Research Institute for the Build Environment, which is part of the Delft University of Technology. The research was facilitated by the Geo-Information Services division of Fugro GeoServices B.V. in Leidschendam. The subject was chosen because it meets the interests of both GISt and Fugro.

With laser scanning (including laser altimetry and multi-beam echo sounding), many data points, called point clouds, are measured, but there is a problem to handle them efficiently. This problem is recognized by GISt and Fugro. GISt is constantly doing research to all kinds of spatial data in a database. Fugro applies laser altimetry to map corridors and laser scanning to capture the shape of buildings and plants. As one of the contractors, Fugro is working on the new actual heights model (AHN-2) of The Netherlands; this yields about 400 billion data points. The resulting data is stored in a file-based environment; and distributed by sending the data on an external hard disk. It is difficult to perform queries and analysis tasks in a file-based environment. Therefore, another method is needed to store all data. This will be the research-case. Introducing a spatial Database Management System (DBMS) will make it easier to perform query and analysis. Some research is done on storing point clouds in a database [Wang and Shan, 2005 and Schön et. all, 2009], but the resulting database has a closed structure, point clouds are stored as blobs and queries are required for just viewing the data. The aim of this research is to design a data model that is suitable for managing point clouds in an efficient way in a spatial Database Management System (DBMS) that it is generally accessible by spatial applications. This thesis includes a benchmark for managing points with different parameters in an Oracle database.

Readers of this report, who are just interested in the conclusions, are referred to Chapter 6 where the conclusions and recommendations for further research are described. I would also refer them to the summary of this thesis. A summary in Dutch (Samenvatting) is added to this report as well.

Of course, I would like to thank some people who assisted me in carrying out this research. At Delft University of Technology, I thank my graduation professor Peter van Oosterom and my supervisor Sisi Zlatanova for their guidance, steering and useful input and comments, and Theo Tijssen for his Oracle-support. At Fugro, I would like to thank my supervisors John Rongen and Martin Kodde, for the conversations and discussions about the subject, and for their confidence and helping me further; everyone who made it possible to perform my thesis at Fugro GeoServices; and the whole Geo-Information Services division for their support and good-fellowship. Also, I would like to thank everybody for their patience.

I would like to give special thanks to my husband, my little girls, my parents, my sisters, Andy and Yvonne, other relatives, and friends, for their encouragement, support, and advice and for just being there when needed.

Thank you.

Zoetermeer, September 2012
Martine Wijga-Hoefsloot

Summary

Introduction
With laser scanning (including laser altimetry and multi-beam echo sounding), many data points, called point clouds, are measured. The interest in point clouds is increasing. Depending on the laser scanner, the measurement set-up and the technique, the number of points in one dataset can vary from a few hundred points to over a billion points. The data volume of a laser survey mission can easily reach Gigabyte or even Terabyte level. It is a problem to handle these voluminous datasets efficiently. One way to manage the data is to partition the data with tiles or grids and then store the data in each tile or grid as a single file in text or binary format; the large volume of data is divided into separate files of a reasonable size. As an alternative, the data can be stored in a database management system (DBMS); each point in a single record based on standard data types. Since the data stored can be accessed in single point level, it is easy to perform queries and analysis on the data server, and only the qualified points are returned to the user. Since the data is spatial data, it has a location; data can be stored based on a spatial data type in a spatial DBMS, spatial objects can be clustered and spatially indexed to limit searches, with the result that spatial queries can be performed.

Storing and retrieving massive datasets as single point records (one point per record) results in storing and retrieving lots of records, but fast retrieval become questionable, as there is overhead per record and because of the number of records to be retrieved. As Oracle is the market leader in the development of DBMS, this research is limited primarily to Oracle. This leads directly to the research question of this thesis: *What is the best design for a data model to store large point clouds in an Oracle DBMS, such that it is generally accessible by spatial applications, that all attributes are preserved and that performance is optimised?*

Analysis
A solution has been found by reducing the number of records by clustering nearby points without loss of information. If points are clustered, associated attributes have to be clustered as well. It has been decided how to record these attributes with the points. The Region Quadtree is implemented to grid the data in logical parts. The Morton space-filling curve is implemented to cluster the data. Hilbert space-filling curve would even be better.

The points have been stored using a well-known data type called SDO_GEOMETRY (GTYPE 3005, POINTCLUSTER), SDO_GEOMETRY is a standard spatial data type implemented in Oracle. Attributes have been stored in one or more arrays (VARRAY), because ordinates are also stored in a VARRAY. This method is been compared to the newer SDO_PC data type.

Conclusion
The presented method using SDO_GEOMETRY (GTYPE 3005, POINTCLUSTER) performs optimal for large point cloud datasets in terms of performance and the points can generally be accessed by spatial applications. The alternative method using SDO_PC is not suitable for the conditions as stipulated in the research question. Because the SDO_PC data type is newly implemented in Oracle, not generally readable by spatial applications, not (yet) fully supported by all vendors and required storage space is double the amount.

Samenvatting (in Dutch)

Inleiding
Met laser scanning (inclusief laser altimetrie en multi-beam echo sounding) worden grote hoeveelheden data punten gemeten, deze data punten worden punten wolken genoemd. De interesse in en het gebruik van punten wolken groeit. Afhankelijk van de laser scanner en de meetopzet kan de ingewonnen dataset variëren van enkele honderden tot miljarden punten. Eén project kan zondermeer, meer dan een GB en zelfs TB aan data bevatten. Het lijkt een probleem om deze grote datasets te verwerken. Eén manier is om de dataset op te splitsen in blokken en deze in een tekst- of binair formaat op te slaan. Een andere manier is om de punten op te slaan in een database. Ieder punt in een record, gebaseerd op standaard data types. Punten opgeslagen in een database zijn eenvoudig te analyseren en te bevragen en alleen de geselecteerde punten worden naar de gebruiker gestuurd. Als de punten met een ruimtelijk data type zijn opgeslagen, hebben ze een locatie. Ruimtelijk opgeslagen punten kunnen worden geclusterd en geïndexeerd met een ruimtelijke index om het ruimtelijk zoeken te versnellen.

Het opslaan en bevragen van grote hoeveelheden data punten (met één punt per record) resulteert in het opslaan en ontvangen van grote hoeveelheden records. En ook al is de data geïndexeerd, het ontvangen van een antwoord kost tijd, enerzijds omdat per record ook veel overhead wordt opgeslagen, anderzijds omdat er heel veel records worden geselecteerd. Omdat Oracle marktleider is, is dit onderzoek alleen gericht op Oracle. Dit alles leidt tot de volgende onderzoeksvraag: *Wat is het beste ontwerp van een data model om grote hoeveelheden punten wolk data in een Oracle database op te slaan, waarbij het mogelijk moet zijn dat de opgeslagen data punten eenduidig gelezen kunnen worden door bestaande software pakketten, dat alle attributen beschikbaar blijven en dat de performance optimaal is?*

Analyse
Een oplossing is gevonden door de punten te groeperen die bij elkaar in de buurt liggen, zonder informatie verlies. Om dit te bewerkstelligen is een manier gevonden om ook de attributen te groeperen. De Region Quadtree is geïmplementeerd om de punten te groeperen in logische delen. De Morton space-filling curve is geïmplementeerd om de data in de DBMS te clusteren. De Hilbert space-filling curve zou nog beter zijn.

De punten worden in groepen opgeslagen gebruikmakend van het standaard data type voor ruimtelijke data in Oracle, SDO_GEOMETRY (GTYPE 3005, POINTCLUSTER). Attributen worden dan opgeslagen in één of meerdere arrays (VARRAY). Deze methode is vergeleken met het nieuwere data type SDO_PC dat door Oracle is ontworpen om punten wolken op te slaan.

Conclusie
De methode die gebruik maakt van SDO_GEOMETRY (GTYPE 3005, POINTCLUSTER) heeft een optimale performance voor het opslaan van grote hoeveelheden punten en de data punten kunnen worden gelezen door bestaande software pakketten. De methode die gebruik maakt van het data type SDO_PC voldoet niet aan de gestelde eisen. Omdat het nieuwe SDO_PC data type niet algemeen leesbaar in bestaande software pakketten, (nog) niet volledig wordt ondersteund door leveranciers van gespecialiseerde software pakketten en benodigde opslagruimte dubbele hoeveelheid is.

Table of Contents

1. Introduction

This introductory chapter gives an overview of the research in this Master Thesis. The first two paragraphs describe the motivation and background (§1.1) and the problem formulation and the research objectives (§1.2). A short overview of related research is in §1.3. The chapter ends with the research methodology in the fourth paragraph (§1.4) and a thesis outline (§1.5).

1.1 Motivation and Background

This thesis is the result of my graduation assignment at the Delft University of Technology. The research associated with this thesis was initiated by the research department GIS-Technology (GISt, GDMC) of the OTB Research Institute for the Build Environment, which is part of the Delft University of Technology. The research was facilitated by the Geo-Information Services division of Fugro GeoServices B.V. in Leidschendam.

With laser scanning (including laser altimetry and multi-beam echo sounding), many data points, called point clouds, are measured. The interest in point clouds is increasing, an important reason for this increasing interest is the problem of rapidly growing datasets due to the evolution of sensor techniques, like laser (from the air, on the road and on the surface) and multi-beam echo sounding (under water). Depending on the laser scanner, the measurement set-up and the technique, the number of points in one dataset can vary from a few hundred points to over a billion points. It is a problem to handle these voluminous datasets efficiently. This problem is recognized by GISt and Fugro. GISt is constantly doing research to all kinds of spatial data. Fugro applies laser altimetry to map corridors and laser scanning to capture the shape of buildings and plants. As one of the contractors, Fugro is working on the new actual heights model (AHN-2, [URL, AHN]) of The Netherlands; this will yield about 400 billion data points.

1.2 Problem Formulation and Research Objectives

The data volume of a laser survey mission can easily reach Gigabyte or even Terabyte level. It is challenging to manage and work with such high volume data. One way to manage the data is to partition the data with tiles or grids and then store the data in each tile or grid as a single file in text or binary format; the large volume of data is divided into separate files of a reasonable size. For example, the grids can be stored as georeferenced images and published on the internet. However, it is difficult to perform query and analysis tasks, especially for large working areas. For example, if a user wants access to all data points that are below sea level, all data in the query area have to be downloaded from the data server then imported into other software to conduct the query. When the working area is large, the size of the data that has to be downloaded can be unmanageable. As an alternative, the data can be stored in a database management system (DBMS); each point in a single record based on standard data types. Since the data stored can be accessed in single point level, it is easy to formulate these kinds of queries and analysis on the data server, and only the qualified points are returned to the user. Since the data is spatial data, it has a location; data can be stored using a spatial data type in a spatial DBMS, with the result that spatial queries can be performed too. Spatial queries can also be mixed with alphanumeric queries to select the required spatial data.

One important property of a DBMS is that data is generally readable by multiple applications. This is a highly preferable property since many different applications are used in

daily practice of an engineering company. In addition, other research towards storage procedures of point clouds, such as that done by Wang and Shan [2005] do not allow for general access using standardised geometry types.

Another property of a DBMS is that besides geometry, room is available to store thematic (alphanumeric) attributes belonging to the spatial data. Normally, attributes are properties such as street names in the case of a road database, or postal codes in the case of a city map. However, point clouds do also feature technical attributes such as colour, intensity, classification and timestamp.

Several spatial DBMSs are available on the market, a lot of them are investigated; but only little R&D is done on managing point clouds with attributes. Only Oracle has developed a new data type that is designed for managing point clouds. As Oracle is the market leader in the development of DBMS, this research is limited to Oracle. Only where necessary, references are made to other database systems.

This leads directly to the research question of this thesis:

What is the best design for a data model to store large point clouds in an Oracle DBMS, such that it is generally accessible by spatial applications, that all attributes are preserved and that performance is optimised?

The performance is of a data model is optimal when data can be stored efficiently and retrieved fast.

The definition of the research problem raises several sub-questions:

1. What is the content of a point cloud dataset? How big is a massive point cloud dataset? (§ 2.2)
2. What is a spatial DBMS? Why is a spatial DBMS needed? Which spatial DBMSs are available? (§ 3.2)
3. What is a (spatial) data type? Which spatial data types are available? Which available spatial data types are suitable for storing point clouds? (§ 3.6)
4. What (number of) attributes are available and useful for users and need to be implemented and tested? (§ 3.7)
5. How can the points be organized? Separate or in clusters, what number of points should be the cluster size? Which methods are suitable for defining the clusters? (chapter 5)
6. How much storage capacity is required for the resulting spatial DBMS? (chapter 5)

Some questions will be answered by studying literature. Other questions will be answered based on implementing and testing the designed method in an Oracle DBMS.

Besides answering the aforementioned questions, another result of this research will be a data model with some configuration parameters that prescribes how to manage massive point clouds in a spatial DBMS. Explicitly defined as out-of-scope are the aspects related to client-server communication. In the test environment client and server are on the same machine, so that client-server communication (or the speed of the Internet) has no influence on the test results. Besides this, laser data is in most applications static, so the temporal aspect will also be out of scope.

1.3 Prior Research

Höfle [2005] presented an information system called “Information System LISA” (LIdar Service Analyses). LISA is built upon the open-source DBMS PostgreSQL with its spatial add-on PostGIS. In LISA, points of a point cloud are stored with one point per record. Liu et. all [2008] proposed a faster database approach to very large lidar data management by introducing the KD tree as an index solution. only for displaying purposes, also based on one point per record.

Storing and retrieving massive datasets as single point records (one point per record) results in storing and retrieving lots of records and efficient storage and fast retrieval become questionable. Because of the overhead stored per record and because of the large number of records to be retrieved.

To solve the problems with single point records, Wang and Shan [2005] proposed to apply the space partition mechanism to a relational DBMS for effective data management; i.e, the data is stored in a spatially indexed relational DBMS. First, the space in which data is embedded is partitioned into a number of 3D cells. The extension of the 3D cells varies according to the data density, such that the number of points in each cell is as even as possible or close to a predefined target value. Next, the 3D cells are ordered in space based on the principle of Hilbert space-filling curve. In this way, cells with adjacent sequential numbers will also be adjacent in space. In order of the Hilbert codes, the points within a 3D cell are then stored into the DBMS as a blob. For spatial query, a hierarchical strategy is applied that initially determines the large cells where points may possibly reside and continues to reach smaller cells when needed. In this way, both the DBMS and on-line transferred data have manageable sizes.

A Hilbert curve (also known as a Hilbert space-filling curve) is a continuous fractal space-filling curve first described by the German mathematician Hilbert [1891], as a variant of the space-filling curve discovered by Peano [1890]. The solution of Wang and Shan might be good, but it has a large shortcoming, as blobs can not be read and understood outside the DBMS. This makes it impossible to visualize, query, manipulate or process the data with external applications. Therefore, their solution does not form an answer to the research question of this thesis.

Oracle has developed a new data type SDO_PC that is designed for managing point clouds. This data type is available since 2008 in Oracle 11g and later. This data type has a disadvantage: point clouds stored as SDO_PC can not readily be analyzed or processed by other applications, because this data type is not yet recognized by external applications, but this might change over time. This data type will be part of this research.

According to Ott [2012], Microsoft’s SQLServer and PostgreSQL failed to provide a dedicated point cloud data type. He proposes to split a point cloud into a set of different blocks and store these blocks and point clouds in a PostgreSQL DBMS similar to Oracle’s SDO_PC solution.

In a publication as a result of the seminar “Management of massive point cloud data: wet and dry” [Oosterom et. all (editors), 2010], Ledoux [pp. 45] proposes a new “Triangle” data structure to store both points and TINs in a spatial database. This data structure contains a straight table (one table with only a fixed number of attributes), where the use of a separate spatial index is not necessary, because one can rely on the topological relationships between the triangles to perform queries. Kodde [pp. 12] presents the use of a point cloud data model in daily practice, the LAS format and the need of a “Web Point Cloud Service” to distribute point clouds to end-users. The LAS format is explained in § 2.3.

1.4 Methodology

The first part of this research consists of a study of literature, followed by a data model design.

To overcome the shortcoming in the solution of Wang and Shan, the points within a 3D cell can be stored in a spatial DBMS as a multi-geometry based on a spatial data type. This has the same advantages as the solution of Wang and Shan, except for the fact that storing the points within a 3D cell as a multi-geometry based on a spatial data type can be read and understood outside the DBMS. This method will be developed and tested in this thesis. The correctness of the results will be checked.

To test the performance of accessing the data in the designed data model, a number of special queries will be identified in this thesis. These special queries contain a selection of a subset of the data. Measuring the speed of the query is not always easy, so larger query windows will be used to give practical results.

As of Oracle 11g, the new SDO_PC data type has become available for handling point clouds. The results from the special query will be compared with similar tests, while the data is stored using this data type.

1.5 Thesis Outline

This report is subdivided into six chapters, which can be ordered into three research phases.

I Introduction
Chapter 2 explains the basics of laser scanning and gives a description of the measurements and the resulting datasets. This results in an answer to the second research question. Chapter 3 goes deeper into the use of databases to store spatial data. Topics will be discussed on the evolution of spatial DBMSs, spatial data access methods, DBMS standards, spatial support in three mainstream DBMSs with Oracle in detail and the use of technical attributes. In this phase, the first four questions will be answered.

II Implementation
In [Oosterom et. all, 2010], Kodde addressed difficulties in daily practice. To be able to conquer these difficulties, a data model is developed. To assess this data model, a special query type is defined that is required for data processing, the execution time taken for these queries should be as short as possible to meet the needs of the users. Chapter 4 deals with the conceptual design of the data model and describes the test procedure. With the tests, the advantages and disadvantages of the data model should become clear. A major part of this research consists of implementing and testing the data model and some parameters recording the data type and group size of point clouds, in chapter 5.

III Conclusion
In Chapter 6, the conclusions and some recommendations for future research can be found. Readers of this report, who are also interested in the SQL codes developed for this research, are referred to Appendix A. Appendix B lists more detailed test results.

2. Data Acquisition

This chapter gives an overview of the data and the acquisition of it. § 2.1 explains three measurement techniques, followed by the post-processing. The combination of the measurement set-up, and the technique used, results in a point cloud dataset, containing many points and possibly other properties of points (attributes). § 2.2 describes a big dataset with billions of points, a subset of which will be used for testing (Chapter 5), also three other datasets are presented. § 2.3 addresses the need of attributes attached to point clouds. In this paragraph, the LAS format will be explained.

2.1 Acquisition Techniques and Post-processing

Three acquisition techniques, based on two measurement techniques, lead to voluminous point cloud datasets. Measurements can be done with either a laser or an acoustic scanner. A laser scanner can be positioned in an airplane, a helicopter or even an unmanned aerial vehicle (UAV), in a car or on earth; these acquisition techniques are called Airborne Laser Scanning (ALS) and Terrestrial Laser Scanning (TLS). An acoustic scanner is positioned on a vessel, and the object to be scanned is frequently the seabed; this technique is called bathymetry. In addition, a more recent development is the generation of point clouds from photogrammetry using dense stereo matching.

A laser scanner measures the line-of-sight vector from the laser scanner to an object. Laser scanners are active instruments, which mean the systems use their own laser beam to measure the distance between the sensor and an object. There are two types of laser scanners, triangulation scanners and ranging (time-of-flight) scanners, see Figure 1 and Figure 2.

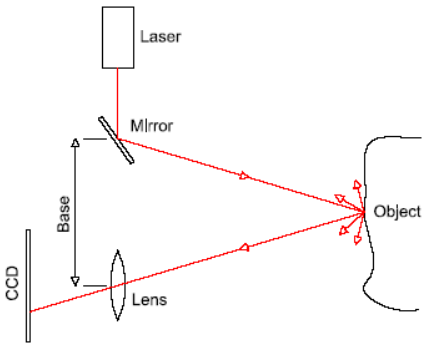


Figure 1: Triangulation principle [Böhler and Marbs, 2002].

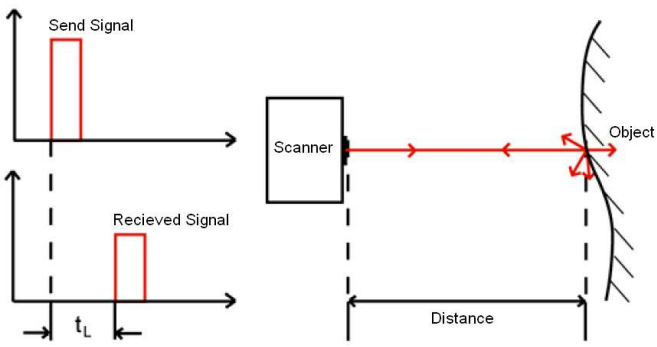


Figure 2: Time of flight of a laser pulse principle [Claassen, 2003].

Triangulation scanners [Böhler and Marbs, 2002]: This type of scanner consists of a transmitting device, sending a laser beam at a defined, incrementally changed angle from one end of a mechanical base onto the object, and a CCD camera at the other end of this base, which detects the laser spot (or line) on the object. The 3D position of the reflecting surface element can be derived from the resulting triangle. These instruments play an important role for short distances and small objects where they are much more accurate than ranging scanners.

Ranging scanners [Böhler and Marbs, 2002]: Ranging scanners measure horizontal and vertical angles. They compute the distance, either by the time-of-flight method or by comparing

the phases of the transmitted and received waveform of a modulated signal.

- **Time of flight of a laser pulse:** A laser pulse is sent to the object and the distance between transmitter and reflecting surface is computed from the travel time between signal transmission and reception.
- **Phase comparison method:** In this case, the transmitted beam is modulated by a harmonic wave and the distance is calculated using the phase difference between transmitted and received wave. Due to the more complicated signal analysis, the results may be more accurate (at the expense of the measuring rate).

A multi-beam echo sounder (MBES) measures the travel time of acoustic waves (see Figure 3). Sound energy is transmitted by a transducer travels through the column of water and is reflected by the target (seabed) back to the receiver. The depth is calculated from the measured travel time ΔT.

$$Depth = c\frac{\Delta T}{2} \tag{1.}$$

where c is the speed of sound in water.

Figure 3: Bathymetry [URL, Dive and Discover].

As the laser beam is reflected differently, depending on various colours and materials, it is possible to detect the different materials and colours in the resulting data. This reflection is called the intensity *(i)* of the signal, in case the full returned signal is recorded, this reflection is called the signature of the signal. With ALS and TLS this intensity can be recorded. The FLI-MAP and DRIVE-MAP systems of Fugro and some TLS scanners have been equipped with optical cameras, in the case of FLI-MAP with two video cameras and in the case of DRIVE-MAP with four metric cameras and one panoramic camera. These cameras provide imagery for even more detail [URL, FLI-MAP]. The colours can be measured as RGB-values, the intensity *(i)* and the RGB-values can be assigned to the points in the point cloud.

With laser altimetry (from the air and on the road) and bathymetry, the 3D position of an object can be computed, if at the time, the measurement is taken, the precise position and orientation of the scanner is known. A “known position” is a position with respect to a coordinate system, for example WGS84. To achieve this known position the scanner system must be supported by a position and orientation system (POS). This can be realised with an integrated differential GPS (DGPS) and an inertial measurement unit (IMU or INS) (see Figure 4). Computation of 3D

positions can be refined with mounting parameters and calibration data of the scanner. Geocoding of laser altimetry (from the air and on the road) and bathymetry measurements requires an exact time synchronisation of all systems: DGPS, IMU, and laser scanner. This results in an overview of post-processing steps in Figure 5.

With TLS, the position and orientation of the scanner is unknown. In order to determine all coordinates of a building, an object or a situation, laser scans are carried out from various positions around the object. Geocoding of measurements requires the coordinates of several reference-points per scan, known in two or more reference systems. After geocoding the measurements, all data is combined into one point cloud.

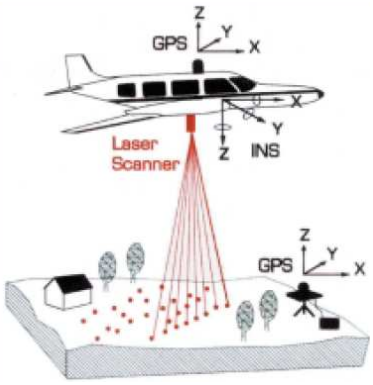


Figure 4: Position and orientation of an airplane [Lemmens and Lohani, 2001].

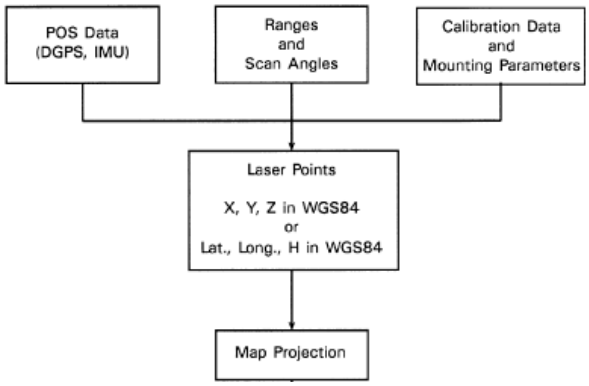


Figure 5: Processing steps for ALS and bathymetry [Wehr and Lohr, 1999].

It is possible to work out information directly after the measurements or to work out information at a later or postponed date. Especially in the case of buildings, information is not needed until restoration work takes place, or until buildings begin to fall into disrepair. In these cases, the data can be retrieved at that time and processed further into completed information.

2.2 Examples of Datasets

The first version of the Actual Height Model (AHN-1) [URL, AHN] of the Netherlands is a detailed elevation model of the whole country measured with an Airborne Laser Scanner. Although the Netherlands is perceived to be relatively flat, this makes high quality elevation data even more important. The resulting dataset has various and divergent applications. For instance, it can be used for town planning, for the design of computer games and as input in a flight simulator. Acquisition of this dataset, which has cost millions of Dutch Guilders, was initially meant to counter the demand for detailed information about elevation, from water boards, provinces and the national government. These institutions require this information for the management of coastal defences, dykes, polders and even higher-level areas in The Netherlands, which seem to be drying out. Also for the construction of infra-structural works, the dataset is of great importance. The elevation information can be used in models that simulate the discharge of the rivers, or that simulate what would happen when a dyke breach happens somewhere. In addition, one can calculate to what level the various high rise flats are subjected to noise pollution caused by the roads, and how this noise can be best reduced by erecting baffle boards (noise-reducing screens or -boards). The accuracy of the dataset depends strongly on the amount of vegetation in and topography of the area. The accuracy of the data for solid topography (such as roads and parking lots) as well as for flat or soft topography (such as beaches and grass-fields)

has a standard deviation of 15 cm maximum with a nominal error of 5 cm maximum, with a minimum point density of one point per 16 square metres. In wooded areas, this degree of accuracy is not feasible, here a minimum point density of one point per 36 square metres is achieved. For this topography a standard deviation of 20 cm maximum and a nominal error of 10 cm maximum is accepted. The data for AHN-1 is delivered in AHN Standard Units of 5 km × 6.25 km in several file formats, such as the ASCII-format (textual file format).

As one of the contractors, Fugro is working on a new version of the AHN, called AHN-2. AHN is owned by water boards and the national government (Rijkswaterstaat). The AHN-2 has a higher density then the old AHN-1 dataset. Because of the higher density and the use of new techniques, filtering for vegetation is much better. This will result in a more accurate (< 5cm) surface model. Besides the laser data, high resolution, forward and downward looking photo and video data has been captured. The data of the photos will be delivered as a GeoTIFF (georeferenced tiff-file, *.tif with *.tfw). The Netherlands has an area of 41.543 km² [URL, Indexmundi], when scanned with a density of at least 10 points per square meter, this will result in a dataset of at least 415 billion points, an AHN Standard Unit will contain 312.5 million points. The AHN-2 will be delivered in subunits of 125 hectare of 1 km × 1.25 km. AHN-2 will be completed in 2013, discussions about new AHN-3 specifications have been started, this AHN-3 will contain even more points.

For testing purposes users can order a pilot dataset of the AHN-2, 10 AHN Standard Units of the area Noord-Beveland and 1 AHN Standard Unit of the city center of Middelburg, this last dataset contains 17,478,245 unfiltered points *(x,y,z)*, intensity and RGB-values are stored as TIFF-files (see Figure 6).

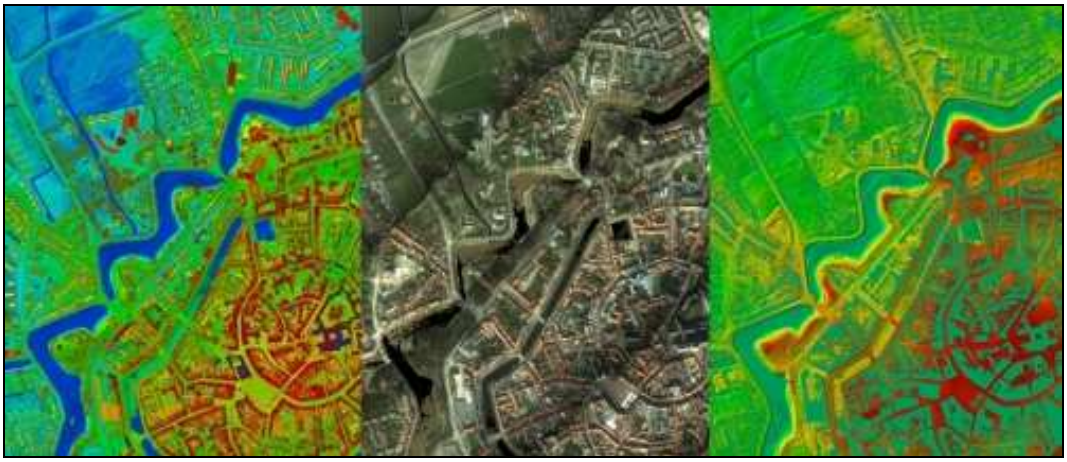


Figure 6: Pilot AHN2 (17.478.245 points) [URL, AHN].

For this research two other datasets have been obtained, from Fugro GeoServices B.V.. One dataset contains the office of a real estate agent in Breda (see Figure 7). The building has been measured with TLS from several positions, registered into a local coordinate system, and translated to the Dutch RD-NAP coordinate system. This dataset contains 1,497,848 points *(x,y,z)* as well as the intensity and RGB-values.

Figure 7: TLS of house agent (1.497.848 points).

The third dataset contains the jetties of IJmuiden. The data for these objects has been obtained by a combination of two measurement methods, ALS, and bathymetry. The resulting dataset contains 1,705,863 points *(x,y,z)* (see Figure 8); no intensity and RGB-values were measured, because part of the dataset was obtained by bathymetry, this part contains only coordinates. This dataset was utilized for determining the size of the jetties above and below the surface of the sea.

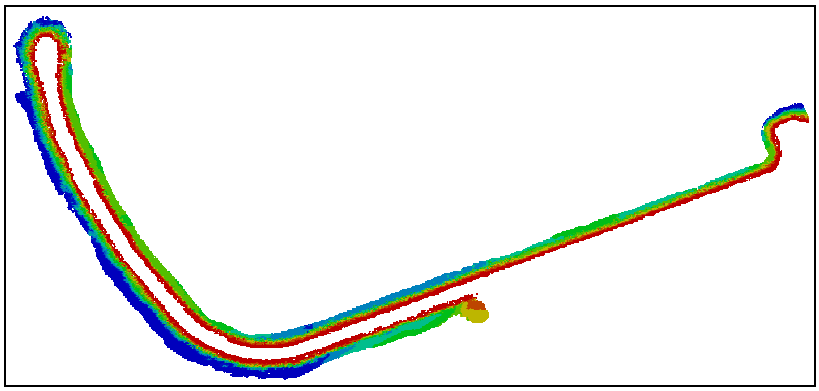


Figure 8: ALS and bathymetry of jetty (1.705.863 points).

Conclusion:
The number of points in a dataset can easily reach a billion (10^9) up to trillion (10^{12}) level. Particularly, in the case of the complete AHN-2 or city models measured with TLS.

2.3 Additional Attributes

A point cloud is a collection of individual points. These points can be stored as a simple list or as georeferenced images. But as is stated in paragraph §2.2 and in Kodde [2010], in daily practice, additional information is collected and can be attributed to these points in order to enhance processing. To be able to store these attributes with the points, a data model has to be developed that can store attributes as well as points and keep them together.

Conventions for storing attributes have been developed. The ASPRS (American Society for Photogrammetry and Remote Sensing [URL, ASPRS]) has defined a file format called LAS. The LAS file format is a public file format for the interchange of 3-dimensional point cloud data between data users. Although developed primarily for exchange of LIDAR point cloud data, this

format supports the exchange of any 3-dimensional *x,y,z* tuplet. The LAS file format is a binary file format that maintains information specific to the LIDAR nature of the data while not being overly complex. This file format has specifications for all the possible attributes that can be present in the data from Airborne Laser Scanning [ASPRS, 2012].

- X, Y, Z,
- Intensity,
- Return number and total number of returns,
- Scan direction, scan angle,
- Edge of flight line,
- Classification,
- Point source ID,
- User Data,
- R, G, B value.

Because this file-format is developed for airborne laser scanning, most fields are directly related to airborne laser scanning, the format does not fit well to other acquisition methods. To accommodate these new needs, a new version of the LAS standard [URL, ASPRS] was proposed, work on this LAS 2.0 file format was previously suspended but will be restarted in 2012. A minor update to LAS 1.4 was approved in 2011.

For mapping purposes as an example of a processing procedure in daily practice of Fugro GeoServices B.V., fields as R, G, B value, Intensity and Classification (especially ground and non-ground) are widely used. Whereas R, G, B and Intensity can be used for identification by visualisation, Classification is needed for a sub-selection on the point cloud.

For these reasons the R, G, B attributes and Intensity will be implemented for each point in this thesis. Point clouds from different classes are stored in separate tables so that a sub-selection can easily be carried out. Because the test data is not classified, all the points are accepted as being in the same class and belong to the same point cloud.

Conclusion:
The attributes Intensity, Red, Green and Blue are frequently used and will be implemented and tested.

3. Spatial Data Storage

In this chapter, theoretical issues with respect to spatial database management systems (Spatial DBMSs) will be discussed. Starting with a small historical background on DBMSs (§ 3.1), followed by an overview of the evolution of spatial DBMSs (§ 3.2). To be able to speed up searching in a spatial dataset, spatial data access methods are developed, they will be introduced and discussed in § 3.3. The chapter ends with standards for spatial DBMSs (§ 3.4) and some main (spatial) DBMS vendors (§ 3.5). Oracle and its data types for storing spatial objects are discussed in detail in § 3.5.3.

3.1 Historical Background

In the 1960s, computers became cost effective for private companies. The increasing storage capability has enabled the hosting of spatial data [Schön et. all, 2009]. The information system for storing, processing and displaying spatial data is called a geographic information system (GIS). GIS is described in several ways, in general and used in this thesis, GIS can be defined as [de By, 2001, pp. 474]:

"A software package that accommodates the capture, analysis, manipulation and presentation of georeferenced data, it is a generic tool applicable to many different types of use."

Early systems utilized a file-based system for representing spatial features and the related attribute information in binary, text, raster, and vector formats. This has advantages and disadvantages.

In the case of storing points, a file-based system is easy to use and does not have a complex structure. With help of tiling, the spatial features can be stored in separate files with a reasonable size. Users can easily select and order the tiles they need for their application, and the associated files can be downloaded on the Internet or sent by DVD or hard disk. If you would like to do a query on few data files, the data files need to be structured and sorted in the most efficient way and a program in for instance Java or C++ is easily written to answer this query. Even if the amount of data in one file is very large, by structuring the data in the most efficient way for this specific query, the answer could be derived quickly. If the working area is larger, all files in the working area have to be imported into the application to conduct the query. When the area is very large, the size of the import can be very time consuming. This has led to the introduction of a database for storing spatial features.

In the non-spatial domain, databases have been in use since the 1960s [de By, 2001, pp. 165], applications utilized in this non-spatial domain have in common that the amount of data is quite large, but that the data itself has a simple and regular structure.

3.2 Evolution of Spatial DBMSs

Based on the evolution and implementation of GIS within organizations, GIS systems have been grouped (based on Vijlbrief and Oosterom [1992] and Bentley Benelux [2006]) into four types: Self-sufficient GIS (§ 3.2.1), hybrid or crossbred GIS (§ 3.2.2), spatial middle layer systems (§ 3.2.3) and spatial DBMS (§ 3.2.4). These are described below.

3.2.1 Self-sufficient GIS

When the GIS field was introduced, the first implementations were based on existing software packages with graphic interface usually from the CAD/CAM world. In a self-sufficient GIS, storage of graphical and non-graphical data uses existing graphics file formats. Non-graphic data are only stored as allowed by the file format used. If the format is not capable, this data is (often) temporary written to a closed file format (see Figure 9 on the left). When using non-spatial data it is necessary to import this data into the system. Once the data is imported, analysis based on spatial and non-spatial data is possible. The main reason for GIS based on specific graphics systems is the functionality provided for the storage of graphics (i.e. spatial) data.

3.2.2 Hybrid or Crossbred GIS

The hybrid GIS is a widely used GIS model. It consists of two components: the graphics core and a relational database. The spatial data and maintaining the connection between spatial and non-spatial data is the job of the graphics core, while the non-spatial data is the responsibility of the database (see Figure 9 on the right). In many respects, this is a combination of the best of both worlds. For example, on the one side there is the benefit of spatial functionality and speed when working with graphics, while on the other hand, administrative data is maintained in a standard environment. The problem with the hybrid system is the link between spatial and non-spatial data. In most cases, this is the graphics reference to database tables and vice versa. This allows spatial elements to be categorized using information stored in the relational database. Compared to the self-sufficient GIS, hybrid GIS is an improvement in the position within the organization. It is possible to make an analysis of spatial and non-spatial data and the most recent version of the latter is always used. In practice, discontinuity between the two components often means that the link between spatial and non-spatial data is corrupt. Selecting objects based on spatial and non-spatial conditions is hard because the query plan for this kind of query is not optimal.

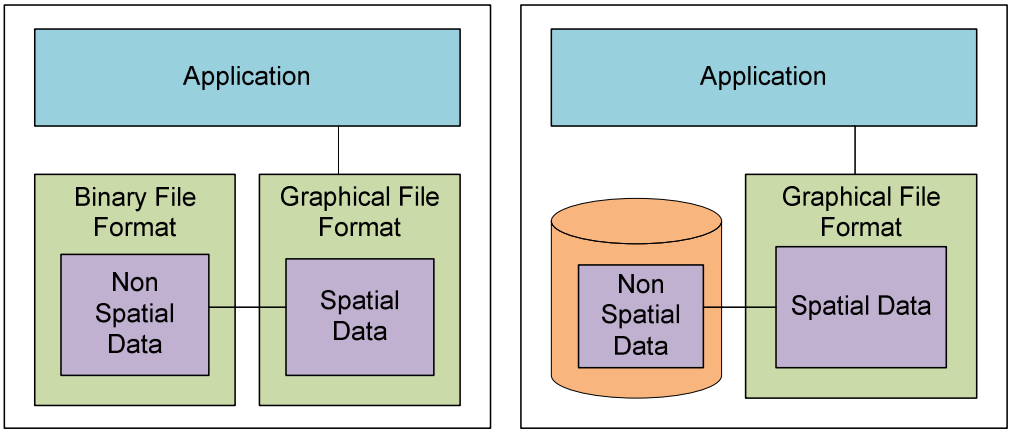


Figure 9: Self-sufficient GIS (left) and Hybrid GIS (right).

3.2.3 Spatial Middle Layer Systems

The emergence of spatial middle layer systems was the result of two developments. In the first place, there was a growing popularity of relational databases, and secondly the increasing need

for the entire organization to use spatial data. The main distinction between hybrid GIS and spatial middle layer systems is located in the storage of the data. All data, both spatial and non-spatial, is stored in the relational database. For the storage of spatial data, spatial middle layer systems are using standard data types in a relational database, mainly BLOBS (binary large objects). Only the middle layer system knows the content and structure of spatial data. Others cannot read the data, since the content and structure of the data type cannot be deduced (see Figure 10 on the left). Again, selecting objects based on spatial and non-spatial conditions is hard because the query plan for this kind of query is not optimal.

The new data type for the storage of point clouds in Oracle 11g, SDO_PC uses BLOBS too, but in the database a query is developed for extracting these BLOBS. The extends of the SDO_PC BLOBS are stored as a spatial data type. The definition of the content and structure of the SDO_PC objects however, is available and can be implemented within applications to read the data.

3.2.4 Spatial DBMS

The development, which initially led to the development of spatial middle layer systems, has continued resulting in the creation of spatial databases. In a sense, the spatial database is the next logical step. Unlike a middle layer system, a spatial database system is characterized by the following: Firstly, it is possible to store spatial data in transparent database tables. Secondly, reading is possible through standard queries and spatial functions, as for instance defined by the Open Geospatial Consortium (OGC) as described in § 3.4. Thirdly, the integrity of the data is secured at the database level. And finally, there is a possibility to index spatial data in a spatial way. This has resulted in spatial data that is seamlessly integrated into the relational database (see Figure 10 on the right). A very important aspect of a spatial database is the separation of application and data. The separation of application and data will merge the strengths of GIS applications (analysis and use) and relational databases (database and storage facilities) into one system. Examples of spatial databases are: Oracle (Oracle Spatial or Oracle Locator), IBM DB2 (with Informix Spatial Data Blade Module), PostgreSQL (with PostGIS), MySQL (with spatial extension) and Microsoft SQL Server.

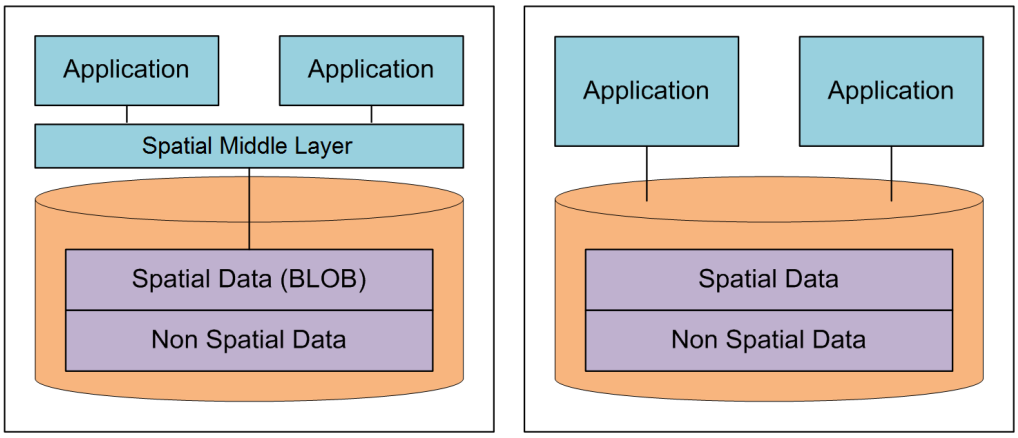


Figure 10: Spatial Middle Layer GIS (left) and Spatial DBMS (right).

In this thesis, spatial data stored in Oracle 11g (using SDO_GEOMETRY and SDO_PC) will be tested.

3.3 Spatial Data Access Methods

There are two techniques needed for rapid searching: indexing (§ 3.3.1) and clustering (§ 3.3.2). If the two techniques are combined, implementation of multi-geometries based on Wang and Shan [2005] would be possible (§ 3.3.3).

3.3.1 Indexing

A spatial index, like any other index, provides a mechanism to limit searches, but in the case of spatial data, the mechanism is based on spatial criteria such as distance, intersection and containment. A spatial index is needed to [Oracle, 2010]:

- Find objects within an indexed data space that interact with a given point or area of interest (window query)
- Find pairs of objects from within two indexed data spaces that interact spatially with each other (spatial join)

The entries in the spatial index are dependent on the location of the geometries in a coordinate space, but the index values are in a different domain. The coordinates for a geometry may be pairs of integer, floating-point, or double-precision numbers.

Common spatial index methods include:

- *Quadtree:* Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four cells. Each cell has a maximum capacity. When maximum capacity is reached, the cell splits. Oracle Spatial has implemented this index.
- *R-tree:* Objects (shapes, lines and points) are grouped using the minimum bounding rectangle (MBR). Objects are added to an MBR within the index that will lead to the smallest increase in its size. Oracle Spatial has implemented this index method by default.

3.3.2 Clustering

The goal of clustering is to minimize the retrieval time by storing nearby objects also nearby in computer memory (disk). In spatial database systems, the notion of clustering is used when spatially adjacent objects, which are often required jointly by queries, are stored physically together in secondary storage. An adequate access mechanism for spatial database systems has to support three types of clustering to perform spatial queries more efficiently [Brinkhoff et al, 1994]:

- *Internal clustering:* In order to speed up access to single objects, the complete representation of one object is stored in one page, assuming its size is smaller than the free space on the page. Otherwise, the object is stored on multiple physically consecutive pages. In this case, the number of pages occupied by the object is at most one higher than the minimum number of pages that are necessary to store the object.
- *Local clustering:* In order to speed up access to several objects, a set of spatial objects (or approximations) is grouped onto one page. This grouping is performed according to the location of the objects (or approximations) in data space.
- *Global clustering:* In contrast to local clustering, a set of spatially adjacent objects are stored not on one but on several physically consecutive pages that can be accessed by one single read request.

According to OGC [1999], multi-geometries like MultiPoint and GeometryCollection are

geometric collections that consist of one or more geometries. The elements of a MultiPoint are restricted to Points, whereas the elements of a GeometryCollection are not restricted at all and can contain all spatial types. The elements in a multi-geometry are not connected or ordered. While creating and using multi-geometries two extra types of clustering can be added:

- *Object clustering:* Spatial indexes are built upon the minimal bounding boxes of objects. During the creation of multi-geometries, the elements can be grouped according to their location. This results in smaller minimum bounding rectangles or boxes, and limits searches with spatial indexes.
- *In-object clustering:* The elements of multi-geometries are by definition not ordered. However, ordering can be applied inside a multi-geometry if this is deliberately done by the user. When querying the dataset, the multi-geometry will be unpacked. A new suggestion proposed in this thesis is to use the ordering in a multi-geometry as a means of implicit generalisation. In effect, this means that the most descriptive points of the area covered by the multi-geometry are stored as the first points within the multi-geometry. A decreasing order of importance should be applied here. Now, if the data needs to be generalised, the user can choose to unpack only the first few points from the multi-geometry object. The number of points to unpack depends on the zoom level of the spatial display window and the processing power of the machine. The ranking of the points can be determined using various point cloud filtering techniques that highlight the most characteristic points. If such a filter is not available, a spatial distribution within the area should be used, i.e. the first point should be in the centre, the next 4 points near the edges, etc. The data model that is developed in this thesis will be designed in such a way that it is compatible with this new in-object clustering approach.

3.3.3 Sorting and Searching – Implementation of Multi-geometries

The theory for clustering and indexing of spatial data can be used to implement multi-geometries. Wang and Shawn [2005] describe a method where the data to be processed is split into cells. The size of the cells varies according to the density of the data such that each cell contains sufficient laser points. Then the cells are organised according to the principals of space-filling curves.

Space-filling curves organize non-overlapping cells into a 1D order, such that cells along the curve that lie close to each other in reality also lie close to each other. There are various space-filling curves. Among them, the Hilbert curve (also known as a Hilbert space-filling curve) first described by Hilbert [1891], as a variant of the space-filling curve discovered by Peano [1890], and the Morton curve (also known as the Z-order, Morton order, or Morton code) introduced by Morton [1966]. These two types with three different levels are shown in Figure 11. A space-filling curve can be made up of several levels by splitting the cells into smaller cells. The index method region Quadtree [Gaede and Gunther, 1998] can be applied to split the data into cells. Using this index method, the data is split into four equal parts, and the number of objects per cell is counted. Each cell that still contains too many objects is split again until the number of objects is just under or equal to the required maximum number of objects in the cell. Then the cells are organised according to the principals of space-filling curves.

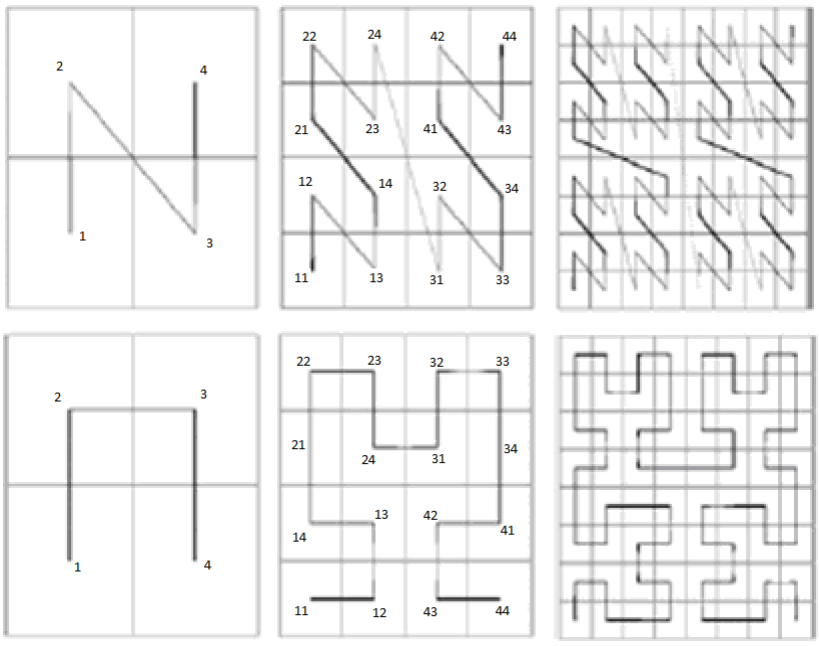


Figure 11: 2D Morton (top) and Hilbert (bottom) curves [based on Wang and Shan, 2005].

3.4 Spatial DBMS Standards

The Open Geospatial Consortium, Inc. (OGC) is a non-profit, international, voluntary consensus standards organization that is leading the development of standards for geographic and location based services. OGC works with governments, private industry, and academia to create open and extensible software application programming interfaces for geographic information systems (GIS) and other mainstream technologies [OGC, 2000]. This paragraph is subdivided into three parts. Subparagraph 3.4.1 explains the functions of the OGC Abstract and Implementation Specifications. The second subparagraph (3.4.2) shows the object model for geometries as defined in the OGC Simple Features Specification.

3.4.1 Abstract and Implementation Specifications

The OGC maintains two types of specification products: Abstract and Implementation specifications. The purpose of the Abstract Specification is to create and document a conceptual model sufficient to allow for the creation of Implementation Specifications. Implementation Specifications are unambiguous technology platform specifications for implementation of industry-standard, application programming interfaces (API) [OGC, 2000].

One of these specifications is the “OGC Simple Features Specification for SQL” [OGC, 1999]. The purpose of this specification is to define a standard SQL schema that supports storage, retrieval, query, and update of simple spatial feature collections. A simple feature is defined by the “OGC Abstract Specification” [OGC, 2000] (see § 3.4.2) to have both spatial and non-spatial attributes. Spatial attributes are geometry values, and simple features are based on 2D geometry with linear interpolation between vertices.

The “OGC Simple Features Specification for SQL” (SFS) [OGC, 1999] does not attempt to standardize and does not depend upon any part of the mechanism by which Types are added and maintained in the SQL environment including:

- The syntax and functionality provided for defining types

- The syntax and functionality provided for defining SQL functions
- The physical storage of type instances in the database
- The specific terminology used to refer to types.

The specification does standardize:

- The names and geometric definitions of the OGC SQL Types for Geometry
- The names, signatures, and geometric definitions of the OGC SQL Functions for Geometry

As can be read in the next subparagraphs, the feature types are defined in two-dimensional coordinate space ($\Re^2$ - *x,y*), the actual implementation of these specifications, however, is extended into a three-dimensional coordinate space ($\Re^3$ - *x,y,z*) and in some cases to a four-dimensional coordinate space ($\Re^4$ - *x,y,z,t*).

3.4.2 Object Model for Geometry

The object model for geometry as specified by the OGC [OGC, 1999] is shown in . The base Geometry class has subclasses for Point (zero-dimensional), Curve (one-dimensional), Surface (two-dimensional), and GeometryCollection (mixed). Each geometric object is associated with a Spatial Reference System (SRS), which describes the coordinate space in which the geometric object is defined. is based on extending the Geometry Model specified in the "OGC Abstract Specification" [OGC, 2000] with specialized zero, one and two-dimensional collection classes named MultiPoint, MultiLineString and MultiPolygon for modelling geometries corresponding to collections of Points, LineStrings and Polygons respectively. The figure shows aggregation lines (ending with a diamond ◊) between the leaf collection classes and their element classes.

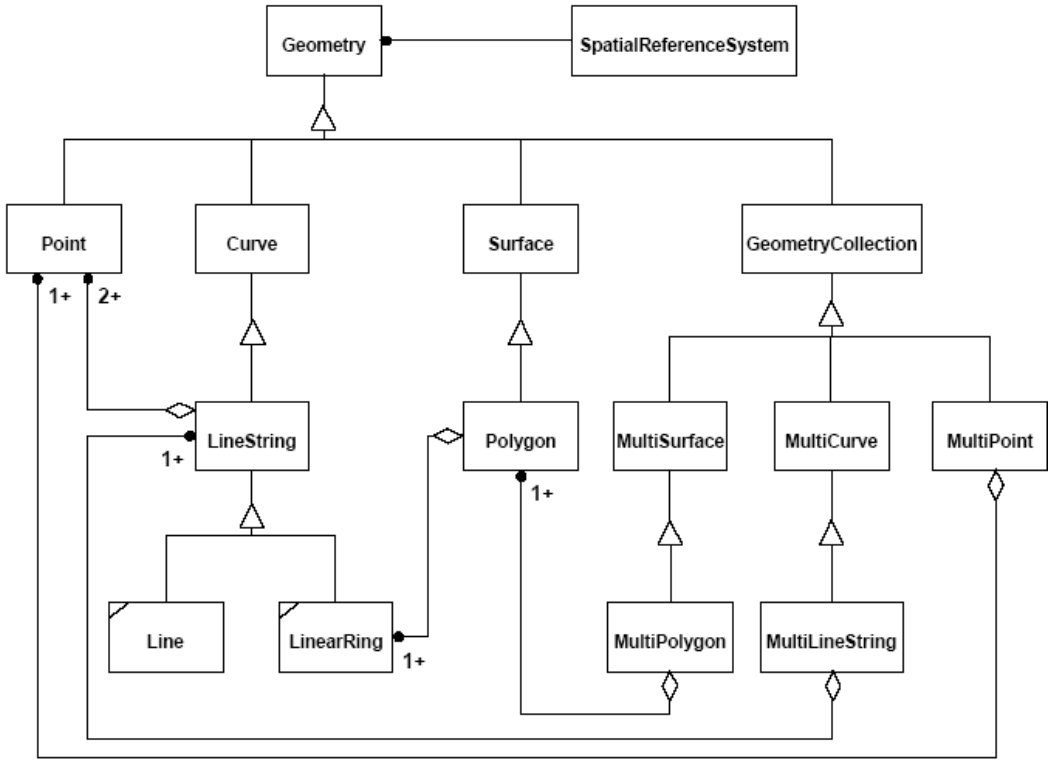


Figure 12: Geometry Class Hierarchy [OGC, 1999].

Geometry is the root class of the specified hierarchy. The subclasses of Geometry defined in

[OGC, 2000] are restricted to zero, one and two-dimensional geometric objects that exist in two-dimensional coordinate space ($\Re^2$). All geometry classes described in [OGC, 2000] are defined such that all instances of a geometry class are valid. According to OGC Simple Features Specifications [OGC, 1999], ten feature types are defined, only the definitions of Point, Multipoint and GeometryCollection are listed:

- **Point:** A Point is a zero-dimensional geometry and represents a single location in coordinate space. A Point has an *x*-coordinate value and a *y*-coordinate value (or *z*-coordinate value). The boundary of a Point is the empty set.
- **MultiPoint**: A MultiPoint is a zero-dimensional geometric collection. The elements of a MultiPoint are restricted to Points. The points are not connected or ordered. A MultiPoint is simple if no two Points in the MultiPoint are equal (have identical coordinate values). The boundary of a MultiPoint is the empty set. All the elements in a MultiPoint must be in the same Spatial Reference. This is also the Spatial Reference for the MultiPoint.
- **GeometryCollection**: A GeometryCollection is a geometry that is a collection of one or more geometries. All the elements in a GeometryCollection must be in the same Spatial Reference. This is also the Spatial Reference for the GeometryCollection.

3.5 Spatial DBMS Implementations and Vendors

Several DBMSs are available on the market. In order to take a closer look at current spatial DBMSs implementations, three mainstream spatial DBMS implementations are chosen and studied in aspects of their supported spatial types, import formats, spatial operators/functions, spatial index, metadata and Spatial Reference System: Microsoft SQL Server – Spatial Data [URL, Microsoft SQL Server 2012] (§ 3.5.1), PostgreSQL - PostGIS [PostgreSQL - PostGIS, 2012] (§ 3.5.2) and Oracle – Oracle Locator/Spatial [Oracle, 2010] (§ 3.5.3). As Oracle is the market leader in the development of DBMS, this research is limited to Oracle. PostgreSQL as well as MySQL are open source DBMSs.

3.5.1 Microsoft SQL Server – Spatial Data

[URL Microsoft SQL Server 2012] SQL Server 2012 provides support for geographical data through the inclusion of spatial data types. SQL Server 2012 provides the geography data type for geodetic spatial data, and the geometry data type for planar spatial data. Both are implemented as Microsoft .NET Framework Common Language Runtime (CLR) types, and can be used to store different kinds of geographical elements such as points, lines, and polygons. Both data types provide properties and methods that can be used to perform spatial operations such as calculating distances between locations and finding geographical features that intersect one another. The geography and geometry data types include methods for importing and exporting data in the Well Known Text (WKT) and Well Known Binary (WKB) formats for geographic data that are defined by the OGC, as well as the Geographic Mark-up Language (GML) format. The geometry data type provides properties and methods that are aligned with the Open Geospatial Consortium (OGC) Simple Features Specification for SQL and enables to perform operations on geometric data. A quad-tree index is implemented to limit spatial searches.

3.5.2 PostgreSQL – PostGIS

PostGIS is an extension to the PostgreSQL object-relational database system that allows GIS (Geographic Information Systems) objects to be stored in the database. POstGIS provides the geography data type for geodetic spatial data, and the geometry data type for planar spatial data. As can be read in the manual [PostGIS, 2012]: The GIS objects supported by PostGIS are a superset of the "Simple Features" defined by the Open Geospatial Consortium (OGC). PostGIS supports all the objects and functions specified in the OGC "Simple Features for SQL" (SFS) specification. PostGIS extends the standard with support for 3DM, 3DZ, 4D coordinates. PostGIS supports seven spatial types in SFS, namely: POINT, MULTIPOINT, LINESTRING, MULTILINESTRING, POLYGON, MULTIPOLYGON and GEOMETRYCOLLECTION. PostGIS 1.4 and above do support the use of Compound Curves in a Curve Polygon. PostGIS includes support for GiST-based R-Tree spatial indexes, and functions for analysis and processing of GIS objects. Ott [2012] is making an attempt towards storing point clouds in PostgreSQL. Some general PostgreSQL limits are included in the table below.

Table 1: Limits on PostgreSQL [URL, PostgreSQL].

Key Feature	***Limit***
Maximum CPU	Unlimited
Maximum RAM	OS Max
Windows Support	yes
Linux Support	yes
Unix Support	yes
64 Bit Support	yes
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

3.5.3 Oracle - Oracle Locator/Spatial 11g

Oracle 11g is available in a choice of editions: Express Edition, Standard Edition One, Standard Edition, and Enterprise Edition. All editions are built using the same common code base, which means that database applications can easily scale from small, single processor servers to clusters of multi-processor servers without changing a line of code. Additional options for enhanced performance, scalability, availability, security and manageability are available with Enterprise Edition. The table below shows the major limitations on each edition.

Oracle Spatial is conformant with "OGC Simple Features Specification for SQL" (SFS, Document 99-049) [OGC, 1999], starting with Oracle Database release 10g (version 10.1.0.4). Conformance with this specification means that Oracle Spatial supports all the types, functions, and language constructs detailed in Section 3.2 of the specification. Within Oracle Spatial a data type called SDO_GEOMETRY is defined. This data type is accepted as being well-known for storing spatial data and can be read by a large number of applications. Starting with Oracle Database release 11g, a new data type called SDO_PC is available for storing point clouds.

All editions include Oracle Locator as its spatial solution. Oracle Locator provides core features and services available in Oracle Spatial. Oracle Spatial is a priced option available only with

Oracle Database 11g Enterprise Edition. Oracle Spatial includes all Oracle Locator features as well as other features that are not available with Oracle Locator. The SDO_PC data type is not licensed and implemented in Oracle Locator. Both SDO_GEOMETRY and SDO_PC data types will be used for testing.

Table 2: Limits on Oracle [URL, Oracle].

Key Feature Summary	***Express Edition***	***Standard Edition One***	***Standard Edition***	***Enterprise Edition***
Maximum CPU	1 CPU	2 Sockets	4 Sockets	Unlimited
Maximum RAM	1GB	OS Max	OS Max	OS Max
Windows Support	yes	yes	yes	yes
Linux Support	yes	yes	yes	yes
Unix Support	no	yes	yes	yes
64 Bit Support	no	yes	yes	yes
Max Database size	4GB	Unlimited	Unlimited	Unlimited
Maximum Table Size		Unlimited		
Maximum Row Size		Unlimited		
Maximum Field Size		Unlimited		
Maximum Rows per Table		Unlimited		
Maximum Columns per Table		1000		
Maximum Indexes per Table		Unlimited		

Oracle enables users to define new data types (user-defined data types) which are made up of several attributes. These attributes can be of basic data types such as NUMBER, VARCHAR2, DATE, or other existing user-defined data types.

SDO_GEOMETRY

With Spatial, the geometric description of a spatial object is stored in a single row, in a single column with the data type SDO_GEOMETRY in a user-defined table. Any table that has a column of type SDO_GEOMETRY must have another column, or set of columns, that defines a unique primary key for that table. Tables of this kind are sometimes referred to as spatial tables or spatial geometry tables. Oracle Spatial defines the object type SDO_GEOMETRY as in SQL-Code 1. As can be seen, the object type SDO_GEOMETRY is made up of the data types SDO_POINT_TYPE, SDO_ELEM_INFO_ARRAY, and SDO_ORDINATE_ARRAY, the definitions of these data types are also shown in SQL-Code 1.

SQL-Code 1: Oracle – SDO_GEOMETRY [Oracle, 2010]

```
CREATE TYPE SDO_POINT_TYPE AS OBJECT (
  X NUMBER,
  Y NUMBER,
  Z NUMBER);

CREATE TYPE SDO_ELEM_INFO_ARRAY AS VARRAY (1048576) OF NUMBER;
CREATE TYPE SDO_ORDINATE_ARRAY AS VARRAY (1048576) OF NUMBER;

CREATE TYPE SDO_GEOMETRY AS OBJECT (
  SDO_GTYPE NUMBER,
  SDO_SRID NUMBER,
  SDO_POINT SDO_POINT_TYPE,
  SDO_ELEM_INFO SDO_ELEM_INFO_ARRAY,
  SDO_ORDINATES SDO_ORDINATE_ARRAY);
```

Because the SDO_ORDINATE_ARRAY may contain up to a maximum of 1,048,576 numbers, the maximum number of vertices in an SDO_GEOMETRY object depends on the dimension of

the dataset: 524,288 vertices for two dimensions, 349,525 vertices for three dimensions, and 262,144 vertices for four dimensions [Oracle, 2010].

Starting with Oracle Database Release 11.1, Oracle Spatial supports the storage and retrieval of three-dimensional spatial data, which can include feature types POINTs and MULTIPOINTs or POINTCLUSTERs. Table 3 shows the relation between the feature type, the SDO_GTYPE and the SDO_ELEM_INFO_ARRAY for point data (for examples see SQL-Code 2).

The SDO_ELEM_INFO_ARRAY attribute is defined using a varying length array of numbers. This attribute lets you know how to interpret the ordinates stored in the SDO_ORDINATES_ARRAY attribute. Each triplet set of numbers is interpreted as follows [Oracle, 2010]:

- SDO_STARTING_OFFSET - Indicates the offset within the SDO_ORDINATES array where the first ordinate for this element is stored.
- SDO_ETYPE - Indicates the type of the element.
- SDO_INTERPRETATION - Means one of two things, depending on whether or not SDO_ETYPE is a compound element. If SDO_ETYPE is a compound element (4, 1005, or 2005), this field specifies how many subsequent triplet values are part of the element. If the SDO_ETYPE is not a compound element (1, 2, 1003, or 2003), the interpretation attribute determines how the sequence of ordinates for this element is interpreted.

Table 3: SDO_GEOMETRY attributes for 3D points [Oracle, 2010]

Feature type	SDO_GTYPE	SDO_STARTING_OFFSET	SDO_ETYPE	SDO_INTERPRETATION
POINT	3001	Does not apply.	Does not apply.	Does not apply.
MULTIPOINT	3005	1,4,7,… starting per point*	1*	1*
POINTCLUSTER	3005	1**	1**	n**

* one triplet for every point in the record ** where n is the number of points, one triplet per record

For storing the data as separate points, the SDO_POINT_TYPE should be used. In that case, the attributes SDO_ELEM_INFO_ARRAY and SDO_ORDINATES_ARRAY are non-existing. When the ordinates of the points are stored in the attribute SDO_ORDINATE_ARRAY, the attribute SDO_POINT_TYPE is non-existing.

The feature type POINTCLUSTER is comparable to the feature type MULTIPOINT, except for the SDO_ELEM_INFO_ARRAY attribute. This contains only one triplet for all points together, where SDO_INTERPRETATION is the number of points in the record (POINTCLUSTER) or it contains one triplet per point (MULTIPOINT).

A spatial table can be created using the CREATE TABLE statement, a spatial table should contain at least one column of object type SDO_GEOMETRY, see for example SQL-Code 2. More information about the data type for storing spatial data can be found in the Oracle Spatial User manual [Oracle, 2010].

The possibilities of each of the above storage methods become clear with the help of the examples below (SQL-Code 2). As can be concluded from the examples below, the data inserted is the same – except for the ROW_ID – although the code to insert it is not. A table with separate POINTs has many more records (each with its own overhead) than a table with MULTIPOINTs or POINTCLUSTERs. Retrieval of points can be obtained in two ways or as a combination of both: Selecting all records (SELECT * FROM …) or selecting separate points by unpacking (using function SDO_UTIL.GETVERTICES).

Suppose a dataset with ten points, with ordinates *(x,y,z)*, in the Dutch RD/NAP spatial reference system *(SRID = 28992)* will be inserted in the database.

ROW_ID	X	Y	Z
1	97395.1	496989.5	-10.450
2	97393.9	496989.9	-10.770
3	97394.7	496989.9	-10.930
4	97394.9	496989.9	-10.950
5	97395.1	496989.9	-10.355
6	97395.3	496989.9	-10.397
7	97395.5	496989.9	-10.430
8	97395.7	496989.9	-10.660
9	97395.9	496989.9	-10.720
10	97397.7	496989.9	-11.180

SQL-Code 2: Oracle – Example – SDO_GEOMETRY

```
POINT
CREATE TABLE MyTable_POINT (ROW_ID NUMBER, GEOM MDSYS.SDO_GEOMETRY);

INSERT INTO MyTable_POINT VALUES (1, MDSYS.SDO_GEOMETRY(3001, 28992,
  MDSYS.SDO_POINT_TYPE(97395.1, 496989.5, -10.450), NULL, NULL));
...
INSERT INTO MyTable_POINT VALUES (10, MDSYS.SDO_GEOMETRY(3001, 28992,
  MDSYS.SDO_POINT_TYPE(97397.7, 496989.9, -11.180), NULL, NULL));

SELECT_RECORDS
SELECT * FROM MyTable_POINT;

UNPACK_POINT
SELECT c.ROW_ID, c.GEOM.SDO_POINT.X, c.GEOM.SDO_POINT.Y, c.GEOM.SDO_POINT.Z
  FROM MyTable_POINT c;

MULTIPOINT
CREATE TABLE MyTable_MULTIPOINT (ROW_ID NUMBER, GEOM MDSYS.SDO_GEOMETRY);

INSERT INTO MyTable_MULTIPOINT VALUES (1, MDSYS.SDO_GEOMETRY(3005, 28992, NULL,
  SDO_ELEM_INFO_ARRAY(1,1,1, 4,1,1, 7,1,1, 10,1,1, 13,1,1, 16,1,1, 19,1,1,
    22,1,1, 25,1,1, 28,1,1),
  SDO_ORDINATE_ARRAY (97395.1, 496989.5, -10.450, 97393.9, 496989.9, -10.770,
    97394.7, 496989.9, -10.930, 97394.9, 496989.9, -10.950, 97395.1,
    496989.9, -10.355, 97395.3, 496989.9, -10.397, 97395.5, 496989.9,
    -10.430, 97395.7, 496989.9, -10.660, 97395.9, 496989.9, -10.720, 97397.7,
    496989.9, -11.180)));

SELECT_RECORDS
SELECT * FROM MyTable_MULTIPOINT;

UNPACK_MULTIPOINT
SELECT c.ROW_ID, t.X, t.Y, t.Z
  FROM MyTable_MULTIPOINT c, TABLE(SDO_UTIL.GETVERTICES(c.GEOM)) t;
SELECT c.ROW_ID, SDO_GEOMETRY(3001,28992,SDO_POINT_TYPE(t.X,t.Y,t.Z),NULL,NULL) GEOM
  FROM MyTable_MULTIPOINT c, TABLE(SDO_UTIL.GETVERTICES(c.GEOM)) t;

POINTCLUSTER
CREATE TABLE MyTable_POINTCLUSTER (ROW_ID NUMBER, GEOM MDSYS.SDO_GEOMETRY);

INSERT INTO MyTable_POINTCLUSTER VALUES (1, MDSYS.SDO_GEOMETRY(3005, 28992, NULL,
  SDO_ELEM_INFO_ARRAY(1,1,10),
  SDO_ORDINATE_ARRAY (97395.1, 496989.5, -10.450, 97393.9, 496989.9, -10.770,
    97394.7, 496989.9, -10.930, 97394.9, 496989.9, -10.950, 97395.1,
    496989.9, -10.355, 97395.3, 496989.9, -10.397, 97395.5, 496989.9,
    -10.430, 97395.7, 496989.9, -10.660, 97395.9, 496989.9, -10.720, 97397.7,
    496989.9, -11.180)));

SELECT_RECORDS
SELECT * FROM MyTable_POINTCLUSTER;
```

```
UNPACK_POINTCLUSTER
SELECT c.ROW_ID, t.X, t.Y, t.Z
  FROM MyTable_POINTCLUSTER c, TABLE(SDO_UTIL.GETVERTICES(c.GEOM)) t;
SELECT c.ROW_ID, SDO_GEOMETRY(3001,28992,SDO_POINT_TYPE(t.X,t.Y,t.Z),NULL,NULL) GEOM
  FROM MyTable_POINTCLUSTER c, TABLE(SDO_UTIL.GETVERTICES(c.GEOM)) t;
```

SDO_PC

Starting with Oracle Database Release 11.1, Oracle Spatial supports the storage and retrieval of three-dimensional spatial data, which can include point clouds. The description of a point cloud is stored in a single row, in a single column of object type SDO_PC in a user-defined table. The object type SDO_PC is defined as in SQL-Code 3.

SQL-Code 3: Oracle – SDO_PC [Oracle, 2010]

```
CREATE TYPE SDO_PC AS OBJECT
  (BASE_TABLE          VARCHAR2(70),
   BASE_TABLE_COL      VARCHAR2(1024),
   PC_ID               NUMBER,
   BLK_TABLE           VARCHAR2(70),
   PTN_PARAMS          VARCHAR2(1024),
   PC_EXTENT           SDO_GEOMETRY,
   PC_TOL              NUMBER,
   PC_TOT_DIMENSIONS   NUMBER,
   PC_DOMAIN           SDO_ORGSCL_TYPE,
   PC_VAL_ATTR_TABLES SDO_STRING_ARRAY,
   PC_OTHER_ATTRS      XMLTYPE);
```

The input data has to be stored in a table or view, this table or view should have the following columns:

- RID (VARCHAR2(24)): Unique ID for each point
- VAL_D1 (NUMBER): Ordinate in dimension 1
- VAL_D2 (NUMBER): Ordinate in dimension 2
- . . .
- VAL_D*n* (NUMBER): Ordinate in dimension *n*, where *n* is the highest-numbered dimension. A point can have a maximum of 8 dimensions.

To store points with the object type SDO_PC, two (optional three) tables have to be created (see SQL-Code 4).

SQL-Code 4: Oracle - SDO_PC - create table.

```
BASE_TABLE (required)
CREATE TABLE <BASE_PC_TABLE_NAME> (PC SDO_PC);

BLOCK_TABLE (required)
CREATE TABLE <BLOCK_TABLE_NAME> AS SELECT * FROM MDSYS.SDO_PC_BLK_TABLE;

RESULT_TABLE (optional)
CREATE TABLE <RESULT_TABLE_NAME> (
  PTN_ID NUMBER,
  POINT_ID NUMBER,
  RID VARCHAR2(24),
  VAL_D1 NUMBER,
  … ,
  VAL_Dn NUMBER);
```

To store points with the object type SDO_PC, two functions have to be executed:

- SDO_PC_PKG.INIT: *Initializes a point cloud*;
- SDO_PC_PKG.CREATE_PC: *Creates a point cloud using the data in the input table*.

The point clouds are created by splitting the points into cells that do not overlap, each cell being filled to its maximum (BLK_CAPACITY). The minimum bounding box, with a data type of SDO_GEOMETRY and the coordinates of each point in binary format are stored in a table (BLOCK_TABLE). By storing the bounding box separately, it is possible to use it as primary filter for spatial data selection. The initial definition of the point cloud input dataset is stored in the BASE_TABLE. These two tables are displayed in . The resulting point data can optionally be stored in the RESULT_TABLE. This table can be used in applications for searching using SQL queries on dimensions other (for example *D4,…, Dn*) than the indexed dimensions (for example *D1,D2,D3*).

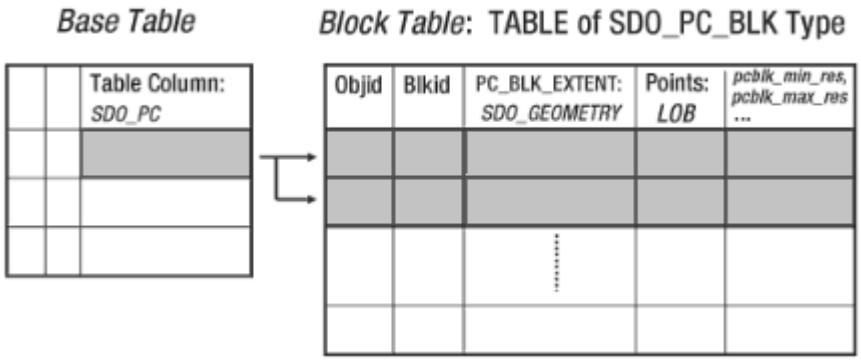


Figure 13: SDO_PC Creation Overview.

Data in a pointcloud can be selected in two ways:

1. Filtering the SDO_GEOMETRY object PC_EXTENT by using spatial operators or functions.
2. Filtering the SDC_PC object by using the SDO_PC_PKG.CLIP_PC function.

Afterwards, the selected SDO_PC objects can be converted to SDO_GEOMETRY objects to enable a more detailed selection by using the SDO_PC_PKG.TO_GEOMETRY function. This conversion results into a SDO_GEOMETRY object, GTYPE *n*005 where *n* is the highest-numbered dimension. The content of the SDO_ELEM_INFO_ARRAY is as feature type POINTCLUSTER (see Table 3). This SDO_GEOMETRY object can be extracted using the function SDO_UTIL.GETVERTICES.

Continuing the example of SQL-Code 2 results in SQL-Code 5 below.

SQL-Code 5: Oracle – Example – SDO_PC

```
INPUT TABLE
CREATE TABLE MyTable_INPUT (RID VARCHAR2(24), VAL_D1 NUMBER, VAL_D2 NUMBER, VAL_D3
NUMBER);

INSERT INTO MyTable_INPUT VALUES (1, 97395.1, 496989.5, -10.450);
 ...
INSERT INTO MyTable_INPUT VALUES (10, 97397.7, 496989.9, -11.180);
```

```
BASE PC TABLE
CREATE TABLE MyTable_BASE_PC (PC SDO_PC);

BLOCK TABLE
CREATE TABLE MyTable_BLOCK_PC AS SELECT * FROM MDSYS.SDO_PC_BLK_TABLE;

RESULT TABLE
CREATE TABLE MyTable_INPUT_RESULT (PTN_ID NUMBER, POINT_ID NUMBER, RID VARCHAR2(24),
VAL_D1 NUMBER, VAL_D2 NUMBER, VAL_D3 NUMBER);
SDO_PC
DECLARE
  PC SDO_PC;
BEGIN
-- initialize sdo_pc
  PC := SDO_PC_PKG.INIT(
           'MyTable_BASE_PC',     -- name of BASE_PC_TABLE
           'PC',                  -- column name of point cloud object in BASE_PC_TABLE
           'MyTable_BLOCK_PC',    -- table to store block of the point cloud
           'BLK_CAPACITY=1000',   -- max # of points per block
           MDSYS.SDO_GEOMETRY(3003,NULL,NULL,MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3),
              MDSYS.SDO_ORDINATE_ARRAY(97393.9,496989.5,-11.2, 97397.7,496989.9,
                 -10.3)),         -- extend
           0.0005,                -- tolerance for point cloud
           3,                     -- total number of dimensions
           NULL,                  -- pc_domain, not implemented yet
           NULL,                  -- pc_val_attr_tables
           NULL);                 -- pc_other_attrs

  INSERT INTO MyTable_BASE_PC VALUES (PC);

-- create sdo_pc
  SDO_PC_PKG.CREATE_PC(
     PC,                          -- initialized point cloud object
     'MyTable_INPUT',             -- name of input table
     'MyTable_INPUT_RESULT');     -- name of output table
END;
/

CLIP_PC
CREATE TABLE MyTabel_QUERY_RESULT_PC AS SELECT * FROM TABLE(
  SDO_PC_PKG.CLIP_PC(
    (SELECT PC FROM MyTable_BASE_PC WHERE ROWNUM = 1),
                                  -- point cloud object
    SDO_GEOMETRY(3003,NULL,NULL,MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3),
      MDSYS.SDO_ORDINATE_ARRAY(97394,496989.6,-12, 97395,496989.8,-10)),
                                  -- query window GTYPE=3003, 3D polygon
                                        SDO_ETYPE=1003, exterior polygon ring
                                        SDO_INTERPRETATION=3, rectangle
    NULL,
    NULL,
    NULL));

PC_TO_GEOMETRY
CREATE TABLE MyTable_QUERY_RESULT_GEOM AS SELECT
  SDO_PC_PKG.TO_GEOMETRY(
    A.POINTS,
    A.NUM_POINTS,
    3,
    28992,
    NULL) GEOM
  FROM MyTabel_QUERY_RESULT_PC A WHERE NUM_POINTS > 0;

UNPACK_PC
SELECT t.X, t.Y, t.Z
  FROM MyTable_QUERY_RESULT_GEOM c, TABLE(SDO_UTIL.GETVERTICES(c.GEOM)) t;
SELECT SDO_GEOMETRY(3001,28992,SDO_POINT_TYPE(t.X,t.Y,t.Z),NULL,NULL) GEOM
  FROM MyTable_QUERY_RESULT_GEOM c, TABLE(SDO_UTIL.GETVERTICES(c.GEOM)) t;
```

4. Conceptual Design

This chapter contains the conceptual design of the data model. The process of designing, involves a number of steps for translating the central concept and the requirements stated in the introduction into the final design. This design includes some parameters. In the first paragraph (§ 4.1), the central concept is described. Based on preliminary research, § 4.2 describes a generic data model that is designed. This generic data model will be the basis for further research. § 4.3 lists the requirements, measurable objectives and constraints for the final design. The test environment (database) (§ 4.4), and the dataset (§ 4.5) that will be used for testing are listed in the following paragraphs. § 4.6 describes the implementation of the generic data model. The test procedure is based on the central concept, the requirements, measurable objectives, and constraints. This procedure is described in § 4.7.

4.1 Central Concept

In the first chapter, the research problem is formulated as:

What is the best design for a data model to store large point clouds in an Oracle DBMS, such that it is generally accessible by spatial applications, that all attributes are preserved and that performance is optimised?

A database management system (DBMS) is computer software that manages databases. A DBMS allows users and other software to store and retrieve data in a structured way. Oracle is a DBMS. In Oracle a schema (with procedures and functions) is implemented to store and retrieve spatial data. The data model that will be designed is based on this schema (called MDSYS). The data model to be designed contains configuration parameters for tuning the DBMS. The goal of this research is to describe a model that ensures efficient storage, has a fast retrieval time, and the possibility to pose simple queries against the data. Therefore:

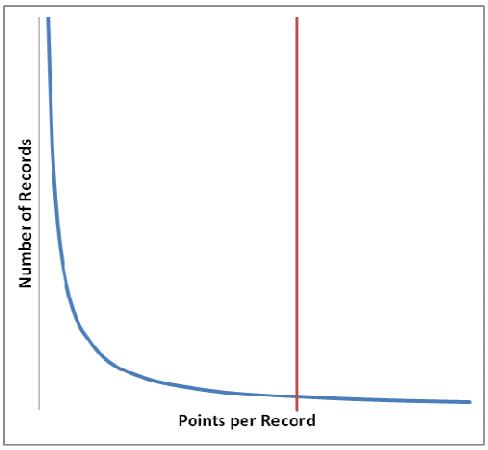
The result of this research will be a data model with configuration parameters that prescribes and realizes how to manage massive point clouds in Oracle Spatial.

Storing and retrieving massive datasets as single point records (one point per record) results in storing and retrieving lots of records and efficient storage, but fast retrieval become questionable. A solution can be found by reducing the number of records by grouping points without loss of information. But, if points are grouped, associated attributes have to be grouped as well. It has to be decided how to record these attributes with the points.

When looking at the ratio of the number of records in relation to the number of points per group (record) with a constant number of points per group (record), then a graph can be obtained as in Graph 1 (blue line). With the number of records inversely proportional to the number of points in one group (record), and while increasing the size of a group, the number of records decreases linear.

There is a certain limit to the number of points in one record; this is shown by the red line. The data type SDO_GEOMETRY is due to its definition limited to 1,048,576 ordinates, in other words 349,525 points per record in 3D [Oracle, 2010]. The SDO_PC data type does not have a limit, but if the user wants to convert the point cloud records retrieved from a query into a SDO_GEOMETRY type, the limit is also 1,048,576 ordinates. But attributes attached to the

point cloud will be converted into the same SDO_GEOMETRY, therefore the limit of a point cloud object (SDO_PC) in 3D with 5 attributes will be 1,048,576/(3+5) = 131,072 points per record.



Graph 1: Number of Records vs. Points per Record (blue) and a Supposed Maximum (red).

When storing points with a spatial data type, an overhead is added into the record for the construction of the points. This takes up an amount of space in the database. If two points are grouped to one record, the overhead halves, because the overhead is per record, not per point. To begin with, the reduction of overhead is quite considerable, but and later on, the same difference in the number of points per record has much less effect. The record length divided through the number of points in one records receives a minimum. The maximum number of points per group will be the tuning parameter.

Considering the coordinates of the point cloud data sets to be stored, it can be stated that for a small areas, coordinates are almost the same, except for the last digits. Besides grouping data, coordinates can be shortened by removing common digits, in other words, by transforming coordinates to a local reference system and storing the transformation parameter in a metadata table.

4.2 Data Model Design

Based on the concepts for data model design presented in the previous section, a new data model for point clouds is developed in this thesis.

To be able to reduce the number of records without loss of information, the data model allows point cloud data to be stored as single points or as multi-points. The data model designed follows the concept of LAS (§ 2.3 [ASPRS, 2012]), with respect to the attributes that can possibly be stored per point. When the point cloud data is stored as multi-point objects, attributes should also be stored in multi-objects. For Oracle VARRAYs can be applied.

In the case of storing point cloud data as multi-points, clustering will be applied in 2D as well as in 3D:

- For object clustering quadtree (2D) and octree (3D) gridding are suitable. The gridding will be started 2D, but when the minimum bounding box of a multi-point has a substantial height, octree gridding will be activated immediately.

- For local clustering the Morton or Hilbert space-filling curves are suitable. From Figure 11, it can be noted that a Morton space-filling curve connects cells that are not related closely, the Hilbert space filling curve does not. This suggests that Hilbert curve has better performance in data retrieval and response time. From Figure 11, it can be noted that the Hilbert space-filling curve pattern rotates along the curve for every deeper level, the Morton space-filling curve pattern does not. For this reason, within the test set-up the Morton space-filling curve is implemented. Both Hilbert and Morton space-filling curves can be extended to 3D [Wang and Shan, 2005].

A spatial index on the point cloud data limits searches. A substantial number of insert and delete operations, degrades the quality of the spatial index and affects query performance. Rebuilding the spatial index may help. To avoid this, point cloud data can better be stored per (sub-)project. In that case, it is possible to store metadata per (sub-)project.

The UML diagram below shows the designed data model. Metadata will be stored in the table *Project*. Every (sub-)project in this table has unique parameters, among them its location and the acquisition method. General parameters about the acquisition method can be stored in a separate *method* table. Every (sub-)project has one *laser points* table, named after the table name in the *project* table. The intensity and the RGB-values can be stored as attributes of the laser points. Whereas the *project* table can contain only the names and numbers of unique attributes. The table with input data is needed for importing the data into the database.

During the clustering phase, the input table will reversely be updated for the cell id's corresponding to the input data.

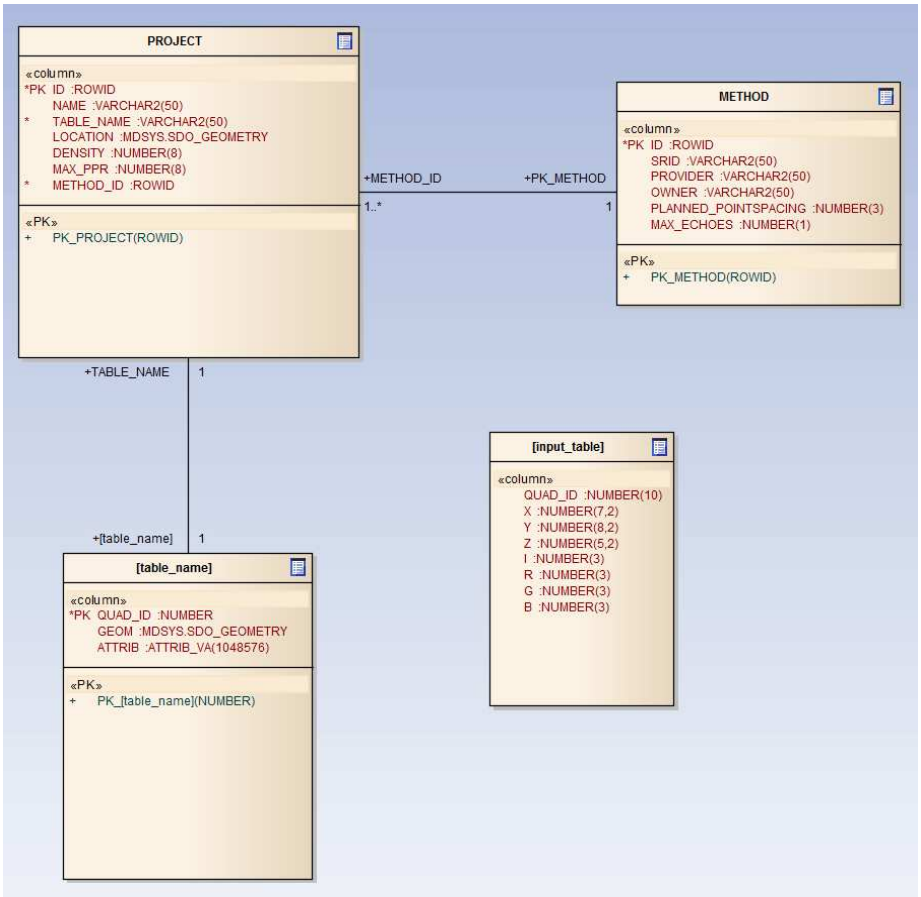


Figure 14: Generic data model for Oracle.

This is the central concept of this master thesis:

The designed generic data model describes how to organize and retrieve massive point cloud datasets in a spatial DBMS and this is tested with a real dataset.

Key issue within the central concept is the way point cloud data is organized.

4.3 Requirements, Measurable Objectives and Constraints

The main problem with massive point cloud datasets is that they are too big to be handled efficiently by today's computers. With the software tools currently available to users, it is very difficult to easily visualise complete datasets. The manipulation and processing of complete datasets is nearly impossible.

It is expected that the designed data model can solve these problems. To be able to validate the data model, to define the tuning parameter (the maximum number of points per group) and to compare the designed data model with the solution to store the points with data type SDO_PC, a special query is developed and the needed storage space on the server will be measured.

The query will be applied to the same dataset several times, while varying the tuning parameter (maximum number of points in a record). The duration of this query will be measured. To make the duration measurements and comparisons more reliable, additional queries are designed to contain larger areas, i.e. containing a large number of points. Consequently, the various other processes running on the PC in the background have a relative smaller influence on the final duration measurement and trends become visible.

The tests described above will not be applied to the metadata (project table and method tables), consequently both tables will not be implemented during the tests.

This research has the following constraints:

- This research will focus on Oracle to store spatial data.
- This research will focus on storing laser points not metadata.
- As formulated in the research problem, spatial data will be stored using a well-known or standard data type, SDO_GEOMETRY is the standard in Oracle, this data type will be compared to the newer SDO_PC data type.
- Storage space needed for storing points in a local and global reference system will be compared. However, as datasets cover a country, the local and global reference system will coincide.
- Gridding and clustering will be applied as described in § 3.3.3. This will result in not all records being completely filled, but minimal bounding boxes are as small as possible and (in 2D and 3D) not overlapping. Theo Morton space-filling curve is implemented in the test set-up, although Hilbert might be better.
- As can be concluded from §2.3 and the data model of §4.1, four non-spatial attributes will be implemented within this research: intensity, red, green, and blue. All within a range from 0 to 255.
- From the introduction (§1.2), in the test set-up, two more restrictions can be added:
  - Client and server are on the same machine;
  - Temporal aspects are out of scope;

4.4 Test Environment

All editions include Oracle Locator as its spatial solution. Oracle Locator provides core features and services available in Oracle Spatial. Oracle Spatial is a priced option available only with Oracle Database 11g Enterprise Edition. Oracle Spatial includes all Oracle Locator features as well as other features that are not available with Oracle Locator. The SDO_PC data type is not licensed and implemented in Oracle Locator. For the set set-up, Oracle is installed on two systems.

Oracle Database 11g Express Edition Release 11.2.0.2.0 - Beta
The system on which the "Oracle Database 11g Express Edition Release 11.2.0.2.0 – Beta" instance is running during this project is a DELL Precision M6500 Notebook. The system has the following specifications: Mobile Intel® Core™ i7 Processor Unit of 1.87 GHz; 12.0 GB main memory; one internal hard disk of 1 TB Raid0; and Windows 7 Professional Version 2009 Service Pack 1 64-bit Operating System. On the client-side, on the same machine, SQL*Plus: Release 11.2.0.2.0 Beta is installed. When connecting with SQL*Plus to the database, the following information is displayed:

```
SQL*Plus: Release 11.2.0.2.0 Beta
Copyright (c) 1982, 2010, Oracle.  All rights reserved.
Connected to:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - Beta
```

The client-side software package runs directly from the command prompt, as this uses less processor capacity.

Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 64-bit
The system on which the "Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 64-bit" instance is running during this project is a SUN V40z server, Solaris 10 operating system 64-bit, 4 AMD Opteron 848 CPU's, 2.2 GHz, 8 GB main memory, 2 internal 73 GB disks - 10K, 12 SCSI-attached 146 GB disks - 10K, 12 SCSI-attached 36 GB disks - 15K configured as 2 (hardware) RAID5 sets.

Many factors affect the performance of a DBMS, only one of them is the operating system. Windows called as GUI because it provides Graphical User Interface. Unix and Linux called as CLUI called Command Line User Interface. On an equal hardware configuration, Linux and Unix provide more performance than Windows due to resource utilization. Oracle database administration on Unix and Linux requires extra knowledge and special expertise. Oracle on Linux and Unix provides best security and performance. Large and critical databases are running on Linux, Unix and Windows [URL DBA Metrix Solutions].

4.5 Test Dataset

The pilot dataset of AHN-2 of the city center of Middelburg, 17,478,245 points *(x,y,z)*, is used for testing. Originally this dataset had TIFF-files containing RGB and intensity values. Per definition intensity, red, green and blue can have a value between 0 and 255. For this research, intensity, red, green, and blue within the test set-up have a dummy value of 255. This results in a real pilot dataset of AHN2 containing 17,478,245 points *(x,y,z)*, and four attributes. This is a representative, but relatively very small dataset compared to the whole AHN-2 project.

4.6 Implementation

The implementation starts with the creation of the input table (see Table 4). This table is a simple table, without spatial objects. The attribute QUAD_ID is added to support spatial gridding and clustering. This attribute will be updates several times during the gridding and clustering phase. After some preliminary testing it is decided to store the points in Oracle either as a point (SDO_GEOMETRY, SDO_POINT, SDO_GTYPE = 3001) or as a point cluster (SDO_GEOMETRY, SDO_GTYPE = 3005, SDO_ETYPE = 1, SDO_INTERPRETATION > 1). When points are stored as point cluster, a solution is needed to store the attributes. It seems the easiest way to store attributes in a VARRAY, one VARRAY for all attributes in a record. This results into the table as in Table 5.

After this, two other tables are designed but not implemented in the test set-up, because they contain only metadata. These tables are relatively small and do not have any contribution (positive and negative) to the testing results performance wize. The content of these two table has to be defined by specialists, except for some columns (TABLE_NAME, SOURCE, MAX_PPR), because they refer to the input table, the point cloud table and the maximum number of points in one record.

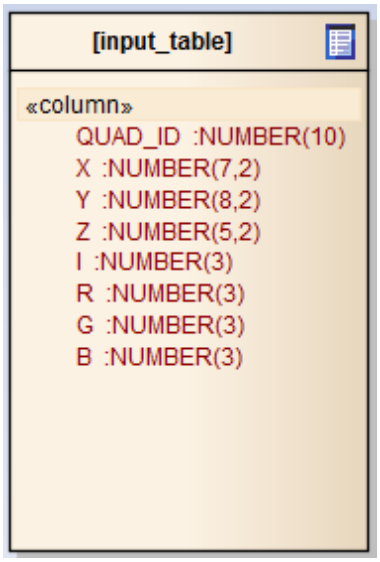


Table 4: Implemented – Input Table

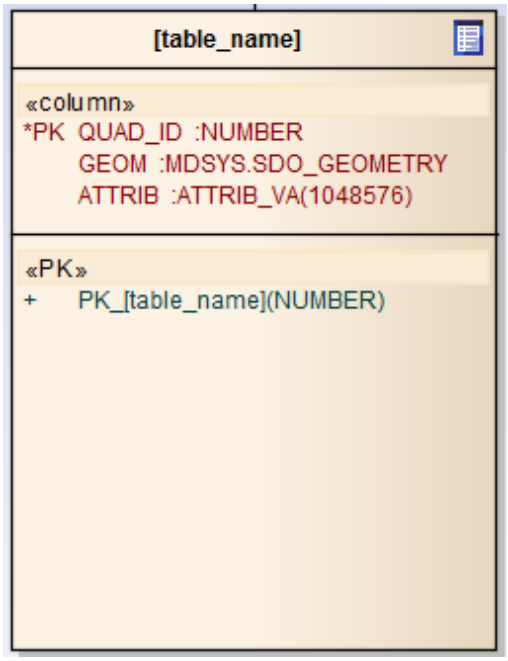


Table 5: Implemented – Point cloud Table

4.7 Test Procedure

In the previous paragraphs, a generic data model for point cloud was described. This paragraph follows with a description on how the generic data model can be evaluated on the real data presented in paragraph 4.5. The results of this test will be presented in the next chapter.

The tests will start with loading data into the database. This involves the conversion of an ASCII dataset using the SQL Loader into a straight table. Once the data is present in this table, the records will be processed following the procedure as described in paragraph 3.3.3. A set of tests

for the evaluation of the method is defined and implemented. The following tests are available:

- *Loading and Gridding timing* — In this test, it is measured how much time is needed to grid and order the data into the cells.
- *Convert to Spatial and Indexing* — In this test, the time needed for converting the ordened data into spatial objects and indexing the objects is measured.
- *Storage Requirements* — In this test, the storage requirements for the objects and index are measured.
- *Local or Global* — In this test, the difference between storage of local coordinates and global coordinates is tested.
- *Special Query* — In this test, the time needed for performing a special query is tested, which involves the use of different query windows to show trends related to query window size.
- *Comparison to SDO_PC* — The same tests on comparable cell sizes (with respect to the number of points) are executed on SDO_PC objects.

The gridding process into cells is based on a Quadtree with Morton space-filling curve. During the gridding process, the above mentioned tests are executed at every cell split occurrence, in other words for every level of gridding, starting when the maximum number of points in one grid cell is below 349,525 (see Graph 1). With the current test data, the Morton space-filing curve has 10 levels.

This will result into many graphs that can be compared to each other for further analysis. Per test, the best cluster-size parameter will be chosen, but the results from the special (range selection) query are crucial. The aggregated test results provide an overall image on the behaviour of the generic point cloud data model with large point cloud data sets. The tests with SDO_PC also results into graphs, advice on the use of SDO_GEOMETRY or SDO_PC can be provided by analyzing the results.

5. Testing and Results

This chapter reviews the testing results in the same order as they are executed, starting with the loading and gridding in § 5.1, followed by the conversion to spatial data in § 5.2. It might be good to store points in a more compact way or to remove information that is the same for all points. One possibility to reduce data is to store local coordinates only. The difference with respect to the required storage capacity, between storing points locally or in a global reference system is tested, results are in § 5.3. Loading, gridding and converting to spatial data has to be done once, but executing spatial queries is done very often so this should be fast. The results from taking spatial queries with help of indexes is in § 5.4. Finally, in § 5.5 the results will be compared with similar tests while the point cloud data is stored with the new developed data type SDO_PC.

5.1 Loading and Gridding

The first step is to load the data into the database. This is done by populating the input table (see Appendix A.1). Populating can take a while when the data file to be loaded is not on the same server as the database. Then communication is the bottleneck. In the test set-up client and server are on the same machine and it takes over one ten minutes to load 17 million points (*x*,*y*,*z*,*i*,*r*,*g*,*b*). If the server has a better performance, specifically the hard drive speed, this can be much shorter.

The AHN-2 project is subdivided into projects of about 20 million points, and will be delivered in AHN Standard Subunits of 1 km × 1.25 km, one subunit is tested within this research. If all data of The Netherlands, that is the entire AHN-2 product, has to be loaded into the database a solution has to be found to load the data files parallel, otherwise it would take too long to load the data files sequentially.

***Conclusions**:*
- *Loading is faster when the data file is on the same machine as the database. This depends completely on the bandwidth of the client-server communication.*
- *Duration of loading the data file depends on the performance of the server.*

***Recommendation**:*
A solution has to be found to load the data files parallel.

For creating point clusters, the second step is gridding the data in a way that points that are next to each other, are in the same cell. Once gridding is performed, each point has a temporary attribute identifying the cell to which it belongs. The input table will be indexed on the temporary cell attribute to speed up the conversion to spatial data. Beforehand, it is not known which points belong to the same object, the real points and objects are not visible at all, there is only a list of coordinates.

In this research, the data is split (into cells) using a combination of Region Quadtree and the Morton space-filling curve. Hilbert space-filling curve might be better, but is more complex to implement. This is done in twelve steps. In each step the number of points in each cell is counted, and if there are still too many points in a cell, it is split in four. Whenever possible, the test process is run after each step. Because an SDO_GEOMETRY data type can only hold

349,525 points in 3D, this was possible after the fourth step. That is level 4 (256 records). At the start of the splitting process all the points are in one cell. The minimum and maximum values of the points form the minimum bounding box of the cell. For the conversion to spatial data these values, together with the counted number of points in the cell and the Morton space-filling curve number of the cell were recorded in another table. In each step a cell is splitted into four equal cells, each of which is given a number following the Morton space-filling curve, the attribute pointing to the new cell number is updated for all the points in the cell. The results of each step are stored in a table for the conversion. In each following step this process is repeated, and each time the number of points in the cell is counted, the Morton space-filling curve number of the cell is updated.

When a line is drawn through the mid-point of all the cells in the number order according to alphanumeric ordering, the Morton space-filling curve becomes visible. Alphanumeric ordering is required, otherwise the cells that are not at the deepest level are not placed at the correct position on the curve, but at the beginning. Table 6 shows the difference between numeric and alphanumeric ordering.

Table 6: (Alpha-) numeric ordering, with cells on different levels.

Random ordering	4	2	31	34	1	32	31
Numeric ordering	1	2	4	31	32	33	34
Alphanumeric ordering	1	2	31	32	33	34	4

According to Oosterom [1990], the Morton space-filling curve numbering is executed by bitwise interleaving of the two coordinates resulting in a one-dimensional key. For this research, the numbering is numeric and starts with 1. Figure 15 show the starting situation (before the splitting process) along with the results after the first, second and third step and the creation of the Morton space-filling curve.

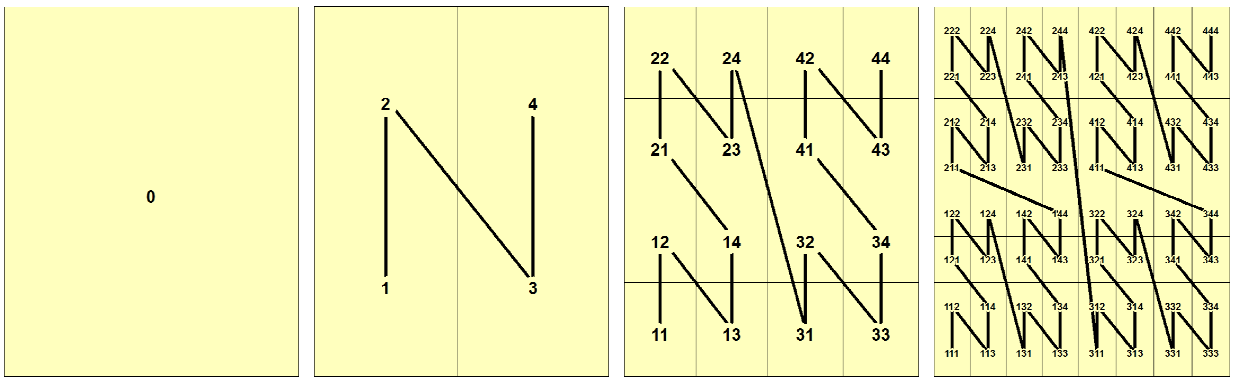
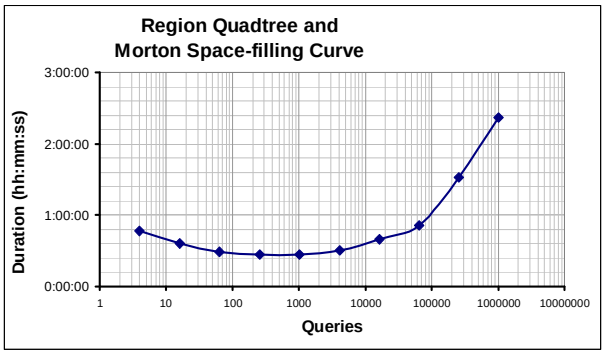


Figure 15: Region Quadtree and Morton Space-filling Curve.

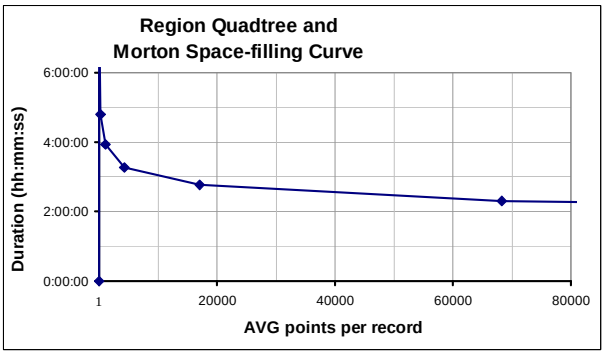
By splitting the cells in this way, few queries are performed on the dataset in the first steps of the splitting process, but large numbers of points are updated. In later steps, many queries are performed on the dataset, but a few points are updated per query. When the elapsed time of the update is compared to the number of queries performed using a logarithmic scale for the number of queries, a trend becomes obvious. (see Graph 2).

Performing the splitting process using this technique, results in Graph 3, where the time and cost

to process the data is compared with the average number of points per cell. The code used to do this work is shown in Appendix A.1 and A.2.



Graph 2: Trendline Queries.



Graph 3: Result - Region Quadtree and Morton Space-filling Curve.

It is not fair to extract conclusions from these graphs as the process in reality will be done differently. By using this method on a real life project, the start level of the splitting process can be determined (see Eq. 2) based on the total number of points in the dataset and the required maximum cell size. Cells that contain more than the required number of points, can be again divided, and cells that contain insufficient points can be combined, assuming they belong to the same parent cell and the parent cell also has less than the required number of points. But this estimation will only work if the distribution of the data is more or less homogeneous.

$$\text{Level} = \left\lceil \frac{ln\left(\dfrac{\text{Total points}}{\text{Points per cell}}\right)}{ln(4)} \right\rceil \tag{2.}$$

The formula shown above is for a 2D division. For Lidar (ALS) this is good, because this is more or less 2,5D, but as terrestrial laser scanning (TLS) is really 3D, the data can also be gridded 3D. It can be stated that as the cell has a substantial height, then the data needs a 3D division according to the Octree. The value 4 in the formula needs to be replaced with an 8. This is because eight cells are needed instead of four at each division. The resultant formula is shown here as Eq. 3. But this estimation too will only work if the distribution of the data is more or less homogeneous. Various implementation of the methods described above have been developed for splitting data in 2D and 3D.

$$\text{Level} = \left\lceil \frac{ln\left(\dfrac{\text{Total points}}{\text{Points per cell}}\right)}{ln(8)} \right\rceil \tag{3.}$$

Splitting cells to level 7 and an aerial photograph of the same area are shown in Figure 16. The waterways of Middelburg are easily found because the water is non-reflective and so the cells for those areas are not filled and even the use of the minimum bounding box produces no image. In the picture of the cell splitting it is possible to see large cells, these are the cells with less than the

required number of points and can therefore not be divided further. These cells are easy to see where there is water, as the laser data has, in general, a very low density which results in a low number of points in a cell.

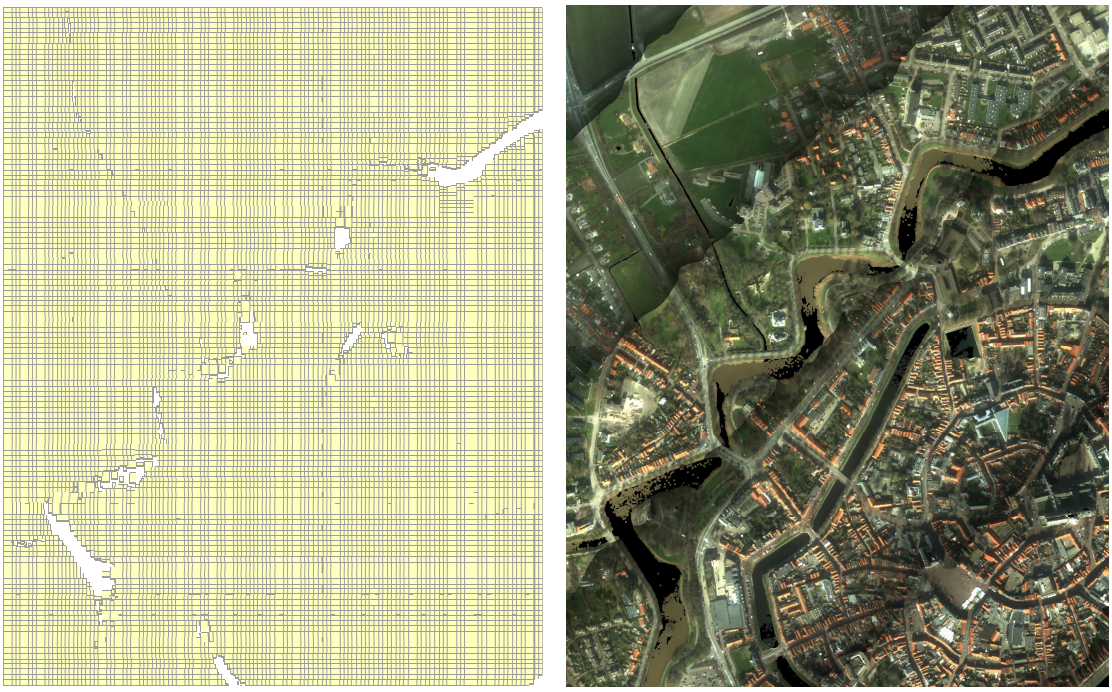


Figure 16: Region Quadtree Level 7 vs. Aerial Picture.
(the cell-size of the smallest cells is approx. 75 square meter)

The end result of splitting the cells is the input for the conversion to spatial points. To determine the correct cell size, a full set of tests are done after each step in the splitting process. The tests are performed based on the statistics in Table 7 below. Because a SDO_GEOMETRY object may only contain a maximum of 349,525 points, it is impossible to do tests before Level 4 (average of 68,274 points per cell).

Table 7: Gridding Results.

Level	*Cell Dimension (m x m)*	*Cell Area (m^2)*	*Average Points per Cell*	*Total Number of Cells*
0	1000 x 1250	1250000.000	17,478,245	1
1	500 x 625	312500.000	4,369,561	4
2	250 x 312.5	78125.000	1,092,390	16
3	125 x 156.25	19531.250	273,098	64
4	62.5 x 78.125	4882.813	68,274	256
5	31.250 x 39.063	1220.703	17,069	1,024
6	15.625 x 19.531	305.176	4,267	4,096
7	7.813 x 9.766	76.294	1,067	16,384
8	3.906 x 4.883	19.073	267	65,536
9	1.953 x 2.441	4.768	67	262,144
10	0.977 x 1.221	1.192	17	1,048,576

***Conclusions**:*

- *Duration of gridding and clustering the data depends on the number of records in the input table.*

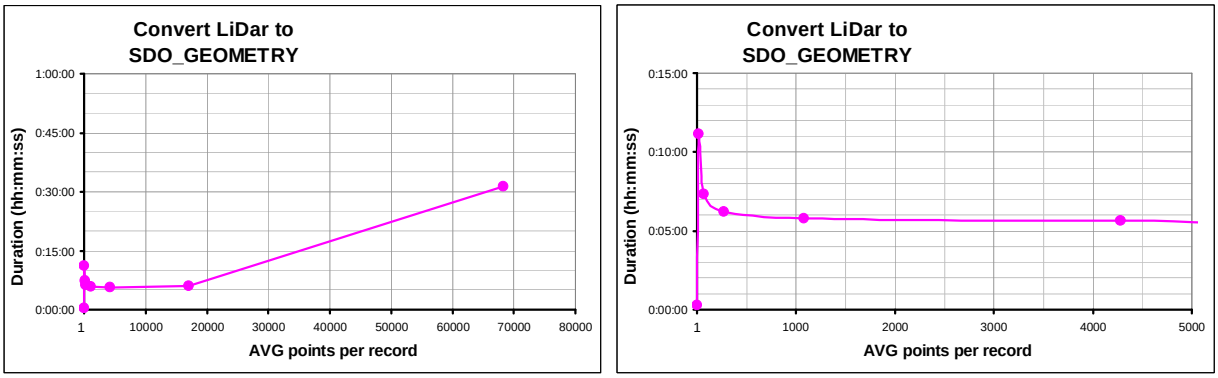
- *Implementing the Region Quadtree is a possibility to grid the data in logical parts.*
- *Implementing the Morton space-filling curve is a possibility to cluster the data. Hilbert space-filling curve would even be better.*

***Recommendations**:*
Based on the results from this thesis, the implementation of the method to grid and cluster the data using the Region Quadtree and Morton or Hilbert space-filling curve should be refined.

5.2 Convert to Spatial and Indexing

Once the loading, gridding and clustering are applied, the conversion to the spatial data type can be performed. For this conversion, a function is programmed (see Appendix A.3). A result of the cell splitting is a table with the minimum and maximum coordinates of each cell, the number of points in the cell and the number of the cell according to the Morton space-filling curve. This table is used as input for the conversion to spatial, by searching the table and selecting the corresponding records from the table of points for each cell, and then writing these in the database as one multi-point (SDO_GEOMETRY, GTYPE=3005, POINTCLUSTER). As one of the steps, points are also converted to single points. When only one point needs to be stored in a record an alternative (simple) conversion can be done using a "INSERT INTO .. AS SELECT …" statement, using SDO_GEOMETRY, SDO_POINT.

Graph 4 shows the relation between the duration of the execution of the function and the number of points per record (this is the only variable in this function) for a constant number of points. The sharp bend in the curve, at approximately 16,000 points per record is a "visualisation artefact" due the method of gridding such that no measurements are possible between 17,000 and 68,000 points per record.

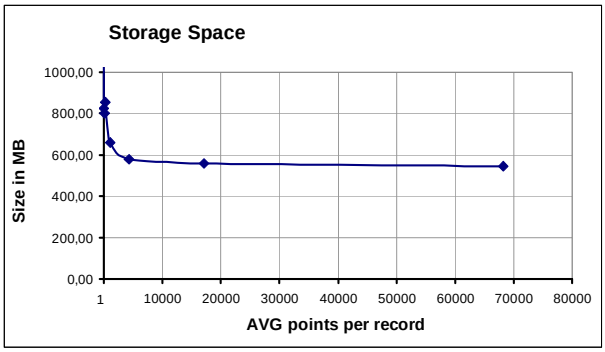


Graph 4: Convert to Spatial Data (right: zoom-in to part of graph).

The tests show, the shortest conversion time occurs when the records have an average of about 4000 points (see detail on right), but durations are nearly equal between 300 and 16,000 points per record. Above 1000 points per record the conversion time is less than six minutes.

The size of the tables containing spatial data is measured, with SQL-code designed by Theo Tijssen (Appendix B.5). The results are in Graph 5. As can be seen in this graph, storing the tested dataset containing 17,478,245 points with intensity, red, green and blue attributes converges to 550 MByte. Above an average of 4,000 points per record the storage space drops to

a size of 600 MByte. Whereas the storage of only one point (SDO_GEOMETRY, SDO_POINT) per record costs as much as 1,500 MByte. AHN-2 will contain at least 415 billion points, this will result in a dataset of 13 TByte, when storing 4000 points per cell.



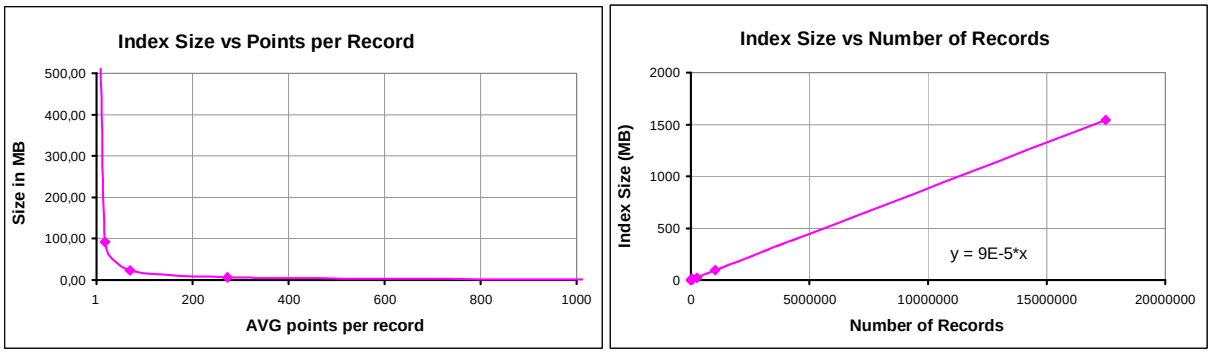
Graph 5: Storage Space.

Once the data is converted into spatial objects (SDO_GEOMETRY), the tables can be indexed using a R-tree spatial index (default in Oracle Spatial) (see SQL-Code 6).

SQL-Code 6: Oracle – Create Spatial Index.

```
CREATE INDEX MyIndex ON MyTable (GEOM) INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

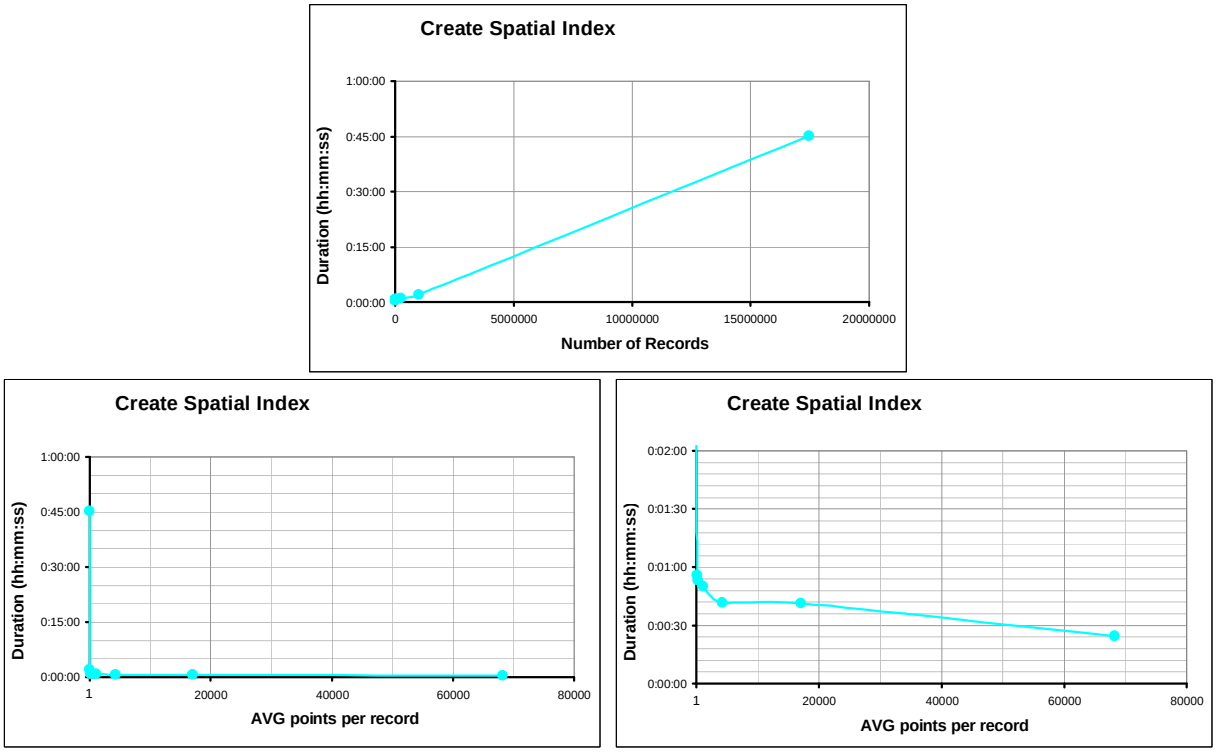
Because a spatial index is considered as a logical index, the size and the duration of creating it will be very much related to the number of records. Graph 6 (left) shows the measured space for an index with the number of points per record as its only variable. Within the testing environment the relation between the number of records and the size of the index was about 1 MByte per 11,320 records, see Graph 6 (right). Decreasing the number of records has great effect on the index size, because the number of cells is growing. Note that the relation between number of records and index size is linear.



Graph 6: Index space (left) – Relation (right).

The duration of creating a spatial index, just like the index space, is dependent on the number of records. (see Graph 7 – top). Reducing the number of records greatly effects the duration. With one point per record (SDO_GEOMETRY, SDO_POINT) it takes Oracle about 45 minutes to create the index (See Graph 7 – left). Above 500 points per record this has dropped to less than a minute (See Graph 7 – right). Also from this test, it was obvious that it is inefficient to store

singe points in the database as opposed to multi-points (SDO_GEOMETRY, POINTCLUSTER), as can be seen from the large space that the index requires in the database and the time it takes to make it.



Graph 7: Create Spatial Index.

***Conclusions**:*

- *Conversion from the input table to spatial data is done very quickly (17 seconds for 17,478,245 points) when each record has only one point (SDO_GEOMETRY, SDO_POINT).*
- *When the number of points per record is between 1000 and 17000 the conversion takes less than 6 minutes 17,478,245.*
- *It seems that the optimum is reached at about 4000 points per record.*
- *Above 4000 points per record, the storage space required for the spatial data drops below the 600 MByte mark, with a limit at 550 MByte.*
- *The space required to store the index, as well as the time taken to make it, is directly proportional to the number of records in the spatial data. The less records the better. Therefore, the computational complexity of this operation is O(n), with n = number of cells*
- *Above 100 points per record, the duration (bellow 1 minute) of the creation and storage of the index has little impact on the time taken for the conversion (below 10 MB) to spatial data, or the space required for storage of the spatial data, and its index.*

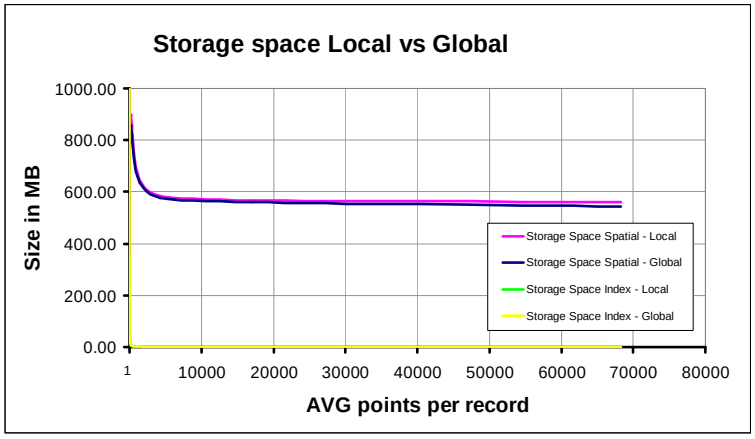
5.3 Local or Global

The next test is to find the difference between storing points with global or local coordinates, in other words, storing the points in the Netherlands in the National RD-NAP reference system or defining a central point as the mean *x,y,z* and store all points as a difference with respect to that

central point. For the tested dataset, three situations can be distinguished:

1. Storing points global
2. Storing points local with respect to a central point per record
3. Storing points local with respect to a central point per input file.

Situations 1 and 3 are tested (see Graph 8), and it is shown that as the SDO_GEOMETRY data type has a predefined format, it can be noted that the difference between global and local are very small (negligible). With Oracle it seems insufficient to store points local compared to the extra work that is needed to convert the data from global to local and back. Particularly as it appears from the test, that more storage space is required in situation 3, which is not logical. It is expected, the table is chained due to the insert statement that has been used to insert the data. This can be solved by moving the table to a new segment in the database.



Graph 8: Storage Space – Local vs. Global.

***Conclusion**:*
Using the SDO_GEOMETRY data type does not improve the storage space when spatial data is stored using a local coordinate system.

5.4 Spatial Query

Within this research for every level of gridding (see Table 7), six queries are tested, all with the same Oracle-function, but with a different query-window size to be able to show trends. The query windows is defined as a circle, using the SDO_WITHIN_DISTANCE operator. All queries use the index to reduce the duration. Preliminary tests showed that using functions without index-support have a duration of over one hour instead of less than two minutes. The results where inserted into a newly created table. The result from every test is checked by counting the number of records that where selected, this count is multiplied by the average number points per cell to get an estimation of the number of points selected.

In SQL-Code 7 the special query is printed with the following variables;
- <spatial_table> Name of the table from which records must be selected;
- <result_table> Name of the table to store the results;
- <x/y_center> xy-coordinates of the middle point of the query window;
- <distance> Radius of the query-window.

SQL-Code 7: Oracle – Special Query.

```
CREATE TABLE MyTable_Result AS SELECT A.* FROM MyTable A WHERE
  SDO_WITHIN_DISTANCE(
    A.GEOM,
    SDO_GEOMETRY(2001,28992,SDO_POINT_TYPE(<x_center>,<y_center>,NULL),NULL,NULL),
    'distance=<distance>') = 'TRUE';
```

In Figure 17, the gridding for the whole test dataset, is shown in light yellow blocks, and the query-window is shown in the red circles. Figure 18 shows the selected records in pastel colours. It can be seen that by the largest grid-cells (Figure 18 - left) cells are often selected that are partially within the query window. These cells also contain many points outside the query-window, and so should not actually be selected. With smaller grid-cells (Figure 18 - right) less points are selected outside the query-window.

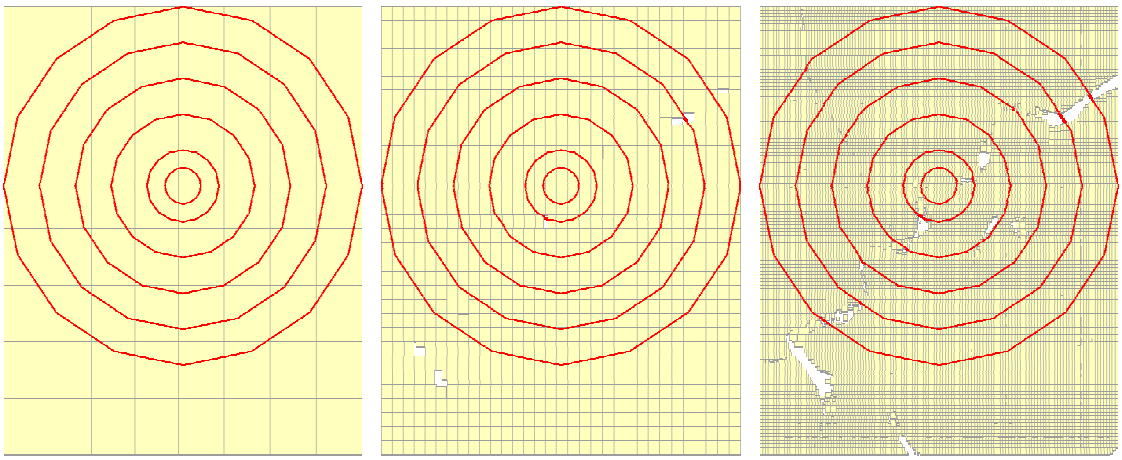


Figure 17: Gridding with Query-windows.

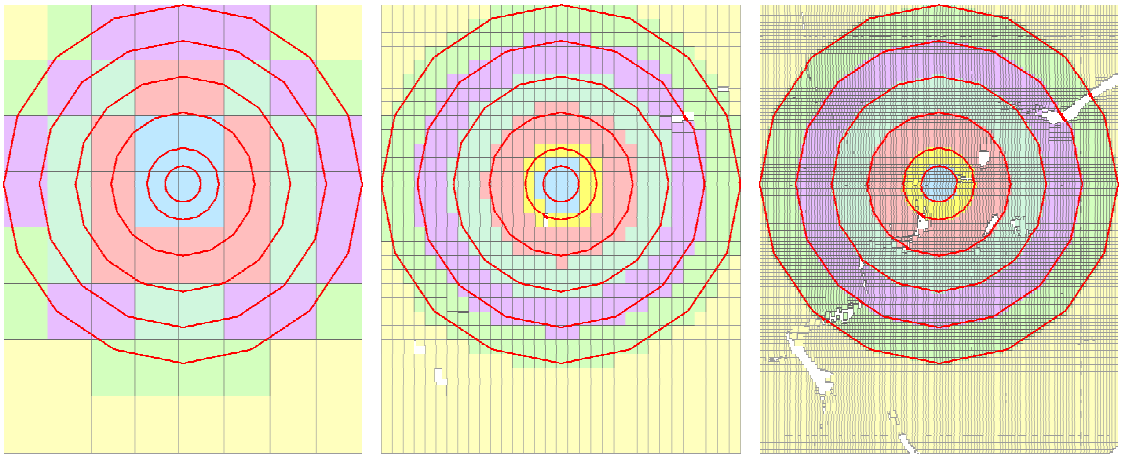
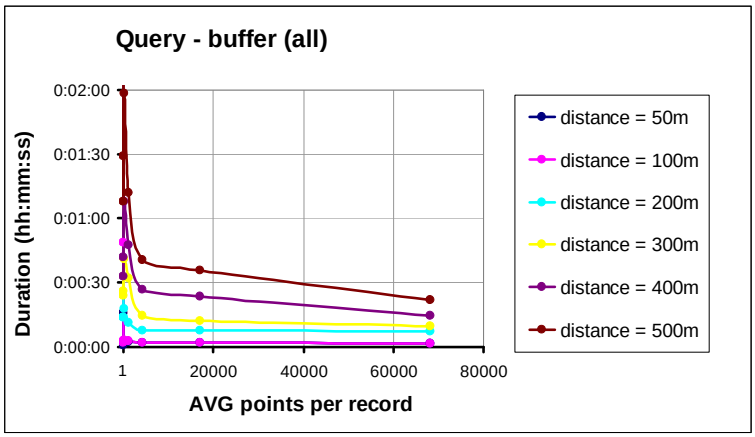


Figure 18: Query-windows with Query-results.

In the case of queries that aim at visualising data or performing measurements within the data, selecting superfluous points outside the query window is not a big problem, provided that the total query time needed to show the result is not influenced negatively. However, when the points are selected for other reasons, for instance the detection of surfaces, then it is important to

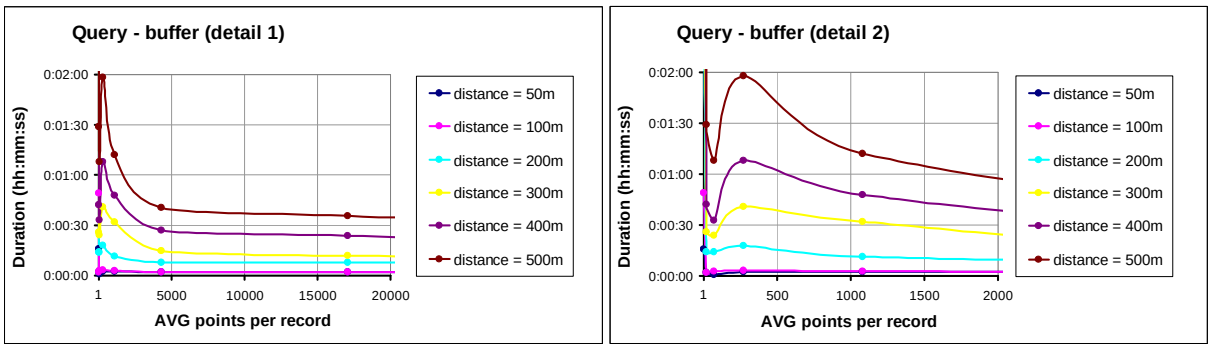
reduce the number of points that most likely do not belong to the object as much as possible. In the first case, this leads to a choice for cells that are as big as possible. For the second query, the choice would be for smaller cells. In both cases, however, the final answer will be returned correctly, but with a potential time penalty.

By asking practically the same query for different query windows, it is possible to detect a trend in the results related to the query window size, especially related to the time needed to complete the query. In Graph 9 the running times of the executed queries are plotted against the average number of points per record.



Graph 9: Query-results (all).

Due to the method of gridding that was applied, the measurements in this test are executed on a limited number of cell sizes. Smooth lines were drawn through all the measurements, even though this is not entirely realistic, because there are no real measurement values available at the intermediate locations between the measurement points and also the opportunities for interpolation are limited. However, the graph still allows deriving some conclusions, especially when zooming in on the smallest grid sizes (Graph 10). Then, we see that a minimum exists at an average of 69 points per record and a maximum exists at an average of 273 points per record. It is remarkable that these two grid sizes have such a large deviation from the larger trend and that these deviations are measured in all query-window sizes. The reason why these deviations exist remains unknown. Possibly, the cause is relates to the fact that records with an average of 69 points just fit in one complete record in the database, while subsequent points require additional tables to fit the records with an average of 273 points.



Graph 10: Query-results (details).

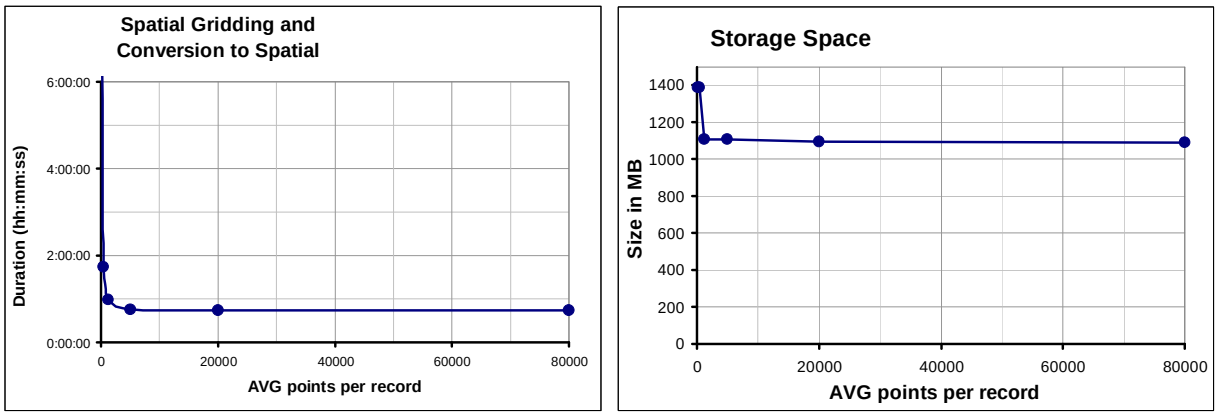
***Conclusion**:*
A clear trend becomes visible by executing the same query at different window sizes. This allows to provide a better answer to the question of what is the ideal cell size. Since the observed minimum and maximum values are located relatively close to each other, for a reason yet unknown, it seems more appropriate to select a cell size of more than 2000 points. At this value, the running time will be almost as short as at the described minimum, but the risk of hitting the almost neighbouring maximum is then limited.

5.5 Compare with SDO_PC Data Type

In section 4.3, it was described how the results from the previous section are compared to the new SDO_PC data type. The average group sizes from the test with SDO_GEOMETRY were taken as a starting point.

In Graph 11 (left) it is shown that the gridding and conversion to the final format is significantly (45 minutes versus over two hours) faster than the method described in section 5.1 and 5.2. In contrast to what was described in section 5.1, the gridding and conversion to the final point cloud is implemented as part of this data type, which may explain the better results.

In Graph 11 (right) it can be seen that it takes more storage space in the database to store point cloud with the SDO_PC data type, compared to the method described in the previous sections, i.e. 600 MB versus 1100 MB. It is difficult to discover the reasons for this difference. Possibly, the answer relates in the method of storing the data. The point cloud coordinates are stored in a binary format and the spatial extend as SDO_GEOMETRY, but possibly also other aspects are stored.

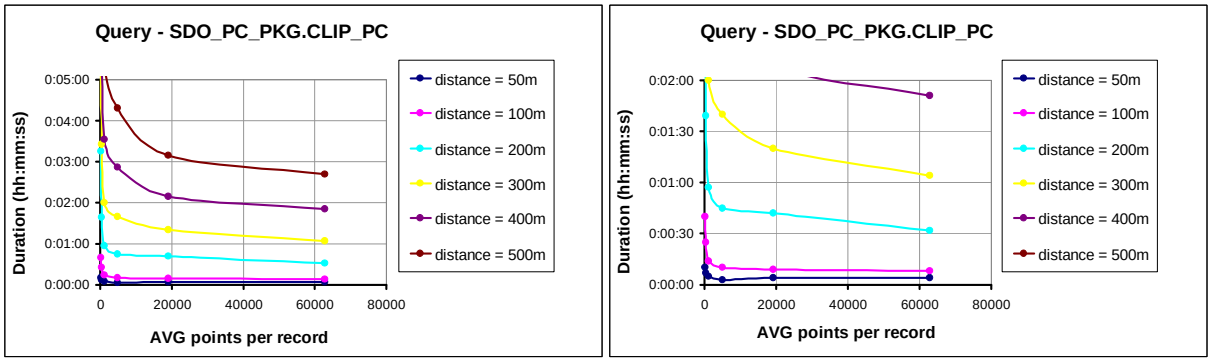


Graph 11: Conversion point clouds to SDO_PC data type, duration (left) and storage space (right, incl. index).

The querying of SDO_PC objects is executed in two phases. In the first phase, the records will be clipped with the function SDO_PC_PKG.CLIP_PC. Then, the selected records can be converted to spatial (SDO_GEOMETRY) multi-points in order to achieve a “readable” result.

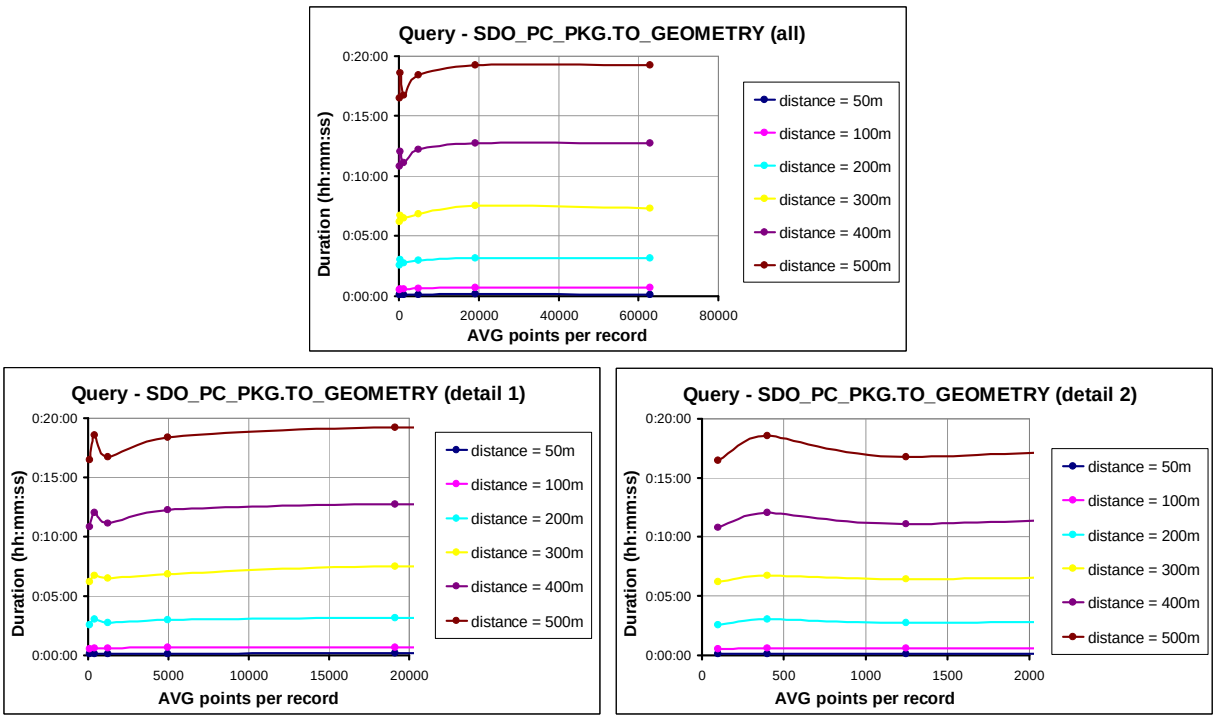
In a method similar to what was described in section 5.4 and with a similar query to select the same point, a total of 6 queries were executed on the data. The first phase results in Graph 12 (to be compared with Graph 9). In the original query the result was available within 45 seconds (SDO_GEOMETRY), starting from 10,000 points per record. Even without the “unwrapping”

and conversion of the result to a “readable” result, the total execution time at over 10,000 points per record (SDO_PC) takes less than 4 minutes. Although the points are gridded differently, is can be stated that the same points are selected, because the selection windows are equal. Graph 12 (right) shows the duration of the CLIP_PC operation with the same y-scale as in Graph 9.



Graph 12: Results of SDO_PC_PKG.CLIP_PC

The total running time for executing the second phase is hardly depended on the number of points per record, but mostly on the number of selected points. Because it is impossible to gain insight in the inner workings of Oracle, the dips at 100 and 1250 points per record, as well as the maximum at 400 points per record, cannot be explained. However, it is clear that the dips and the maximum occur at each of the query that was executed.



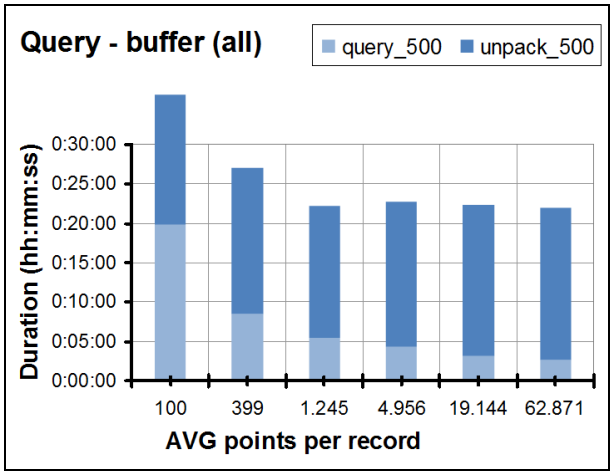
Graph 13: Results of SDO_PC_PKG.TO_GEOMETRY

After completing SDO_PC_PKG.TO_GEOMETRY, the largest limitations of SDO_PC become apparent:

- Coordinates and attributes are being converted to ordinates in SDO_GEOMETRY format with SDO_GTYPE = 7005 if the following values are stored per point: X, Y, Z, I, R, G, B.

The 7 in the SDO_GTYPE identifier defines the number of dimensions, or attributes in this case. An object in SDO_GEOMETRY can be 8 dimensional at most, hence the SDO_PC format is limited to 1 coordinate with 5 attributes.

- All coordinates and attributes are being stored in the SDO_ORDINATE_ARRAY. This array is defined as a VARRAY with 1048576 numbers. With an SDO_GTYPE = 8005, this means that an SDO_PC object may contain at maximum 131,072 points. In addition, all point attributes should be numeric.
- Unwrapping the SDO_PC objects takes relatively much time (see Graph 14), compared to executing the query itself. However, unwrapping the SDO_PC data type is often a necessity as SDO_PC objects are not supported by specialist applications (yet).
- The data type SDO_PC is not implemented in Oracle 11.1g Express Edition. It is implemented in Oracle Spatial, which is a priced option for Oracle Enterprise Edition.



Graph 14: Oracle - SDO_PC - Query and Unwrap.

***Conclusions**:*

- *Storing a point cloud of the data type SDO_PC, compared to storing the points in a SDO_GEOMETRY point cloud has just one advantage: the execution of the gridding, conversion to SDO_PC and indexing are implemented within the function SDO_PC_PKG.INIT and SDO_PCPPKG.CREATE_PC.*
- *The required storage capacity in the database seems to be double.*
- *It takes longer before the results of a query are known, especially because the first result from a spatial query is not readable to every program (45 seconds versus over 20 minutes for cells containing 16,000 points per record).*
- *The conversion to the readable SDO_GEOMETRY takes relatively much time.*
- *SDO_PC data type is limited to points with maximum 5 attributes.*
- *When using 5 attributes, the SDO_PC type is limited to 131,072 points per record.*
- *All attributes have to be of format NUMBER.*

6. Conclusions, Recommendations and Further Research

This chapter contains the conclusions (6.1), recommendations for further research in § 6.2. § 6.3 contains a reflection on the research.

6.1 Conclusions

This research is executed to give an answer to the question:

What is the best design for a data model to store large point clouds in a DBMS, such that it is generally accessible by spatial applications, that all attributes are preserved and that performance is optimised?

A modern point dataset can easily reach a billion (10^9) up to trillion (10^{12}) points and it may contain attributes such as colour, intensity, classification and time.

In this thesis a new data storage model, based on SDO_GEOMETRY (by using POINTCLUSTER) is presented to store these point datasets, called point cloud datasets. This method performs optimal for large point cloud datasets in terms of performance and the point cloud can generally be accessed by spatial applications. The new method preserves all attributes related to point clouds and works in a DBMS.

Data in a spatial DBMS requires special data types. In Oracle a spatial data type called SDO_GEOMETRY is implemented, this data type is accepted as a well-known data type for storing spatial data. Many spatial applications can read or write this data type.

Two different ways of storing points in a database were proposed and tested: either as single points (SDO_GEOMETRY, SDO_POINT) or grouped in multi-points (SDO_GEOMETRY, POINTCLUSTER). The advantages using multi-points, as found in the experiments in this thesis, are:

- Creating indexes is much faster, because of the reduced number of records.
- Less storage overhead, which reduces the total storage space required.
- Smaller indexes which require less storage space as well.
- Much faster querying results.
- Even better results in terms of query time for nearest neighbour questions.

The disadvantage of grouping points in point clouds is that the grouping and ungrouping of the data takes extra time.

Implementing the Region Quadtree is a possibility to group the data in logical parts. Implementing the Morton space-filling curve is a possibility to order the data. Hilbert space-filling curve would even be better.

In addition, it can be concluded from the tests that:

- Duration of gridding and clustering the data depends on the number of records in the input table.
- The storage space does not improve when spatial data is stored using a local coordinate system using the SDO_GEOMETRY data type.

- Conversion to spatial data is done very quickly when each record has only one point (SDO_GEOMETRY, SDO_POINT).
- When the number of points per record is between 1000 and 17000 the conversion takes less than 6 minutes for 17,478,245.
- It seems that the optimum conversion speed is reached at about 4000 points per record, but durations are nearly equal between 300 and 16,000 points per record.
- Above 4000 points per record, the storage space required for the spatial data drops below the 600 MByte mark, with a lower limit at 550 MByte.
- The space required to store the index, as well as the time taken to make it, is directly proportional to the number of records in the spatial data. The less records the better. Therefore, the computational complexity of this operation is $O(n)$.
- Above 100 points per record, the creation and storage of the index has little impact on the time taken for the conversion to spatial data, or the space required for storage of the spatial data, and its index.
- A clear trend becomes visible by executing the same query at different scales. This allows to provide a better answer to the question of what is the ideal cell size. Since the observed minimum and maximum values are located relatively close to each other, for a reason yet unknown, it seems more appropriate to select a cell size of more than 2000 points. At this value, the running time will be almost as short as at the described minimum, but the risk of hitting the almost neighbouring maximum is then limited.

It can be concluded that the number of points in one record is at least 2000 points and the records have an average of 4000 points per record.

An alternative storage method was developed by Oracle and is known as SDO_PC. Advantages of the newly developed storage method (with respect to SDO_PC) are:

- The required storage space for storing a point cloud in the SDO_GEOMETRY data type is half the amount of the space required for storing the same point cloud with the SDO_PC data type.
- Querying an amount of data is quicker when compared to querying the same amount of data from the SDO_PC storage method (45 seconds versus over 20 minutes for cells containing 16,000 points per record).
- The SDO_GEOMETRY data type can generally be accessed by spatial applications, SDO_PC is newly implemented in Oracle and not (yet) fully supported by all vendors.
- The SDO_GEOMETRY data type is unlimited to the number of attributes per point, and can therefore store all attribute data that belongs to a point cloud according to the LAS-specifications.
- If the maximum number of attributes are applied, the maximum point cloud size for SDO_GEOMETRY is 349,525, whereas the SDO_PC data type with *x,y,z* and 5 other attributes is limited to 131,072 points.

Storing the point cloud with data type SDO_PC has the advantage, that all required functions to convert the point cloud are included in Oracle.

Therefore, it is concluded that the SDO_PC data type is not suitable for the conditions as stipulated in the research question above.

Besides Oracle, another mainstream database for spatial data is PostgreSQL. At this moment, PostgreSQL does not have a dedicated data type for point clouds. However, it is technically feasible to implement the new SDO_GEOMETRY based point cloud data storage method in PostgreSQL, as PostGIS (spatial implementation in PostgreSQL) supports also multi-points.

6.2 Recommendations and Further Research

Based on this research a few recommendations can be done:

- By better programming, the duration of loading, gridding, and converting can be improved.
- The Hilbert space-filling curve should be implemented for even better clustering.
- As terrestrial laser scanning (TLS) is really 3D, the data must be gridded 3D. It can be stated that as the cell has a substantial height, then the data needs a 3D division according to the Octree. The code can easily be rewritten to support 3D gridding and clustering.
- The code can easily be rewritten to save every kind of attribute in a separate array in a separate column.
- This research is focussed on spatial queries for validating the data model. A method is required to query other attributes.
- The code can easily be rewritten to be implemented into PostgreSQL.

From both the academic users as well as non-academic users, there is a growing need for the development of an infrastructure for the storage, the management, the visualisation and the manipulation of point cloud datasets such as AHN-2. This was also addressed by Kodde [Oosterom et. all, 2010]. This infrastructure should be able to handle these very large datasets and enable users to work with it by reducing query time to a minimum. This thesis can be the starting point of this further research.

By storing points in point clusters, some extra functions are needed for finding points that are close to each other can be reduced by knowing that points that are close to each other must be in the same record as the point itself or in neighbouring records. When selecting multi-point objects, sometimes a precise result is needed. New functions are required to query points and attributes inside a multi-point.

In § 3.3.2 the subject in-object clustering is defined as a way to order points inside a multi-point as a means of implicit generalisation. In effect, this means that the most descriptive points of the area covered by the multi-point are stored as the first points within the multi-point. A decreasing order of importance should be applied here. Now, if the data needs to be generalised, the user can choose to unwrap only the first few points from the multi-geometry object. The number of points to unwrap depends on the zoom level of the spatial display window and the processing power of the machine. The ranking of the points can be determined using various point cloud filtering techniques that highlight the most characteristic points. If such a filter is not available, a spatial distribution within the area should be used, i.e. the first point should be in the centre, the next 4 points near the edges, etc. This can be a subject for future research.

6.3 Reflection

During the research period, which took a long period, the research subject was changed. But due to new techniques the research is still useful.

In the engineering world, for sure within Fugro there is a growing need to share post-processed laser data around the world. Sending harddisks by airplane is outdated, but as the data volume is growing, it seems the fastest way to send data to other offices inside the company. Within the company, R&D is working on a solution to share raw data together with correction parameters, to be able to do the post-processing on the client system instead of on the server.

References

Bibliography

- ASPRS (2012) *LAS Specification version 1.4 – R12*, American Society for Photogrammetry and Remote Sensing.
- Bentley Benelux (2006) *Handleiding Workshop Oracle Spatial 10g*, Versie 2.3, Bentley Benelux Culemborg, Nederland.
- De By R.A. (ed.) (2001) *Principles of Geographic Information Systems*, Second Edition, ITC Educational Textbook Series; 1, ITC Enschede, ISBN 90-6164-200-0
- Böhler, W. and A. Marbs (2002) *3D scanning instruments*, Proceedings of the CIPA wg6 international workshop on scanning for cultural heritage recording, ISPRS commission V, pp. 9-12 (also available at http://www.i3mainz.fh-mainz.de).
- Brinkhoff, T. and H.P. Kriegel (1994) The impact of global clustering on spatial database systems, In Proceedings of the Twentieth International Conference on Very Large Data Bases, pp. 168–179.
- Claassen E.P. (2003) *Registratie van airborne en terrestrische laserdata*, Afstudeerscriptie, Technische Universiteit Delft – Fugro GeoServices B.V.
- Gaede, V. and O. Gunther (1998) *Multidimensional access methods*, ACM Computing Surveys (CSUR), 30(2): 170-231.
- Hilbert D. (1891) *Über die stetige Abbildung einer Linie auf ein Flächenstück*, Mathematische Annalen 38, pages 459-460.
- Höfle B. (2005) *Entwicklung eines Informationssystems für Laserscannerdaten mit open source Software*, Unveröffentlichte Diplomarbeit am Institut für Geographie der Universität Innsbruck.
- Lemmens M.J.P.M., Lohani B. (2001) *Geo-Information from LiDAR - How India May Benefit from Airborne Laser-Altimetry*, GIM International, July 2001, pp. 30.
- Liu H., Huang Z., Zhan Q., Lin P. (2008) *A Database Approach to Very Large LiDAR Data Management*, ISPRS 2008, pp. 463 – 468.
- Morton G. M. (1966) *A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing*, Technical Report, Ottawa, Canada: IBM Ltd.
- OGC (1999) Open Geospatial Consortium, *The OpenGIS Simple Features Specification for SQL*, Technical Report Revision 1.1, OpenGIS Project Document 99-049, Open Geospatial Consortium, Wayland, Mass., USA.
- OGC (2000) Open Geospatial Consortium, *The OpenGIS Abstract Specification, Topic 0: Abstract Specification Overview*, Version 4, Technical Report, OpenGIS Project Document Number 99-100r1, Open Geospatial Consortium, Wayland, Mass., USA.
- Oosterom P.J.M. van (1990) *Reactive Data Structures for Geographic Information Systems*, Proefschrift Rijksuniversiteit Leiden.
- Oosterom P.J.M. van, Vosselman M.G., Dijk Th.A.G.P. van and Uitentuis M. (Editors) (2010) *Management of massive point cloud data: wet and dry*, NCG 49, ISBN: 978-90-6132-322-8.
- Oracle (2010) *Oracle Spatial - Developer's Guide - 11g release 2 (11.2)*, E11830-07.
- Ott M. (2012) *Towards Storing Point Clouds in PostgreSQL*, Seminar Database Systems, Master of Science in Engineering Major Software and Systems, HSR Hochschule für Technik Rapperswil.
- Peano G. (1890) *Sur une courbe, qui remplit toute une aire plane*, Mathematische Annalen 36, pages 157–160.
- PostGIS (2012) *PostGIS 2.0.1 Manual*, pp. 24-51.
- Schön B., Bertolotto M., Laefer D.F., Morrisch S.W. (2009) *Storage, manipulation, and visualization of lidar data*,

- Vijlbrief T., and Oosterom P.J.M. van (1992) *GEO++: An extensible GIS*, In proceedings 5th International Symposium on Spatial Data Handling, Charleston, South Carolina, pages 40-50.
- Wang, J., and Shan J. (2005) *Lidar Data Management with 3-D Hilbert Space-filling Curves*, ASPRS Annual Conference, Baltimore, USA.
- Wehr A., and Lohr U. (1999) *Airborne laser scanning - an introduction and overview*, ISPRS Journal of Photogrammetry & Remote Sensing 54, pp. 68-82.

URL's

- *AHN*, last visit: 2012-09-17,
  URL: http://www.ahn.nl/wat_is_het_ahn
- *ASPRS*, last visit: 2012-09-17,
  URL: http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html
- DBA Metrix Solutions, last visit: 2012-09-19,
  URL: http://www.dbametrix.com/ora-unix-linux-win.html
- *Dive and Discover*, last visit: 2012-09-17,
  URL: http://www.divediscover.whoi.edu/tools/sonar.html
- *FLI-MAP*, last visit: 2012-09-17,
  URL: http://www.fli-map.nl/about-flimap/fli-map
- *Indexmundi*, last visit: 2012-09-17,
  URL: http://www.indexmundi.com/netherlands/area.html
- *Microsoft SQL Server 2012*, last visit: 2012-09-17,
  URL*:* http://msdn.microsoft.com/en-us/library/bb933790.aspx
- *Oracle*, last visit: 2012-09-17,
  URL: http://www.oracle.com/us/products/database/product-editions-066501.html
- *PostgreSQL*, last visit: 2012-09-17,
  URL: http://www.postgresql.org/about/

List of Tables

List of Figures

List of Graphs

List of SQL-Code

Appendix

Appendix A: SQL-code

A.1 Loading

```
DROP TABLE STRAIGHT7D;
CREATE TABLE STRAIGHT7D (QUAD_ID  NUMBER DEFAULT 0,
                         X        NUMBER,
                         Y        NUMBER,
                         Z        NUMBER,
                         I        NUMBER,
                         R        NUMBER,
                         G        NUMBER,
                         B        NUMBER);
```

```
LOAD DATA
INFILE '05_31000_391250.xyz'
APPEND INTO TABLE STRAIGHT7D
FIELDS TERMINATED BY ' '
TRAILING NULLCOLS
(QUAD_ID CONSTANT '0',
 X "replace (:X, '.', ',')",
 Y "replace (:Y, '.', ',')",
 Z "replace (:Z, '.', ',')",
 I CONSTANT '255',
 R CONSTANT '255',
 G CONSTANT '255',
 B CONSTANT '255')
```

```
DELETE FROM STRAIGHT7D WHERE X IS NULL;
DELETE FROM STRAIGHT7D WHERE Y IS NULL;
DELETE FROM STRAIGHT7D WHERE Z IS NULL;

ANALYZE TABLE STRAIGHT7D COMPUTE STATISTICS;
CREATE INDEX STRAIGHT_idx ON STRAIGHT7D (QUAD_ID,X,Y);
```

A.2 Region Quadtree and Morton Space-filling Curve

```
CREATE TABLE RESULT_QUAD
 AS SELECT QUAD_ID,
           COUNT(*) TOTAL,
           MIN(X) MIN_X,
           MAX(X) MAX_X,
           MIN(Y) MIN_Y,
           MAX(Y) MAX_Y,
           MIN(Z) MIN_Z,
           MAX(Z) MAX_Z
 FROM STRAIGHT7D GROUP BY QUAD_ID;

ANALYZE TABLE RESULT_QUAD COMPUTE STATISTICS;

DROP INDEX RESULT_QUAD_idx;
CREATE INDEX RESULT_QUAD_idx ON RESULT_QUAD (QUAD_ID);
```

```
CREATE OR REPLACE PROCEDURE GRID_STRAIGHT(input_table IN VARCHAR2, size_pointcluster
IN INTEGER, result_table IN VARCHAR2) AUTHID CURRENT_USER AS

-- Declarations
n_gem_x NUMBER;                                -- variable for mean x
n_gem_y NUMBER;                                -- variable for mean y
i NUMBER;                                      -- loop integer total
```

```
-- Begin
BEGIN

  EXECUTE IMMEDIATE 'ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ".,"';

  i := 0;

  FOR curs_rec IN (SELECT QUAD_ID,TOTAL,MIN_X,MAX_X,MIN_Y,MAX_Y FROM RESULT_QUAD WHERE
TOTAL > size_pointcluster)

  LOOP

    n_gem_x := (curs_rec.MIN_X + curs_rec.MAX_X)/2;
    n_gem_y := (curs_rec.MIN_Y + curs_rec.MAX_Y)/2;

    EXECUTE IMMEDIATE 'UPDATE ' || input_table || ' SET QUAD_ID = ' ||
curs_rec.QUAD_ID || '1 WHERE (QUAD_ID = '|| curs_rec.QUAD_ID || ' AND X < ' || n_gem_x
|| ' AND Y < ' || n_gem_y || ')';
    EXECUTE IMMEDIATE 'UPDATE ' || input_table || ' SET QUAD_ID = ' ||
curs_rec.QUAD_ID || '2 WHERE (QUAD_ID = '|| curs_rec.QUAD_ID || ' AND X < ' || n_gem_x
|| ' AND Y >= ' || n_gem_y || ')';
    EXECUTE IMMEDIATE 'UPDATE ' || input_table || ' SET QUAD_ID = ' ||
curs_rec.QUAD_ID || '3 WHERE (QUAD_ID = '|| curs_rec.QUAD_ID || ' AND X >= ' ||
n_gem_x || ' AND Y < ' || n_gem_y || ')';
    EXECUTE IMMEDIATE 'UPDATE ' || input_table || ' SET QUAD_ID = ' ||
curs_rec.QUAD_ID || '4 WHERE (QUAD_ID = '|| curs_rec.QUAD_ID || ' AND X >= ' ||
n_gem_x || ' AND Y >= ' || n_gem_y || ')';
    COMMIT;

    i := i+1;

    IF mod(i, 5000) = 0 THEN -- return every 5000 QUAD_ID updates
        DBMS_OUTPUT.PUT_LINE (TO_CHAR (i) || ' QUAD_ID updated sofar');
    END IF;


  END LOOP;

  DBMS_OUTPUT.PUT_LINE (TO_CHAR (i) || ' QUAD_ID updated in total');

END;
/
```

```
EXEC GRID_STRAIGHT('STRAIGHT7D',<maximum_size_pointcluster>,'RESULT_QUAD');

ANALYZE TABLE STRAIGHT7D COMPUTE STATISTICS;

DROP INDEX STRAIGHT_idx;
CREATE INDEX STRAIGHT_idx ON STRAIGHT7D (QUAD_ID,X,Y);

TRUNCATE TABLE RESULT_QUAD;
INSERT INTO RESULT_QUAD SELECT QUAD_ID,
           COUNT(*) TOTAL,
           MIN(X) MIN_X,
           MAX(X) MAX_X,
           MIN(Y) MIN_Y,
           MAX(Y) MAX_Y,
           MIN(Z) MIN_Z,
           MAX(Z) MAX_Z
 FROM STRAIGHT7D GROUP BY QUAD_ID;

ANALYZE TABLE RESULT_QUAD COMPUTE STATISTICS;

DROP INDEX RESULT_QUAD_idx;
CREATE INDEX RESULT_QUAD_idx ON RESULT_QUAD (QUAD_ID);

SELECT COUNT(*) FROM RESULT_QUAD WHERE TOTAL > <maximum_size_pointcluster>;
```

*) Repeat procedure GRID_STRAIGHT until the total number of points in every cell is below the defined maximum.

A.3 Conversion to SDO_GEOMETRY

```
CREATE OR REPLACE TYPE ATTRIB_VA AS VARRAY (4000000) OF NUMBER;
/
```

```
CREATE OR REPLACE PROCEDURE STRAIGHT_TO_SPATIAL(input_table IN VARCHAR2, result_table
IN VARCHAR2, spatial_table IN VARCHAR2) AUTHID CURRENT_USER AS

-- Declarations
i_drop NUMBER(1);                              -- count if table_spatial exists

type NUM_LIST IS VARRAY(1048576) OF NUMBER;    -- varray definition
AX NUM_LIST;                                -- varray for X
AY NUM_LIST;                                -- varray for Y
AZ NUM_LIST;                                -- varray for Z
AI NUM_LIST;                                -- varray for I
AR NUM_LIST;                                -- varray for R
AG NUM_LIST;                                -- varray for G
AB NUM_LIST;                                -- varray for B

t_geom SDO_GEOMETRY;                        -- SDO_GEOMETRY constructor
t_attrib ATTRIB_VA := ATTRIB_VA();          -- varray for attributes

t_dim RESULT_QUAD%ROWTYPE;                  -- get dimensions from table_quad

i BINARY_INTEGER;                           -- loop integer for t_geom
j BINARY_INTEGER;                           -- loop integer for t_attrib
k INTEGER;                                  -- loop integer total

-- Begin
BEGIN

  EXECUTE IMMEDIATE 'ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ".,"';

  SELECT COUNT(*) INTO i_drop FROM USER_TABLES WHERE TABLE_NAME = '' || spatial_table
|| '';
  IF i_drop = 1 THEN
    EXECUTE IMMEDIATE 'DROP TABLE ' || spatial_table;
  END IF;

  EXECUTE IMMEDIATE 'CREATE TABLE ' || spatial_table || '(QUAD_ID NUMBER, GEOM
MDSYS.SDO_GEOMETRY, ATTRIB ATTRIB_VA)';

  k := 0;

  FOR curs_result IN (SELECT QUAD_ID FROM RESULT_QUAD WHERE TOTAL > 1 ORDER BY
QUAD_ID)

  LOOP

    EXECUTE IMMEDIATE 'SELECT X,Y,Z,I,R,G,B FROM ' || input_table || ' WHERE QUAD_ID =
:id' BULK COLLECT INTO AX,AY,AZ,AI,AR,AG,AB USING curs_result.QUAD_ID;

      t_geom :=
SDO_GEOMETRY(3005,28992,NULL,SDO_ELEM_INFO_ARRAY(1,1,AX.COUNT),SDO_ORDINATE_ARRAY());
      t_attrib := ATTRIB_VA();

      t_geom.SDO_ORDINATES.EXTEND(3*AX.COUNT);
      t_attrib.EXTEND(4*AX.COUNT);

      i := 0;
      j := 0;

      FOR p in 1..AX.COUNT LOOP

        t_geom.SDO_ORDINATES(i+1) := AX(p);
        t_geom.SDO_ORDINATES(i+2) := AY(p);
```

```
        t_geom.SDO_ORDINATES(i+3) := AZ(p);

        t_attrib(j+1) := AI(p);
        t_attrib(j+2) := AR(p);
        t_attrib(j+3) := AG(p);
        t_attrib(j+4) := AB(p);

        i := i+3;
        j := j+4;

      END LOOP;

    EXECUTE IMMEDIATE 'INSERT INTO ' || spatial_table || ' VALUES (:id,:geom,:attrib)'
USING curs_result.QUAD_ID,t_geom,t_attrib;
    k := k+1;

    IF mod(k, 1000) = 0 THEN -- commit every 1000 records
        COMMIT;
        DBMS_OUTPUT.PUT_LINE (TO_CHAR (k) || ' records inserted sofar');
    END IF;

  END LOOP;

  DBMS_OUTPUT.PUT_LINE (TO_CHAR (k) || ' records inserted');

  EXECUTE IMMEDIATE'INSERT INTO ' || spatial_table || ' SELECT
A.QUAD_ID,SDO_GEOMETRY(3001,28992,SDO_POINT_TYPE(B.X,B.Y,B.Z),NULL,NULL),ATTRIB_VA(B.I
,B.R,B.G,B.B) FROM ' || result_table || ' A,' || input_table || ' B WHERE A.TOTAL = 1
AND A.QUAD_ID = B.QUAD_ID';

  COMMIT;

  EXECUTE IMMEDIATE 'SELECT
MIN(QUAD_ID),MIN(TOTAL),MIN(MIN_X),MAX(MAX_X),MIN(MIN_Y),MAX(MAX_Y),MIN(MIN_Z),MAX(MAX
_Z) FROM ' || result_table INTO t_dim;

  DELETE FROM USER_SDO_GEOM_METADATA WHERE TABLE_NAME='' || spatial_table || '';
  INSERT INTO USER_SDO_GEOM_METADATA VALUES ('' || spatial_table || '', 'GEOM',
MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X',t_dim.MIN_X,t_dim.MAX_X,.005),
MDSYS.SDO_DIM_ELEMENT('Y',t_dim.MIN_Y,t_dim.MAX_Y,.005),
MDSYS.SDO_DIM_ELEMENT('Z',t_dim.MIN_Z,t_dim.MAX_Z,.005)),28992);

  COMMIT;

END;
/
```

```
EXEC STRAIGHT_TO_SPATIAL('STRAIGHT7D','RESULT_QUAD','<spatial_table_name>');

ANALYZE TABLE <spatial_table_name> COMPUTE STATISTICS;

DROP INDEX SPATIAL_idx;
CREATE INDEX SPATIAL_idx ON <spatial_table_name> (GEOM) INDEXTYPE IS
MDSYS.SPATIAL_INDEX;

ANALYZE INDEX SPATIAL_idx COMPUTE STATISTICS;
```

A.4 Special Query

```
DROP TABLE RESULT_QUERY;
CREATE TABLE RESULT_QUERY AS SELECT A.* FROM <spatial_table_name> A WHERE
SDO_WITHIN_DISTANCE(A.GEOM,SDO_GEOMETRY(2001,28992,SDO_POINT_TYPE(<x>,<y>,NULL),NULL,N
ULL),'distance=<distance>') = 'TRUE';

SELECT COUNT(*) FROM RESULT_QUERY;
```

A.5 Statistics

```
SELECT TABLE_NAME,(BLOCKS*8/1024) "SIZE_MB",BLOCKS, EMPTY_BLOCKS "EMPTY",
NUM_ROWS,CHAIN_CNT "CHAINED",AVG_ROW_LEN "AVROWLEN",AVG_SPACE "FRSPACE" FROM
USER_TABLES WHERE TABLESPACE_NAME = 'USERS' AND (BLOCKS*8/1024) > 1 AND TABLE_NAME NOT
LIKE 'MDRT%' ORDER BY NUM_ROWS;

SELECT TABLE_NAME,SEGMENT_NAME "LOB_SEGMENT_NAME",(SELECT SUM(BYTES/(1024*1024)) FROM
USER_EXTENTS e WHERE e.SEGMENT_NAME = l.SEGMENT_NAME) "SIZE_MB",(SELECT UNIQUE
e.TABLESPACE_NAME FROM USER_EXTENTS e WHERE e.SEGMENT_NAME = l.SEGMENT_NAME) "TSPACE"
FROM USER_LOBS l WHERE (SELECT SUM(BYTES/(1024*1024)) FROM USER_EXTENTS e WHERE
e.SEGMENT_NAME = l.SEGMENT_NAME) > 1 ORDER BY TABLE_NAME,COLUMN_NAME,SEGMENT_NAME;

SELECT t.TABLE_NAME,i.INDEX_NAME,(t.BLOCKS*8/1024) "SIZE_MB" FROM USER_TABLES
t,USER_SDO_INDEX_INFO i WHERE t.TABLE_NAME = i.SDO_INDEX_TABLE ORDER BY
TABLESPACE_NAME,INDEX_NAME;

SELECT INDEX_NAME,TABLE_NAME,(LEAF_BLOCKS*8/1024) "LF_BLK_MB" FROM USER_INDEXES WHERE
INDEX_TYPE != 'LOB' AND INDEX_TYPE = 'NORMAL' AND TABLESPACE_NAME = 'USERS' ORDER BY
TABLE_NAME;

DROP VIEW TABLE_SIZE;

CREATE VIEW TABLE_SIZE AS SELECT SEGMENT_NAME,COUNT(*) EXTENTS,SUM(BYTES/(1024*1024))
"SIZE_MB",SUM(BLOCKS) "BLOCKS",TABLESPACE_NAME "TSPACE" FROM USER_EXTENTS GROUP BY
SEGMENT_NAME,TABLESPACE_NAME ORDER BY TABLESPACE_NAME,SEGMENT_NAME;

SELECT l.TABLE_NAME,e.SEGMENT_NAME,e.EXTENTS,e.SIZE_MB,e.BLOCKS FROM TABLE_SIZE
e,USER_LOBS l WHERE e.SEGMENT_NAME = l.SEGMENT_NAME AND e.SEGMENT_NAME LIKE 'SYS_LOB%'
AND e.EXTENTS > 1 AND e.TSPACE = 'USERS' ORDER BY SEGMENT_NAME;

SELECT * FROM TABLE_SIZE WHERE EXTENTS > 1 AND TSPACE = 'USERS' AND SEGMENT_NAME NOT
LIKE 'SYS_LOB%' AND SEGMENT_NAME NOT LIKE 'MDRT_%' ORDER BY SEGMENT_NAME;

SELECT i.INDEX_NAME,e.SEGMENT_NAME,e.EXTENTS,e.SIZE_MB,e.BLOCKS FROM TABLE_SIZE
e,USER_SDO_INDEX_INFO i WHERE e.SEGMENT_NAME = i.SDO_INDEX_TABLE AND e.EXTENTS > 1 AND
e.TSPACE = 'USERS' ORDER BY INDEX_NAME;

-- -- -- -- -- -- -- -- -- -- --

SELECT COUNT(*),MIN(QUAD_ID),MAX(QUAD_ID),MIN(TOTAL),MAX(TOTAL),AVG(TOTAL) FROM
RESULT_QUAD;
SELECT * FROM <spatial_table_name> WHERE ROWNUM < 2;

SELECT QUAD_ID, (MAX_X-MIN_X) DIFF_X, (MAX_Y-MIN_Y) DIFF_Y FROM RESULT_QUAD WHERE
QUAD_ID = (SELECT MIN(QUAD_ID) FROM RESULT_QUAD);
SELECT QUAD_ID, (MAX_X-MIN_X) DIFF_X, (MAX_Y-MIN_Y) DIFF_Y FROM RESULT_QUAD WHERE
QUAD_ID = (SELECT MAX(QUAD_ID) FROM RESULT_QUAD);
```

A.6 Clean-up DBMS

```
CREATE OR REPLACE PROCEDURE CLEANUP_TU(dummy IN VARCHAR2) AUTHID CURRENT_USER AS

-- Begin
BEGIN

  EXECUTE IMMEDIATE 'ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ".,"';

  FOR index_rec IN (SELECT INDEX_NAME FROM USER_INDEXES WHERE INDEX_NAME NOT LIKE
'SYS_%')

  LOOP

    EXECUTE IMMEDIATE 'DROP INDEX ' || index_rec.INDEX_NAME;
```

```
    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Index ' || index_rec.INDEX_NAME || ' dropped');

  END LOOP;

  FOR table_rec IN (SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME NOT LIKE
'MDRT_%')

  LOOP

    IF table_rec.TABLE_NAME = 'STRAIGHT7D' THEN
      CONTINUE;
    END IF;

    EXECUTE IMMEDIATE 'DROP TABLE ' || table_rec.TABLE_NAME;
    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Table ' || table_rec.TABLE_NAME || ' dropped');

  END LOOP;

END;
/
```

```
EXEC CLEANUP_TU('test');
```

A.7 SDO_PC Prepare

```
DROP TABLE SDO_PC_STRAIGHT7D;
CREATE TABLE SDO_PC_STRAIGHT7D (RID VARCHAR2(24),VAL_D1 NUMBER,VAL_D2 NUMBER,VAL_D3
NUMBER,VAL_D4 NUMBER,VAL_D5 NUMBER,VAL_D6 NUMBER,VAL_D7 NUMBER,VAL_D8 NUMBER);

DROP SEQUENCE RID_SEQ;
CREATE SEQUENCE RID_SEQ INCREMENT BY 1 START WITH 1 ORDER;

INSERT INTO SDO_PC_STRAIGHT7D (RID,VAL_D1,VAL_D2,VAL_D3,VAL_D4,VAL_D5,VAL_D6,VAL_D7)
SELECT RID_SEQ.nextval,X,Y,Z,I,R,G,B FROM STRAIGHT7D;

DROP INDEX SDO_PC_idx;
CREATE INDEX SDO_PC_idx ON SDO_PC_STRAIGHT7D (VAL_D1,VAL_D2);
```

A.8 SDO_PC Create

```
DROP TABLE PC_BASE;
CREATE TABLE PC_BASE (PNTCLOUD SDO_PC);

DROP TABLE PC_BLOCK;
CREATE TABLE PC_BLOCK AS SELECT * FROM MDSYS.SDO_PC_BLK_TABLE;
```

A.9 SDO_PC Fill

```
DECLARE

  DIM        NUMBER       := 7;
  TOL        NUMBER       := 0.001;
  PTN_PARAMS VARCHAR2(80) := 'BLK_CAPACITY=<max_points_per_record>';
  EXTENT     SDO_GEOMETRY;
```

```
  PC         SDO_PC;

BEGIN

  EXTENT :=
SDO_GEOMETRY(2003,28992,NULL,SDO_ELEM_INFO_ARRAY(1,1003,3),SDO_ORDINATE_ARRAY(1,2,3,4)
);

  SELECT MIN(VAL_D1),MIN(VAL_D2),MAX(VAL_D1),MAX(VAL_D2) INTO
EXTENT.SDO_ORDINATES(1),EXTENT.SDO_ORDINATES(2),EXTENT.SDO_ORDINATES(3),EXTENT.SDO_ORD
INATES(4) FROM SDO_PC_STRAIGHT7D;

  PC :=
SDO_PC_PKG.INIT('PC_BASE','PNTCLOUD','PC_BLOCK',PTN_PARAMS,EXTENT,TOL,DIM,NULL,NULL,NU
LL);

  INSERT INTO PC_BASE VALUES (PC);
  COMMIT;

  SDO_PC_PKG.CREATE_PC(PC,'SDO_PC_STRAIGHT7D',NULL);

END;
/

SELECT COUNT(*) FROM PC_BLOCK;
SELECT SUM(NUM_POINT) FROM PC_BLOCK;
SELECT * FROM PC_BASE;

ANALYZE TABLE PC_BLOCK COMPUTE STATISTICS;
ANALYZE TABLE PC_BASE COMPUTE STATISTICS;
```

A.10 SDO_PC Query

```
DROP TABLE RESULT_QUERY_SDO_PC;
CREATE TABLE RESULT_QUERY_SDO_PC AS SELECT * FROM TABLE(SDO_PC_PKG.CLIP_PC((SELECT
PNTCLOUD FROM PC_BASE WHERE
ROWNUM=1),SDO_GEOMETRY(2003,28992,NULL,MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,4),MDSYS.SDO_O
RDINATE_ARRAY(<x>-<distance>,<y>, <x>,<y>-<distance>, <x>+<distance>,<y>)), NULL, NULL,
NULL));
SELECT COUNT(*) FROM RESULT_QUERY_SDO_PC;

DROP TABLE RESULT_QUERY;
CREATE TABLE RESULT_QUERY AS SELECT
SDO_PC_PKG.TO_GEOMETRY(A.POINTS,A.NUM_POINTS,7,28992,NULL) GEOM FROM
RESULT_QUERY_SDO_PC A WHERE A.NUM_POINTS>0;

SELECT COUNT(*) FROM RESULT_QUERY;
SELECT * FROM RESULT_QUERY WHERE ROWNUM < 2;
```

Appendix B: Detailed Benchmark Results

Graph 1 (page 28): Number of Records vs. Points per Record (blue) and a Supposed Maximum (red).

Level	Number of records	Points per record	Maximum Points per record
1	4	4369561	349525
2	16	1092390	349525
3	64	273098	349525
4	256	68274	349525
5	1024	17069	349525
6	4082	4282	349525
7	16165	1081	349525
8	64075	273	349525
9	254691	69	349525
10	1011841	17	349525
99	17478245	1	349525

Graph 2 (page 37): Trendline Queries.

Level	Gridding duration	Records updated
1	0:46:30	4
2	0:36:20	16
3	0:29:01	64
4	0:27:08	256
5	0:27:04	1024
6	0:30:13	4082
7	0:39:31	16165
8	0:51:32	64075
9	1:31:45	254691
10	2:21:44	1011841

Graph 3 (page 37): Result - Region Quadtree and Morton Space-filling Curve.

Level	Points per record	Gridding duration
1	4369561	0:46:30
2	1092390	1:22:50
3	273098	1:51:51
4	68274	2:18:59
5	17069	2:46:03
6	4282	3:16:16
7	1081	3:55:47
8	273	4:47:19
9	69	6:19:04
10	17	8:40:48
99	1	-

Graph 4 (page 39): Convert to Spatial Data (right: detail).

Level	Points per record	Conversion to Spatial
1	4369561	-
2	1092390	-
3	273098	-
4	68274	0:31:22
5	17069	0:06:05
6	4282	0:05:40
7	1081	0:05:46
8	273	0:06:11
9	69	0:07:19
10	17	0:11:10
99	1	0:00:17

Graph 5 (page 40): Storage Space.

Level	Points per record	Storage Space Spatial (MB)
1	4369561	-
2	1092390	-
3	273098	-
4	68274	544,13
5	17069	560,31
6	4282	578,00
7	1081	661,00
8	273	856,00
9	69	801,00
10	17	824,00
99	1	1811,00

Graph 6 (page 40): Index space.
Graph 6 (page 40): Relation.

Level	Points per record	Number of records	Storage Space Index (MB)
1	4369561	4	-
2	1092390	16	-
3	273098	64	-
4	68274	256	0,01
5	17069	1024	0,13
6	4282	4082	0,44
7	1081	16165	2,00
8	273	64075	6,00
9	69	254691	23,00
10	17	1011841	92,00
99	1	17478245	1544,00

Graph 7 (page 41): Create Spatial Index.

Level	Points per record	Number of records	Create index
1	4369561	4	-
2	1092390	16	-
3	273098	64	-
4	68274	256	0:00:24
5	17069	1024	0:00:41
6	4282	4082	0:00:42
7	1081	16165	0:00:50
8	273	64075	0:00:53
9	69	254691	0:00:56
10	17	1011841	0:02:06
99	1	17478245	0:45:11

Graph 8 (page 42): Storage Space – Local vs. Global.

Level	Points per record	Storage spatial (global)	Storage index (global)	Storage spatial (local)	Storage index (local)
1	4369561	-	-	-	-
2	1092390	-	-	-	-
3	273098	-	-	-	-
4	68274	544,13	0,01	560,56	0,01
5	17069	560,31	0,13	568,50	0,13
6	4282	578,00	0,44	586,00	0,44
7	1081	661,00	2,00	674,00	2,00
8	273	856,00	6,00	895,00	6,00
9	69	801,00	23,00	815,00	23,00
10	17	824,00	92,00	838,00	96,00
99	1	1811,00	1544,00	1811,00	1544,00

Graph 9 (page 44): Query-results (all). Graph 10 (page 44): Query-results (details).

Level	Points per record	Number of records	distance = 50m	distance = 100m	distance = 200m	distance = 300m	distance = 400m	distance = 500m
1	4369561	4	-	-	-	-	-	-
2	1092390	16	-	-	-	-	-	-
3	273098	64	-	-	-	-	-	-
4	68274	256	0:00:01	0:00:02	0:00:07	0:00:10	0:00:15	0:00:22
5	17069	1024	0:00:02	0:00:02	0:00:08	0:00:12	0:00:24	0:00:36
6	4282	4082	0:00:02	0:00:02	0:00:08	0:00:15	0:00:27	0:00:41
7	1081	16165	0:00:02	0:00:03	0:00:11	0:00:32	0:00:48	0:01:12
8	273	64075	0:00:02	0:00:03	0:00:18	0:00:41	0:01:08	0:01:58
9	69	254691	0:00:01	0:00:03	0:00:14	0:00:24	0:00:33	0:01:08
10	17	1011841	0:00:02	0:00:02	0:00:14	0:00:26	0:00:42	0:01:29
99	1	17478245	0:00:16	0:00:49	0:03:19	0:07:12	0:17:07	0:26:11

Graph 11 (page 45): Conversion point clouds to SDO_PC data type, duration.
Graph 11 (page 45): Conversion point clouds to SDO_PC data type, storage space (incl. index).

Level	Number of records	Points per record	Gridding and conversion to spatial and create index duration	Storage Space (MB)
1	-	-	-	-
2	-	-	-	-
3	-	-	-	-
4	278	62871,38489	0:43:41	1092,50
5	913	19143,75137	0:43:53	1096,00
6	3527	4955,555713	0:45:22	1110,00
7	14035	1245,33274	0:59:20	1110,00
8	43758	399,4297043	1:43:50	1387,50
9	174821	99,97794887	8:19:57	1387,50
10			Over 12 hours	

Graph 12 (page 46): Results of SDO_PC_PKG.CLIP_PC.

Level	Number of records	Points per record	distance = 50m	distance = 100m	distance = 200m	distance = 300m	distance = 400m	distance = 500m
1	-	-			-	-	-	-
2	-	-			-	-	-	-
3	-	-			-	-	-	-
4	278	62871	0:00:04	0:00:08	0:00:32	0:01:04	0:01:51	0:02:42
5	913	19144	0:00:04	0:00:09	0:00:42	0:01:20	0:02:09	0:03:09
6	3527	4956	0:00:03	0:00:10	0:00:45	0:01:40	0:02:52	0:04:18
7	14035	1245	0:00:05	0:00:14	0:00:57	0:02:00	0:03:33	0:05:28
8	43758	399	0:00:07	0:00:25	0:01:39	0:03:25	0:06:03	0:08:29
9	174821	100	0:00:10	0:00:40	0:03:15	0:06:14	0:12:13	0:19:49

Graph 13 (page 46): Results of SDO_PC_PKG.TO_GEOMETRY.

Grid-Level	Number of records	Points per record	distance = 50m	distance = 100m	distance = 200m	distance = 300m	distance = 400m	distance = 500m
1	-	-			-	-	-	-
2	-	-			-	-	-	-
3	-	-			-	-	-	-
4	278	62871	0:00:08	0:00:41	0:03:09	0:07:18	0:12:45	0:19:16
5	913	19144	0:00:09	0:00:41	0:03:11	0:07:31	0:12:44	0:19:13
6	3527	4956	0:00:08	0:00:38	0:02:58	0:06:52	0:12:14	0:18:25
7	14035	1245	0:00:07	0:00:35	0:02:45	0:06:28	0:11:08	0:16:45
8	43758	399	0:00:08	0:00:36	0:03:01	0:06:45	0:12:02	0:18:35
9	174821	100	0:00:07	0:00:32	0:02:35	0:06:12	0:10:50	0:16:30

Graph 14 (page 47): Oracle – SDO_PC – Query and Unwrap.

Column “Points per record” and “distance = 500m” of Graph 12 and column “distance = 500m” of Graph 13.