MSc thesis in Geomatics for the Built Environment

Utilizing a Discrete Global Grid System For Handling Point Clouds With Varying Locations, Times, and Levels Of Detail

Neeraj Sirdeshmukh

2018

TU Delft

FUGRO

UTILIZING A DISCRETE GLOBAL GRID SYSTEM FOR HANDLING POINT CLOUDS WITH VARYING LOCATIONS, TIMES, AND LEVELS OF DETAIL

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

Master of Science in Geomatics for the Built Environment

by

Neeraj Sirdeshmukh

June 2018

ISBN 999–99–9999–999–9

The work in this thesis was made in the:

Regional Innovation Centre Europe
Fugro
and
Department OTB - Research for the Built Environment.
Faculty of Architecture and the Built Environment
Delft University Of Technology

Supervisors: ir. Edward Verbree
prof.dr.ir. Peter van Oosterom
Co-reader: Dr. Ravi Peters

ABSTRACT

Point clouds are getting increasingly larger and there is an ever greater need to use a scalable, high performance, multi-dimensional spatial framework that can be used to work with point clouds at scales from local to global. First, point clouds stemming from disparate sources possess distinct geometrical, temporal, and scale characteristics and these pose challenges for the interoperability of these datasets in certain applications, for example when point clouds of different countries need to be combined into a single point cloud. A second challenge is the provision of this enormous amount of data to anyone, anywhere via the Internet. This is the vision of Open Point Cloud Map (OPCM), a proposed 'OpenStreetMap of point clouds' [Verbree et al., 2017].

To address the above challenges, having a common underlying data structure would be ideal. A Discrete Global Grid System (DGGS) can be used as a common global data structure for the storage, visualization, and analysis of disparate point clouds as it provides a means to encode locations on the entire Earth using a hierarchical tessellation of non-overlapping cells that can support data at multiple levels of spatial resolution. DGGS's have emerged in recent years as unconventional Earth reference systems for working with global heterogeneous datasets in a Digital Earth, a term that refers to a digital representation of the Earth will all geospatial information attached to it. This research seeks to analyze the extent to which a DGGS can be used to handle point clouds with varying location, time, and scale, and compare a DGGS with conventional reference systems.

There exist several kinds of DGGS, each posing their own unique advantages and disadvantages. These are compared to one another, and a choice is made for a suitable DGGS for the purpose of handling heterogeneous global point clouds. For appealing visualization, point clouds are better displayed using variable-scale instead of multi-scale data structures, and the means to achieve a variable-scale representation of point clouds using the hierarchical nature of a DGGS is stated and implemented. The temporal dimension of point clouds can be utilized to perform change detection, and this is illustrated using a DGGS extended from 2D into 3D and 4D. Finally, a DGGS is compared and contrasted with conventional systems such as European Terrestrial Reference System (ETRS), International Terrestrial Reference System (ITRS), and the Dutch *Rijksdriehoeksstelsel (RD)* system. Several contributions to the field are being offered with this thesis: first, to date, there have been no studies on using point clouds with DGGS; second, a DGGS has never been applied to the indexing of a global point cloud; third, it is shown how a DGGS can be used in higher than 2 dimensions; fourth, it is shown how the precision-encoding nature of a DGGS can be used for variable-scale smooth zoom visualization; and fifth a DGGS is compared and contrasted with conventional Coordinate Reference System (CRS)'s. It can be concluded that a DGGS-based approach provides for addressing all three potentially problematic aspects of point cloud integration - location, time, and levels of detail- in a single, consistent structure.

As a prototype implementation to demonstrate the feasibility of this method, a viewer is developed that can stream point clouds in a DGGS-based system. This further contributes to the OPCM vision and promotes the advancement of open data. Collectively, addressing the issues of location, time, and scale ultimately leads to better interoperability of point clouds and consequently to more effective decision making.

ACKNOWLEDGEMENTS

I would like to thank several people for providing their fullest support to me during my thesis. First and foremost, my supervisors from the university Edward Verbree and Peter van Oosterom, whose guidance and supervision was extremely beneficial. The discussions I held with them provided me with new ideas, a sense of challenge, and a proper direction for my research. They also provided me the opportunity to attend the March 2018 Open Geospatial Consortium (OGC) Meeting in Orléans, France, where I presented the results of my research to the DGGS and Point Cloud Domain Working Group (DWG)'s. I would also like to thank Martijn Meijers and Marien de Vries from the Geomatics faculty for their willingness to provide advice when requested. From the company Fugro, where this thesis took place, I would like to thank Stella Psomadaki, who acted as a company supervisor, held meetings, and was available whenever needed to provide support, and Martin Kodde, who provided a broad, comprehensive direction at which to aim the research. Martin also provided me the opportunity to present the plan for my research at the September 2017 OGC meeting in Southampton, United Kingdom, where I first got to personally meet and interact with the experts on DGGS from the OGC and gained invaluable advice. I also thank the members of the DGGS DWG, such as Dr. Matthew Purss, Perry Peterson, and Robert Gibb, for their readiness to provide advice when requested. It was a great experience interacting with them and exchanging ideas. It has been an amazing 2 years at the Delft University Of Technology and I have fully enjoyed my time here. I have honed my skills in geomatics and gained an invaluable education.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

ALS Airborne Laser Scanner . . . 4
API Application Programming Interface . . . 16
ASCII American Standard Code for Information Interchange . . . 26
AWS Amazon Web Services . . . 87
CAD Computer Aided Design . . . 2
CRS Coordinate Reference System . . . v
CSV Comma Separated Values . . . 63
CZML Cesium Language . . . 54
DBMS Database Management System . . . 8
DE-9IM Dimensionally Extended 9-Intersection Model . . . 23
DGG Discrete Global Grid . . . xi
DGGS Discrete Global Grid System . . . v
DWG Domain Working Group . . . vii
ECEF Earth Centered Earth Fixed . . . 29
EC2 Elastic Compute Cloud . . . 87
ETRS European Terrestrial Reference System . . . v
GB Gigabyte . . . 56
GIS Geographic Information Systems . . . 3
GLONASS Global Navigation Satellite System . . . 75
GLTF GL Transmission Format . . . 54
GNSS Global Navigation Satellite System . . . 26
GPS Global Positioning System . . . 3
GPU Graphics Processing Unit . . . 16
GRS80 Geodetic Reference System of 1980 . . . 11
GUID globally unique identifier . . . 35
HD Hausdorff Distance . . . 30
HPC High Performance Computing . . . 56
IMU Inertial Measurement Unit . . . 33
ISEA Icosahedral Snyder Equal Area . . . xi
ITRF International Terrestrial Reference Frame . . . 12
ITRS International Terrestrial Reference System . . . v
LAEA Lambert Azimuthal Equal Area . . . 11
LAS LASer . . . 26
LIDAR Light Detection And Ranging . . . xi
LOD Level Of Detail . . . 5
LOD Levels Of Detail . . . 4
LOPOCS Light Open Source Point Cloud Server . . . 54

1 INTRODUCTION

Point clouds are becoming increasingly popular ways of mapping the Earth's surface. Several technologies exist to generate these massive point clouds, with one of the major ones being LIDAR [Wang et al., 2018], which uses laser scanning to generate a digital 3D representation of the target of interest. Point clouds can be used towards a number of applications, from creating 3D Computer Aided Design (CAD) models, rendering, quality inspection, change detection, vegetation mapping, and others [Pfeifer, 2018].

One of the most significant challenges in point cloud data processing lies in handling its increasing data volume. Contemporary remote sensing data acquisition technologies have the potential to generate point clouds with 10E12 or even 10E15 of 3D points [van Oosterom et al., 2015]. But a further challenge and one more important from a practical perspective is the provision of this enormous amount of point cloud data to the general public at free or reasonably low-cost rates. The shining examples of countries that have provided all or most of their point cloud data as open data are the United States, the Netherlands, Denmark, and Finland [Open Data Diliman, 2017]. In general, however, point cloud data remains inaccessible, licensed and/or proprietary and out of reach of the general public. It is therefore of enormous added value to research whether it is possible to establish an open-source platform for the accessibility of point clouds and to investigate the key challenges of point cloud data management in the creation of this platform.

1.1 OPEN POINT CLOUD MAP

Volunteered Geographic Information (VGI) refers to the harnessing of tools to create, assemble, and disseminate geographic data provided voluntarily by individuals [Goodchild, 2007]. Over the past decade or so, there has been a tremendous growth in the engagement of private citizens, most of whom do not possess any formal training in the discipline of geography, in the creation of geographic information, and this has been further reinforced by the explosive growth of the Internet. These individuals are participating in the Spatial Data Infrastructure (SDI) on a purely voluntary basis, and their input into the geographic data creation process might or might not be entirely accurate. However, as more and more individuals are gaining access to the Internet, they are better able to contribute to VGI-based projects.

One of the prime examples of a VGI-based initiative is Open Street Map (OSM), a map of the entire world built by a global community of mappers that is free for anyone to edit, use, and contribute [Neis and Zipf, 2012]. OSM is purely *open data*: data which is available in its entirety at no more

than a reasonable reproduction cost, and that which can be freely used, re-used, and re-distributed by anyone, without any discriminatory policies against any certain group of individuals [Open Knowledge International, 2017]. Anyone with an Internet connection can contribute to the expansion of OSM. The advent of cheap portable hand-held Global Positioning System (GPS) devices, aerial imagery, field survey equipment and unmanned data collection systems such as Unmanned Aerial System (UAS)'s and the original lack of map data for many regions of the world have fueled OSM's growth to a community of more than 2 million registered users [Neis and Zipf, 2012]. Although many areas of the world are still not completely mapped in digital form in OSM, this enormous effort by a global community of Geographic Information Systems (GIS) professionals, engineers, humanitarians working in disaster zones, and others will significantly narrow the gap in the accessibility and availability of geographic information to a typical end-user.

The success of VGI initiatives such as OSM would propel one to wonder why the same cannot be accomplished with point clouds. Just as OSM provides a global map of vector features such as roads, railways, parks, restaurants, lakes, cycle routes, and others, an 'OpenStreetMap of point clouds' could also be created (as a web application) to allow for the use and dissemination of point cloud data. Point cloud data has often been overlooked [Verbree et al., 2017], but its growing popularity among both professional and non-professional users and the broad range of applications for which it can be utilized further motivate the creation of an Open Point Cloud Map (OPCM).

The vision of the OPCM [Verbree et al., 2017] is to make point cloud data accessible to anyone, everywhere via an Internet connection. The objective of this project is to research and implement an open source platform for the accessibility of point clouds, similar to an 'OpenStreetMap of point clouds' [Verbree et al., 2017]. Open data, in general, provides a wide range of benefits, including improved efficiency of public administrations, economic growth in the private sector, improved transparency and accountability and development of innovative services [European Data Portal, 2017]. Creating a global OPCM will allow anyone, anywhere, with access to the Internet to upload, download, edit, use, and contribute to the existing repository of crowd-sourced point cloud data, and this will further propel the expansion of VGI and user-generated content.

1.2 PROBLEM STATEMENT

The creation of the OPCM is no easy task. The management and visualization of massive volumes of point clouds requires a scalable framework to begin with, and their incorporation into a global OPCM necessitates the addressing of three of the most common challenges faced when trying to integrate point clouds coming from different origins. These challenges in the creation of the OPCM pertain to the aspects of location, time, and densities of point clouds [Verbree et al., 2017].

The first aspect pertains to the geometry of the point cloud data. Different regions of the world use disparate CRS's and point clouds in these

regions will likely also be in these distinct CRS's, each of which could have its own unique origin, orientation, and scale. Therefore, a problem arises when trying to integrate or combine these datasets that are each linked to a different CRS. If the points aren't projected into a common system, the coordinates in the combined point cloud would be meaningless. A common CRS needs to be devised to handle such varying point clouds. Moreover, for a project such as the OPCM, where point cloud data of the entire world will be incorporated into a common reference framework, a common CRS is a fundamental requirement.

Second, point clouds coming from different sources could have varying Levels Of Detail (LOD); a point cloud collected using a Terrestrial Laser Scanner (TLS) could, for example, have a point density of 1000 points per square meter whereas another point cloud of the same area but collected using an Airborne Laser Scanner (ALS) could have a much lower point density of 10 points per square meter, 100 times lower than the TLS point cloud. If one naively combines such point clouds of the same area with significantly varying densities into one dataset, sharp jumps between the densities will be seen at the boundary between the two point clouds [van Oosterom et al., 2015]. Using such a multi-scale approach means that there are a discrete number of LOD's in the combined point cloud and the viewer might notice the difference in the point densities between neighboring blocks at different levels of detail [van Oosterom et al., 2015]. Therefore, this is yet another problem that needs to be addressed.

Finally, in time-stamped point clouds each point has an attribute that indicates the time at which the laser pulse that captured the point was emitted from the point cloud data collection platform (for example, aircraft or TLS). The time component of point clouds is of prime importance if point clouds of the same area but that were acquired at two different times need to be analyzed for the purposes of change detection, or to create time-dynamic visualizations that showcase how an area has changed in between times $t1$ and $t2$. Having a fixed, regular structure would be ideal for performing such a change analysis. Collectively, addressing these three issues will ultimately lead to better integration of point clouds from different sources, and will eventually promote more effective decision making.

To address the above challenges, having a common underlying data structure would be ideal. A DGGS can be used as a common global reference frame for the storage, visualization, and analysis of point clouds as it provides a means to encode locations on the entire Earth using a hierarchical tessellation of non-overlapping cells that can support data at multiple levels of spatial resolution [Purss et al., 2017]. A DGGS provides for rapid spatial data integration, storage, and analytics at scales from local to global in a single consistent framework [Open Geospatial Consortium, 2017b]. Although at present it is used in 2 dimensions, in this research it has been extended into 3D and 4D to allow for analysis of higher-dimensional data such as point clouds. Therefore, a DGGS-based approach to handling massive point clouds stemming from across the world has been investigated in this research.

1.3 SCIENTIFIC RELEVANCE

Addressing these above challenges will allow for more interoperability among point clouds stemming from different origins with respect to location (geometry), time, and densities. That is, these datasets can more frequently be used in tandem with one another in geospatial projects. The transformation of point clouds into a common reference system provides several advantages: we can study the systematic errors in-between the various datasets; moreover, the internationalization of recent decades and growth in cross-border projects has necessitated the need for a common reference system, and transforming data into a common system will greatly aid the exchange of geographic information [Geonovum, 2013], allowing for more international cooperation and strengthening of ties between countries. Public authorities at all levels need to regularly exchange geographic information with other authorities [European Environment Agency, 2017], especially with those with whom they share a political border. In these situations, it is of prime importance to exchange geographic information in a common CRS to avoid positional errors and facilitate improved decision making and planning. Doing otherwise will hinder the success of cross-border projects.

Point clouds stemming from different origins are likely to have varying initial point densities. Moreover, point densities are also likely to differ for different applications towards which point clouds are utilized. For example, for basic 3D surface models or forest inventory purposes, point density is likely to be around 0.5 - 1 point/square meter, for flood modeling around 1-2 points/square meter whereas for detailed 3D city models it could be up to 10 points/square meter [Rohrbach, 2015b]. The density is dictated by the requirements for the usage of the generated point cloud, and is also an indicator of the Level Of Detail (LOD): more dense point clouds will have high LOD whereas less dense point clouds will have low LOD. When trying to combine such point clouds with varying densities into a composite point cloud, it is of added value to create a continuous LOD representation of the composite point cloud rather than a discrete LOD representation (with multiple LOD's). Doing so will make for more appealing point cloud visualization, and will suppress any artefacts. Using continuous LOD based upon a continuous dimension added to the points will suppress the density shocks that discrete LOD based representations have [van Oosterom et al., 2015]. Such continuous LOD representations can be performed not only on points, but also on the other primitive vector data types such as lines and polygons. These so-called variable-scale representations provide a gradual change in the display of vector features when they are zoomed in or out upon, without any split and/or merge operations causing a sudden local 'shock' [Meijers, 2011]. It is desirable to prevent such breaks or jumps in the data, to provide a positive end-user experience. It must also be taken into consideration that most of the end users of the OPCM will be part of a non-expert audience [de Haan, 2010], and therefore this requirement becomes even more crucial. As a DGGS inherently provides a system of multiple resolutions, the advantages and disadvantages of using it as a variable-scale structure can be studied. This research seeks to analyze this aspect of DGGS.

Time-stamped point clouds allow for the spatial analysis of changes that could have taken place in an area of interest; we can study what has remained unchanged, or has been been added, removed or modified across an area in-between times $t1$ and $t2$. Timely and accurate change detection of features on the Earth's surface is invaluable for understanding relationships between human and natural phenomena [Lu et al., 2004] and helps maintain

and update databases [Xiao et al., 2013]. Point clouds can also indicate the land cover of an area based upon their spectral characteristics, and the study of land cover change using point clouds can help in the understanding of how the Earth is changing, for determining which factors are contributing to these changes, and to predict how the landscape will look like in the future [Boriah et al., 2008]. Ultimately, this promotes more effective decision making.

In the academic literature, the use of DGGS's for point clouds is non-existent; a key contribution of this thesis is to the application of DGGS technology for "handling" (i.e. analyzing/manipulating/processing) point clouds. Moreover, at present DGGS's are 2D systems; this thesis presents an inaugural conceptualization of how they would appear in 3D and 4D. Integrating four dimensions into one DGGS cell index rather than storing them as attributes results in improved query performance, and allows us to fully exploit the multi-dimensional nature of point cloud data.

1.4 RESEARCH QUESTIONS

The main research question for this thesis is:

To what extent can a Discrete Global Grid System be used to handle point clouds with varying locations, times, and densities/levels of detail?

The goal of this research is to analyze to which extent a DGGS is suitable for handling point clouds stemming from different origins with respect to location, time, and LOD's and create a DGGS-based viewer allowing for visualization, analysis, and upload of global open point cloud data. Sub-questions are:

- How can a variable-scale structure be implemented to support smooth zoom in DGGS?
- How can a DGGS be constructed to store the time component associated with point cloud data for analysis of spatio-temporal point clouds?
- What are the advantages and disadvantages of using a DGGS as compared to using conventional CRS's?

1.5 RESEARCH SCOPE

The following remarks outline the scope of this research.

- There exist several methods of constructing a DGGS, based on different choices of initial parameters. These methods have been briefly compared to one another. However, only the *ISEA aperture 4 rhombus* DGGS method was implemented. More explanation follows in Section 2.5.
- Change detection using the time component of point clouds is one of the focus areas of this research. However, this research does not aim to identify the most suitable method for change detection, nor does it aim to perform any sophisticated change detection itself. It simply aims to

study how changes in time-enabled point clouds can be analyzed and visualized with a DGGS.

- Automatic feature detection/identification and object reconstruction from point clouds is outside the scope of this research.
- Different ways to generate point clouds exist. This thesis has focused on point clouds acquired only from LIDAR, and this could be terrestrial or airborne.

1.6 OVERVIEW OF RESULTS

An icosahedral rhombus-based DGGS approach was utilized for this thesis, as it provides a congruent tessellation of cells covering the Earth at various resolutions, simpler geometry as compared to other shapes such as hexagons, and the application of quadrant-recursive indexing algorithms such as Morton SFC's. The key results of this research include:

- As the current DGGS specification is 2D on the surface of the Earth, extensions to 3D and 4D DGGS are proposed.
- As a DGGS is a hierarchical system with multiple resolutions, it can be used to encode the discrete precision of point observations collected in the real world. Moreover, this additional dimension can be utilized for a variable-scale visualization of point clouds, which is explained.
- A DGGS is also ideal for analyzing changes between time-stamped point clouds as it provides a fixed global reference frame that, once defined, does not move due to geophysical forces. The use of DGGS for identifying and visualizing changes between two moments in time is studied.
- Finally, as they are not widely-used technologies in the geomatics community, DGGS's are compared and contrasted with existing coordinate reference systems.

It can be concluded that DGGS's are an ideal tool for handling point clouds with varying locations, times, and densities/levels of detail.

1.7 OVERVIEW OF THESIS

This thesis is organized as follows:

- Chapter 2 provides a broad introduction to the concepts that are necessary to understand the rest of this thesis. This includes topics such as coordinate systems, datums, map projections, ellipsoids, DGGS's, indexing approaches using SFC's, and point cloud visualization and change analysis with DGGS.
- Chapter 3 elaborates on the methodology utilized in order to answer the research questions for this thesis. The procedure followed to encode and decode a 2D, 3D, and 4D Morton code for any location on Earth is explained. The methodology used to generate a variable-scale

visualization and conduct change analysis is elucidated. Also, the operations that can be conducted using a DGGS are briefly summarized, in conceptual terms.

- Chapter 4 discusses the datasets used and the implementation of the methodology developed.
- Chapter 5 presents the results of this thesis.
- Chapter 6 provides answers to the main research questions, discusses drawbacks of the current approach, and provides recommendations for future work.

The work performed for this thesis started in July 2017 and took approximately 9 months. The work was performed at Fugro GeoServices in the Netherlands, with academic supervision from TU Delft. I started work on it a bit earlier than the usual university period, because it's always good to begin something early in case extra time is needed towards the end, especially with a large research project such as a Master's thesis. Over the course of this period, I got the opportunity to study in-depth an up-and-coming topic in the field of geomatics, DGGS. As far as I know, this topic has rarely been researched in academia and therefore there was a lot of new ground to explore. This was as daunting at first as it was exciting. However, with a sense of adventure, I decided to step foot into unexplored territory.

The Geomatics programme at TU Delft provided me the core foundation needed for conducting such a kind of research, as it taught me invaluable skills in data collection, processing, analysis, and visualization. My research was a combination of several topics, such as point clouds and remote sensing, databases, cartography, geodesy, positioning, and web GIS and visualization. I was able to utilize the knowledge from several of the core foundation courses of the first year, such as Sensing Technologies, GIS and Cartography, Positioning and Location Awareness, Geo-Database Management System (DBMS), and Geo-Web Technology, to name the most important. Sensing Technologies taught me skills in data acquisition; GIS and Cartography skills in spatial analysis and map generation; Positioning and Location Awareness skills in geodesy (projections, CRS's, datums, etc.), surveying, and navigation; Geo-DBMS skills in data storage and retrieval; and Geo-Web Technology skills in sharing geospatial data with others across the Web. I also honed my skills in Python in the Python Programming class, and this greatly aided me in writing the code for my thesis, which was almost all done using this language. I also wrote code in Javascript and C#, and although these are different languages, having a strong foundation in one language allows one to easily grasp other languages. My thesis followed a methodical procedure much similar to the order in which the core foundation courses were taught. For example, I first had to acquire the point cloud data, then store it in a DBMS, then process and analyze it, and finally use it for web visualization.

Over the course of my thesis, I was also given the privilege to present the state of my research at the OGC Technical and Planning Committee Meetings of September 2017 and March 2018, in Southampton, UK and Orléans, France, respectively. These were both invaluable opportunities to meet and network with experts in the DGGS and point cloud domains, exchange ideas, understand the work of others, and showcase my own work in front of an international audience. I got to meet the chairs and members of both the DGGS

and Point clouds DWG's and developed strong professional associations. As I had just begun work on my thesis at that time, I presented a proposed plan for my research during the September 2017 meeting. At this meeting I met Dr. Matthew Purss, whom I thank wholeheartedly for being available anytime to provide guidance and/or feedback on my questions. During the March 2018 meeting, I mainly presented the results from my research. I greatly thank my supervisors from both TU Delft and Fugro for allowing me the opportunity to attend these meetings.

My thesis topic is highly germane to the field of geomatics, as it involves many relevant aspects. It has a lot to do with geodesy and mapping and modeling the Earth. Moreover, due to the large volume of data in a point cloud and an equally large data structure such as a DGGS, scalability is important and it becomes consequential to use a spatial DBMS for indexing, clustering, and querying such massive datasets. Finally, data has to be shared with/displayed to others somehow, and spatial web visualization tools are needed. These are all various parts of the science of geomatics, which according to the website of the Master's in Geomatics programme is "concerned with the acquisition, analysis, management and visualization of geographic data".

2 THEORETICAL BACKGROUND AND RELATED WORK

This chapter provides an overview of all the topics relevant to and necessary to understand this thesis. It is organized as follows: sections 2.1,2.2,2.3, and section 2.4 provide an introduction to the techniques used to map and model the Earth. Section 2.5 discusses the key components of a DGGS, while section 2.6 elucidates various forms of indexing strategies for use on a sphere or an ellipsoid. Section 2.8 provides a detailed overview of the ISEA projection that has been used. Finally, section 2.9 elaborates on the key challenges in the handling of point clouds that are dealt with in this thesis.

2.1 COORDINATE REFERENCE SYSTEMS

A CRS is a means to describe the position of a point in space. CRS's are usually one of three types: 2-dimensional Cartesian systems, in which case separate X, Y horizontal coordinates are used; 3-dimensional Cartesian systems, in which separate X, Y, and Z coordinates describe a point's position; or *geodetic* coordinates, also known as *ellipsoidal* or *geographic* coordinates, in which latitude, longitude, and height above a reference ellipsoid that is used to model the Earth are used to describe a point's position [van der Marel, 2016]. 3D Cartesian systems usually originate from the center of mass of the Earth, in which case they are referred to as *geocentric* coordinates; however, sometimes the origin is placed somewhere on the surface of the Earth, in which case the resulting coordinates are referred to as *topocentric* coordinates [van der Marel, 2016]. Each of these kinds of CRS's have their own unique advantages and disadvantages. Geodetic coordinates are actually angular measurements that describe position using curvilinear coordinates on a sphere or an ellipsoid. A DGGS greatly differs from such conventional CRS's and provides an alternative approach to representing the locations of objects on the Earth's surface (see Section 5.3).

2.2 MAP PROJECTIONS

Locations on a 3-dimensional model (for example, a sphere or an ellipsoid) that represents the Earth are usually provided in angular geodetic (latitude, longitude) coordinates. However, in practice, it is more common to transform these angular coordinates into 2-dimensional X, Y *grid* coordinates on a map projection since these *projected* coordinates are easier to work with than their angular counterparts. A map projection is a systematic transformation of the geodetic coordinates of locations on the surface of an Earth model such as a sphere or an ellipsoid into coordinates describing locations on a flat, 2D Cartesian plane [Snyder and Voxland, 1989] without any loss

of information [van der Marel, 2016]. Grid coordinates can originate from any point on the projection; in many cases, however, they are set to originate from the lower left of the projection, thereby turning them into positive *False Eastings*(x) and *False Northings*(y) [van der Marel, 2016]. In theory, there are an infinite amount of map projections [Snyder, 1993]. A map projection can preserve angles, areas, distances, or directions, but never a combination of all of the above. Furthermore, no map projection can preserve both area and angles (i.e. be both conformal and equal-area). The map projection to be mainly used in this thesis is the ISEA projection, more information about which is provided in section 2.8. Most DGGS implementations are based upon a map projection, and usually an equal-area one, such as ISEA.

Examples of some commonly used projections are the Mercator, Robinson, and Lambert Azimuthal Equal Area (LAEA). 2-dimensional CRS's are defined on top of these projections for locating features on the surface of the Earth. For example, the Universal Transverse Mercator (UTM) coordinate system is defined on top of a worldwide Transverse Mercator projection.

2.3 DATUMS

A *geodetic datum* defines a means to link a coordinate system to a model of the Earth, such as a sphere or an ellipsoid. More specifically, it defines the size and shape of the Earth model and the *origin*, *orientation*, and *scale* of the coordinate systems that are linked to that model [Dana, 2017]. Separate datums can be used for horizontal and vertical position measurements. Every CRS has its own datum. Examples of some common datums are North American Datum of 1983 (NAD83) and North American Datum of 1927 (NAD27) (*horizontal*), North American Vertical Datum of 1988 (NAVD88) (*vertical*), and WGS84 and ETRS 1989 (*composite, both horizontal and vertical*). Every observation in the real world, such as a vector linear feature or a point in a point cloud, is collected in some original CRS. It is important to remember that transformations between CRS's on two or more different datums lead to a certain loss of information or error, whereas transformations between CRS's *on the same datum* are lossless i.e. they do not introduce any errors or loss of information on the original observations[van der Marel, 2016].

2.4 EARTH MODELS

The Earth is best approximated as an *oblate spheroid*, a mathematical construction resembling a sphere squashed on the top and bottom sides so that the diameter from pole to pole is less than the diameter of a line going from the Equator through the center to the other side on the Equator again. It is almost similar to an *ellipsoid*. The two most common ellipsoids in use today are the Geodetic Reference System of 1980 (GRS80) and WGS84 ellipsoids [ICSM, 2016]. The two ellipsoids are nearly identical to each other, with the only major difference being the length of their semi-minor axis (half the distance from the North to the South pole) [Neacsu, 2011].

Using a global network of monitoring stations in a *reference frame*, the location of the axis of rotation (the line connecting the North and South poles of the Earth) and the position of any point relative to the center of the Earth can be determined in 3D geocentric coordinates. These geocentric coordinates can then easily be converted into ellipsoidal coordinates by fitting a reference ellipsoid into this 3D Cartesian reference frame and converting the geocentric coordinates into latitude, longitude, and ellipsoidal height. The two main reference frames in use today are the International Terrestrial Reference Frame (ITRF) and WGS84 reference frames. The ITRF is linked with the GRS80 ellipsoid and the WGS84 reference frame is linked with the WGS84 ellipsoid [ICSM, 2016]. It is important to note that due to plate tectonics, reference frames have to be continuously updated to correct for plate motion. These corrections are known as *realizations* and reflect improved measurements of station positions and velocities, datum definitions, and newly added or discontinued stations [van der Marel, 2016].

For the rest of this thesis, the WGS84 *ellipsoid*, and not a sphere, was used as a basis on which to construct a DGGS, as it is one of the most popular global terrestrial reference systems, not the least because it is also the basis for the American GPS system. Moreover, Vincenty's direct and inverse formulas can be used on the WGS84 ellipsoid, and they are more accurate than methods that assume a spherical Earth such as great-circle distance, because they assume the Earth to be an oblate ellipsoid (which is also what is used by WGS84) [Rooy, 2016]. The direct formula computes a point's location on an ellipsoid given a distance and bearing/direction from another point; the inverse formula computes the geographical distance and bearing/direction between two points. WGS84 is also the most globally accurate Earth model. In some cases, however, concepts are simpler to explain if a sphere is assumed, and this is stated wherever necessary.

2.5 DISCRETE GLOBAL GRID SYSTEMS

A DGG is a hierarchical tessellation of the Earth's surface into a set of cells at discrete resolutions that forms a complete partition of the surface. That is, there are no gaps or overlaps between the cells and the cells completely cover the Earth's surface. Each cell in the tessellation has a single point inside of it that is representative of it [Sahr et al., 2003]. Usually, this is chosen to be the centroid of that cell. The cells do not necessarily have to be of the same shape or size in between resolutions. The cells at a single resolution constitute a DGG. A collection of these DGG's constitutes a DGGS. As each of the cells of the DGG's that form a DGGS is of a different size/resolution, a DGGS consists of a series of increasingly finer resolution grids [Sahr et al., 2003] organized in the form of a hierarchy.

The conventional latitude/longitude grid or *graticule* is a popular example of a DGGS. It partitions the Earth's surface into a series of cells, each of which is bounded by a certain latitude and longitude. A limitation of this approach to subdividing the Earth is that these cells are not equal-area [Sahr et al., 2003], and become progressively distorted in area, shape, and inter-cell spacing as one moves north or south of the Equator [Sahr et al., 2003]. This makes it increasingly difficult to perform any kind of geostatistical

analysis using such grids, as the underlying cells all do not have the same area. The North and South poles map to 1-dimensional lines in such grids, when in reality they are simple 0-dimensional points on the Earth's surface. Furthermore, the top and bottom rows in this form of grid structure are squares, when in reality they should be triangles on the plane, as they share a common point (i.e. the pole). Although the graticule is familiar to most people, for statistical surveying or sampling purposes [Sahr et al., 2003] it is not apt for use. Due to the large distortions in area, each cell does not provide an equal probability of contributing to an analysis using such a grid.

It would be ideal if there was an alternative to the graticule that provides an equal-area tessellation of the Earth's surface into a hierarchy of resolutions. Indeed, this is exactly what is provided by a geodesic equal-area DGGS. DGGS's are polyhedral reference systems on the surface of a base polyhedron's circumscribed ellipsoid [Purss et al., 2017]. One of a number of polyhedrons, usually either a Platonic or an Archimedean solid, can be chosen as the base polyhedron for a DGGS, and its faces subdivided into a series of finer resolution cells, which are then inversely projected using an equal-area projection onto the surface of a sphere or an ellipsoid that represents the Earth. The basic idea of the equal area projection is that the area of an infinitesimal disk on a polyhedron and its mapping to the sphere are preserved. Since the area of a cell on the planar face of a polyhedron is the integration of all of these infinitesimal disks, the area of a cell is also preserved. Since, at any particular resolution, all the cells have the same area in the refined polyhedron (i.e. they are equal-area), their area after mapping to a sphere or an ellipsoid is also the same within that resolution. Unfortunately, no map projection can preserve both shape (angles) and area; that is, it is impossible to have a projection that is both conformal and equal-area [Olson, 2006]. Therefore, although it is possible to achieve an equal-area tessellation of the ellipsoid using a base polyhedron in a DGGS, the *shapes* of the cells on the ellipsoid after the inverse projection can be heavily distorted.

A DGGS consists of a number of design parameters. These include: 1) a base regular polyhedron 2) an orientation of the polyhedron relative to the chosen Earth model 3) a subdivision method defined on each face of the polyhedron, including the choice of a shape 4) the amount of refinement applied to each parent cell at a particular resolution in the hierarchy to yield the corresponding children cells at the next higher resolution, known as the *refinement ratio* or *aperture* and 5) a method to transform the cells from a planar surface to that of the chosen Earth model [Sahr et al., 2003] i.e. from the planar faces of the polyhedron to a sphere or an ellipsoid that represents the Earth. Therefore, it is readily apparent that there are various kinds of DGGS's, each with their own unique properties. A DGGS can be chosen that is optimal for a particular application or use-case. The use-case for this thesis is in handling point clouds that have varying initial CRS's, LOD's, and/or times.

2.5.1 Base Polyhedron

The most commonly used polyhedrons in the creation of a DGGS are the 5 Platonic solids, namely the tetrahedron, cube, octahedron, dodecahedron,

and icosahedron. These are depicted in Figure 2.1. Less commonly explored polyhedrons are the 13 Archimedean solids, such as the truncated octahedron, or the icosidodecahedron. The key difference between Platonic and Archimedean solids is that Platonic solids consist of surfaces of only a single kind of regular polygon, whereas Archimedean solids consist of surfaces of more than a single kind of regular polygon [Snyder, 1992]. Many of the Archimedean solids are truncated, or "cut-off", versions of the Platonic solids or combinations of two Platonic solids. The Archimedean solids are depicted in Figure 2.3. Unfortunately, in general these polyhedra remain unexplored as choices for a base polyhedron in DGGS. Therefore, this thesis has focused on the Platonic solids, and specifically the icosahedron as it is a better approximation of the Earth than any of the other four Platonic solids [Amiri et al., 2015a, 2016]. The icosahedron has smaller face sizes than all of these other solids, and therefore the resulting shape and area distortions when it is mapped to the ellipsoid are minimized [White et al., 1998].

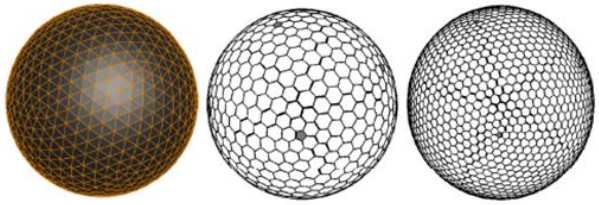

Figure 2.1: Triangular cells (left), and hexagonal cells at two levels of resolution (middle and right) in a DGGS. Figure from Purss et al., 2017.

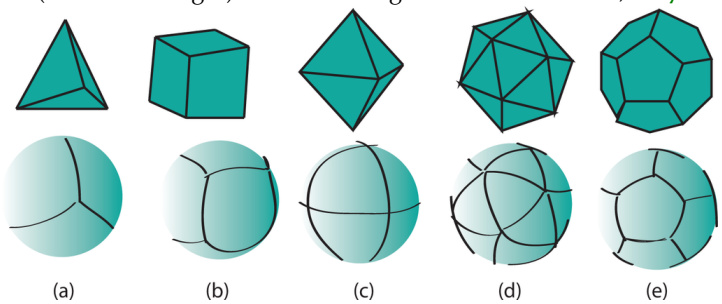

Figure 2.2: A base polyhedron (top) and its initial equal-area tessellation of the sphere (bottom). *a* = tetrahedron, *b* = cube, *c* = octahedron, *d* = icosahedron, and *e* = dodecahedron. Figure from Purss et al., 2017.

2.5.2 Polyhedron Orientation

Once a polyhedron has been defined, its orientation relative to the Earth model can be chosen. For example, one vertex of the icosahedron can be aligned with the 'North Pole' of the sphere or ellipsoid that represents the Earth, which due to the nature of the icosahedron would also mean that there is another vertex that is tangent to the 'South Pole' of the Earth. With this orientation, bearings between points on an icosahedron projected onto the ellipsoid can be computed easily, and it is also relatively trivial to tell on which icosahedral face a point is located. Another possible orientation tries to minimize the number of icosahedron vertices that fall on land, to avoid any ruptures in the landmass when the icosahedron is unfolded onto the plane [Sahr et al., 2003]. However, for the purposes of making the OPCM, polyhedron orientation is not considered to be a significant factor, as theoretically point clouds could be located anywhere on the surface of the Earth,

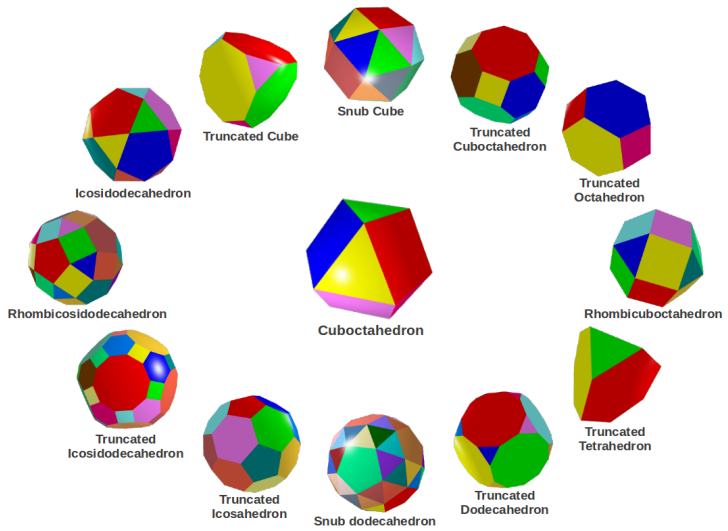

Figure 2.3: The Archimedean solids. Figure from Sacred Geometry, 2017.

including in the oceans. Therefore, the classic pole-aligned orientation was chosen for this thesis, as it provides for ease of use and computations.

2.5.3 Subdivision Shape

A variety of shapes can be used to subdivide the planar faces of the icosahedron into cells of finer resolution. These include the hexagon, pentagon, rhombus, and triangle. It is wise to compare the advantages and disadvantages of using each of these shapes before making a selection.

Squares, although they have a simple and familiar geometry, have the disadvantage that their geometry makes them unusable on triangle-faced polyhedrons such as the icosahedron [Sahr et al., 2003] that are better approximations of the Earth than other polyhedrons such as the cube. However, their skewed counterpart - *rhombuses* - can be used on the triangle-faced icosahedron (see Figures 3.2 and 3.3 for an illustration of how rhombuses can be used on an icosahedron). Although in layman's terms a rhombus is also known as a *diamond*, the proper mathematical term for this shape is a rhombus and this is what was used throughout this thesis.

Hexagons possess several favorable properties such as maximal compactness [Sahr et al., 2003], greatest angular resolution [Golay, 2000], and *uniform adjacency*, meaning that the centers of adjacent hexagons are the same distance from one another[Gregory et al., 2008]. However, a significant disadvantage lies in the fact that hexagonal DGGS's are *incongruent*: it is impossible to decompose perfectly a parent hexagon into smaller children hexagons. This complicates implementing a hierarchical indexing scheme on hexagonal cells. Furthermore, it is impossible to completely subdivide a spherical icosahedron into hexagons [Sahr et al., 2003], as pentagons will be formed at each of the vertices of an icosahedron. This necessitates a modification of existing algorithms that need to take both of these shapes into account. It should be noted, however, that as hexagons are the most compact of all the shapes, they are ideal if Tobler's First Law Of Geography is to

be considered: "everything is related to everything else, but near things are more related than distant things" [Miller, 2004]. One of the objectives of this thesis is to formulate a method to *index* and retrieve global point clouds in the OPCM. That is, it should be possible to retrieve points in a highly efficient manner based on a spatial, temporal, or spatio-temporal query. Therefore, a hierarchical indexing mechanism is needed in order to assign a proper identifier to each point in a global point cloud. It would also be ideal to spatially/temporally *cluster* the points; that is, to find a method that can be used to store points that in reality are close to one another also close to one another in a database. The cell identifier should indicate the approximate location of the point on the Earth model, and it should be possible then to retrieve points based upon this identifier. Due to their non-hierarchical nature, hexagons are not a proper choice for a DGGS for this thesis.

Triangles have the advantage of allowing for rapid visualization and rendering, as many pipelines in the cross-platform, cross-language Application Programming Interface (API) OpenGL framework support such structures [Amiri et al., 2015a]. Modern Graphics Processing Unit (GPU)'s are particularly optimized at rendering triangles [Loop and Blinn, 2017]. However, triangles suffer from non-uniform orientation at successive resolutions. At each successive resolution, the orientation of the inner triangle flips. This complicates the implementation of algorithms such as adjacency analysis, spatial queries, and data update [Bai et al., 2005] on DGG's built from them, as they must take this changing orientation into account. One of the earliest implementations of a DGGS was the Octahedral Quaternary Triangular Mesh (O-QTM) [Dutton, 1996]; however, this uses a triangle subdivision that suffers from such a non-uniform orientation at different levels of the subdivision; as a result, spatial-analytical operations are much more complicated.

This leads to the choice to use a *rhombus* as a spatial partitioning method in a DGGS. [White, 2000] and [Bai et al., 2005] provide several advantages rhombuses provide over other shapes. First, the geometry of a rhombus is simpler than that of a triangle or a hexagon [Bai et al., 2005]. The rhombus tessellation has uniform orientation at successive resolutions (see Figure 2.4) [Bai et al., 2005], as well as *radial symmetry* (see Figure 2.5) and *translation congruence* (see Figure 2.6) [White, 2000]. Radial symmetry refers to the fact that there is a point the same distance in the opposite direction from the center of the rhombus as another point. [White, 2000]; this provides a certain balance in how far away a point observation can be from the centroid of a cell and in the assignment of an observation to a cell. Translation congruence refers to the fact that rhombuses maintain the same size, shape, and orientation at each level in the DGGS hierarchy [White, 2000]. Perhaps the most beneficial property of a rhombus tessellation is that quadrant-recursive orderings, such as Morton or Hilbert SFC's, can be used to index the cells of a rhombus-based DGGS (more information about SFC's can be found in Section 2.6.3). The rhombus hierarchy is nested, such that each parent rhombus completely contains all its children rhombuses [White, 2000]. Moreover, the two most important needs of the global data user community are equal-area cells and a nested hierarchy among cells [Gregory et al., 1999], and a rhombus satisfies both, unlike a hexagon. This property is very desirable for indexing a global point cloud in a rhombus-based DGGS. Using a Morton curve, the cells of a rhombus-based DGGS can be uniquely indexed in *both* a hierarchical and space-filling manner. Therefore, this allows for the implementation of both an efficient indexing *and* clustering mechanism. One of

the key requirements of any DGGS implementation is that each cell across the entire domain of a DGGS has a spatial reference (index) assigned to it via a spatial referencing method that uniquely identifies it within the DGGS [Purss et al., 2017]. Morton indexing and its relationship to a hypercube can be used to accomplish this task on a rhombus-based DGGS.

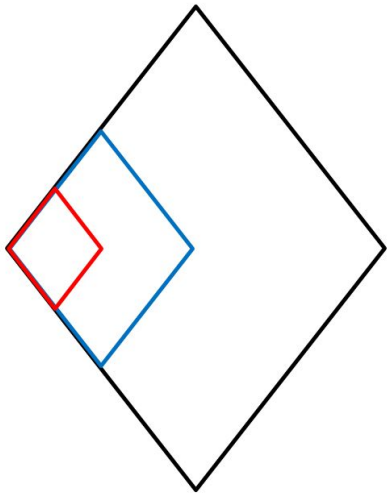

Figure 2.4: A rhombus tessellation offers uniform orientation at successive levels of refinement / resolution

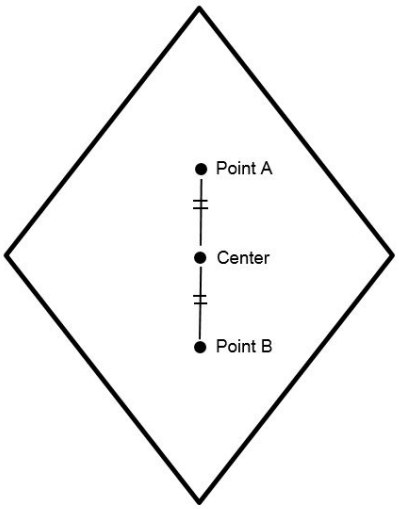

Figure 2.5: A rhombus tessellation possesses radial symmetry about its center. The distance from point A to the center is identical to the distance from point B to the center.

2.5.4 Refinement Ratio and Transformation

Rhombuses allow for a nested, congruent aperture 4 hierarchy: each parent rhombus can be perfectly subdivided into 4 children rhombuses. This allows for quadtree-based algorithms, such as Morton indexing, to be applied. Quadtrees are explained in more detail in Section 2.6.1. It should also be noted that rhombuses can also be subdivided into n^2 children, with

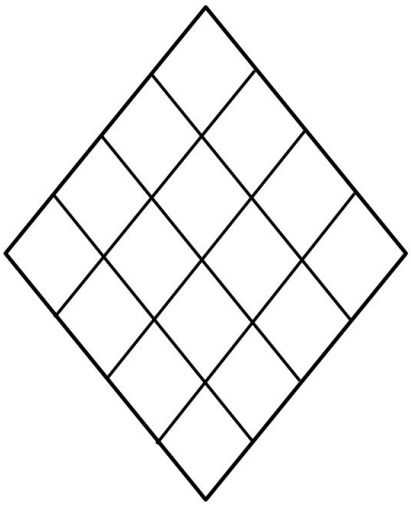

Figure 2.6: All rhombuses in a single resolution DGG grid have the same size, shape, and orientation.

n being a positive integer larger than 2; however, by definition, a quadtree cannot be used in this case.

Finally, a method is needed to transform the cells created on the planar faces of the icosahedron onto the ellipsoid. One of the fundamental requirements of any DGGS is that the cells at any resolution should all be equal-area or approximately equal-area [Purss et al., 2017]. This provides each area on the Earth's surface an equal probability of contributing to an analysis [Purss et al., 2017]. The rhombus cells on the triangular-faced icosahedron all have the same areas and shapes (internal angles). To preserve their areas on a sphere, an (inverse) equal-area projection is needed. The ISEA projection can be used for this task, as it provides for relatively low linear scale variation and angular deformation [Snyder, 1992] and provides an equal-area mapping between the icosahedron and the ellipsoid. Figure 2.2 shows the Platonic polyhedrons along with their initial equal-area tessellations on a sphere. The initial equal-area tessellation represents Resolution 0 of a DGGS. The ISEA projection is explained in more detail in Section 2.8. Therefore, the DGGS that has resulted from the above considerations is an icosahedral, pole-oriented, rhombus-based DGGS with a refinement ratio of 4 and the ISEA projection to be used to transform from the icosahedron to the WGS84 ellipsoid.

Although DGGS's have been in existence since at least the 20th century, they were officially standardized and released as a specification by the OGC in October 2017. Therefore, as of this writing, DGGS's remain not widely understood or used technologies throughout the geomatics community [Open Geospatial Consortium, 2017a]. Moreover, their use with respect to point clouds is even more limited. The OGC DGGS specification does mention the term 'point cloud' in several places in its writing [Purss et al., 2017]. In contrast to a growing amount of literature on vector and raster data handling with a DGGS, however, there does not exist any academic literature on point cloud data handling with a DGGS. However, a DGGS is designed to work with vectors, rasters, and/or point clouds in a seemingly interoperable way. All of these data structures can ultimately be reduced to a set of points; a

vector polygon is composed of lines (edges) that are in turn composed of points (vertices); a raster is a gridded data structure in which each cell contains one value and the cell's center point represents that value, so it can also be reduced to a lattice of point values; and a point cloud by definition is a collection of points. Therefore, the same kinds of spatial analytical operations that can be applied on vector and raster datasets in a DGGS can also be applied to point clouds.

2.6 INDEXING STRATEGIES

The practical use of DGGS-based systems has been hampered by a lack of proper spatial indexing techniques [Sahr, 2008]. However, in order to be usable, a DGGS implementation must assign a unique location code to each cell of a DGGS [Purss et al., 2017], thereby converting the DGGS into a spatial database that can be efficiently queried and analyzed [Sahr, 2008]. A unique location code for each cell at every resolution provides a means to associate observations collected in the real world to a single location code, that of the cell to which they are assigned. A rhombus DGGS conveniently lends itself to efficient indexing strategies for querying and retrieval of point clouds.

2.6.1 Quadtrees

A *quadtree* is a tree data structure in which, at each level, each internal node forks into four children nodes [Ottoson and Hauska, 2002]. A quadtree is picture in Figure 2.7. Quadtrees are usually used in a 2-dimensional coordinate system that does not take the curvature of the Earth into account. At progressively large scales, however, this approach can lead to increasingly large errors, as the areas of cells within a quadtree differ more and more from their 'true' areas on the curved surface of the Earth. As a rhombus DGGS is essentially a linear quadtree [Bai et al., 2005] on the plane, a quadtree-based decomposition of the surface of the Earth can be achieved by indexing cells and their contained observations on the ISEA map projection plane and inverse-projecting them onto the ellipsoid. It is important to remember that not all shapes in a DGGS can be indexed as a quadtree. For example, pentagons or hexagons do not lend themselves easily to hierarchy-based quadtree algorithms [Sahr et al., 2003].

A quadtree is invaluable in an aperture 4 based 2D DGGS, and can be viewed as a means to index a 2-dimensional hypercube. For indexing higher dimensional hypercubes (such as regular cubes and tesseracts, in 3D and 4D respectively), a quadtree can be extended into an octree and a 4-dimensional hypercube, respectively.

2.6.2 Pyramid And Path Addressing

[Sahr, 2008] introduces the two types of multi-resolution location coding systems in use with DGGS: *pyramid addressing* and *path addressing*. Pyramid addressing assigns a unique location code to a cell within its containing DGG. Therefore, pyramid addressing is only useful for *single-resolution* spatial or statistical analysis. The multi-resolution DGG location code for a location

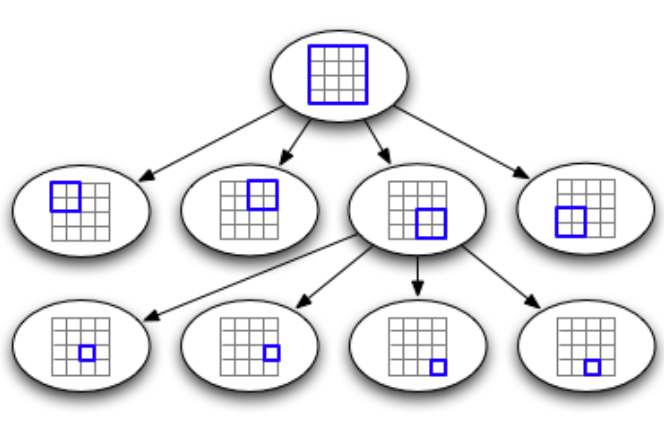

Figure 2.7: A quadtree in a plane. At every iteration, the parent cell is subdivided into 4 children. Figure from Johnson, 2009.

can be obtained by listing the series of single-resolution pyramid addresses for that location, ordered by increasing resolution [Sahr, 2008]. A significant limitation of this approach is that due to its single-resolution nature, relationships between cells at different levels of a DGGS hierarchy cannot be known based only on a single pyramid address. Therefore, interpolation and/or decimation of point observations through the DGGS hierarchy cannot be performed as no hierarchical structure is available. Furthermore, pyramid addressing can lead to a lot of redundancy [Sahr, 2008] in data storage as the location of the same point on the Earth's surface is recorded once for *every* resolution in the DGGS, which is unnecessary. Multiple representations are stored, and potentially separate databases will need to be maintained and harmonized [Dutton, 1996], one for each resolution. For efficient indexing of and retrieval from a global point cloud, this is undesirable.

A *path* address, on the other hand, is essentially just that: a path through the tree structure of a DGGS beginning at the root (i.e. the whole Earth) that encodes the location of a point on the Earth's surface. Path addresses provide several advantages: the length of the address indicates the approximate resolution and precision level in a DGGS, eliminating the need to store the (discrete) precision separately as metadata; they do not store redundant information [Sahr, 2008] and can reduce data size [Bartholdi and Goldsman, 2001], as only a single address can be used to identify the location of a feature at any resolution in the hierarchy, and this address can also be used to find the path addresses at *all* coarser resolutions; they also simplify the implementation of hierarchical algorithms and greatly improve the performance of spatial queries, such as containment, intersection, or adjacency, on the data [Sahr, 2008]. A rhombus DGGS conveniently lends itself to a path addressing system due to its congruent, nested nature, and this is what was utilized in this thesis. Consecutive numbers in the complete path address identify sub-regions within the parent rhombus, and thereby all observations (i.e. points) that are not located within the corresponding child rhombus can be ignored instantaneously in a spatial query, thereby significantly speeding up query execution time. Also, for the DGGS chosen for this thesis, each parent rhombus cell has four smaller children cells. Therefore, in 2D DGGS, one of four digits (0-3) is added to the location code of the parent cell to yield the location code of the child cell. Each additional digit can be rep-

resented using 2 bits (base 4), and therefore no bits are wasted during index assignment. For 3D and 4D DGGS (see Sections 3.2.1 and 3.2.1), one of 8 or 16 digits/letters, respectively, is added to the parent location code to yield the child location code. Each path address location code for a rhombus also automatically encodes: the number of cells in the whole-Earth DGG at that resolution and the inter-cell spacing between cells at that resolution.

A disadvantage of path addresses is that they can get rather long, especially for extremely high precision/resolution levels. For example, to achieve *sub-meter* precision, up to at least 23 digits could be needed to encode the location of a point with that precision using a path address in an icosahedral rhombus DGGS. However, these addresses can be truncated to yield a coarse filter (representation of the same location at a coarser resolution) for spatial analytical operations such as containment, intersection, and/or adjacency [Sahr, 2008]. This means that coarser resolution cells can be used as a filter for spatial queries at finer resolutions, because they completely cover these finer resolution cells (i.e. are congruent), thereby speeding query execution time.

2.6.3 Space Filling Curves

A SFC can be defined as a linear traversal of discrete finite multidimensional space [Gotsman and Lindenbaum, 1996] . It is a path through space that traverses through all the elements of that space *at a given resolution*. It therefore provides a mapping from n-dimensions into 1 dimension, a line. The mapping is a linear ordering of the elements along the SFC [Psomadaki et al., 2016]. Consequently, utilizing a SFC can allow for efficient *clustering* of spatial data. Spatial clustering refers to grouping together objects that are close together in reality (i.e. on the Earth's surface) [Pfoser et al., 2011]. Clustering can also be performed temporally. Spatio-temporal clustering provides for better performance of queries in a DBMS, as objects close to one another in the real world in space and time are also stored close together in the database so that they can be accessed together in a query. The number of disk accesses is reduced [Bartholdi and Goldsman, 2001]. Furthermore, the derived one-dimensional locations along the SFC can be sorted [Bartholdi and Goldsman, 2001] and indexed using data structures such as a B-Tree [Psomadaki et al., 2016].

Several types of SFC's exist. Some common ones are Morton, Hilbert, Peano, Row, Row-Prime, and Moore. Each has a certain set of unique properties. The SFC to be used in this thesis is the Morton curve, particularly because it can be used in a rhombus-based DGGS and because of its innate relationship with a quadtree. The Morton curve on a plane and its relationship to a quadtree is depicted in Figure 2.8.

It is important to remember that the indexing will ultimately need to be implemented on an ellipsoid, and not on a flat Cartesian plane, thereby facilitating spatial analysis at increasingly large scales where the curvature of the Earth becomes a limiting factor. A DGGS structure provides a means to achieve this goal. An *ellipsoidal* space-filling curve provides a 1-1 mapping from a one-dimensional location on the curve to a point on the ellipsoidal surface [Bartholdi and Goldsman, 2001] and vice-versa. In general, this conversion is not perfectly reversible; that is, converting from a SFC index

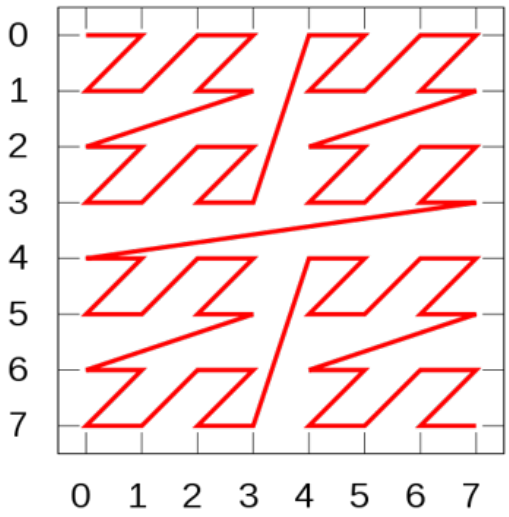

Figure 2.8: The quadrant-recursive Morton SFC on a planar surface.

to a position on the ellipsoid and back *might* not always result in the *exact* same SFC location, but it will still be approximately close [Bartholdi and Goldsman, 2001].

A SFC ordering is said to be *continuous* when cells with sequential indices on the curve at least share a common vertex [Bartholdi and Goldsman, 2001]. Most of the indexing schemes proposed to-date have been discontinuous, including Dutton's O-QTM [Dutton, 1996], which uses triangles. Indexing the Earth using a rhombus-based DGGS provides such a continuous ordering, where all of the cell centroids at any one resolution are connected on the SFC without any discontinuities and taking into account their spatial proximity to one another. The rhombus centroids that represent the 'start' and 'end' point positions of the curve are also sequential positions on the curve. Continuous orderings are usually much shorter than discontinuous orderings, and are invaluable in spatial, analytical, or logistical operations [Bartholdi and Goldsman, 2001]. A rhombus-based ordering is also *order-consistent*, that is, the order of cells along the SFC does not change at different levels of the subdivision [Bartholdi and Goldsman, 2001]. This means that rhombus cells at resolution k will have children at resolution $(k + 1)$ that are in the same order along the SFC as the cells at resolution k. The rhombus SFC ordering is also *adjacency-preserving* because the relationships with neighboring rhombus cells can be easily found using their Morton codes. Furthermore, an SFC encoding can be extended into n-dimensions in a DGGS; for this thesis, $n \leq 4$.

In conclusion, a host of beneficial properties are provided by the above proposed indexing of the Earth using a rhombus DGGS, and these are summarized below. The ordering is:

- Path-based
- Hierarchical

- Space-filling
- Continuous
- Order-consistent
- Adjacency-preserving
- Precision-encoding
- N-dimensional

2.7 DIMENSIONALLY EXTENDED 9-INTERSECTION MODEL

Spatial objects of any type of geometry (point, line, or polygon) can have spatial relationships with other objects. The Dimensionally Extended 9-Intersection Model (DE-9IM) is a framework that provides an approach to determining the spatial relationships between features by considering the dimensions of the *intersections* of the interiors, boundaries, and exteriors of these features [Strobl, 2008]. The dimension returned by a DE-9IM query is a number from the set [-1,0,1,2], with -1 a null set (i.e. no intersection), 0 a point, 1 a line, and 2 an area intersection [ESRI, 2017b]. A host of spatial relationships are described by the DE-9IM, such as "equals", "intersects", "disjoint", "contains", and "within" [Strobl, 2008]. Although mainly used in spatial databases such as PostGIS, the DE-9IM is also the fundamental model on which a DGGS operates. A DGGS is essentially a large spatial database. Once constructed, the cell ID's at each resolution along with their geometry can be stored in the form of an array or list, and the various spatial relations provided by the DE-9IM can be tested as array processes, up and down the DGGS hierarchy. This is scalable. However, it must be noted that the construction, comparison, and querying of geometries is a procedure that requires considerable overhead. Although the geometries of cells can be compared with geometries of points in a point cloud, this is not what was utilized for the reasons mentioned above. This can be avoided completely by using an alternative query procedure, and this is described in Section 4.7.

2.8 ICOSAHEDRAL SNYDER EQUAL AREA (ISEA) PROJECTION

Area-preserving projections such as ISEA are important for various reasons. Spatial analysis is more meaningful when using cells that are all of the same area. Each cell would then represent an equal portion of the Earth's surface, and have an equal probability of contributing to the analysis at hand. For example, if we wish to extract intensity or elevation information from a point cloud to create a map showing the mean distribution of these values across a region, it is more preferable to conduct this analysis using cells that are all of the same area (even though their shapes could be distorted). Then, each resulting intensity or elevation value will represent the measured amount of that variable in an equally sized area. In many ways,

this is similar to the population vs. population density paradigm: even though a country could have an extremely high population, it might not necessarily be distributed evenly across its area. What is more meaningful in this case is to analyze the population *per unit area*, or population density. With equal-area cells, each measured value of the population density then represents the number of people living in an identically-sized area as any other measured value. Population density maps convey information at a finer LOD than simple population maps. Furthermore, equal area cells also enable standardized interoperability between DGGS and other geospatial data infrastructures [Purss, 2016].

The ISEA projection is a function that outputs coordinates on a flattened icosahedron given coordinates on a sphere (this can also be extended to an ellipsoid). The projection is shown in Figure 2.9. The 20 triangular faces of the icosahedron have been flattened onto a plane. The projection is shown in its normal orientation (the orientation chosen for this thesis), with the North and South poles of the Earth each being vertices where 5 unique triangles converge. The projection essentially uses a modified form of the more popular LAEA projection, and centers it for each face of the icosahedron [Harrison et al., 2011].

The projection consists of a number of steps, with the overall goal being to equalize the areas of two right triangles, one defined on the planar icosahedral face and one defined on the spherical (or ellipsoidal) Earth surface [Snyder, 1992]. For a point on the sphere for which we wish to compute its planar ISEA coordinates, a sequence of steps are followed. First, a sphere is circumscribed around the icosahedron, and a *scaling factor* is determined between this sphere and the sphere that represents the Earth. Second, the areas of two scalene triangles (one on the sphere, the other on the plane) are equated for a given azimuth (direction or bearing) from a vertex of the spherical scalene triangle , and the matching azimuth from a vertex of the planar scalene triangle is calculated appropriately to obtain local areal equivalence. Finally, a point P is positioned along this computed planar azimuth to preserve overall areal scale [Harrison et al., 2011]. A proportionality factor is computed, and this is then used to compute rectangular/grid coordinates (x,y) on the ISEA projection. The coordinates initially originate from the center of the projection, that is, from the point with latitude 0^{o}, longitude 0^{o} on Earth. These coordinates can then be translated as needed to any other point on the projection. The mathematics behind this projection is far more complicated than the description above, and relevant formulae for forward and inverse projections are provided in [Snyder, 1992]. Although [Snyder, 1992] provides formulae that assume a spherical Earth, in order to get them to work on the WGS84 ellipsoid, Vincenty's formulae can be used. The in-depth discussion of these formulae is outside the scope of this thesis.

The ISEA projection can be used to construct a compliant DGGS. The faces of the flattened icosahedron (i.e. the triangles) can be subdivided into finer and finer cells until the desired resolution in the DGGS hierarchy is reached. The triangles can then be subdivided into hexagons, rhombuses, or smaller triangles. Two neighboring triangles can be combined to yield rhombuses. For the icosahedron, this projection yields a maximum angular deformation of 17.27%, a maximum scale factor of 1.163, and a minimum scale factor of 0.860 [Snyder, 1992]. This means that the linear scale does not vary less than

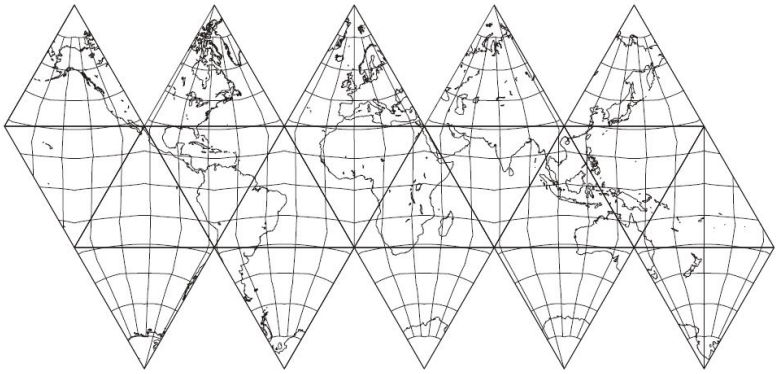

Figure 2.9: The ISEA projection provides a mapping from a sphere/ellipsoid to a flattened icosahedron. The 20 triangular faces are pictured. Figure from Harrison et al., 2011.

.

14% and more than 16.3% from the nominal map scale. This means that even if it is an equal-area projection, it does not *perfectly* preserve areas. However, it is convenient in the sense that it provides a 1-1 mapping between an icosahedron face and a sphere/ellipsoid, and is one of the most popular projections used in DGGS. In fact, the dodecahedron has less angular and areal distortion than the icosahedron, but due to its pentagonal faces, it is not of much use for the global indexing of point clouds.

2.9 POINT CLOUDS

A point cloud is a collection of data points in a given CRS. The CRS could be 2-dimensional (X, Y), but is usually 3-dimensional (X, Y, and Z). Furthermore, each point in a point cloud usually has not only positional attributes that identify its location in space in a given CRS, but also other attributes such as a GPS timestamp, color, classification, and intensity [ESRI, 2017d]. It inherently possesses an n-dimensional nature. An explanation of some of the more important attributes associated with points in a LIDAR point cloud follows, with the ones most relevant to this thesis listed first:

- **X, Y, Z**: These are the coordinates of the point in a 2 or 3 dimensional coordinate system. X and Y are horizontal coordinates, whereas Z is a vertical coordinate that represents the elevation of the point above the surface of the Earth in the corresponding coordinate system. For a bathymetric point cloud, Z will be negative. Many times, if these coordinates are really large numbers, they can be converted into a *local coordinate system* using topocentric coordinates, yielding smaller numbers. This is achieved by translating/shifting and/or rescaling the original coordinates. This is the approach used by the popular point cloud processing software *CloudCompare* [CloudCompareWiki, 2016].

- **Time**: This is the time of acquisition of the point, that is, the time at which the pulse that captured the point was emitted from the data collection platform. This is usually stored in the form of *GPS Time* for

LIDAR datasets. GPS Time is a uniformly counting time scale beginning at 12 AM on January 6, 1980. GPS Time represents the current time in weeks and seconds of that week since this date. One of the main differences between GPS Time and regular UTC time is that UTC time takes into account the Earth's rotation by adjusting using leap seconds, whereas GPS Time does not incorporate leap seconds [NPS, 2016]. However, GPS Time can be converted to UTC Time and vice versa.

- **Intensity**: This refers to the strength of the pulse that was returned to the scanner by an object that reflected the pulse. Many factors can influence the intensity, including the reflectivity and composition of the target object [ESRI, 2017c], near infrared energy absorption and the amount of obstruction to the pulse as it travels through space.

- **Classification**: The classification defines the type of object that reflected the laser pulse that generated the point. Every point can have a classification, such as "bare ground", "water", or "vegetation". The industry-standard LASer (LAS) file format, in its 1.4 version, contains up to 256 possible classification codes [ESRI, 2017a].

- **RGB**: An acronym for *Red, Green, Blue*, RGB refers to the reflectance in these three bands of the electromagnetic spectrum. Each point in a point cloud can be attributed with its RGB values. Many point clouds already come with these values, but if not provided, they can be assigned using color aerial imagery.

- **Return number**: Each pulse can have many potential returns. This simply identifies the number of the return that generated the respective point.

- **Number of returns**: This is the total number of returns for the pulse that generated the respective point.

It is important to state here that point clouds acquired from photogrammetry can have different attributes than those acquired from LIDAR; this thesis, however, has only focused on LIDAR point clouds.

The most common file format for handling point cloud data remains the LAS format, and this is what was utilized throughout this thesis. Point clouds, although most frequently acquired with LIDAR techniques, can also be acquired using Global Navigation Satellite System (GNSS), Terrestrial Laser Scanning (TLS), Radio Detection And Ranging (RADAR), Sound Navigation And Ranging (SONAR), leveling, or photogrammetry [Lemmens, 2014]. The LAS format can be used to work with any of these kinds of data, and not only laser data [ASPRS, 2013]. It is a public binary file format that allows for more interoperability between systems in comparison with its contender, the American Standard Code for Information Interchange (ASCII) format, and can be used to work with any 3-dimensional data. Furthermore, many existing software systems are already capable of working with LAS files, such as *FugroViewer*, *LASTools*, *CloudCompare*, and *FME*.

2.9.1 An Additional Dimension

A point in a point cloud can have not only a unique X, Y, Z and time value, it can also be assigned an additional dimension that indicates its *importance* as compared to all of the other points in the point cloud. 'Importance' is just that: the *relevance* or *significance* of a point as compared to other points. A variety of methods can be used to compute this importance value, including a random function [van Oosterom et al., 2015], minimum distance to the nearest point, distance from the data collection platform (i.e. scanner exit point where pulse originated), and importance based upon neighborhood-based feature detection and identification using the point and its surrounding points within a certain distance. Importance is an indicator of the LOD of a point: points with a higher importance have a higher LOD than points with a lower importance. This added dimension can then be used to efficiently visualize point clouds that have varying densities in a smooth, continuous manner (see Section 2.9.2 for more information). The hierarchical nature of a DGGS can be utilized to create such a variable-scale representation of massive amounts of points.

Time is an extremely valuable dimension for point clouds, as it allows us to spatially analyze the changes that have taken place across a scanned area between two different times in history. A DGGS can be utilized to answer one of the most fundamental questions of big geospatial data analytics: "How has it changed?" [Purss et al., 2017], referring to how an area has changed over a span of time. This thesis has attempted to study the effective uses of a DGGS in monitoring spatial change.

Therefore, in essence, although there can be many additional dimensions for each point in a point cloud, five separate dimensions and their applicability to a DGGS were studied in this thesis. These include the positional dimensions X, Y, Z, in addition to importance/LOD and time.

2.9.2 Variable-Scale Visualization With DGGS

The density of a point cloud refers to the amount of measurements that have been sampled on the surface of the Earth in a given area within the point cloud [Rohrbach, 2015b]. This is the most discriminating feature of a point cloud when used for the purpose of visualization of heterogeneous point clouds in a common environment. That is, two or more point clouds used in the same visualization environment can be readily identified as separate entities if they have relatively varying initial point densities. Density is usually provided in units of *number of points per square meter*, and varies depending on the application at hand. For example, for basic surface models or forest inventory, density is usually in the range of 0.5-1 point/m^2; for basic 3D models in the range of 5-10 points/m^2, and for highly detailed (LOD3 or LOD4) 3D city models in the range of more than 10 points/m^2 [Rohrbach, 2015b].

A problem arises when trying to integrate points clouds with varying initial densities in a common visualization environment. Sharp "jumps" in point density will be seen at the borders between the point clouds, when each has a distinct discrete LOD. For example, the density of a less-detailed point cloud could be 5 points/m^2 whereas another point cloud with a higher LOD in the same area could have a density of 15 points/m^2. These discrete-LOD-based data representations, also known as multi-scale representations

[van Oosterom, 2005], although desired, when used as-is have the disadvantage of producing potentially drastic density shocks in the data [van Oosterom et al., 2015]. In an attempt to address this situation, an additional continuous *importance* dimension can be added to the points, and the points displayed according to a *perspective view query* that uses this additional dimension.

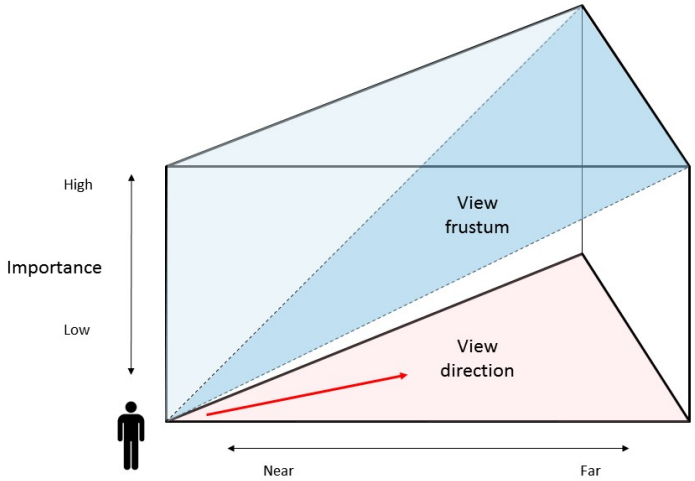

Figure 2.10: Depiction of a variable-scale view frustum. During visualization, both high and low important points are displayed closer to the observer position, and a gradual change is made to display only the most important points far from the observer.

This additional continuous dimension is called 'importance' [van Oosterom et al., 2015], and can be based upon any user-defined function. In the context of a DGGS, points in a point cloud are assigned to a cell based upon their *precision*: the level of measurement and exactness of description of observations [Foote and Huebner, 1995]; that is, it is the degree to which independent measurements of locations *of the same observation* agree to one another. An 'observation' as applicable to this thesis refers to a point in a point cloud, but in general could be any other type of feature such as a vector line, vector polygon, or raster cell. Precision is composed of two aspects: *repeatability* and *reproducibility*. Repeatability refers to the degree to which multiple measurements of a point's location will show the same result (i.e. same location). Reproducibility refers to the degree to which an experiment or study can be accurately reproduced or replicated by someone else. Precision is also an indicator of the spatial *uncertainty* of an observation: the higher the precision, the lower the spatial uncertainty, and the lower the precision, the higher the spatial uncertainty of a point observation.

A highly regular global grid should document the precision and location of spatial data on the globe [Gregory et al., 1999]. Every point in a point cloud has a spatial uncertainty that can be represented as an area feature around that point. A DGGS's hierarchical, multi-resolution structure is ideal for encoding the *discrete* precisions of points directly *on the surface of an ellipsoid*, and not in any map projection. If only an approximate indicator of precision is desired, this obviates the need to store the precision separately

as additional metadata, which is the usual case in the conventional GIS approach. If exact, continuous precision is desired, then DGGS by themselves do not inherently allow for encoding continuous precision. This has to be stored as additional metadata along with the observations, as precision is a continuous variable and DGGS's are by definition discrete. Once the continuous precision is stored alongside every observation, it can be modeled as an additional continuous dimension that can be used to visualize the points in a more effective and appealing manner. The smaller the size of a DGGS cell, the higher the precision because the area covered by the cell is small, indicating that the spatial uncertainty of points that are assigned to that cell is also small and that the points have been measured very precisely. Conversely, the larger the size of a DGGS cell, the lower the precision and the higher the spatial uncertainty of a point. Points which are not very precise are also not very important, and therefore during visualization they should be shown only at larger scales (i.e. when zoomed in sufficiently close). Points that have high precision are shown at both smaller and larger scales and are stored in higher levels of the structure during visualization.

Through assigning a point observation to a DGGS cell defined on the surface of the Earth that has *approximately* the same area as the precision of that point (a process known as *quantization* [Sahr, 2008], explained in more detail in Section 3.1), and visualizing the points based upon this continuous dimension, a true variable-scale visualization can be created. DGGS's by definition are *discrete*; that is, each cell at a particular resolution in the hierarchy of a DGGS has a fixed area. However, precision is continuous. Therefore, it can be utilized to provide a variable-scale visualization of points. That is, the points are not stored for any predefined scales (i.e. the discrete levels of a DGGS), but can rather provide levels of detail for an entire scale range, at any variable scale [Meijers, 2011]. Within a perspective view query, the lower importance points are gradually ignored when moving further from the observer position [van Oosterom et al., 2015]. Only the higher importance points are shown far from the observer, and both higher and lower importance points are displayed while close to the observer [van Oosterom et al., 2015]. As the observer position moves through space, the *view frustum*, a 3-dimensional polyhedron, changes. The view frustum is simply the field of view of the observer, that is, the region of space that appears on the screen in a visualization environment. It is useless to render points that do not intersect with the view frustum, as these points are not even visible. These points can be discarded using a process known as *view frustum culling*.

2.9.3 Change Identification With DGGS

A DGGS is an Earth Centered Earth Fixed (ECEF) reference system [Purss et al., 2017]. That is, the location and orientation of the polyhedron utilized for a DGGS are defined in ECEF coordinates. An ECEF system is a geocentric Cartesian coordinate system which represents positions on the surface of the Earth in coordinates originating from the center of mass of the Earth, hence "earth centered". It is "earth fixed" in the sense that the coordinates of any point defined in ECEF do not change due to tectonic movement and are not subject to any other motion (such as erosion or orogenesis), as ECEF rotates along with the Earth [Clynch, 2006]. Any point with coordinates in latitude, longitude can be readily converted into ECEF , and vice-versa.

Given that a DGGS is an ECEF system, the cells of a DGGS are fixed onto the surface of the ellipsoid that represents the Earth. One can think of them as lying in a static position 'below' the mobile tectonic plates above. It is only the *orientation* of a polyhedron in a DGGS that can be changed. Changing the orientation will also change the ECEF coordinates of all cell centroids. The fact that the cells of a DGGS are fixed once the polyhedron orientation is defined, is, however, ideal for the purposes of using a DGGS for identifying changes between two moments in time. Indeed, one of the three most fundamental questions of spatial analytics, namely "How has it changed?", can be effectively answered using a DGGS [Purss et al., 2017]. Even though the cells are fixed in position, observations such as points in a point cloud, vectors, or raster cells are mobile due to geophysical forces. Therefore, these will move across the DGGS reference frame over a period of time. Although in theory the movement of a point between two moments in time is a single numerical value, in practice the rate of movement varies depending on the datum used to measure the points; for example, station positions and velocities in the ITRS are on the order of a few centimeters/year, whereas in the ETRS they are only a few millimeters/year [van der Marel, 2016]. Therefore, metadata regarding the datum used to measure observations might need to be stored for certain applications.

As the cells of a DGGS are fixed onto the Earth model, changes between two moments in time can be analyzed on a cell-by-cell basis, as the cell is the fundamental atomic object of a DGGS reference frame [Purss et al., 2017]. The cells are all of equal-area (in 2D), and pseudo-equal volume (in 3D), which makes any kind of spatial or geostatistical analysis much more meaningful than when non equal-area/volume cells are being used. Statistics such as the mean, median, mode, minimum, maximum, or range of an attribute can be found in every cell, and the cells color-coded based upon these values.

One of the most common metrics for measuring the amount of change in the *surface* between two moments in time is Hausdorff Distance (HD). This involves computing the shortest distances between the two point clouds, for every point in the first point cloud. A disadvantage is that the HD is extremely sensitive to any possible variation in point density between two scans [Montaut et al., 2005]. Moreover, a DGGS encoding cannot be used directly to compute shortest distances between points, without performing a decode operation. However, other metrics can be used to compute change, not necessarily in the surface but in other quantities. For example, changes in point density, elevations, intensities, Red, Green, Blue (RGB) values, or other attributes can all be studied on a cell-by-cell basis, as the cell is the fundamental element of a DGGS. Changes in point density were studied in this thesis; the same procedure can be used to work with most other metrics. More information about the analysis and visualization aspect is provided in Section 3.4.

2.10 OTHER CONSIDERATIONS

[Sahr et al., 2003] provides a broad overview of the different parameters that go into constructing a DGGS (discussed in Section 2.5) and compares several existing DGGS's with each other to decide on an good general-purpose one, namely an icosahedral hexagon-based DGGS with a refinement ratio of

3 and symmetry about the Equator, which also minimizes the number of icosahedron vertices that fall on land. Although this is a valid design choice, for the purposes of this thesis an efficient method to index a global point cloud is needed, and polyhedron orientation is not a mitigating factor (see Section 2.5.2). However, [Sahr et al., 2003] also concludes that the rhombus DGGS may prove popular due to its direct relationship with the square quadtree. [Gregory et al., 1999] provide a thorough comparison of various global grids with respect to metrics that define an 'ideal' global grid system. Their conclusion is that the ISEA method is the best current choice for a global grid system, more efficient to compute and easier to implement than other methods [Gregory et al., 1999]. Hexagons are currently the most popular choice of shapes for a DGGS, but as stated earlier they also have several disadvantages. Therefore, an icosahedral rhombus DGGS was investigated in this thesis.

3 METHODOLOGY

This chapter provides a *conceptual* overview of the methodology utilized for this research. Section 3.1 describes the methodology used to assign a point to a DGGS cell. Section 3.2 describes the procedure followed to generate a Morton code for a point in 2D using the ISEA method, and extends this approach into 3D and 4D for handling the distance of a point above or below the Earth's surface (as an additional dimension) and also the time at which the point was captured. Section 3.3 explains how the decoding of a 2D, 3D, or 4D Morton code works. Section 3.4 describes how a DGGS can be used to study changes between two point clouds taken at two different moments in time. Section 3.5 compares existing point cloud web visualization software frameworks, and also discusses 3D Tiles, an up-and-coming format for visualization of massive amounts of points in a web browser. Section 3.6 concludes by providing examples of spatial-analytical operations that can be performed using a DGGS.

3.1 QUANTIZATION

Quantization refers to the mechanism for assignment of data to cells and retrieval of data from cells [Purss et al., 2017]. There exist several different types of quantization, each well-suited for a specific purpose. For example, there are *data tile*, *data cell*, *tag*, and *graphic cell* quantization. Of particular relevance for this thesis is *coordinate quantization*. *Every* spatial dataset is ultimately composed of point features: a vector feature is a set of points topologically connected to one another in some way, a raster is a gridded data structure obtained from interpolation of a set of points, and a point cloud by definition is a collection of points. Each of these points has a coordinate value indicating its position on or relative to the surface of the Earth model and each coordinate has a certain degree of precision. Within a DGGS, a spatial observation is assigned to a cell whose area is approximately commensurate with that observation's precision or area of uncertainty. This is because DGGS's are error-minimizing spatial data structures, whereby the error in the data, once mapped onto a DGGS, is contained. The size of a DGGS cell should ideally be as close as possible to the spatial uncertainty of an observation in order for that observation to be assigned to that DGGS cell. This obviates the need to store the closest (discrete) precision separately as additional metadata. When heterogeneous data from multiple sources are to be combined in a manner that minimizes the accumulated error in the combined dataset, the precision of the measurements is important to consider, and the metadata for these data need to be looked at for acquiring such information. This could be a time-consuming process. Within a DGGS, however, discrete precision is explicit in the size of the cells.

It is important to mention here that, innately, points are infinitesimally small 0-dimensional features. Projecting or transforming the coordinates of these points into another map projection, CRS, or datum will *not* result in any loss in precision (the transformations are completely *lossless*) if the points are viewed as 0-dimensional features. However, every point has a certain spatial uncertainty, and this can be represented as an area feature centered around that point. Transforming this areal feature into another system *will* result in a 'warping' in the area and a change in the spatial uncertainty of the observation. Consequently, point observations in a point cloud need to be viewed as area features (and not points) when it comes to mapping them to a DGGS reference frame. This is because the cells of a (2D) DGGS are precision-encoding *areas*, and a feature can only be incorporated into such a system if it itself is also an area. Therefore, for the purposes of this research, points have been viewed not as 0-dimensional features but as areas (or volumes, in 3D) whose extent is an indicator of their spatial uncertainty.

A laser beam signal scatters more the further away from the source of emission (i.e. scanner exit point). This is depicted in Figure 3.1. It follows a cone-shaped volumetric region in between the scanner exit point and the target. The amount of dispersion is indicated by the size of the beam divergence angle, usually expressed in milliradians. A larger angle yields a lower Signal-Noise Ratio (SNR) as the beam disperses across a larger volume, thereby inflicting more noise on the measurement, whereas a smaller angle yields a higher SNR [Rohrbach, 2015a]. The SNR affects the precision of the final measurement. The beam divergence is directly related to the size of the laser beam footprint: a larger beam divergence yields a larger beam footprint, and a smaller beam divergence yields a smaller footprint. It is also worth noting that the distance from scanner to target also influences the size of the resulting beam footprint: a larger distance (i.e. higher altitude) yields a larger footprint [Rohrbach, 2015a]. In reality, however, for laser scanned point clouds there are several other factors that ultimately contribute to the precision of point measurements; these include the quality and operation of the GPS sensor, Inertial Measurement Unit (IMU), mirror, and aircraft, and also how the point cloud has been processed and how well the entire system has been calibrated [Tully, 2012]. For the purposes of this research, however, the simplistic assumption of point precision as mainly dependent on the ranging from the laser was made. It is also important to note that other means of generating point clouds, such as dense image matching from photos, photogrammetry, multi-beam echo sounding, and others [van Oosterom et al., 2014], each have their own unique sources of error that affects point precision differently. This needs to be taken into account when utilizing point clouds acquired from such sources. This thesis has focused on point clouds acquired only from LIDAR, and this could be terrestrial or airborne.

With any DGGS, one should store observations at a level in the DGGS hierarchy that best represents the spatial resolution/uncertainty of the observation (i.e. point in point cloud). The spatial resolution of each point is indicated by the size of the footprint of the laser beam that generated the point. This size can be inferred by examining the distance from the laser scanner to the point; the larger this distance, the larger the footprint due to beam divergence, and the lower the spatial resolution or precision of the point.

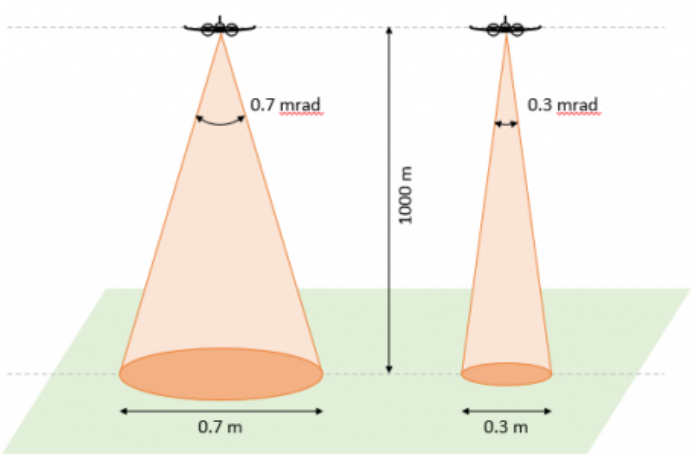

Figure 3.1: The precision of a LIDAR point observation is mainly determined by the size of the laser beam footprint at the location where it hit the point. Figure from Rohrbach, 2015a.

By calculating the sizes of the laser beam footprints at each point in a point cloud, point precision can be determined. Then, using the principles of coordinate quantization, the point can be assigned to a DGGS cell covering the same location as the point and which has approximately the same area as the point's precision. The precision needs to be computed in projection space in units such as meters, and brought over (as an attribute) before quantization onto a DGGS ellipsoid. Ultimately, this means that the various points constituting a real-world object such as a church could be assigned to different DGGS cells depending on their ranging (distance) from the scanner. A higher precision point is contained within a lower precision point, and the containment is a property of their cell ID's.

It should be noted that this approach only works if all the points in the respective point cloud have different point precisions, thereby implying (under this simplified assumption) that they are not at the same range (distance) from the scanner.

For overview analysis or visualization, high precision points are stored in the top levels (with important points, and not a lot of data) of the structure as these should be shown at both smaller and larger scales. Points with a lower precision / higher spatial uncertainty are stored in the bottom levels of the structure as these are only shown at larger scales (i.e. only when zoomed in sufficiently close). High-precision points are also highly important, and this additional continuous dimension can be used to visualize the points in an appealing manner using a perspective view query (see Section 2.9.2).

3.2 MORTON INDEXING ON A CURVED SURFACE

One of the most beneficial properties of a rhombus tessellation in a DGGS is that quadrant-recursive orderings, such as Morton or Hilbert curves, can be used to index the cells in the tessellation. This is because a rhombus

tessellation can be viewed as a skewed transformation of a square tessellation [White, 2000]. The geometry of the cells and the topological relationships between neighboring cells can be used to assign globally unique identifier (GUID)'s to the cells at any resolution [Purss et al., 2017]. Addressing the cells of a DGGS using such an ordering allows the preservation of spatial locality that can then be utilized for efficient spatial database and geometric operations such as parent-child hierarchical operations, spatial clustering, neighbor-finding and adjacency testing. Furthermore, addressing the entire globe using an array of cells in a multi-resolution structure provides a common reference frame for the entire world that obviates the need to use separate map projections for every region. As indicated in Table 3.1, different countries use heteregenous map projections, CRS's and datums for their point clouds, and a common map projection, datum, and/or coordinate system is needed to work with data spanning international borders. A DGGS provides a common underlying reference frame for such data, and the cell indices provide a way to address/index the individual elements (i.e cells) within such a structure. Each cell is defined on the curved surface of the Earth ellipsoid, thereby facilitating improved spatial analysis at scales where the curvature of the Earth becomes a significant factor, and lies in a fixed position that, once defined, never changes.

As a DGGS is an ECEF system, it is tied to a particular model of the Earth, such as the GRS80 or WGS84 ellipsoids. It can be tied to any model. Furthermore, even though currently they are Earth-based reference frames, they could theoretically be used for other planetary or celestial bodies, such as the moon or Mars. Therefore, the cells of a DGGS hierarchically cover the surface of the Earth model without any overlaps or underlaps (similar to a *planar partition* on a plane). Therefore, to assign a unique identifier or index to each cell at every resolution of a DGGS, a *spherical* or *ellipsoidal* SFC is needed, and not a planar SFC. The ISEA projection, on which the icosahedral rhombus DGGS used in this thesis is based, can be used to generate unique Morton codes for each cell. More information about this projection is provided in Section 2.8 . The ISEA projection projects the faces of an icosahedron onto a flat 2D plane. An icosahedron, by definition, consists of 20 triangles, and these can be numbered as shown in Figure 3.2. Although its faces are initially triangles, pairs of adjacent triangles can be connected to form rhombuses that form the *initial equal-area tessellation* to be used for a rhombus DGGS. As the icosahedron has 20 triangles, there are 10 total rhombuses. The numbering of these rhombuses to be used in this thesis is shown in Figure 3.3. Such a numbering yields a continuous SFC ordering.

In this way, the entire surface of the Earth can be covered with rhombus cells. Each parent rhombus has four smaller children rhombuses (see Figure 3.12 for an illustration), thereby discretizing the surface of the Earth into a linear quadtree on top of which Morton indexing (or any other kind of SFC ordering, for that matter) can be applied. This indexing can be extended into n dimensions, as explained in section 3.2.1, so that not only the spatial but also the temporal dimension is integrated into one code. Forward and inverse formulas for the ISEA projection are well-documented in [Snyder, 1992]. These can be used to convert geographic coordinates (a latitude, longitude location) into rectangular (x,y) coordinates on the ISEA projection plane (see Figure 3.4). Then, pairs of adjacent triangles can be combined, and recursively subdivided to yield the hierarchy of cells that constitute the chosen DGGS.

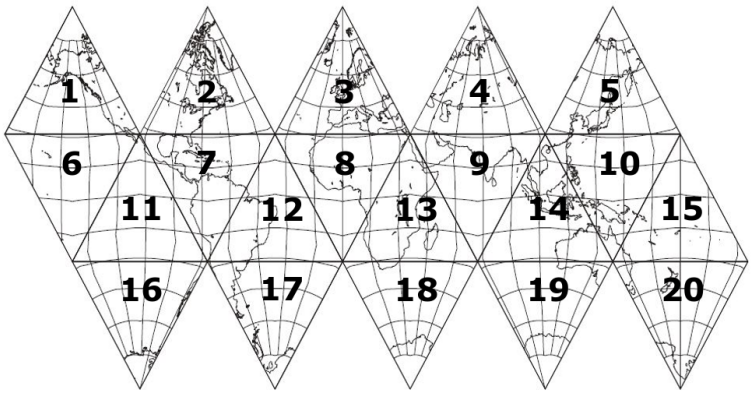

Figure 3.2: The initial numbering of triangles on a flattened icosahedron using the ISEA projection.

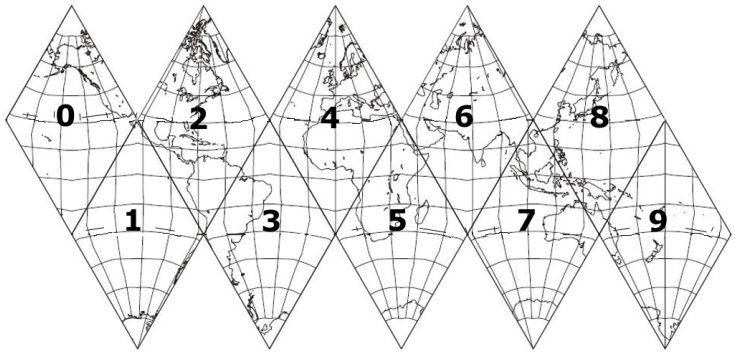

Figure 3.3: Pairs of adjacent triangles can be combined to yield rhombuses on a flattened icosahedron.

As there are 10 total rhombuses, each can be assigned a unique label number from 0-9 that indicates its location on the Earth. To preserve a degree of spatial locality and maintain a continuous ordering (i.e with no sharp 'breaks' or 'jumps' in the Morton codes), these numbers have been assigned as follows: triangles 1 and 6 are connected to form rhombus 0; 11 and 16 to form rhombus 1; 2 and 7 to form rhombus 2; 12 and 17 to form rhombus 3; 3 and 8 to form rhombus 4; 13 and 18 to form rhombus 5; 4 and 9 to form rhombus 6; 14 and 19 to form rhombus 7; 5 and 10 to form rhombus 8; and 15 and 20 to form rhombus 9. This is illustrated in Figure 3.3. It can be readily seen that nearby rhombuses also have closer label numbers. This is similar to the approach that was utilized by [Bai et al., 2005], although for an octahedral rhombus DGGS; this thesis has utilized it for an icosahedron.

Then, to generate each cell at every resolution within the rhombus, each rhombus face is subdivided into finer and finer cells. For example, to generate the tessellation at the second level, the midpoints of opposite-facing sides of an initial rhombus are connected to yield four smaller, nested children rhombus cells (2 along every dimension). There will therefore be 4^n

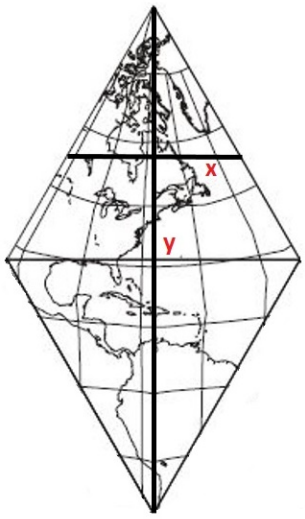

Figure 3.4: The ISEA projection can be used to convert geographic coordinates into grid (x,y) coordinates on the projection.

cells at resolution n (the initial equal-area tessellation can be considered as 'resolution zero'). It is important to mention here some of the favorable properties of a rhombus as opposed to other shapes. By definition, a rhombus is a quadrilateral whose four sides all have the same length. Connecting the midpoints of opposite sides will therefore yield 4 smaller *equal-area* rhombuses. Every rhombus has two diagonals that connect pairs of opposite vertices, and these diagonals bisect (divide into two equal parts) their opposite angles. The diagonals also intersect one another at right angles (see Figure 3.5). On an icosahedral rhombus face, there are two interior angles of 120°(angle b in Figure 3.5) and two of 60°(angle a in Figure 3.5) located opposite of one another [White, 2000].

It is important to state that all SFC's are based on hypercubes (i.e. n-dimensional analogues of squares (n=2) and cubes(n=3)), and there is a separate hypercube on every rhombus face of the icosahedron. However, the numbering of the faces shown in Figure 3.3 yields a continuous ordering of n-dimensional space. As each cell has a unique Morton code based on its location on the SFC, all of the points that are physically contained in that cell automatically inherit the same Morton code as the cell. A point is assigned to a cell in a DGGS based upon its precision (explained in Section 2.9.2). With the above information, an elaboration of the entire procedure to generate an n-dimensional Morton code for a point in a point cloud is readily facilitated.

1. First, the (continuous) precision of the point in a standard map projection is calculated. Usually, point clouds of different countries will use their country's national map projection and coordinate system, and not geographic coordinates. For example, the CRS's used by the point cloud datasets of the Netherlands, Belgium, and Germany are presented in Table 3.1. All of these CRS's use units of meters. The precision of a point in a point cloud can be calculated according to the procedure described in Section 3.1. This precision needs to be stored as an attribute of the point.

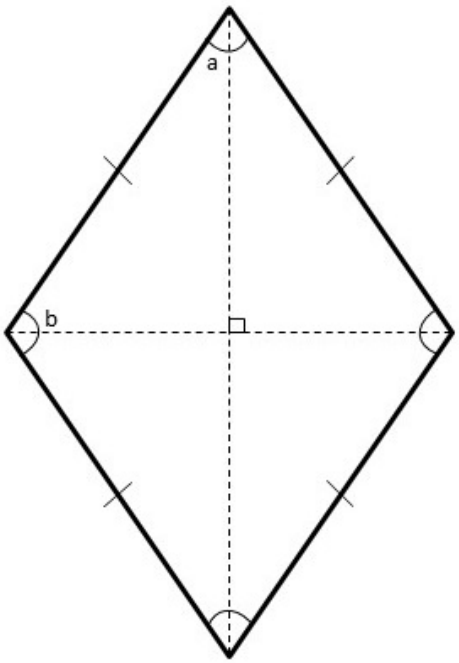

Figure 3.5: A rhombus, along with its diagonals. The diagonals are perpendicular, and the lengths of the sides of the rhombus are equal.

Country	Horizontal CRS	Vertical CRS
Netherlands	Amersfoort / RD New	Normaal Amsterdams Peil (NAP)
Belgium	Belgian Lambert 72	Ostend
Germany	ETRS89 / UTM Zone 32N	DHHN92

Table 3.1: The CRS's of national point clouds of three countries.

2. Then, the point is converted from its initial CRS/datum and/or ellipsoid into geographic coordinates on the WGS84 ellipsoid (used for this thesis), with the precision in its original CRS brought over as an attribute. This conversion will involve a loss in point positional accuracy if the two ellipsoids are different (i.e it is not completely lossless and reversible). For example, the Dutch RD system uses the Bessel ellipsoid of 1841 [van der Marel, 2016], therefore there will be a minor loss in point accuracy after conversion of points on this ellipsoid onto the WGS84 ellipsoid.

3. The precision associated with the point is linked to the closest discrete resolution of a DGGS. This is the point's discrete precision. For example, if a point has a spatial uncertainty of (+ -) 3 meters (i.e. an area of approximately 28 square meters), then this point should be associated with a DGGS cell that has an area close to 28 square meters. The cell areas at all resolutions of an icosahedral rhombus aperture 4 DGGS are provided in Table A.1. In theory, cells can be defined for infinite possible resolutions. At this stage, both discrete and continuous point precisions have been computed.

4. Then, to compute the Morton code for use in a global indexing framework, the point is projected onto the ISEA projection. This involves a number of steps. First, the icosahedral triangle face on which the point is located needs to be found. The numbering of these faces are shown in Figure 3.2. This is a recursive procedure that tests each of the 20 faces one by one until it finds the proper face. Internally, this uses Vincenty's Inverse Formula for quickly determining on which face a point is located. Then, grid coordinates originating from the geographic center of that face need to be computed, formulas for which are provided in [Snyder, 1992]. The geographic coordinates for all triangle centers are also provided in [Snyder, 1992]. The coordinate axes on which these coordinates are based have different orientations due to the dissimilar orientation of triangles on the icosahedron. For example, for triangles 1-5 and 11-15, the x-axis will be positive to the right of the center and negative to the left of the center, whereas the y-axis will be positive up and negative down from the center; for triangles 6-10 and 16-20, however, due to their opposite orientation, this will be the reverse: the x-axis will be positive to the left and negative to the right, and the y-axis will be positive 'down' and negative 'up'. This is illustrated in Figure 3.6.

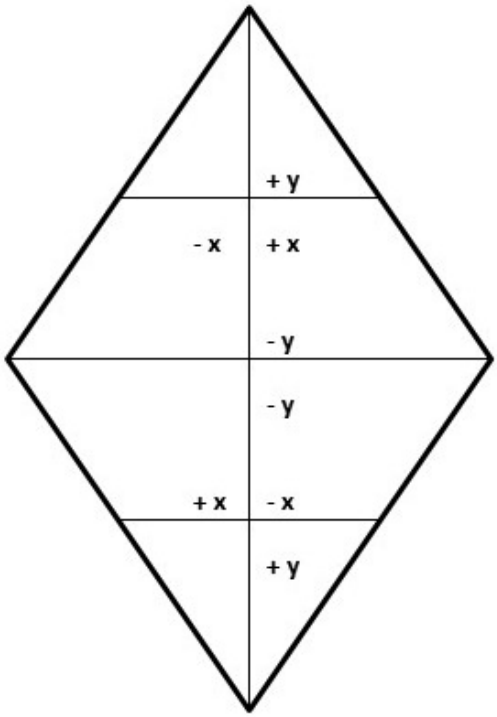

Figure 3.6: The opposite orientations of the triangles need to be taken into account, as the orientation of the axes differs depending on the triangle orientation.

5. The point's rhombus face number is then determined based upon its triangle face number. This is simple as the rhombus-triangle containment relationships are known. The rhombus face number uses the numbering defined in Figure 3.3.

6. Taking into account the orientation of the axes mentioned earlier, the point's coordinates are then translated to the left vertex of their con-

taining rhombus, because this is the origin of the Morton SFC for every face. This results in only positive coordinates for x and positive or negative coordinates for y in a right-handed Cartesian system.

7. Since the interior angle of an icosahedral rhombus at the left vertex is 120°(angle b in Figure 3.5), its x and y axes are also skewed with an interior angle of 120°. Therefore, the coordinates obtained in step 6 need to be rotated into this skewed system. The equations to convert an (x,y) coordinate from the Cartesian system of step 6 into those of a skewed system on an icosahedral rhombus are:

$$\mathcal{X}_{skewed} = x - \frac{y}{\sqrt{3}} \tag{3.1}$$

$$\mathcal{Y}_{skewed} = x + \frac{y}{\sqrt{3}} \tag{3.2}$$

8. The rotated x and y coordinates from step 7 are converted into binary form with the same number of bits as the DGGS resolution of the point. For example, if a point has been assigned to a cell at resolution 3, and its rotated (x,y) coordinates from step 7 are (5,4), then these coordinates should be converted into 3-bit binary form as (101,100). The binary coordinates are then converted into integers. Then, the below formula can be used to generate the Morton code $\mathbb{M}_C$ for the point in 2 dimensions [Zhao et al., 2006]:

$$\mathcal{M}_C = 2 * (Y_{bin}) + (X_{bin}) \tag{3.3}$$

This formula can easily be extended into n dimensions:

$$\mathcal{M}_C = 8 * (T_{bin}) + 4 * (Z_{bin}) + 2 * (Y_{bin}) + (X_{bin}) \tag{3.4}$$

9. The complete Morton code for the point is calculated by concatenating the rhombus face number found in step 5 with the Morton code computed in step 8. Therefore, the first digit in the index is the face number, and this is then followed by the actual Morton code.

10. This Morton code is then brought back to the original point on the WGS84 ellipsoid defined in step 2 as an attribute. As in this thesis, a DGGS is being constructed on top of the WGS84 ellipsoid, this step also constitutes a 'mapping'/import of point clouds to a DGGS reference frame.

Figure 3.7 provides an overview of the above process that is being used to map/import a point cloud into a DGGS.

After step 10, there is no need to re-project the point positioned on the WGS84 ellipsoid in latitude, longitude, and (optionally) height and time dimensions. *Every* map projection involves a certain degree of error. This error accumulates over the course of time as projections are applied to the data and coordinate conversions between different datums and ellipsoids are performed. For example, even for a projection such as ISEA, there is a maximum angular deformation of 17.27% and maximum and minimum areal

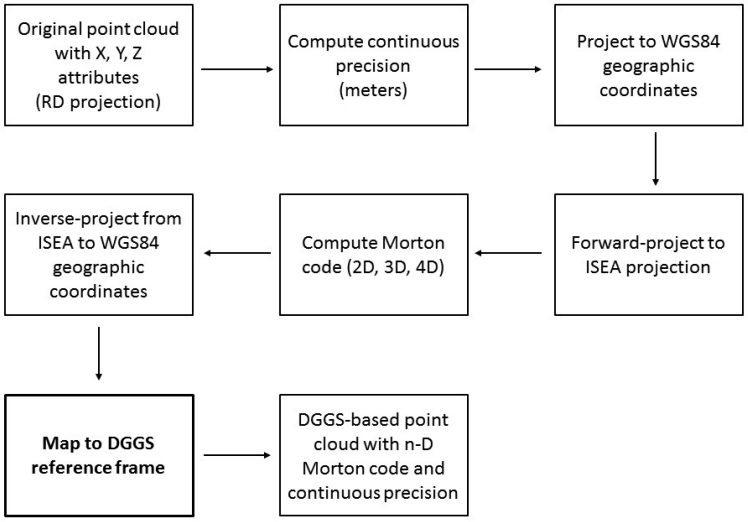

Figure 3.7: Diagram of the process used to import data into a DGGS reference frame for the Dutch point cloud datasets, which use grid coordinates in the RD CRS.

deformations of 16.3% and 14%, respectively [Snyder, 1992]. Re-projecting or inverse-projecting a point will 'warp' its fuzzy region of uncertainty in unpredictable, erroneous, and *irreversible* ways. Although if one is working at significantly large scales (i.e. continent or whole-Earth) these warps become insignificant, if working at local scales this is an important consideration. This is precisely what a DGGS is designed to avoid. Once spatial observations are imported into a DGGS, there is no need to apply any projection or datum transformation on the data. The error in the original observations is contained and not exacerbated due to the repeated projections and datum conversions that could have been applied to it had a DGGS approach not been used.

In summary, repeated map projections and datum transformations (i.e. between different ellipsoids) introduce compounding errors on spatial data (see Figure 3.8). *Usually*, this error increases further from the center of the projection, on a line referred to as the *central meridian*, and can be variable. The amount of spatial uncertainty is indeterminate. A DGGS minimizes the amount of error accumulation on spatial data because it requires the data to be mapped into it only once at a particular level in its hierarchy. Every spatial analysis after this point is an array process using set theory and/or the DE-9IM (explained in Section 2.7).

3.2.1 Extensions to 3D and 4D DGGS

At present, the OGC DGGS Abstract Specification is 2D on the surface of the globe. That is, the cells of a DGGS cover the curved surface of the Earth in a non-overlapping manner with their normal vectors pointing in different directions. They are not volumetric; they do not inherently allow the storage and representation of 3 dimensional values, such as those that include

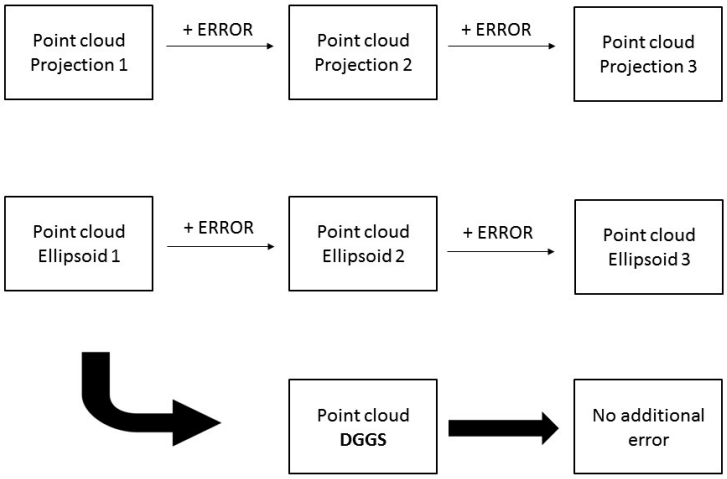

Figure 3.8: Repeated map projections and datum transformations introduce compounding errors on spatial data. A DGGS approach makes it unnecessary to apply repeated map projections and datum transformations.

elevations, and much less 4 dimensional ones, such as those that include time. However, a point in a point cloud contains such attributes and it is important to preserve this information in the spatial location code for that point. This speeds up the processing of spatio-temporal queries on the data as the query does not have to perform yet another sequential search on the 3rd or 4th dimension attributes after narrowing the database records based on the 2 dimensional attributes.

This section presents an approach to extend DGGS into 3D or 4D, assuming the 3rd dimension to be the distance of the point above or below the Earth's surface and the fourth to be the time that the point was captured. It also assumes the Earth to be a sphere, and *not* an ellipsoid; however, in general the same procedure can be applied on an ellipsoid, but some corrections are needed, and this is noted as appropriate.

3D DGGS

In 2D DGGS the cells of the tessellation at any one level are all of equal-area, but are distorted in shape (as no projection can preserve both area and shape). They tessellate the *surface* of the Earth, and are therefore 'flat' on the curved surface of the Earth (although on a spherical/ellipsoidal Earth volume, the vertices that form the cells do not all lie on the same plane). It would be invaluable if DGGS could be extended into higher dimensions to take advantage of the multidimensional nature of geographic data, and especially point clouds. Every point can have an n number of dimensions, including its 3D position (latitude, longitude, height), time, and other attributes such as intensity or color. Therefore, in 3D DGGS it makes sense to make them all of *equal volume* at any one resolution in the hierarchy, above or below the Earth's surface.

The 2D cells can be extended into 3D either on the base polyhedron or directly on the sphere/ellipsoid. This is ideal because DGGS's, although usually constructed using a polyhedron, can also be built on the sphere or ellipsoid directly (using a method known as *direct spherical subdivision* [Sahr et al., 2003]). If a polyhedron is used as a base to construct the 3D cells, then a 3D hypercube is constructed on each face of the polyhedron and a SFC is created that traverses through all of the cells of the hypercube. This provides a unique ID for each cell. Then, the cell vertices are inverse-projected onto the Earth model (sphere or ellipsoid) using an equal-area projection, and the cells appear on a spherical Earth as shown in Figure 3.9. The 'height' above the face of the polyhedron (i.e. above the plane of the DGGS projection) is a scaled version of the height above the surface of the Earth model. If a DGGS is built on a sphere directly, the 2D cells can be extended into 3D by computing the normal to the surface of the Earth at each of their vertices (see Figure 3.9). The normal to the surface also automatically passes through the center of a spherical Earth. For an ellipsoid, however, this is not generally true (see Figure 3.10). The Earth is slightly bulged at the Equator and flattened at the poles (i.e. is an ellipsoid), therefore the normal to the surface does not always pass through its center. Ellipsoidal DGGS's are more accurate than spherical DGGS's, as the Earth itself is most closely resembled as an oblate ellipsoid. The general idea, however, is to connect each of the vertices of all of the cells of a DGGS to the center of the Earth model (be it a sphere or an ellipsoid) in an earth-centered system such as ECEF.

Unfortunately, 3D cells at any one resolution in the hierarchy will not be of the *exact* same volume, as the amount of space expands outwards from the Earth's center. However, the cells in a concentric ring at any one radius (distance) outwards from the center of the Earth within the same *resolution will* be of the same volume, if the Earth is assumed to be a sphere. For other Earth models, such as ellipsoids, some corrections are needed, and the cells above or below the surface in equatorial regions will be of a larger volume than those closer to the poles because the Earth, being an oblate spheroid, bulges out at the Equator and is more flattened at the poles. However, for most purposes these differences in volume are negligible. For point clouds, an elevation range of +- 5000 meters, or 5 kilometers, is sufficient, as even high-altitude airborne LIDAR acquisition surveys are usually performed at around 3000 meters [Hopkinson, 2007]. In this elevation range, the volume of the cells is roughly identical and the differences in volume can be ignored. Therefore, for all practical purposes, 3D cells of the same volume are being used when it comes to handling point clouds or even other near-surface features. This elevation range can be user-defined and customized to suit application-specific needs.

Although the faces of the 3D cells are not planar, for all practical purposes one can think of them in terms of simple primitives such as 3D planes. The 3D cells extend both above and below the Earth's surface, and as stated earlier, an elevation range of +-5000 meters from the surface is sufficient to incorporate both terrestrial and bathymetric point clouds. Just as a parent cell is subdivided into 4 smaller children on the flat 2D icosahedral plane, a volumetric 3D cell can be subdivided into 8 smaller children by splitting the 3D hypercube on the polyhedron by 3 splitting planes, when an aperture 4 2D DGGS is extended into 3D. A 3D SFC can be defined that traverses through all of the cells at each resolution in a hierarchical manner, thereby allowing the storage and representation of the 3D location in space of the point that

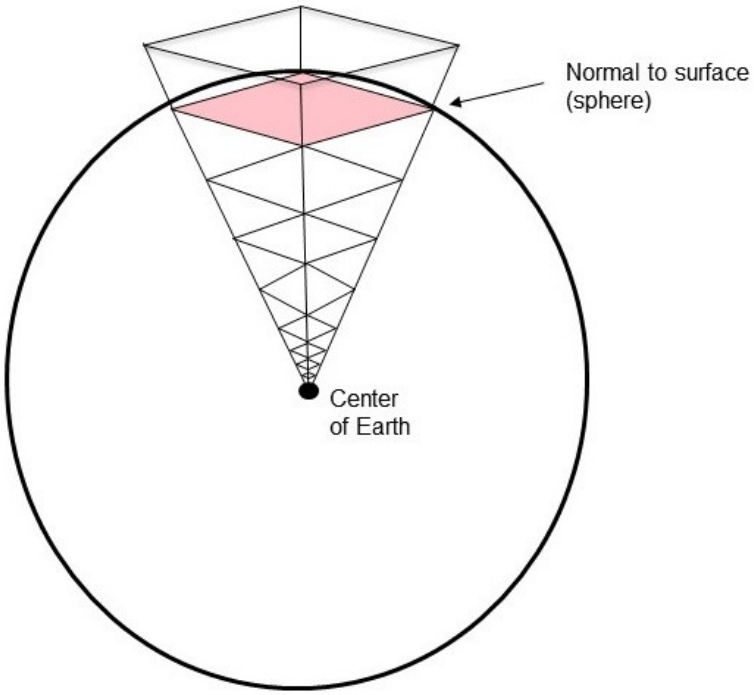

Figure 3.9: A DGGS extended into 3D on a spherical Earth. Only a single resolution is shown. The cell in red is a 2D cell on the curved surface of the Earth, and shown for reference.

was generated by the laser pulse, in an earth centered system such as ECEF or latitude-longitude.

As the 3D cells extend below the surface of the Earth model, they can be used to work with bathymetric or other point clouds acquired below the surface. If the elevation range extends below the Earth's surface, negative elevation values are also automatically encoded: the range of elevation values (maximum – minimum), always a positive number, is computed, and hierarchically split into 4 smaller elevation ranges at every resolution of an aperture-4 DGGS. With other apertures, the amount of refinement would be different.

It is important to note that cells of *any* shape- such as triangles, squares, and hexagons- can be extended into 3D in the same manner; this procedure is not limited to rhombuses. Moreover, *any* polyhedron can be used, and not only the icosahedron. Once the orientation of the polyhedron is set, the 3D cells stay in the same position with respect to the surface of the Earth, similar to in 2D. Changing the polyhedron orientation allows for adjusting the positions of 3D cells for application-specific purposes.

4D DGGS

All points are captured at a specific moment in time, and this additional dimension can also be encoded into a DGGS cell, thereby turning it into a 4D cell that is fine enough in temporal resolution to encode not just the date but the exact observation time of each point. The fineness in time (i.e. hours,

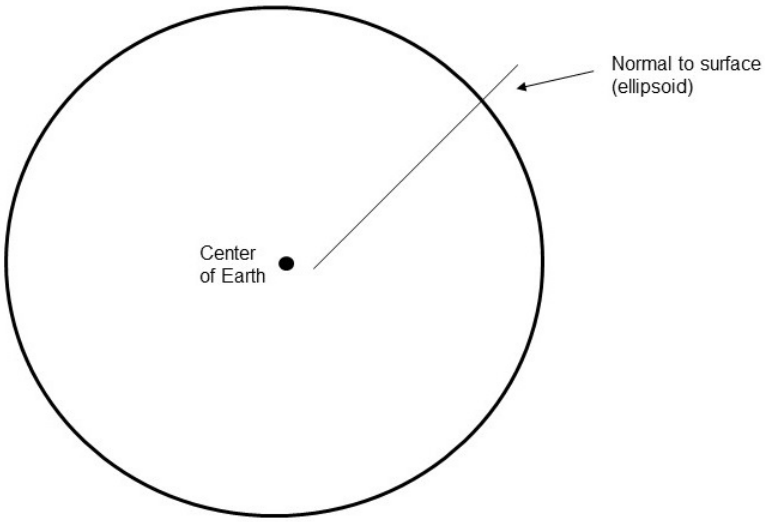

Figure 3.10: The normal to the surface of an ellipsoidal Earth does not always pass through its center. Therefore, it cannot be directly used on an ellipsoid to create 3D DGGS cells.

minutes, seconds) can be set as appropriate. In 4D DGGS, the behavior of the time dimension is much similar to the behavior of the spatial dimensions, in which at each resolution the range of a dimension is hierarchically split into a certain number of finer parts. Just as how space is subdivided into a hierarchy of nested spatial areas or volumes in 2D and 3D DGGS respectively, in 4D DGGS time is discretized as a hierarchy of nested temporal ranges. For example, a large temporal range is subdivided into exactly four smaller temporal ranges at every resolution in any aperture-4 DGGS (see Figure 3.11 for an illustration). With other apertures, the refinement ratio would be different. A 4D SFC will traverse through all four dimensions and allow for efficient indexing, clustering, and querying of 4D point cloud data.

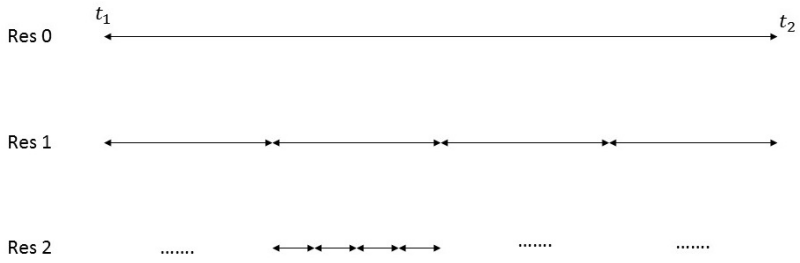

Figure 3.11: Time, a continuous dimension, can be discretized into a hierarchy of nested temporal ranges for encoding the temporal dimension of point clouds, or any other observations for that matter.

Before encoding all dimensions into a proper SFC code, their range of values should be scaled to a common range. This range is dependent on the resolution of the DGGS. For example, at resolution 3 in 2D DGGS there are 8 possible X values and 8 possible Y values, because space is subdivided three times hierarchically. This can be seen in Figure 3.12. Similarly, in 3D DGGS, Z should be scaled to a range of 0-8 at this resolution. In 4D DGGS, time should also be scaled to the same range. Time uses different units from

space, as it is measured using years, months, weeks, days, hours, minutes, and seconds. The amount of change applied to the time to get it to be the same range as the X, Y, and Z dimensions is known as the *scaling* and is the factor of how much time is integrated with how much space [Psomadaki et al., 2016]. In the context of a DGGS, the scaling differs depending on the resolution, with a larger scaling applied applied to smaller resolutions (i.e. those containing larger cells), and a smaller scaling applied to larger resolutions (i.e. those containing smaller cells). As time can be measured using a variety of different units (i.e. year, month, day, etc.), it should first be converted into a single unit that is fine enough so as to be able to distinguish each point separately in a point cloud, but coarse enough so as to not yield unnecessarily large numbers that might pose an overhead for storage and retrieval.

In 4D DGGS, time is represented on a continuous scale since a certain moment in history. For the dataset used in this project, it is in the form of GPS Time, a scale beginning at 12 AM on January 6, 1980 that is currently 18 seconds ahead of the more popularly used UTC time, because it does not take into account leap seconds [QPS, 2018]. The time is given in seconds. By setting an appropriate end value for the time range, the entire range of values can be discretized into a hierarchy of nested temporal ranges. The end value chosen here is 12 AM on January 6, 2018, exactly 38 years after the beginning of GPS Time. This range is sufficient for studies of most recent point clouds. The end value for the time range could also be set to update dynamically, as time itself is dynamic. This would mean that, the 4D DGGS SFC codes of a point observation would also change dynamically as time passes by. For this thesis, however, a static time range was used.

The user-friendly hexadecimal base-16 system lends itself exceptionally well to its use in encoding 4D DGGS observations. The system uses the digits 0 - 9 to represent values from zero to nine, and the letters A - F to represent values from ten to fifteen, respectively. This means that there are a total of 16 possible values that can be represented using hexadecimal. This is also the case with a 4D DGGS at every resolution, whereby one out of sixteen values is used. Due to its user-friendly nature and analogy to a 4D DGGS, it was used in this research to encode 4D point clouds.

When time is encoded into the DGGS cell and its ID, spatio-temporal queries on the data can run a lot faster as the database engine does not have to perform a complete sequential scan of the data as when time is stored only as an additional attribute. 3D space and time are incorporated into one SFC key and this key can be decoded into its original space and time values (i.e. the point's latitude, longitude, distance from the surface of the Earth, and time of acquisition).

Algorithm 3.1 provides a high-level overview of the Morton encoding procedure in 4-dimensions.

Algorithm 3.1: 4D Morton Encode

Input: Point X, Y, Z, and time values, A maximum DGGS resolution number *maxRes*, an elevation range above and/or below surface of Earth *hRange*, a time range *tRange* in an established system such as GPS Time

Output: 4D Morton code *MC* for point

```
Point (Long,Lat,Height in WGS84) ← Project Point(X, Y, Z) in projection
while triangle face not found do
    for i ← 1 to 21 do
        Find triangle face
        Find x,y coordinates from center of face
        XTrans, YTrans ← Translate coordinates to lower left/upper left origin
        XSkewed, YSkewed ← Rotate axes so that interior angle is 120° using formulas
        —- XSkewed = XTrans - YTrans / sqrt(3)
        —- YSkewed = XTrans + YTrans / sqrt(3)
        Round down to nearest integer
        Tbin, Zbin, Ybin, Xbin ← Convert to binary
        Convert binary to integer
        Find rhombus face using triangle face
        MC ← Compute nD Morton code using formula 8 * (Tbin) + 4 * (Zbin) + 2 * (Ybin) + (Xbin)
return MC
```

The resulting Morton code has some unique properties. The first digit of the code indicates the icosahedron rhombus face on which the point is located. For a spatial query, this allows a rapid determination of where on the Earth the point could be located. Points on all other faces can be immediately discarded from further consideration. Depending on the use case, n dimensions can be integrated into one code. For this thesis, the three spatial dimensions (X, Y, Z) have been integrated into a 3D DGGS code, an attempt to draw a 3D DGGS tessellation has been made (see Figure 3.9), and it also has been demonstrated how in addition time (T) can be incorporated into a single 4D DGGS code. The length of the code indicates the resolution of a DGGS in which a point is contained, and also that point's discrete precision. As there are a fixed number of cells at every resolution and a fixed minimum spacing between the cells, a Morton code also automatically encodes the number of cells and the DGGS inter-cell spacing at that resolution.

3.2.2 Order of dimensions

The order of the space/time dimensions (latitude, longitude, distance from the surface of the ellipsoid, and time) used in the encoding of an observation's Morton code influences the resulting Morton code (or any other SFC code, for that matter). For example, using the order of *latitude, longitude, altitude, time* will yield a different Morton code from the order of *time, altitude, longitude, latitude*. However, as long as all points in all datasets have been *consistently* encoded in the same manner, the order of the dimensions will not result in any problem for querying and retrieval.

3.2.3 Storage of Morton codes

For storage purposes, there are two options:

1. The Morton code only up to the resolution to which the point has been assigned can be stored, thereby obviating the need to store the (discrete) precision separately as additional metadata as the length of the entire sequence of numbers in the code indicates the *discrete* precision/resolution level in a DGGS hierarchy to which the point is assigned [Bai et al., 2005]. However, in order to uniquely identify and get access to the spatial and temporal properties of each and every point, additional information would have to be stored. Latitude, longitude, distance from the surface of the Earth, and time are all continuous dimensions. DGGS's, on the other hand, are by definition discrete. Unless a SFC with dimensions represented by only integer values is used, converting to a DGGS will result in a loss of information as continuous values are rounded down to their closest integers. To preserve the original information, the differences in the continuous and discrete values, for each dimension, will need to be stored separately in addition to the Morton code. This will result in 4 additional fields as there are 4 total dimensions. This approach provides a good way to encode a point's discrete precision without the need to store it as an additional attribute. However, it also requires storing a lot of additional data.

2. The full-resolution Morton code can be stored for each and every point, thereby eliminating the need to store the additional fields for each dimension. In case the discrete precision/resolution level in a DGGS has to be stored, this approach would necessitate its storage as a separate attribute, because the length of the code itself cannot be used to recover the discrete DGGS resolution to which the point has been assigned. However, this approach requires considerably less storage.

The second approach to DBMS storage for points was used as overall it requires the lowest storage. If the cells of this DGGS are to be stored, only the leaf cells (i.e. cells at the highest resolution) need to be stored, as the location of all the other cells can be obtained through simple spatial aggregation using their Morton codes, benefiting from the congruent nature of the rhombus tessellation. However, for query purposes, storing the cells is unnecessary as the Morton codes of the cells are also the Morton codes of their assigned points.

3.2.4 SFC code convergence

As resolution increases, a DGGS SFC code slowly converges to the true values of the its encoded dimensions. For this thesis, the Morton code gradually converges to the true latitude, longitude, elevation, and time values of a point as full-resolution is approached. This is illustrated in Table 3.2 for an example location. The table shows the values for the four dimensions obtained from decoding a 4D Morton code for a hypothetical point observation at various resolution levels. The decoded values are slowly converging to the actual values. Therefore, although latitude, longitude, elevation, and time are all continuous variables, storing a single full-resolution code allows

for encoding the continuous nature of the data as the code encodes these dimensions up to such an arbitrarily high LOD so as to converge to the true values of these dimensions for that point.

Point Observation:

Latitude: 36 ° N
Longitude: 25 ° E
Elevation: 44.0 meter (WGS84)
Time: February 1, 2008 00:00:00 (12 AM) UTC / 885,859,218 seconds GPS Time (seconds since 12 AM January 6, 1980)

4D DGGS Morton code: 4F38AAA1D23699081A69D5B67D79A2928B (hexadecimal)

Resolution	Latitude	Longitude	Elevation (m)	GPS Time (s)
2	30.2809	18.7822	0.0	599,616,027
5	35.7917	23.7038	0.0	861,948,031
10	35.9975	24.9653	39.0625	885,370,531
15	36.0006	24.9989	43.9453	885,846,301
25	36.0000	24.9999	43.9999	885,859,203
32	36.0000	25.0000	44.0000	885,859,218

Table 3.2: With increasing resolution, the Morton code converges to the true latitude, longitude, elevation, and time values of every point.

Resolution	GPS Time (s)	Equivalent UTC
2	599,616,027	Jan 6, 1999 00:00:09 AM
5	861,948,031	Apr 30, 2007 06:00:13 AM
10	885,370,531	Jan 26, 2008 08:15:13 AM
15	885,846,301	Jan 31, 2008 20:24:43 PM
25	885,859,203	Jan 31, 2008 11:59:45 PM
32	885,859,218	Feb 1, 2008 00:00:00 AM

Table 3.3: The equivalent time in UTC for every GPS Time value provided in Table 3.2.

A 4D Morton code at resolution 10, when decoded, will not simply return a GPS Time that is double the time returned by decoding a Morton code at resolution 5. For example, as can be seen in Table 3.2, a decoded resolution 5 4D DGGS code returns a GPS Time of 861,948,031 seconds, whereas a code at resolution 10 returns a value of 885,370,531 seconds, which is not simply double the first amount. What is observed, however, is that the difference in decoded values at consecutive resolutions *decreases* with increasing resolution. For example, there is a difference of only about 15 seconds in time between resolutions 25 and 33, but a difference of more than 9 years between resolutions 2 and 10 (see Table 3.3). The greatest differences in successive resolutions lie in-between smaller resolutions, whereas the smallest differences lie in-between larger resolutions.

3.3 DECODING A MORTON CODE

A Morton code for a point can be decoded into its latitude, longitude, and elevation on the WGS84 ellipsoid, and time. However, an important point to

note is that converting between coordinates and spherical SFC positions is not completely reversible; the exact latitude, longitude coordinate may not be recovered, but it will be *sufficiently* close [Bartholdi and Goldsman, 2001]. Unfortunately, if a projection contains many formulas that are not lossless, the error accumulates.

3.3.1 2D DGGS

Since the Morton code is computed on a projection (ISEA), inverse-projection formulas, all of which are documented in [Snyder, 1992], are needed in order to transform an encoded point into geographic coordinates. First, however, the Morton code needs to be converted into the skewed x, y coordinates mentioned in Section 3.2. This requires the determination of two unknowns from one equation (see Equation 3.3). Although this is mathematically impossible, it can be done using a simple process of elimination in any quadtree-based DGGS by observing the digits comprising the code. The length of the code indicates the resolution level, which in turn indicates the number of possible unique values for the x and y coordinates; for example, resolution 4 means 2^4 or 16 potential x or y values (0-15). The first number in the code is simply the rhombus face number, and can be ignored temporarily. To determine the rotated x coordinate, one can loop through the remaining numbers in the code, one by one, and eliminate the possible values for the x and y coordinates to yield only 1 possible value after the series of iterations. For the x coordinates, this works as follows: each successive digit is tested to see if it is even or odd. This restricts the set of possible values to half the original amount at every iteration. Odd numbers (1 or 3) are on the upper side of the current range, whereas even numbers (0 or 2) are on the lower side of the range. For the y coordinates, a digit of 2 or 3 restricts the set of possible values to the upper half of the current range, whereas a digit of 0 or 1 restricts it to the lower half of the current range. This is illustrated in Figure 3.12. By looping through the entire Morton code, the X and Y coordinates of the point in the skewed system can be determined.

The x,y coordinates are rotated back into a right-handed Cartesian system originating at the left vertex of each rhombus face using the system of equations 3.1 and 3.2. The triangle face number then needs to be computed from the rhombus face number. This is simple: if the Cartesian y coordinate is negative, the triangle is one out of either (6-10) or (16-20); if positive, it is one out of either (1-5) or (11-15). The triangles associated with each rhombus are known beforehand. The coordinates can then be translated to the geographic center of their respective triangle, taking into account triangle orientation. Finally, the inverse ISEA projection formulas from [Snyder, 1992] can be applied to yield a latitude, longitude value. This essentially involves the determination of the azimuth Az and the spherical distance z from the center of the icosahedral triangle face. The geographic centers of all icosahedral triangles are also known beforehand. With an azimuth and spherical distance along with a starting location in geographic coordinates, the geographic coordinates of a point can be calculated using Vincenty's Direct Formula.

3.3.2 3D and 4D DGGS

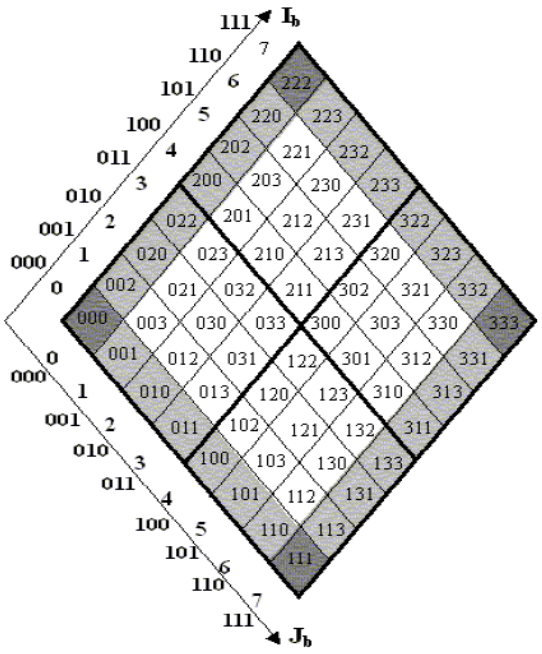

Figure 3.12: The X and Y coordinates in a skewed coordinate system on a rhombus can be found by observing the pattern of numbers in the Morton code of a cell: even (0,2) and odd (1,3) values for X, and (0,1) and (2,3) for Y. Figure from Zhao et al., 2006.

In 3D DGGS, the distance from the Earth's surface is computed using the exact same method as in 2D, except that the range of possible values for the Z-dimension extends both above and below the surface, and this needs to be taken into account in the decoding process. By looping one by one through the numbers comprising the Morton code, the exact encoded Z value can be found.

For 4D DGGS, the hexadecimal base-16 system has been used in this research. As each digit/letter represents a number from 00 - 15, each digit/letter needs to first be decoded into this number. Once all digits/letters are decoded, by looping through the sequence of numbers comprising the full-resolution code and observing every *two* successive digits at a time, the time of acquisition of the point can be computed to an arbitrarily high level of precision. Algorithm 3.2 provides a high-level overview of the Morton decoding procedure in *4*-dimensions.

Algorithm 3.2: 4D DGGS Morton Decode

Input: Morton code *MC* for point, DGGS full resolution number *maxRes*, elevation range *hrange*, and time range *trange*
Output: latitude *lat*, longitude *lon*, elevation *H* and time *T* for point

Face Number $f \leftarrow$ first digit in *MC*
Code $c \leftarrow$ remaining digits in *MC*
Resolution *res* $\leftarrow$ length of *code*
Total range of values *totRange* $\leftarrow$ 2 ^*maxRes*
numXValues, numYValues, numZValues, numTValues $\leftarrow$ 2 pow *maxRes*
xMarker, yMarker, zMarker, tMarker $\leftarrow$ (*numXValues*/2, *numYValues*/2), (*numZValues*/2), (*numTValues*/2)
yVals $\leftarrow$ [0,1,4,5,8,9,12,13]
zVals $\leftarrow$ [0,1,2,3,8,9,10,11]
Code *cNum* with numbers $\leftarrow$ Code *c* of hexadecimal
for $i \leftarrow 0$ **to** *(maxRes - 1)* **do**
    Observe every pair of two digits in *cNum* and narrow down possible values for *xMarker, yMarker, zMarker, tMarker*
Look at last two digits of *cNum* and find precise values of *xMarker, yMarker, zMarker, tMarker*
H $\leftarrow$ (-1) * *hrange* + (*zMarker* / *totRange*) * (2 * *hrange*)
T $\leftarrow$ round((*tMarker*/*totRange*) * *trange*) + 18
Find triangle coordinates from rhombus coordinates
Find azimuth *Az* and spherical distance *z*
lat, *lon* $\leftarrow$ Use Vincenty direct formula
return *lat, lon, H, T*

3.4 CHANGE VISUALIZATION

In order to analyze changes in an area between two moments in time, a separate dataset is needed for each moment (old and new). Unfortunately, DGGS's by themselves are not good frameworks for calculating *distances* between points. These are better calculated using a conventional CRS defined on a map projection. That is, there is no formula that can compute distances between two observations using their DGGS SFC codes directly, without performing a decoding operation. A DGGS is fundamentally designed as an information grid, and not a navigation grid [Purss et al., 2017]. Therefore, changes in the *surface*, which require the computation of minimum distances between points in two point clouds, was not studied in this thesis. Other metrics, such as changes in point density, can be studied. DGGS's offer a scalable approach to spatial data manipulation and one of the three most fundamental questions of spatial analytics, namely *"How has it changed?"* [Purss et al., 2017], can be answered effectively using a DGGS. The SFC codes of observations can be used directly to perform the change analysis.

DGGS's hierarchical, multi-resolution nature is ideal for analyzing spatio-temporal phenomena at arbitrary discrete LOD's. This allows the user much flexibility in choosing a resolution that is most suitable for the application at hand. In order to visualize the above changes, a particular single DGGS

resolution (for example, 17) can be chosen (in 2D), and the point density of all of the points that fall in every cell can be computed by dividing the total number of points in a cell by the cell's area. The areas of the cells are known beforehand, for every resolution, and are fixed (i.e. do not change). The cells can then be symbolized based upon their point densities. The result will yield a visual spatial indication of where most of the changes in point density have taken place. This approach works for point clouds of different densities where the density is consistent throughout the extent of every individual point cloud but varies in-between point clouds, and also for point clouds of different densities where the density varies internally within a single point cloud across its extent. Once the arbitrary resolution at which to conduct the analysis has been chosen, the Morton codes of points only up to that resolution need to be found. This is straightforward as the full-resolution Morton codes are stored for every point; a simple truncation/substring operation needs to be performed to acquire the Morton code only up to a certain length (i.e. resolution). The truncated Morton code indicates that the point could be located anywhere within the 2D DGGS cell that shares the same Morton code. If all of the points are truncated to the same length, the point density within each cell at that resolution can be easily found. Truncation and concatenation operations on the Morton codes allow for moving observations up and down the DGGS hierarchy, without increasing the error in the original observations.

3.5 POINT CLOUD WEB VISUALIZATION

In order to create a viewer allowing for visualization of DGGS-based point clouds, an existing framework is needed.

3.5.1 Comparison of existing solutions

There exist several commercial and open-source Earth viewers, many based on Web Graphics Library (WEBGL), that can be considered. WEBGL is an extension of the underlying Open Graphics Library (OPENGL), which is aimed at providing GPU-rendering capabilities to webpages on many kinds of devices, such as desktops, mobile phones, and tablets [Schûtz, 2016]. A thorough comparison of the existing viewers is outside the scope of this thesis, however a brief overview of some of the more popular ones is provided here.

- **Potree**: Potree is an open-source WEBGL viewer that uses a multi-resolution octree to structure and render the points, in which every internal node has eight children nodes. It uses the *three.js* Javascript library as a coding and rendering environment. The octree allows for efficient view frustum culling and perspective view queries [Martinez-Rubi et al., 2015]. Points are stored in one of the nodes of the octree based upon the spacing between them and their surrounding points. However, Potree renders a point cloud in a flat, Cartesian environment. For the purposes of this thesis, a globe-like representation is more just and appealing.

- **Plasio**: A WEBGL-based viewer still in its early stages of development, it supports a functional implementation of the LAS and LASzip file formats. Also, it only works in Google Chrome. Due to its currently limited capabilities and Cartesian environment, it is not suitable for this thesis.

- **Cesium**: An open-source Javascript library to render 3D globes and maps in a web browser. Cesium offers support for not only point clouds, but also vectors, rasters, terrain, imagery, 3D models, and geometrical objects. As a DGGS can be used to integrate all of these kinds of data, this functionality is ideal. Point clouds can be rendered in the 3D Tiles format (see Section 3.5.2, a relatively new specification for streaming massive heterogeneous 3D geospatial datasets. With its integrated time-slider and support for Cesium Language (CZML) for displaying movements or changes of objects between different times, along with its 3D globe-view, Cesium forms an ideal visualization environment for the purposes of this thesis. Another advantage of using Cesium is that the Light Open Source Point Cloud Server (LOPOCS) (written in Python) can be used to load point cloud data from a spatial database such as PostgreSQL into Cesium using the 3DTiles format.

- **iTowns**:A Javascript/WEBGL framework for 3D geospatial data visualization, iTowns is also based on the three.js library, like Potree. It also supports many formats, including 3D Tiles, 3D textured models, imagery, elevation, GeoJSON, Web Map Service (WMS)/Web Map Tile Service (WMTS)/Tiled Map Service (TMS) services, in both a globe and planar view. It also allows for making precise measurements in 3D. Despite providing the above benefits, iTowns is still in an early stage of development. Also, LOPOCS support is non-existent with iTowns.

After reflecting upon the above considerations, Cesium was chosen as the visualization environment for this thesis.

3.5.2 3D Tiles

An open specification for streaming massive heterogeneous 3D geospatial datasets, 3DTiles can be used to stream 3D content such as buildings, trees, point clouds, and vector data into Cesium. 3D Tiles was created by the Cesium team and is based on the popular GL Transmission Format (GLTF) 3D format for geospatial data. The primary goal of 3D Tiles is to improve the streaming and rendering performance of these datasets. Each tile in a 3D Tile is one WEBGL draw-call. A collection of individual 3D Tiles organized in the form of a hierarchy is referred to as a *tileset*. A tileset is a hierarchical data structure consisting of a number of individual *tiles* located at a certain level within the hierarchy, each of which is enclosed by a bounding volume. Every tile can contain one or many *features*, such as points in a point cloud or 3D models of buildings. Each tile has a number of properties, such as *bounding volume* (defines a bounding box for the tileset in geographic coordinates on the WGS84 ellipsoid), *geometric error* (defines the error, in meters, introduced if this tile is rendered and its children are not), *content* (contains metadata about the tile's content), *refine* (specifies which kind of refinement is desired while moving the observer position), and *children* (an array defining the children of this tile). The *geometric error* property

can be used to perform a perspective view query-based visualization in the Cesium viewer, so that only the most 'important' points are shown when farther from the viewer and successively lesser important points are shown closer to the viewer. 3D Tiles is internally based on a Cartesian ECEF system using the WGS84 ellipsoid, with units in meters.

Point clouds in 3D Tiles are stored in tiles in the .PNTS (points) format. The .PNTS format enables efficient streaming of massive point clouds for 3D visualization. Every tile is composed of a *feature table* and a *batch table*. The feature table stores positions and colors for each point, whereas the batch table allows for storing distinct per-point properties that can be used for declarative styling. The hierarchical nature of 3D Tiles is similar to that of a DGGS. Both are 'tree-like' structures. 3D Tiles allows for cells with increasing volume from the center of the Earth in its specification, useful for 3D DGGS. It also allows for overlap between tiles in a tileset, although for this thesis this approach was not pursued as the cells of a 3D DGGS are non-overlapping.

3.6 ANALYSIS IN A DGGS

A DGGS is essentially a large spatial/temporal database spanning the Earth at many discrete resolutions. Therefore, many of the kinds of spatial operations that can be applied in a spatial database such as PostGIS can be applied on a DGGS. Once a set of observations has been assigned a Morton code, two distinct kinds of algebraic operations can be applied: *cell navigation operations*, which exploit the hierarchical nature of a DGGS for parent-child-sibling relationships, adjacency and neighbor-finding operations, and *spatial analysis operations*, which utilize the framework of the DE-9IM to determine the relationships between DGGS cells and spatial query objects [Purss et al., 2017] and/or conduct spatial analysis using any of the resolutions of a DGGS. Once imported into a DGGS, spatial observations do not need to be further projected or transformed, thereby minimizing the additional error introduced by repeated forward and inverse projections. The cell indices along the Morton SFC are used to perform the above operations on the DGGS.

Every point has an associated cell whose spatial resolution and precision are explicit. To retrieve a representation (i.e. cell) for the same point at a lower resolution, the Morton code of that point can be trimmed to the necessary length and the cell corresponding to the trimmed Morton code can be selected. Reference data stored at a particular level in a DGGS hierarchy can be interpolated or decimated as an array process to move up or down the hierarchy without increasing the initial error in the data. Storing the cells or their contained observations as arrays (i.e. lists) is a scalable approach to spatial analysis. For example, suppose that there are point observations stored at levels 15, 16, and 17 in a DGGS. To generate a map of interpolated LIDAR intensity in a cell at resolution 15, all of the points falling into the geographical extent of that cell (at any resolution) can be "moved" up to level 15 and the points interpolated to form a grid or coverage on the ellipsoid. The points in resolutions 16 and 17 are selected solely based upon their Morton codes, whose prefix will indicate whether they are children of the

cell at resolution 15. The same operation can be applied to traverse down the hierarchy if points at resolutions 15, 16, and 17 need to be spatially analyzed alongside points at resolutions 19 or 20. If points with only a certain range of precision need to be analyzed, then only the points stored in those resolutions can be selected.

As all of the points belonging to a cell share the same Morton code, spatial and statistical operations can be carried out on these points by simply selecting and analyzing the points having a common Morton code. This does not require CPU-intensive operations such as Point In Polygon (PIP) queries. Examples of such operations are finding the average, maximum, median, or most frequent RGB, intensity, or elevation value within a study area. If the geographic area to be analyzed is to be extended, then performing the same spatial analysis on the bigger area (i.e. larger size cells) is simple: all of the points sharing the same prefix (up to a certain length) in their Morton code can be quickly selected and the spatial analysis performed. Only if the area is to be extended into another icosahedron rhombus face will the prefixes be different. Most of the times, however, the study area is usually smaller than the area of a 2D rhombus face, which is 51,006,562.17 square kilometers on the surface of the Earth (see Table A.1). However, since the locations of every rhombus face are known beforehand, the adjacent faces can be easily found. As the difference between the sizes of cells at successive resolutions decreases with increasing resolution (i.e. when going from larger to smaller cells), it is more likely for spatially near points at higher resolutions to have distinct Morton codes than near points at lower resolutions.

The volume of data in a project such as OPCM that is based upon a DGGS is massive. For example, simply storing the 2D cells at resolution 30 across the entire Earth could take at least 15,300 Gigabyte (GB) or 15 Terabyte (TB) of storage space if the cells are stored in Keyhole Markup Language (KML) format; for other less redundant formats, such as GeoJSON, this will be lower, but still very large. Storing a global point cloud will require even more space. For instance, a nationwide point cloud scan of only the United States is expected to generate close to 27 10E15 points, with an estimated storage need of 540 TB [Schûtz, 2016]. Other formats, such as GeoJSON, pose less of a storage overhead as they are less verbose, but regardless of the format used the amount of cells and points in a global point cloud necessitates the need for an advanced computing infrastructure to store and process such large datasets. This very problem makes DGGS exceptionally well suited for High Performance Computing (HPC) and big data infrastructures such as Apache Spark and Hadoop MapReduce. Instead of allocating a workload to only a single node (computer), a cluster of nodes can be utilized to spread out the processing in parallel, distributed algorithms across several nodes, each handling a "chunk" of the workflow. It is worth noting, however, that spatial/algebraic operations requiring access to *all* of the cells of a DGGS at the same time are less common than other operations which require access to only a small portion of the cells at a time. Furthermore, the nested, hierarchical nature of an icosahedral rhombus DGGS necessitates the storage of observations in only the leaf cells (i.e. cells at the largest resolution) along with their cell indices in a database; the extent of data in all of the other cells can be recovered through simple aggregation based upon the cell indices [Bai et al., 2005].

4 IMPLEMENTATION

4.1 TOOLS AND DATA

A variety of tools were used throughout this research.The analysis was tested on two datasets of the same area that have been acquired at different times.

4.1.1 Software

- **DGGRID**: This is a software available in the public domain for creating and manipulating DGGS, and was created mainly by Kevin Sahr. DGGRID allows one to create many kinds of different DGGS, such as triangular, hexagonal, and rhombus ones with varying apertures. This software was used *purely for research purposes*; for example, to verify if the 2D Morton codes have been accurately computed. It was also used to visualize the changes in point density across two moments in time.
- **LASTools**: One of the most popular tools for processing vast LIDAR datasets in a memory-efficient manner, LasTools commands were used for various purposes, such as retrieving information about a LAS file and tiling a LAS file into chunks.
- **C#**: C Sharp was used to compute the scanner-point distances for every point, and store this into the 'Red' and 'Green' color fields of the LAS files where the 'Red' field contains the distance rounded to the nearest meter, and the 'Green' field contains the added distance in millimeters on top of the distance in meters. Therefore, when converted into the same unit and summed, the total of the 'Red' and 'Green' fields yields the total distance between the LIDAR scanner's exit aperture and the point. The C# implementation was performed in Microsoft Visual Studio 2017.
- **Python**: The Python programming language was utilized for most of the coding, including the projection of points from the RD coordinate system to geographic coordinates in WGS84, the conversion to the 2D, 3D, and 4D DGGS Morton codes, and the posting to and retrieval from the database. The Python version used is 2.7.
- **PostgreSQL/pgPointCloud**: As a database engine, the open-source PostgreSQL database and its extension for the storage of point clouds, *pgPointCloud*, was utilized. *pgPointCloud* provides two data types, namely *PcPoint* and *PcPatch*, for storage of points in a flat table (one row per point) or a blocks table (a collection of points in every block/patch), respectively.

- **FME**: The Feature Manipulation Engine suite of software was used to create a translation to convert the points into a vario-scale 3D Tiles tileset for display in Cesium JS.
- **Cesium JS**: As a front-end open-source library for 3D geospatial visualization, Cesium JS was used for the creation of the 3D point cloud viewer. The point clouds are converted into the relatively new 3D Tiles format for streaming massive heterogeneous points, that can be consumed by Cesium.

4.1.2 Hardware

All tests during this research were carried out on a Windows 7 Intel Core i7-2820QM CPU with a speed of 2.30 GHz, 8 GB Random Access Memory (RAM), and a 64-bit operating system.

DGGS's and point clouds, due to their vast sizes, are inherently well-suited for big data infrastructures and cluster computing engines such as Apache Spark, a high-performance framework with built-in modules for streaming, SQL, machine learning and graph processing. At its core, Spark utilizes a data structure known as a Resilient Distributed Dataset (RDD), an immutable, distributed collection of elements that are divided into logical partitions, each of which could be located on a different node (i.e. computer) of the cluster. RDD's are *resilient* in the sense that they keep a record of the transformations used to build them and this can be used to recompute lost data which ensures fault tolerance. *Immutable* means that once created, they cannot be changed, only copied into another RDD. Transformations on RDD's are lazily evaluated (only at runtime). Therefore, running the exact same analysis across a cluster of machines is expected to yield much faster execution times than those provided in this research. Although this is ideal, it is outside the scope of this thesis.

4.1.3 Data

The datasets used in this research include two mobile LIDAR scans of the Forepark area in the Hague, Netherlands. The scans were performed as a part of the Fugro DRIVE-MAP project, in which a vehicle simultaneously collects accurate 3D data and imagery of an area. The first scan was taken in 2010 and the second one was acquired in July 2016. As there was only one laser scanner at the time, the first scan has a lower density of points than the second one. Therefore, taken together, these two datasets of the same area can be used to identify changes between the points clouds at two different moments in time, and can also be used for the purposes of vario-scale visualization. The area of Forepark is approximately 0.83 square kilometers.

As a DGGS is inherently global, for the purpose of handling point clouds it does not really matter where the point cloud is coming from. The same procedure followed in this research can be used on a point cloud originating from any area on the Earth, both above or below the surface. Table 4.1 shows some more details about the two datasets used.

Dataset	File Size	Number Of Points
2010	5.44 GB	162,918,748
2016	6.64 GB	198,365,000

Table 4.1: Details about the datasets used in this thesis

4.2 COMPUTING POINT PRECISIONS

In order to assign a point observation to a DGGS cell, first its (continuous) precision needs to be determined in the units of its original CRS/projection, and stored as an attribute of that point. The original CRS in which the Fugro point clouds are captured is the Dutch RD system, with units in meters. Each point acquired in the Fugro DRIVE-MAP project is time-stamped with its acquisition time, given in GPS Time. Furthermore, the trajectory of the car that collected the points is known along with the times when the car was at each point on the trajectory; this is stored in a file named *poses.ipz* for each scan (i.e. there is a separate poses.ipz file for the 2010 and 2016 scans). The *IPZ* format comes from the manufacturer of the old DRIVE-MAP system. The scanner-point distances can be computed based on matching the time component, and using the known direct linear relationship between distance from the scanner's exit aperture and the size of the beam footprint (provided in Table 4.2), the diameter of the beam footprint can be calculated at the exact location where the beam hit the point. C# was used to perform this procedure. This gives an indication of the 'true' location of the point, which could be anywhere within the 'block' of spatial uncertainty given by the beam footprint; furthermore, any repeated measurements of the same point will likely fall somewhere in the same area. The distinction between precision and accuracy is important to note here.If a point's *accuracy* is to be determined, the *centroids* of every DGGS cell (in n-dimensions) are usually computed and assumed to be the 'true' location of the point, and then the distances from the measured/observed location and the centroid are taken to be the accuracy [Amiri et al., 2015b].

The scanner utilized for capturing the point clouds of Forepark is a Riegl VQ-250 scanner featuring a high speed, non-contact profile measuring system with a narrow infrared laser beam and a fast line scanning mechanism, enabling a full 360 degree beam deflection without any gaps. It is optimally suited for mobile mapping from a variety of moving platforms, such as cars, ships, boats, and railways, similar to what is being done in the DRIVE-MAP project.

Distance	Beam Footprint Width
Exit aperture	7 mm
50 m	18 mm
100 m	36 mm

Table 4.2: The laser beam footprints at various distances from the Riegl VQ-250 LIDAR scanner. A direct linear relationship exists between the distance and width of the beam footprint.

4.3 TILING THE LAS FILES

With some testing, it can be determined that tiling the entire point cloud files into smaller chunks for subsequent processing yields significantly faster execution times than operating on the entire point cloud as one LAS file. For example, the size of the entire 2010 dataset for the Forepark area is 5.44 GB and contains 162,918,748 points. Breaking this up into tiles of different sizes will significantly reduce the processing time for the point cloud than operating on it in its entirety at once. This can be seen in Table 4.3, where for the 2010 point cloud a tile size of 5,000 points each yields an estimated processing and loading time of approximately 31 hours, whereas a tile size of 40,000 points yields an estimated time of 233 hours, almost 8 times longer. The numbers provided are for one of the 4 bulk-loading methods tested (see Section 4.5). It is interesting to note that the total execution time for both the Morton conversion and bulk loading into the database actually increases as tile sizes go lower than 5,000 points each. Therefore, choosing an appropriate tile size is extremely important for fast execution.

Points/Tile	Number Of Tiles	File Size (KB)	Est. Total Time (Hrs.)
40,000	4,073	1,329	233.25
8,000	20,365	266	38.91
5,000	32,584	167	30.98
4,000	40,730	134	31.85
500	325,838	17	34.93

Table 4.3: File sizes and processing times (Morton conversion and bulk loading combined) for chunks of the 2010 point cloud. A tile size of 5,000 points per tile provides the most optimal processing times.

The actual tiling operation was then performed using the *lassplit* function in LasTools, which splits an input file into a set of output files (tiles), with a maximum size parameter of 5,000 points per tile.

4.4 MORTON CONVERSION

Prior to the Morton conversion, the points were projected from their original CRS's into the reference ellipsoid used by the DGGS implementation, WGS84, using the *PyProj* library (see Figure 3.7 for an illustration).

The Morton conversion itself was made using the procedure followed in Section 3.2, extended into 3D DGGS. Figure 4.2 provides visualizations of the Morton SFC (in 2D) traversing the entire Earth, at resolutions 5 and 6. Both 2D and 3D Morton codes were calculated for every point: the 2D code for performing change identification using the cells of a DGGS, and the 3D code for the purposes of querying and visualization with 3D Tiles. Only the full-resolution codes were computed, thereby eliminating the need to store the differences between the discrete and continuous representations of space and time as additional attributes for every point. For the highest possible resolution, a resolution of 32 was used as this provides close to millimeter-level precision, and is enough to uniquely distinguish every point in space and in time (see Table A.1). Moreover, there are already many existing

computer systems that are based on 32-bit storage for numbers. The full-resolution codes originate from the whole Earth. As the study area is very small, especially when viewed at from national, regional, or global scales, most of the digits in the front of the codes are identical. Therefore, if in case only datasets from a smaller region are to be used for a project, there is no need to use/store the entire sequence of numbers originating from the whole Earth; only those numbers distinct to each and every point beginning at a certain position (i.e. certain resolution DGGS cell) in the full resolution Morton code until its end can be used and/or stored, to speed up processing and reduce storage overhead. The first number in the Morton code indicates the icosahedron rhombus face on which the point is located. Therefore, all points on any one face can be easily retrieved by selecting all Morton codes having the same first digit. For all of the points in the study areas used for this thesis, this number is 4 (see Figure 3.3).

The Morton conversion showed a linear relationship between the number of points and the total time needed to acquire the Morton codes for those points, as shown in Figure 4.1. For the 2010 dataset with 162 million points, the conversion took approximately 420 minutes, or 7 hours, and for the 2016 dataset with 198 million points, it took approximately 510 minutes, or 8.5 hours.

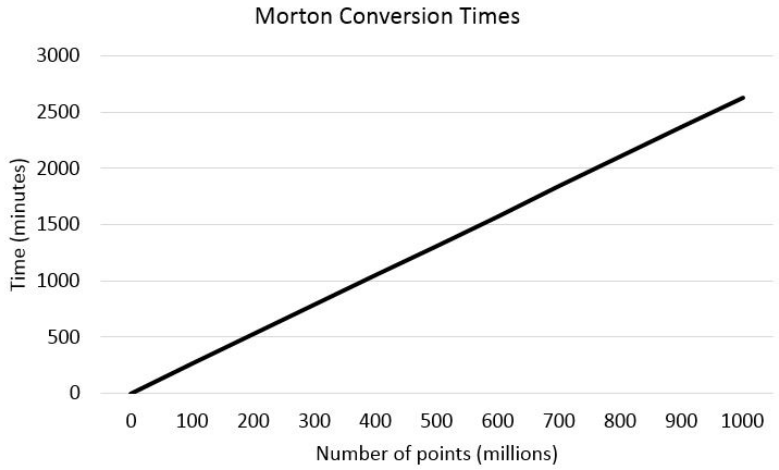

Figure 4.1: The linear relationship between the number of points and amount of time needed to obtain their Morton codes.

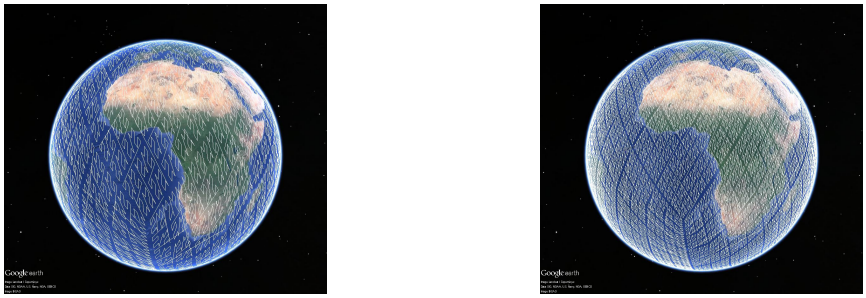

Figure 4.2: Ellipsoidal SFC's at resolutions 5 (left) and 6 (right) of an icosahedral rhombus DGGS. There exists a separate SFC at every resolution of a DGGS.

4.5 LOADING OF DATA

The nature of point cloud data is such that there is just so much of it. This not only poses an overhead for storage but also for retrieval of the data from a database. Furthermore, point clouds are growing in popularity and size every year, and modern technologies have the potential to generate massive point clouds of 10E12 or 10E15 points, each with their own unique attribute values [van Oosterom et al., 2014]. Therefore, the operations that are performed on point clouds need to be scalable so as to not fail or deliver slow response times for large point clouds. For example, the Dutch *AHN2* national point cloud is stored and distributed in 60,000 LAZ files, and for simple query purposes storing this dataset as files is not an efficient use of resources [van Oosterom et al., 2014]. This necessitates the need for a DBMS solution. A DGGS is ultimately a large spatial, or in this case *spatio-temporal*, database. Moreover, a DGGS can be used not only for handling point clouds, but also for incorporating vector and raster data sources in a single framework. If users want to combine different types of data in their queries, a standardized and generic DBMS solution is preferable to file-based solutions [van Oosterom et al., 2014]. Therefore, it is essential to utilize a DBMS for the management and storage of DGGS data.

Examples of existing DBMS's that allow for point cloud storage include Oracle and PostgreSQL, with the *sdo_pc* and *PCPOINT/PCPATCH* data types, respectively. For this research, the open-source *pgpointcloud* extension for PostgreSQL, with its in-built PCPOINT/PCPATCH data types, was used. Both DBMS's allow for the storage of points as either a *flat table* or a *blocks table* (see Section 4.6).

A new database named 'dggs' was created in PostgreSQL that would ultimately store the test datasets. A variety of methods can be used to load data into the database. It is quicker and more efficient to store the data from the Python script directly into the database rather than have to first create LAS/LAZ files with attributes such as the 2D, 3D, or 4D Morton code, continuous point precision, or DGGS discrete resolution, and then load the files into the DBMS. Moreover, the Morton codes are large numbers; 2D, 3D, and 4D full-resolution Morton codes (resolution 32) are all 33 digits long (one digit for the face number and the other digits/letters for the actual Morton code). There does not exist any in-built attribute in the LAS file format specification that is capable of storing such a large number in one field. If the Morton codes are to be stored directly into LAS files first, then they would have to be broken up into multiple smaller components, and each of these would have to be stored in a separate attribute in every chunk of a LAS file. This poses potential problems for data management and is an error-prone process that can be avoided by using Python to connect to and store point data in a database directly.

Psycopg2 is the most popular PostgreSQL adapter for the Python programming language. Most of the features of PostgreSQL can be used via this module that was designed for heavily multi-threaded applications that create and destroy lots of cursors and make a large number of concurrent inserts and updates [Python Software Foundation, 2018]. Psycopg2 also features client-side and server-side cursors and asynchronous communication.

Large tables can have over 10E12 rows of points, and inclusive of any indexes defined on them can occupy over 400 GB of disk space. When inserting data into tables that will eventually become this large, it is important to choose a procedure that allows for the fastest insertion times possible, taking into account memory and disk spaces on the machine on which the insertion is taking place. These can be achieved using bulk-loading instead of loading a single point in every insertion query.

The most naive way of loading data would be to run an individual insert operation for every point, one at a time, into the database. This requires Python to communicate with the database server once for every point, thereby initiating as many query executions as points. This is an extremely slow process, and for only the 162 10E6 points that are a part of the 2010 test dataset for this research, is expected to take close to 25 days. Of course, this is something to be avoided at any cost. Another slightly faster method would be an *executemany* operation, which is essentially the same as the former, but combines multiple INSERT statements into one giant list of INSERT statements (each with only one row in the VALUES clause), and executes them all. This is not performance-optimized and suffers from exceptionally slow response times, much like its predecessor. Moreover, the Python community itself does not recommend usage of this method. A third way of loading data would be to formulate a Structured Query Language (SQL) INSERT statement with a very large VALUES clause as a string and send this to the database server once for every group of chunks/tiles of point cloud data. So, if the whole point cloud is broken up into tiles of 1,000 points each, data from several hundred of these tiles can be combined into the VALUES clause of an INSERT statement, and an INSERT query can be formulated and posted to the database for all of these points. The VALUES clause allows for such a kind of operation, as multiple rows of data to insert can be incorporated into the clause. This is much faster than the previous two approaches. Table 4.3 shows some statistics regarding the insertion of points into the database using this method.

As is evident, a tile size of 5,000 points per tile provides the fastest processing and loading times. It is clear that the times drop significantly when utilizing tiles of less than 10,000 points each, but go up again after making tiles of fewer than 5,000 points each. However, these times are still not fast enough; loading a set of 32,584 files containing a total of 162 10E6 points would take approximately 31 hours with such a tile size. Other tile sizes, for example, 40000, 8000, and 500, would take respectively, 233, 39, and 35 hours. This is clearly something to be avoided, even if it is faster than the earlier two methods. Furthermore, with this method, only at most about 800,000 points can be inserted at any one time without crashing the process, because there are limits to query size. So, for 162 10E6 points, this would require more than 200 post operations.

The fastest way to bulk-load data from Python to PostgreSQL comes from the SQL command dedicated to bulk-loading and unloading of data, *COPY*. COPY allows one to either use an in-memory string of data for input, or load data from an existing file on disk such as a Comma Separated Values (CSV) file. It would be far better if the data is loaded from memory so as to not have to write to an external file first, read the entire file into memory again, and then post the data. If the data is to be read into memory anyway while copying from a file, there is no gain made by first writing the data to a file.

Therefore, data was written to memory as a StringIO object and then posted to the database. It was found that COPY reduced the loading time to only 6 seconds per 1,000,000 points. This means that the loading time for the entire dataset (of 162 10E6 points) would be only 17 minutes. This is currently the fastest existing implementation, given the limitations of Psycopg2, Python, PostgreSQL, and the machine on which the loading has been tested. A tile size of 5,000 points was used, and for every 5 10E6 points (or 1,000 tiles) an insertion query was executed. This approach provides for rapid insertion and fewer overall posts.

4.6 STORAGE OF POINTS

Points can be stored in either a *flat table*, with one point and its properties per row, or as a *blocks table*, with a collection of points that are nearby in space and/or time in one block. PgPointCloud provides the PCPOINT data type for use in a flat table, and the PCPATCH data type for use in a blocks table.

If blocks are to be used, there are two possible approaches as to how they can be used:

1. The blocks could be made to have a maximum size limit, to avoid having large inconsistencies in the number of points in between blocks. In this case, if there are too few points in a block, the block's size can be extended to incorporate a larger range on the Morton curve and thereby more points, however this approach will likely lead to overlapping blocks that contain the same points. The cells of 3D DGGS are, however, non-overlapping and form a complete partition of volumetric space. Each 3D DGGS cell can have a different number of points, and this can potentially vary significantly in-between cells. Therefore, extending some blocks until a threshold is met and not others is not a suitable method for use with DGGS.

2. A block could be made to hold as many points as are physically contained in that block without any maximum threshold. This approach ensures that every point is assigned to only a *single* DGGS cell. This is preferred over the first approach because then the DGGS structure itself can be exploited to generate the blocks. Furthermore, the tiles that are a part of 3D Tiles are usually non-overlapping volumes that, similar to 3D DGGS, form a complete partition of volumetric space both above and below the Earth. Every block can then be also treated as a tile in a 3D Tiles tileset. Storage of the points as non-overlapping blocks rather than as a flat table, therefore, provides a 1-1 relationship between a block and a 3D DGGS cell, and a block and a 3D Tile.

However, blocks are only really useful if their geometries can be created and indexed with a spatial index such as an R-Tree. This greatly aids the query procedure as first the query will determine the blocks intersecting the query region by comparing the geometry of the query region with only the *bounding box* of the block, and not the block's geometry itself. The construction of geometries on blocks is a rather time-consuming procedure. If the cells of a DGGS themselves are exploited to create the blocks, with points

assigned to the closest 3D DGGS cell as to their precision, most of the blocks are expected to contain only one point, because the cells to which they are assigned are so very small in size. For example, for the 2010 dataset, the number of total points is 162,918,748 and the number of total blocks is 162,883,995. Therefore, approximately 99.97% of the points are falling into only one block each directly based upon their discrete precision. The points could theoretically be 'moved' to a block/3D cell at a lower resolution/precision. However, the use of blocks necessitates the need for two queries to be run on the data: first, to select all the blocks intersecting the query region, and second, to select all the points falling into those blocks. It also requires storing a separate flat table containing unique per-point properties anyway, so creating a blocks table will only add to existing storage overheads. The creation of the blocks table, much less the creation of any geometries, is itself a time-consuming process.

With a flat table, only *one* query is needed, one that operates on the points directly. Properties which are unique to every point, such as its continuous precision (i.e. importance) can be stored in their own separate fields. With the use of the query procedure described in Section 4.7, no construction and/or comparison of geometries is needed, and the Morton codes of the points themselves can be used to retrieve all of the points in the cells overlapping the query region. Because of the above justifications, a flat table approach was utilized for this thesis.

Storage of a 2D DGGS Morton code requires 2 bits per digit (level), because each digit can take the values 0-3. Storage of a 3D DGGS code requires 3 bits per digit, with values from 0-7, and with 4D DGGS this requires 4 bits per level, with values from 0-F (with two groups of hexadecimal digits, 10 with hex numbers from 0-9 and 6 with hex numbers from A-F). Furthermore, storing the initial top-level rhombus face number (1 out of the 10 options) requires 1-4 bits itself. Therefore, the total number of bits required to store a full-resolution (i.e. resolution 32) Morton code is 64 bits in 2D, 96 bits in 3D, and 128 bits in 4D exclusive of the rhombus face number. Inclusive of the face number, this could be 4 bits larger. 2D, 3D, and 4D (if the hexadecimal system is used) DGGS full-resolution Morton codes are 32 digits long, exclusive of the rhombus face number. Therefore, the maximum number of bits needed at resolution k is $4 + (n * k)$ for n dimensions, whereas the maximum number of digits is $k + 1$ for 2D, 3D, and 4D (if hexadecimal is used). Of course, the number of bits needed will decrease for smaller resolutions. As a consequence, the only two numeric PostgreSQL data types capable of storing such large numbers are DECIMAL and NUMERIC. The codes could also be stored as strings in a TEXT data type field. The TEXT data type was ultimately used, and it provides a variable storage size and allows for easy string manipulation on the Morton codes. For example, to find the parents or children of a rhombus, simple truncation or concatenation of the Morton codes is needed, and storing the codes as strings allows for such operations without introducing any additional overhead. If the codes are stored as numeric values, a type-cast to text and back to numeric would have to be performed. This is something to be avoided if fast query execution times are desired.

Unfortunately, as an icosahedral rhombus tessellation has 10 faces, storing the face number necessitates up to 4 bits of storage, thereby wasting 6 other possible values (4 bits is 16 possible values, while only 10 - the numbers 0 to

9 - are being used). In 2D and 3D DGGS, however, this is only the situation with the first number in the location code. For all numbers after the first number, no bits are wasted.

As point clouds acquired in two different years are being analyzed (2010 and 2016), one would either have to store a flag variable (i.e. field) indicating the year for every point or block, but this would necessitate storing redundant information and is undesirable. What is better is to store the point clouds from 2010 and 2016 in completely separate flat tables. There would then be no need to store the year along with every point record, since the points from different years are stored in separate tables themselves. Furthermore, as the points are spaced 6 years apart in time, the 4D Morton codes of points at (approximately) the same X, Y, Z location but taken in the two different years are significantly different from one another. The attributes that need to be stored per point include the *full-resolution* 2D and 3D Morton codes, discrete DGGS resolution, and (continuous) point precision. With the full-resolution Morton code, storing the latitude, longitude, and height values for every point is unnecessary as the code encodes the point's location up to an exceptionally high level of detail and can be decoded to yield these values. Using Psycopg2's *COPY _FROM* command, the insertion into the flat table took only 6 seconds for every one 10E6 points, approximately 17 minutes for all of the 162 10E6 points in the 2010 dataset, and only slightly longer for the 198 10E6 points in the 2016 dataset (with a tile size of 5,000 points per tile).

4.7 QUERYING A DGGS

Once the points are stored into separate flat tables, a B-Tree index was created on the 3D Morton code to allow for faster retrieval. A B-Tree is a self-balancing tree structure that maintains data in a sorted order and allows for searches, sequential access, insertions, and deletions in logarithmic time [Comer, 1979]. Although the index itself takes up several GB's of disk space, it's creation is a one-time process. Making an index on the *full-resolution* Morton code, however, does not make much sense, because the the cells at resolution 32 are so very small that they reduce to a 0-D point geometry. Querying these 'fake' cells at such a high resolution would essentially mean that we are querying the points directly. In DGGS, however, the *cell* is the fundamental unit of spatial analytics [Amiri et al., 2015b], and the operations that are performed using a DGGS (see Section 3.6) are all cell-based processes.

The power of a DGGS comes from the fact that real-world observations that have been mapped to a DGGS structure can be 'moved' up or down the hierarchy to obtain a finer/coarser representation of the same observation at a higher/lower LOD. This process does *not* add to the error present in the original data; we are only retrieving a representation of an observation at a lower (or higher) LOD. Therefore, even though all of the points have been stored with a full-resolution Morton code that is 33 digits long, for the purpose of querying an index was added on only a *truncated* full-resolution Morton code. This can be an arbitrarily defined number of digits long and application-specific. It is dependent on the resolution of a DGGS that needs

to be used to perform the analysis at hand. The decimation of observations to a lower level in the hierarchy is made by simply truncating the Morton codes to a specific length (since the length of the Morton code indicates the resolution). For the purposes of this thesis, a resolution of 17 was used, at which the cells have an inter-cell spacing of 53 meters and an area of 2,969 square meters on the ellipsoid. This was chosen so as to get a reasonably average number of cells (not too many, not too few) to cover the study area of Forepark. Therefore, in the index creation SQL command, a *substring* of the full-resolution Morton codes was used up to the 18th digit (yielding a representation of points with codes that are 18 digits long; i.e. one digit for the face number and the other digits comprising the actual code). The B-Tree index creation took approximately 22 minutes on a table of 162 10E6 points, and only slightly longer for the 2016 table. It is important to mention here that creating a B-Tree on a numeric Morton code column takes slightly less: 17 minutes for the 2010 table. However, the advantages provided by storing the Morton codes as a string (i.e. text) clearly surpass those provided by storing it as a numeric value. Next, the flat tables were also clustered based upon the substring (at resolution 17) and not the full-resolution Morton code. This physically reorders the data in the table based on the index information, so that the amount of disk pages to be fetched is reduced. Once an index has been created on a truncated Morton code, spatial or spatio-temporal queries can be performed at the particular level to which the codes have been truncated. This means that coarser-resolution cells can be used as a filter for spatial queries at higher resolutions.

A possible method of querying a DGGS would be to first compare the geometries of all cells at a particular resolution with the geometry of the query region, to see which ones intersect. This can be speeded up by adding an R-Tree index on the cell geometries so only bounding boxes are compared, as explained earlier. However, the comparison of cell geometries, much less their construction and storage in a DBMS, is a costly procedure, and can be avoided completely. Moreover, a DGGS based query does not require any storage of cell geometries, because the points have the same Morton codes as their assigned cell and they can be queried directly based upon these codes.

The query procedure utilized for this thesis was performed using Python and is as follows:

1. A query region in minimum and maximum latitude, longitude, and height coordinates on the WGS84 ellipsoid is defined. Figure 4.3 shows an example illustration of a query polygon.

2. An equally spaced grid of points in latitude/longitude (2D) and/or height (for 3D) is created inside this query region where the spacing between the points is small enough so as to encompass all of the cells that overlap the query region. This inter-point spacing is different for every resolution of a DGGS, because the inter-cell spacing is also different for every resolution. For best performance, the inter-point spacing should be set to a value such that it is not too small to result in too many points, but also not too large so that some cells are 'skipped'.

3. The Morton codes of these points are found. As the Morton codes of the cells are the same as their contained points, this also means that the Morton codes of the cells that overlap the query region are found.

4. As there can be more than one point inside a cell in the created grid, duplicate Morton codes are removed. This yields a set of codes that represent all of the cells overlapping the query region, without any duplicates.

5. All of the points having the same Morton codes as these cells are then retrieved from the DBMS. This retrieval is relatively fast as the flat table has been indexed with a B-Tree on a substring of the Morton code. The retrieval is only done using a truncated version of the full-resolution Morton code; that is, the full-resolution codes are retrieved, but they are only retrieved using a decimated version of the codes. The attributes retrieved include the 2D and 3D Morton codes and the continuous precision (used for vario-scale visualization, discussed in Section 5.1).

6. The retrieved full-resolution Morton codes are then decoded back into their latitude, longitude, and height values on the WGS84 ellipsoid, and these decoded coordinates can then be visualized as points. With the decoded values, distance and/or algebraic operations in geographic space can be easily performed, or the points could be projected onto a map projection to work with units such as feet or meters.

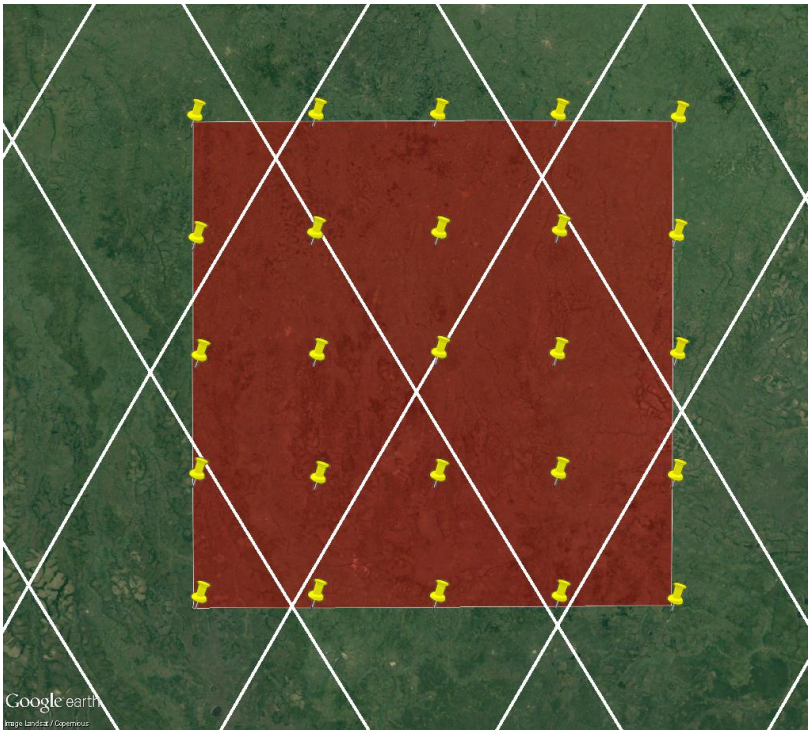

Figure 4.3: Visualization of the query procedure used for this thesis to retrieve point observations in a query region.

This yields a set of all of the points falling into *all* the cells that *intersect* the query region.

Although for the purpose of this thesis latitude, longitude, and altitude information was not stored alongside every point as additional attributes, this can be performed as-needed for other applications. Storing these coordinates as additional attributes makes it unnecessary to decode the selected Morton codes from Step 5 of the query procedure. The points would then have to be *selected* based on their indexed and clustered Morton codes,

but only the latitude, longitude, and altitude coordinates would have to be *retrieved* from the DBMS. This results in faster visualization and/or other applications towards which geographic coordinates can be directly utilized, such as distance operations. However, as there are additional attributes to store in the DBMS, more overhead is posed for the loading procedure explained in Section 4.5.

4.7.1 Exact match

For some applications, an *exact* match might be desired to select only the points falling perfectly into the n-D query region. It is then a simple matter of reducing the superset of points retrieved in the previous step into a set of points falling perfectly within the query region through comparison of the latitude, longitude, and height values of the retrieved points with the minimum and maximum latitude, longitude, and height values of the query region to determine whether a point is inside or outside of the query region. This requires no comparison of geometries, only of simple numeric values. However, this additional step is application-dependent and might not be required at all times.

4.7.2 Index On Full Key

If a B-Tree index is added on the full-resolution Morton code rather than on a truncated Morton code, the index itself will be very large. This is because the full-resolution keys are unique to each and every point. Only when they are truncated to a lower level in the DGGS hierarchy are there many duplicate keys as the cells become larger and larger. Moreover, if an index is added on the full-resolution code, the query procedure stated in 4.7 cannot be utilized. At full-resolution, the cells become infinitesimally small so as to behave like points. A regular grid of points equally spaced in latitude, longitude, height above/below the ellipsoid, and/or time will not encompass all of these infinitesimally small cells and therefore there will be several points left unreturned by the query. Moreover, the spacing between the points would also have to be made to be very small, and this will also pose an overhead for the creation of an index and query execution time, as the query would have to retrieve points directly (without using the *cells* to retrieve the points). Moreover, if any kind of spatial analysis such as change detection is to be performed across a study area, it would make more sense to use cells which are not in the range of a square-millimeter (i.e. at full-resolution) in size to perform such a kind of analysis. A resolution that provides cells which are sufficiently large (but not too large) for the analysis at hand should be used. Therefore, adding an index on a truncated Morton code and performing analysis at that respective resolution would allow one to exploit the equal-area nature of a DGGS *cell* and its power of moving an observation to any level in the hierarchy to perform the intended spatial analysis.

5 RESULTS

This chapter provides an overview of the results of this research in relation to the research questions posed in Chapter 1.

5.1 WEB VISUALIZATION

The goal of this section is to provide an answer to the sub-research question: *"How can a variable-scale structure be implemented to support smooth zoom in DGGS?"*

A prototype viewer was developed using the Cesium frontend Javascript API along with Node.JS, a Javascript runtime built on Google Chrome's V8 Javascript engine that allows the execution of Javascript code server-side. In the past, Javascript was used as merely a client-side language to add functionality to dynamic web pages that also used HTML and CSS. Node.JS allows server-side execution of Javascript code, with scripts running on a web server that respond to requests from the client and produce a response customized for each user's (client's) request. Javascript uses an event-driven, non-blocking asynchronous I/O model, allowing high throughput and scalability for web applications that are built using it. The user can draw a 3D bounding box in WGS84 coordinates, and make a request to the server to retrieve all of the points from the DBMS that are physically contained within the box. As the user can draw any arbitrary box, the algorithm first converts the query window into a latitude, longitude, altitude aligned box. It then sends these bounding coordinates to the server, which feeds them as arguments into a Python script. Python can be executed from a Node.JS server using Python Shell. A Python script is then triggered, which connects to the PostgreSQL database server where the points are stored as a flat table, and the query procedure described in Section 4.7 is performed.

The full-resolution Morton codes retrieved at the end of the query procedure were then decoded into their latitude, longitude, and height values (above or below the surface) on the WGS84 ellipsoid using the procedure outlined in Section 3.3. The full-resolution code allows for accessing the 3D position of a point at an extremely fine LOD. These points can then be used for web visualization in 3D Tiles. More information about 3D Tiles is provided in Section 3.5.2.

There are many different methods that can be used in order to allow for smooth zoom in/zoom out; these are simply different existing implementations. The *geometric error* property of a 3D tile can be used. Geometric error is a non-negative number (i.e. greater than or equal to 0) in meters, introduced if a tile is rendered and its children are not. At runtime, the geometric

error is used to compute Screen Space Error (SSE), i.e., the error measured in pixels. The SSE determines if a tile is sufficiently detailed for the current view or if its children should be considered [Analytical Graphics Inc., 2018]. Therefore, a pre-processing process can be performed to put the high importance points into tiles with a lower geometric error and the low importance points into tiles with a higher geometric error. The points retrieved from the previous step are then sorted according to importance (i.e. low importance to high importance). They are then grouped into blocks/tiles and input into an FME translation that creates a 3D Tiles tileset. This means that the high importance points are always rendered (i.e. both closer to and farther from the observer position) as they have a lower geometric error, and the low importance points are only sometimes rendered (i.e. when closer to the observer position). Although this approach does not allow for *true*-smooth zoom, it is sufficiently close. Moreover, the magnitude of smooth zoom that can be achieved can be increased by creating more tiles. The more the number of tiles, the larger the smooth zoom effect. The reason this approach does not allow for *true* smooth-zoom is because a single geometric error value is used for all of the points inside a tile. However, the importance is distinct for each and every point. In order to get a true smooth-zoom visualization, a unique *per-point* property named 'importance' would have to be stored, and this can only be done using the *Batch table* of a .PNTS file. Unfortunately, there are no existing open-source tools that allow for manipulating the data stored in the batch table of a .PNTS file in 3D Tiles. Furthermore, the data in the batch table is stored in binary form and cannot be modified using a simple text editor.

Another property that can be set to allow for smooth zoom is a variable named *cesium_priority*, in FME. This attribute allows for control of LOD, and can be used to change the relative placement of features within the tile hierarchy. Features with a higher priority value will be closer to the top of the 3D Tiles tree, meaning that they will be rendered from higher zoom levels than features with a lower value. For example, points with a higher *cesium_priority* value will be rendered first, before points with a lower *cesium_priority* value, as the user zooms in from a smaller scale to a larger scale (i.e. larger area to smaller area, or global to continental or nationwide scales). If only the most important (most precise) points are to be displayed when zoomed far out, the *inverse* of the importance values needs to be used. This is because more important points are more precise, and therefore their precision is a smaller number than lesser important points; however, larger *cesium_priority* values are rendered first, so to put the most important points at the top of the 3D Tiles tree structure, the inverse of their importances needs to be used. This can be done by setting the *cesium_priority* attribute to a large value such as 100 divided by the continuous point precision (i.e. area of the LIDAR beam footprint in square meters or square millimeters, for example).

Other methods also exist; however, this thesis has utilized the *geometric error* property of a 3D tileset. Figure 5.1 shows a visualization of a 3D DGGS query result using Google Earth. The DGGS-based Morton codes stored in the DBMS were queried on-the-fly according to the procedure described in Section 4.7, decoded, converted into 3D Tiles, and visualized in a web browser using Cesium. However, as Google Earth provides a readymade 3D building and terrain model for the study area, for illustration purposes it has been shown instead of Cesium in Figure 5.1.

Figure 5.1: A 3D DGGS query result visualized in Google Earth. This particular 3D query returned only one DGGS cell at resolution 17, and its shape clearly resembles that of a rhombus. The query selected all points in all cells overlapping a 3D bounding box in between 42 and 45 meters above the WGS84 ellipsoid.

5.2 DGGS FOR TEMPORAL ANALYSIS

To provide a visual indication of the areas in a point cloud where the most changes have taken place, a separate dataset is needed at two different moments in time. For this thesis, two datasets of the same area - one taken in 2010 and the other in 2016- were used (see Table 4.1). The datasets are stored in two separate flat tables and are queried using the procedure outlined in Section 4.7.

Two queries are made in a user-defined latitude, longitude aligned polygon, one for the 2010 and one for the 2016 dataset. The points are retrieved at resolution 17 of the DGGS. The attribute retrieved is the full-resolution 2D Morton code. As both flat tables have been indexed (using a B-Tree) and clustered based upon a substring at resolution 17 of their Morton codes, this retrieval is very fast, even for tens of 10E6 of points. Then, the codes are grouped into chunks based upon their truncated Morton codes at resolution 17. This grouping process (known colloquially as *binning*) ensures that every point in the same 2D DGGS cell at this resolution is put in the same chunk. As arrays or lists are used to work with the retrieved points, this binning process is exceptionally fast and scalable. Once the points are grouped, the point density is calculated in each chunk by dividing the total number of points by the area of a cell. Figures 5.2 and 5.3 show the point density per cell in the 2010 and 2016 datasets, respectively, in a sample query region. The differences in the densities (2016 - 2010 dataset) are then visualized on a cell-by-cell basis as shown in Figure 5.4. Unfortunately, the datasets greatly differ from one another in the distribution of points across the study area; that is, there exist areas with many points in the 2010 dataset, but only a few to no points in the 2016 dataset, or many points in the 2016 dataset but only a few to no points in the 2010 dataset. Therefore, there exist some DGGS cells (in light gray in the figures) that contain no points. Only cells containing points in both the datasets are shown in Figure 5.4. It can be

readily observed that the 2016 dataset is in general more dense than the 2010 dataset.

A clear benefit of a DGGS-based approach in this application is that equal-area cells are being used to conduct the analysis.

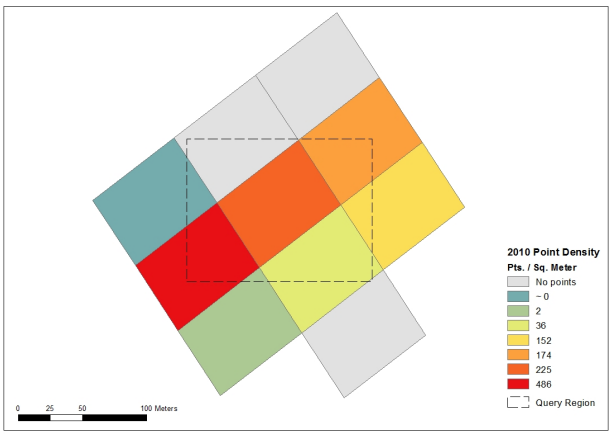

Figure 5.2: Point density in the 2010 dataset in a small subset of the study area.

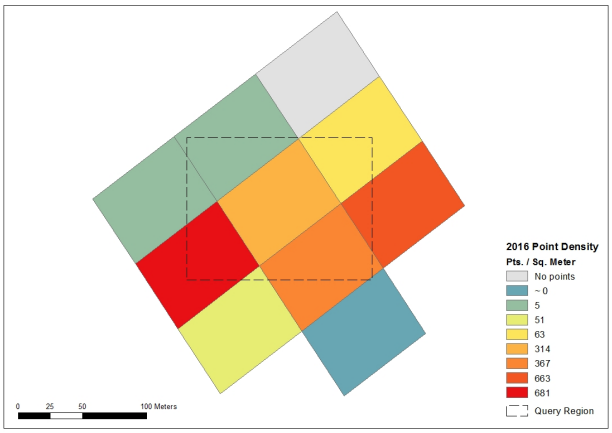

Figure 5.3: Point density in the 2016 dataset in a small subset of the study area.

5.3 COMPARISON OF DGGS WITH CONVENTIONAL REFERENCE SYSTEMS

This section provides an answer to the sub-research question *"What are the advantages and disadvantages of using a DGGS as compared to using conventional CRS's?"*

It provides a detailed overview of the similarities and differences, and advantages and disadvantages of utilizing a DGGS as compared with conventional CRS's, map projections, and datums. More specifically, four separate systems are compared with a DGGS: the latitude/longitude graticule, ITRS, ETRS, and the Dutch RD system. The motivation for this section stems from the question,

"Can we simply discontinue the use of existing standard coordinate reference systems and switch to DGGS?"

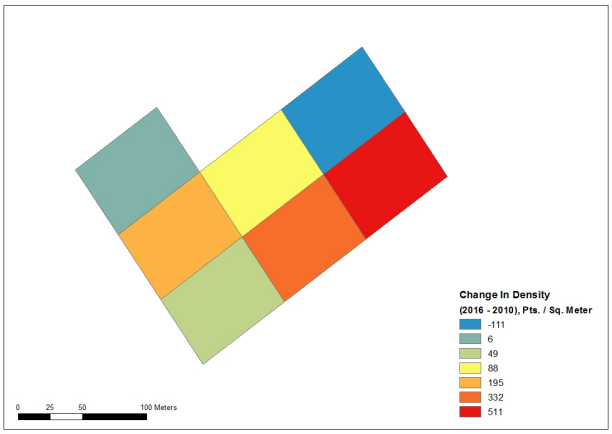

Figure 5.4: The difference in point densities in between the two years. Only cells containing points in both datasets are shown. The use of equal-area cells makes such a kind of spatial analysis more meaningful.

The Dutch system is studied as this research is being carried out in the Netherlands. The motivation for this comparison stems from the lack of any previous studies on this topic. Although the use of DGGS's goes back until at least the mid-20th century, they were released as a new standard by the OGC only in October 2017. They are only in their far early stages of commercial use as of this writing, and much of the conventional spatial community is finding it hard to comprehend the advantages of a DGGS as opposed to a conventional CRS.

Some general differences between traditional CRS's and DGGS's include:

1. In standard CRS's, coordinates are given as linear measurements along an axis, whereas in DGGS they are locations along an n-dimensional SFC. Although many DGGS's are based on equal-area projections, the coordinates are ultimately converted into geographic coordinates that are encoded into SFC codes. Therefore, coordinates in standard CRS's are simple integer or floating point numbers, whereas those in DGGS are cell ID's along a SFC.

2. Standard CRS's contain coordinates on a flat projected plane, whereas DGGS's use coordinates on a spherical/ellipsoidal surface. For analysis on continental or global scales, where the curvature of the Earth becomes an important factor to consider, DGGS's allow for better preservation of error.

3. One can apply algebraic operations (addition, subtraction, multiplication, division) on standard coordinate values, and find distances between two points. With DGGS, however, direct distances between two points on the SFC cannot be found unless a decode is performed. Angles and surface areas are also better computed in coordinate space. Coordinates are more apt for computations.

4. Standard CRS's are single-resolution, whereas DGGS's are multi-resolution, thereby allowing observations to be studied and analyzed at any discrete LOD.

5.3.1 Latitude and longitude

The latitude/longitude graticule is also a DGGS, as it subdivides the surface of the Earth into a set of cells that are regularly spaced apart at equal increments of latitude and longitude. These cells can be further refined by smaller equal increments of degrees to generate a hierarchy of cells. However, equally-spaced grid lines become increasingly distorted in area and shape as one moves north or south from the Equator; the distortion increases even further with the use of an ellipsoid instead of a sphere as a model for the Earth. The top and bottom rows of cells (that touch the poles) are not quadrilaterals but instead triangles (as they converge onto the poles), and the poles, which are points, become lines. This is precisely why special grids have been implemented for polar regions [Sahr et al., 2003]; for example, the UTM system uses the Universal Polar Stereographic (UPS) system for mapping polar regions. Therefore, this is not an equal-area system. Moreover, implementing a SFC-based indexing technique becomes even more challenging, as in theory there can be infinitely many SFC codes for the North and South poles. Furthermore, geostatistical analysis loses its efficacy as non-equal area cells are being used. These problems have been instrumental to the growth of equal-area DGGS's as an alternative to mapping and modeling the Earth.

5.3.2 The Dutch Rijksdriehoeksstelsel (RD) system

Pros

- The RD system has a history dating from the 19th century [van der Marel, 2016]. As such, many existing geospatial databases and paper documents in the Netherlands are based on the RD [Geonovum, 2014].
- Despite some recent efforts for a transition to the European standard ETRS, the RD remains the most commonly used system. The transition to ETRS itself will be a complicated process that will require the involvement of all stakeholders [Geonovum, 2014], and will likely take a couple of decades. With DGGS, this will be an even more intensive, long, and complicated process. Furthermore, this will likely also be financially burdensome.
- Many users already find it difficult to work with geographic coordinates and rather prefer grid (map) coordinates. With DGGS the user needs to work with something completely different, *cell indices*, thereby introducing a new form of coordinate representation that will require a certain level of understanding of how SFC's work and how DGGS's are constructed.
- RD is linked to ETRS through a transformation procedure named *RDNAPTRANS*, which allows users to convert their coordinates to ETRS or keep working with RD. The availability of this transformation means that users can choose to work in whichever of the two systems they prefer. For example, users of GPS or Global Navigation Satellite System (GLONASS) can work with geographic coordinates in ETRS. However, this necessitates the need to maintain separate databases for RD

and ETRS. In the current working version of ISO 19152 LADM, for example, coordinates in both systems are stored as two separate attributes [Buren et al., 2008]. This leads to the need for multiple systems and duplicate storage, risk of error in transformations back and forth between the two systems, and possible ambiguity in an international context. According for changing coordinates in databases due to the movement of tectonic plates is already a big challenge in standard CRS's. DGGS, on the other hand, are not yet implemented in conventional software, and are not widely used reference systems in the geospatial community [Open Geospatial Consortium, 2017a].

Cons

- RD is based on an oblique stereographic double projection, a *conformal* projection [van der Marel, 2016], whereas the majority of DGGS's utilize *equal-area* projections that are better suited for cartographic and geodata analytics purposes where each individual cell has an equal probability of contributing to an analysis.
- Although RD has now also been defined for the North Sea area in addition to the mainland Netherlands, it is, at least as of this writing, not used for marine applications.
- RD requires a "correction grid" to correct the local distortions in the grid coordinates as the planar grid has been based on triangulated geodetic networks established more than 100 years ago. Unfortunately, the correction grid is not supported by most existing GIS software, and the oblique stereographic double projection is considered obsolete [van der Marel, 2016]. If a user fails to utilize the correction grid, results will be erroneous; distortions in X and Y coordinates could reach up to 25 centimeters.
- RDNAPTRANS is supported by only a few GIS software packages. For example, the open-source QGIS software does not support RDNAPTRANS.

5.3.3 European Terrestrial Reference System of 1989

Another alternative for GIS users in the Netherlands is the European ETRS 1989 system that uses geographic coordinates on the ETRS datum.

Pros

- Being the standard CRS for Europe, it is the reference system of choice for most global geospatial projects in Europe.
- ETRS is fixed to the stable part of the Eurasian tectonic plate, making it opportune for high-precision mapping and land surveying applications. There is usually no need to publish movements of coordinates over time in ETRS, as velocities can be at most a few millimeters per year; just the coordinates themselves are sufficient.

- Similar to the RDNAPTRANS procedure, many other countries have also implemented transformations between their own systems and ETRS through the use of GPS.

- Eurocontrol, the European air safety and traffic management organization, has adopted ETRS as the European realization of WGS84 for aviation applications, and several other organizations such as EuroGeographics, Eurostat, the European Union, European Environment Agency and the Open GIS Consortium have recommended using ETRS for continental Europe.

- The European Union has mandated ETRS as a CRS for INSPIRE [INSPIRE TWG CRS/GGS, 2017], an initiative facilitating exchange, combining and use of environmental geographical information for all Member States of the European Union by defining new standards for GI as well as setting up geodata infrastructure.

- ETRS is also defined for the sea, unlike RD, which is only defined for the mainland Netherlands.

- ETRS has been tied to ITRS (the most accurate terrestrial reference frame to date) in the year 1989. Therefore, ITRS coordinates can be derived from ETRS coordinates and vice-versa, in case the user wants to work with coordinates in an international setting.

- ETRS is implemented in almost all popular GIS software.

- With the growth and popularity of applications such as Google Earth, and with the use of GPS, the generic public is becoming more aware than ever before of working with geographic coordinates, and this is making them more competent in working with ETRS.

Cons

- Geodata exchange in Europe in ETRS is uncommon, as most countries are still using their own national systems. For example, Belgium uses the Belgian Lambert 2008 and 1972 projections [Voet, 2011], and France uses a Lambert conformal conic projection named Lambert-93.

- The user will typically be working with geographic coordinates, which are not suited for computations in planar space such as distances, lengths, or areas. However, the user is free to convert these coordinates into grid coordinates in any map projection. This can be problematic as different municipalities or regions in a country could potentially use their own unique map projections for use with ETRS, and geospatial analysis conducted on a nationwide scale will suffer from the problems of dissimilar data integration.

5.3.4 International Terrestrial Reference System

Pros

- ITRS is the most accurate terrestrial CRS to date. It can be used to study the movements of features on the Earth's surface as it takes into account tectonic plate movement and regional subsidence. Every ITRS realization, known as an ITRF, contains positions and velocities of reference stations across the world that have been surveyed and measured.
- ITRS is an ECEF system with 3D Cartesian coordinates, and these can be transformed into latitude, longitude, height triplets on any ellipsoid, such as GRS80 or WGS84. Both GRS80 and WGS84 are linked to ITRS [van der Marel, 2016]. *Every* DGGS is also linked to some model of the Earth. So, if it is linked to the same ellipsoid as the ITRF, coordinates in both systems can be used in tandem.

Cons

- ITRS uses geographic coordinates. For industrial workers such as those in the construction industry, for example, projected coordinates in units such as meters or feet are usually desired. This necessitates the conversion and/or datum transformation of ITRS coordinates into a suitable map projection (such as UTM) for use with ITRS, or into topocentric coordinates, which could introduce a certain degree of error.
- As it takes into account plate motion, a dynamic correction grid is needed to deal with deformations due to geological forces. Station positions and velocities need to be continuously updated for each realization, and so do geospatial databases. This requires additional investment of time, money, and effort.

5.3.5 DGGS

Finally, the pros and cons of DGGS are listed.

Pros

- As has been demonstrated in this thesis, a DGGS is an ideal structure to use for indexing, clustering, and querying global point cloud data.
- Some spatial analysis, such as adjacency or nearest neighbor queries, can be more efficiently executed by computer algorithms operating on a regular cell structure as in a DGGS where coordinates are stored as locations along a SFC in arrays than on geometric features whose coordinates are stored as floating-point numbers [Lott, 2017]. Array/list storage and manipulation is also scalable; i.e. storing point observations in an array with their Morton codes and other attributes, and performing aggregation, grouping, sorting, filtering, mapping, or other similar tasks on these observations using the Morton codes is an extremely scalable procedure, and can be used to perform tasks

such as computing the mean, median, maximum, minimum, or modal values of these attributes in every containing DGGS cell. These are precisely the kinds of operations that are now being standardized as a part of the DGGS Modeling Language [Peterson, 2018]. Examples of attributes that can be stored along with every point include intensity, color, elevation, or classification, and cell-by-cell analysis can be performed using SFC codes along with these attributes; computing systems are adept at such kinds of cell-based operations. With respect to this thesis, examples of such operations include finding the mean point density within every cell by grouping and mapping all observations on a cell-by-cell basis using their Morton codes. The other attributes listed above could also be analyzed statistically using the Morton codes.

- The vast majority of DGGS's utilize equal-area cells, which allow for uniform coverage of data in an area of interest, and a hierarchy of resolutions, which allow the encoding of observations acquired at any level of detail. The user is free to choose a cell size or resolution that is most applicable for the task at hand.

- If all geodata is aligned to the same global grid, the cost introduced by repeated projections and datum transformations on the data in the form of error and affected outcomes is kept to a minimum. With DGGS, the user needs to only map the data *once* at a particular level in the hierarchy; all further operations involve not error-introducing projections, but array processes using set theory. The cost of data integration is also largely reduced.

- Once the orientation of the polyhedron is defined with respect to the model of the Earth, the location of the cells remains constant. This is ideal for change detection applications where changes in the *exact* same area/volume on the Earth's surface need to be studied. Over time, the locations of observations will 'march' across the DGGS, and if appropriate metadata is stored alongside every observation, either the cells can be queried for a particular set of observations, or an observation for the set of associated DGGS cells.

- DGGS further the 'Digital Earth' vision as conceptualized by former United States vice president Al Gore, who imagined a digital representation of the Earth with all kinds of data describing georeferenced observations on its surface attached to it in an integrated knowledge framework. DGGS's are optimal tools for geodata integration from various sources, regardless of their resolution or dimensions.

- DGGS were recently approved as an official OGC standard for interoperability. This provides a strong foundation for their use and implementation.

- DGGS can be used to work with data stemming from any location on Earth (2D), both above and below the surface (3D), and acquired at different times (4D), as demonstrated in this thesis. In theory, they can be extended into infinitely many dimensions; a 3 and 4 dimensional DGGS was introduced and studied in this thesis. Moreover, they could even be used on other celestial bodies, such as the Moon or Mars.

Cons

- Classical algebraic operations (addition, subtraction, multiplication, comparisons, etc) are not possible as cell indices on SFC's are being used. Distances or directions between DGGS SFC-encoded observations cannot be found without performing a decoding operation first. The only possible operations are cell navigation and spatial relations between two geometries as described by the DE-9IM. Coordinates are better suited for computations.
- There are already many existing reference systems and ways for notation of coordinates, and these can cause a lot of confusion; introducing another approach to encoding coordinates could potentially make things worse, as the number of options increases. Moreover, there are many different kinds of DGGS's themselves.
- SFC codes can become rather long, especially when working at extremely high resolutions. If the study area is small, however, there is no need to store the entire sequence of numbers originating from the whole Earth (only a part of the code beginning at a particular level in the hierarchy and ending at a level that is sufficient to differentiate observations is needed). For continental or global scales, though, this poses a problem. For high-precision applications, such as surveying or positioning, where high-precision codes are needed, lengthy codes might not be particularly useful.
- A slight disadvantage could also be that, due to their innate discrete nature, an observation's spatial uncertainty could be grossly misrepresented after assignment to the closest DGGS cell, if the difference between the cell's area and the amount of spatial uncertainty of that observation is considerably large. Furthermore, once the orientation of a polyhedron is defined, the location of all DGGS cells are fixed. If a point does not fall in the center of a containing DGGS cell (which is very likely), its area of spatial uncertainty could be misrepresented due to it being assigned to a cell that physically contains the point but that only *partially* overlaps with the beam footprint (a polygonal region representing the spatial uncertainty) at that point. This is less of a problem for observations with relatively low spatial uncertainties (such as points in a point cloud), since the difference in cell areas at higher resolution DGG's is much lower than that at lower resolution DGG's.
- DGGS's by themselves do not account for tectonic motion and other geological forces, like ITRS. The n-D cells are fixed in location as relative to an Earth model once the orientation of the polyhedron is defined. DGGS internally 'depend' on systems such as ITRS to provide them reference station position and velocity information. Every DGGS is based upon a conventional CRS and/or datum. Therefore, DGGS's cannot be used by themselves if information about the system on which they are based is not available. In practice, however, DGGS's are usually based upon popularly used reference frames such as the ITRS, for which all associated information is available.
- Although this might change in the future, DGGS are not currently implemented in conventional GIS software.

- Understanding how a DGGS works itself can be a daunting task for a novice. To fully understand and be able to use DGGS, one requires a reasonable level of understanding of mathematics, geodesy, databases, and geometric concepts such as set theory.

After the above considerations, it is the *author's view* that DGGS are more likely to be used *alongside* standard CRS's, and are *not* a *replacement* for them. When there is a need to integrate different kinds of geospatial data (such as vector, raster, and point cloud) into one solution, DGGS's offer a viable alternative as they bridge the divide between these various kinds of data. Any of these kinds of data can be mapped onto a DGGS. DGGS's are in no way a replacement for existing CRS's, as these are better suited for distance or area computations.

It must also be stated that a DGGS is *not* a tool for point cloud *alignment*. That is, if the task is to resolve the discrepancies between point clouds that are based in different CRS's (i.e. align point clouds so that they 'fit'/'match' one another by moving one of the point clouds a certain distance one way and another point cloud a certain distance another way), DGGS's are not a solution. Alignment in this context implies concepts such as *distance* and/or *direction*), and DGGS's cannot be used directly to calculate distances between observations. The observations are encoded using SFC codes, which cannot be directly used for calculating distances on a sphere or an ellipsoid. The codes themselves need to first be decoded into latitude, longitude, and altitude values that are then used for the distance calculation. This means that a DGGS itself is not used. Computations such as distance and/or direction are much better performed using a standard map projection/ CRS. A DGGS is not meant for navigation, rather for data analytics purposes. Integrating point cloud data into a DGGS assumes that the point clouds are already aligned, and a DGGS provides a fundamentally critical, seamless common reference frame for storing, analyzing, and visualizing the data. Therefore, a DGGS does provide a common reference frame, but it cannot be directly used for any distance-based operations. The only exception to this is if a hexagonal DGGS is used on a spherical Earth model. Hexagons provide for uniform adjacency [Sahr et al., 2003]. When hexagons are inverse-projected from a polyhedron onto a sphere, they maintain their uniform adjacency, and as the inter-cell spacing is known beforehand for all resolutions, the distances between hexagon centroids at all resolutions are also known. Therefore, distance-based calculations between hexagon centroids can be performed on a spherical Earth model. However, a SFC might not be directly usable for performing this calculation, unless a decode is performed.

So, the author does not conclude that the geospatial community in the Netherlands can simply switch to a DGGS and discontinue the use of RD or ETRS. It is rather more likely for DGGS's to be used as an alternative to these existing CRS's. The full-fledged adoption of DGGS could take several decades, as DGGS's are only in their very early stages of commercial use. Their use is also dependent to a large extent on the application at hand. For example, for indexing and clustering global point cloud data or for visualizing changes in spatio-temporal point clouds over time, as has been done in this thesis, their use makes perfect sense.

6 CONCLUSION

My thesis offers several innovative aspects of study. To-date, there have been no studies on using point clouds with DGGS. Moreover, the DGGS I ultimately chose (an icosahedral rhombus DGGS) has also never in previous academic literature been applied to the indexing of any kind of spatial data. There were many aspects of this topic that posed a challenge for me as I began my research. First, simply understanding the mathematics and science behind DGGS's was a challenge by itself. After a thorough academic literature review of around 3 months, I came to know that the use of DGGS's with point clouds is non-existent. There has been no similar research performed in academia or industry on integrating point clouds with varying locations, times, and densities. Moreover, as DGGS's have only been used in 2 dimensions to-date, it was scientifically stimulating to ponder how they could be extended into 3D and 4D. However, with sufficient thought and research, I was able to think of a way to extend them into higher dimensions to truly harness the multi-dimensional nature of point cloud data. To my knowledge, this is *the first-ever* conceptualization of higher dimensional DGGS's made. Third, there does not exist much research on generating a SFC through the cells of a DGGS to index and cluster (in a DBMS) and then later query spatial observations, and no research on the utilization of the hierarchical nature of DGGS's to yield a vario-scale visualization of massive amounts of points in a web browser. Finally, there also exists no in-depth comparison of DGGS's with conventional CRS's. Therefore, these are all new aspects of DGGS's being introduced by this thesis. With this thesis, I hope to provide not only the university but also the wider geospatial community with a detailed overview of the applicability of DGGS's to handling point clouds and the pros and cons of using DGGS's as opposed to conventional CRS's.

This section provides an answer to the main research question of this thesis posed in Chapter 1, that reads as follows:

To what extent can a Discrete Global Grid System be used to handle point clouds with varying locations, times, and densities/levels of detail?

It can be concluded that a DGGS is an ideal reference system for the integration of point clouds coming from varying locations, having varying initial levels of detail, and acquired at different times. A fundamental requirement in the creation of the OPCM is a common underlying reference frame for the access, storage, processing, integration, analysis, and visualization of global point clouds. Moreover, a scalable approach to big geospatial data analytics is needed. DGGS's provide a scalable approach to spatial analysis as every operation conducted inside a DGGS is an array process using set theory and the DE-9IM.

Indubitably, DGGS's transcend map projections and can be used as a common reference frame to integrate data from many different projections, datums,

ellipsoids, and CRS's. Moreover, the reference frame is global, allowing for the encoding of any location on Earth. It is also seamless, without any boundaries. One of their main uses in this thesis is with regards to the indexing, clustering, and querying of global point cloud data. For analysis at scales where the curvature of the Earth becomes a determining factor, DGGS's are more optimal than map projections. The fundamental difference between a standard map projection/CRS and a DGGS is in its operation; error-prone data needs to be mapped into a DGGS only once, without the need to apply repeated error-introducing map projections and datum conversions. With their multi-resolution hierarchical framework, observations can be represented at any discrete LOD. DGGS's allow for unconventional *cell-based* spatial analysis methods in sharp contrast to existing standard coordinate computations. Furthermore, they can be extended into n-dimensions using an n-dimensional SFC, as has been shown in this thesis; this allows us to fully harness the power of the multi-dimensional nature of point cloud data.

DGGS's by definition are discrete, so they allow for the encoding of vector features, raster cells, and points in a point cloud at a fixed set of discrete resolutions. They are precision-encoding systems so the area of a DGGS cell to which an observation has been assigned should approximately be the same as the spatial uncertainty of that observation. Due to their discrete nature, a slight disadvantage is that this leads to a certain loss in the encoding of precision (which is continuous) as the observation is assigned to the *closest* discrete cell. The area of the assigned DGGS cell might differ significantly from the spatial uncertainty of that observation; this is especially true for observations with a very large spatial uncertainty that are assigned to lower resolution DGGS cells; for points in a point cloud, however, it is *not* a critical issue, because points usually do not have extremely large spatial uncertainties. In order to achieve a true vario-scale visualization of points, the precision of a point can be used as its importance, as more precise points are shown at both smaller and larger scales and are stored in higher levels (larger cells) of the structure. Points which are not very precise are also not very important, and therefore during visualization they should be shown only at larger scales (i.e. when zoomed in sufficiently close). Most of the points will fall somewhere 'in between' the layered discrete set of cells constituting a DGGS, as precision is a continuous variable and not limited to discrete values. However, this added continuous dimension can be used to visualize the points in a web browser using a perspective view query using smooth zoom. Although in this thesis only a pseudo-smooth zoom effect was reached, this is only due to an implementation issue with 3D Tiles, as there are no open-source tools that allow for storing and working with per-point properties in the batch table of a .PNTS file.

As the n-D cells are fixed in position relative to the Earth model once the orientation of a DGGS is specified, a DGGS provides a framework where one of the three fundamental questions of big geospatial data analytics can be answered: "How has it changed?" [Purss, 2016]. Over time, the observations will march across the DGGS, and if appropriate metadata is stored alongside the observations, they can be queried for all of the cells in which they were present throughout a course of time. In this thesis, 'moving' observations were not studied, however point clouds taken in two different years (spaced 6 years apart) were analyzed for changes using a DGGS structure.

Another contribution of this thesis is to the study of the similarities and differences, and advantages and disadvantages, of DGGS's in contrast with conventional CRS's. Although DGGS's have their own limitations, when it comes to indexing, clustering, and conducting analysis on global point cloud data their use makes perfect sense. Their hierarchical, multi-resolution nature allows the possibility of moving point observations up or down the hierarchy *without* increasing the error in the original point observations. Some major limitations are that algebraic operations or computations cannot be easily performed and SFC codes can become rather long. Regardless, they serve as an attractive alternative to conventional CRS's.

With the above considerations, it can be concluded that a DGGS is an ideal reference frame to handle varying point clouds.

6.1 PROBLEMS FACED IN RESEARCH

Although in general the research questions were answered exceptionally well, there were also some problems faced throughout the course of the research. Most of these are *implementation-related* and are not conceptual/-theoretical. A list of some of them follows:

- The visualization of the mean HD per containing DGGS cell requires the creation of the cell geometries themselves. The only freely provided tool that allows for the creation of these cells as a KML file or shapefile is DGGRID. Unfortunately, as cells at a relatively high resolution (20) are being used to perform the analysis, the generation of the geometries of these cells for the whole Earth would take an extremely long time. Furthermore, there is no working functionality to clip the DGG layer by a clipping polygon, so that only those cells intersecting the study area are being generated.
- For achieving true smooth zoom, a unique per-point importance property needs to be stored. The batch table (and not the feature table) of a 3D Tiles point cloud (.PNTS) file is where per-point properties reside. However, as 3D Tiles is still under development, there do not exist any free or open-source tools that allow for manipulating the data stored in the batch table of a .PNTS file. This being said, an almost true level of smooth zoom was achieved in this thesis, using some other properties of a 3D Tiles tileset. Moreover, the effect can be increased by creating more tiles.
- The loading into the DBMS was in general fast, given the limitations of the Intel Core i7-2820QM CPU on which it was tested. However, utilizing an even better performance, network, or memory optimized CPU is expected to speed up the process significantly.
- As the cells in 2D DGGS are all of the same area, it makes sense to make them all of the same volume in 3D. However, as the amount of space expands outwards (almost linearly) as a function of distance from the center of the Earth, the volume of the cells is not the same within a DGG in a 3D DGGS. To achieve a sufficiently close level of equal-volume cells throughout a resolution would require excessive additional manipulation such as the addition of more cells or removal of existing cells at

different distances from the center of the Earth model. This introduces many potential challenges for DGGS's, including a potential loss of the hierarchical nature of the cells, which is key. For example, adding more cells at higher elevation levels within a DGG and not at other levels would mean that the parent-child relationships get affected (if, for example, one of the newly added cells falls into *two* parent cells and not one), and it becomes tedious to use the beneficial characteristics of the subdivision shapes for any practical purpose (for example, using the aperture 4, congruent and nested nature of a rhombus tessellation to index global point cloud data). For point clouds, which are usually acquired close to the surface where the volumes are roughly identical, this is not a big issue, but for other observations that span a larger range of elevation values this could be problematic.

6.2 FUTURE WORK

This section presents some recommendations on future work.

6.2.1 Utilization of a different DGGS

As mentioned in Section 2.5, many kinds of DGGS's exist, each with their own unique advantages and disadvantages. The hexagon could be investigated as a spatial partitioning method due to its maximal compactness, uniform adjacency, and shape that more closely resembles the footprint of a LIDAR beam than a rhombus. The octahedron could be tested as a polyhedron, and although it is a worser approximation of the Earth than an icosahedron, it has the advantage of having 8 faces, which makes it easy to align with the latitude, longitude graticule [Dutton, 1996], and provides for lesser storage, as 3 bits are needed to store face numbers and no bits are wasted during index assignment. Furthermore, some of the Archimedean polyhedrons are even better approximations of the Earth than the Platonic polyhedrons; examples of these are the rhombicosidodecahedron, truncated icosidodecahedron, or snub dodecahedron (see Figure 2.3); although the construction of a DGGS on these polyhedrons is much more complicated due to them having multiple shapes, and their use in commercial implementations is limited, they present an interesting alternative choice of options worthy of exploration.

6.2.2 Utilization of a different SFC

Within this thesis, the Morton SFC was utilized for indexing, clustering, and retrieval of point cloud data. However, many different SFC's exist, each with their own unique properties. The key difference is in their method of mapping n-dimensional space to a 1-dimensional line. The Morton curve has the advantage that it is relatively easy to program, although in the context of a DGGS many complicated mathematical formulae need to first be implemented in order to encode and decode a Morton code for a point location. DGGS's can work with any kind of SFC's, and examples of such other curves are the Hilbert, Gray, Sierpinski, and Cantor SFC. In particular, the

Hilbert curve preserves spatial locality far better than the Morton curve and could also be investigated on a DGGS structure, although it is more complicated to program.

6.2.3 Using a DGGS to study moving observations

A point, although a 0-dimensional object, can also be represented as a polygon whose area represents that point's spatial uncertainty (as has been done in this thesis). As observations will move across the DGGS reference frame due to geophysical processes such as erosion, orogenesis, volcanic activity, or tectonic motion, they will 'visit' different DGGS cells over the course of time. If a DGGS is tied to a model of the Earth such as the GRS80 ellipsoid, which is also the base for the ITRS, then the information regarding the speed and direction of movement or deformation can be acquired from the ITRS. If this is stored alongside every point observation, then it is possible to retrieve the cells visited by the point over time, or the points that fell into a cell at any given moment in time. The handling of such dynamic and/or unpredictable datasets is a viable topic for further research [Amiri et al., 2015b].

6.2.4 Implementing a distance or direction metric on a DGGS

The cell indices in DGGS's are locations along a SFC. However, as has been stated earlier, SFC codes cannot be directly used to compute distances between point observations on a sphere or an ellipsoid *unless* a decoding operation is performed. If a SFC code is decoded into its latitude, longitude, and height coordinates, standard great-circle or Vincenty formulae can be used to compute distances between points, but this means that a DGGS itself has not been used for these calculations. It would be phenomenal if a distance metric was introduced on a DGGS structure directly, that allows the computation of *geodesic* distances using, for example, the Morton SFC codes. The same can be said about *directions* between two DGGS-based Morton-encoded points. These are precisely the kinds of benefits standard CRS's have over DGGS's, and what would propel one to use a standard CRS in favor of a DGGS. Using a DGGS structure directly for this task might not be possible, but an attempt can be made. Other SFC's could be tested to see if they are better fit for the computation of distances.

6.2.5 Standardization of a 3D and/or 4D DGGS

The OGC DGGS Abstract Specification is an excellent first step to standardization and interoperability of various 2D DGGS implementations. As there currently does not exist any implementation of a 3D or 4D DGGS, this thesis aspires to promote the use of such higher-dimensional DGGS's and provides for an approach to how the distance of a point from the surface of the Earth model and the time at which a point was acquired can be incorporated into a DGGS structure, thereby turning it into a 3D and/or 4D DGGS. This results in much faster spatio-temporal queries. However, there is an additional need for *standardization* of such higher-dimensional DGGS's. For example, with respect to time, there are several aspects that can be standardized, such as

the use of leap seconds, amount of temporal refinement applied at different DGGS resolutions, the choice of which system to use (i.e GPS Time or other system), which end value to use for the time range, temporal CRS's or whether or not to use time zones. With 3D DGGS, the conceptualization proposed in this thesis could be considered for standardization. Moreover, different Earth models need to be taken into account, not only mathematical formulations such as the sphere or ellipsoid but also physical ones like the geoid.

6.2.6 Using parallel processing

Due to their vast sizes, DGGS's are inherently well-suited for scalable, distributed, HPC infrastructures. The algorithms and/or spatial analysis performed in a DGGS is conducted as an array process using set theory and/or the DE-9IM. This is a scalable approach to spatial data manipulation. Moreover, the nature of point cloud data is such that there is just so much of it. Amazon Web Services (AWS) can be used to run a HPC cluster of nodes in the cloud in a parallel and distributed manner. AWS provides various kinds of servers on demand, optimized for specific purposes, such as memory, network, computation or storage. HPC workloads on AWS run on these servers, powered by Amazon Elastic Compute Cloud (EC2). All of the tests in this thesis have been run on a single node (computer); using a cluster of nodes is expected to significantly speed up execution times.

BIBLIOGRAPHY

Ali M. Amiri, Faramarz Samavati, and Perry Peterson. Categorization and conversions for indexing methods of discrete global grid systems. *ISPRS International Journal of Geo-Information*, 4(1):320–336, 2015a.

Ali-Mahdavi Amiri, Troy Alderson, and Faramarz Samavati. A survey of digital earth. *Computers and Graphics*, 53:95–117, 2015b.

Ali-Mahdavi Amiri, Troy Alderson, and Faramarz Samavati. Data management possibilities for aperture 3 hexagonal discrete global grid systems. 2016.

Analytical Graphics Inc. 3d tiles, 2018. URL https://github.com/AnalyticalGraphicsInc/3d-tiles.

ASPRS. Laser (las) file format exchange activities, 2013. URL https://www.asprs.org/committee-general/laser-las-file-format-exchange-activities.html.

Jianjun Bai, Xuesheng Zhao, and Jun Chen. Indexing of the discrete global grid using linear quadtree. *ISPRS Workshop On Service And Application Of Spatial Data Infrastructure*, pages 267–270, 2005.

John Bartholdi and Paul Goldsman. Continuous indexing of hierarchical subdivisions of the globe. *International Journal of Geographical Information Science*, 15(6):489–522, 2001.

S. Boriah, V. Kumar, M. Steinbach, C. Potter, and S. Klooster. Land cover change detection: A case study. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 857–865. ACM Press, August 2008.

Joop Van Buren, Ebrahim Hemmatnia, Chrit Lemmen, and Peter Van Oosterom. Europees coöordinatensysteem is voor nederland vooral een kans. *Geodata-inwinning; Special of Geo-Info and Vi Matrix*, pages 20–23, 2008.

CloudCompareWiki. Global shift and scale, April 2016. URL http://www.cloudcompare.org/doc/wiki/index.php?title=Global_Shift_and_Scale.

James Clynch. Earth coordinates, February 2006. URL https://web.archive.org/web/20150418092513/http://www.sage.unsw.edu.au/snap/gps/clynch_pdfs/coorddef.pdf.

Douglas Comer. The ubiquitous b-tree. *Computing Surveys*, 1I(2):121–137, 1979.

Peter Dana. Introduction to geodetic datums, 2017. URL https://www.colorado.edu/geography/gcraft/notes/datum/datum_f.html.

G. de Haan. Scalable visualization of massive point clouds. *Management of massive point cloud data: wet and dry*, pages 59–67, 2010.

Geoffrey Dutton. Encoding and handling geospatial data with hierarchical triangular meshes. 1996.

ESRI. Lidar point classification, November 2017a. URL http://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/lidar-point-classification.htm.

ESRI. Understanding spatial relations, 2017b. URL http://edndoc.esri.com/arcsde/9.0/general_topics/understand_spatial_relations.htm.

ESRI. What is lidar intensity data?, April 2017c. URL http://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/what-is-intensity-data-.htm.

ESRI. What is lidar data?, October 2017d. URL http://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/what-is-lidar-data-.htm?_sm_au_=iHVsN7PqnnTDftJR.

European Data Portal. Benefits of open data, August 2017. URL https://www.europeandataportal.eu/en/using-data/benefits-of-open-data.

European Environment Agency. Inspire, 2017. URL https://www.eea.europa.eu/about-us/what/seis-initiatives/inspire-directive.

Kenneth Foote and Donald Huebner. More about gis error, accuracy, and precision, 1995. URL https://www.e-education.psu.edu/geog469/node/262.

Geonovum. Van rd naar etrs89: kans of dreiging, 2013. URL https://docs.google.com/file/d/0B2JKk84LeoLJYzZlNkVDRTk4Y2s/edit.

Geonovum. Onderzoek overstap rd naar etrs89, 2014. URL https://www.geonovum.nl/onderwerp-artikel/onderzoek-overstap-rd-naar-etrs89.

M.J.E. Golay. Hexagonal parallel pattern transformations. *IEEE Transactions on Computers*, C-18(8):733 – 740, 2000.

Michael Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69:211–221, 2007.

C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE Transactions on Image Processing*, 5:794–797, 1996.

Matthew Gregory, A. Kimerling, Denis White, and Kevin Sahr. Comparing geometrical properties of global grids. *Cartography and Geographic Information Science*, 26(4):271–288, 1999.

Matthew Gregory, A. Kimerling, Denis White, and Kevin Sahr. A comparison of intercell metrics on discrete global grid systems. computers, environment and urban systems. *Computers, Environment and Urban Systems*, 32 (3):188–203, 2008.

Erika Harrison, Ali Mahdavi-Amiri, and Faramarz Samavati. Optimization of inverse snyder polyhedral projection. In *International Conference on Cyberworlds (CW)*, pages 136–143. IEEE Computer Society, October 2011.

Chris Hopkinson. The influence of flying altitude, beam divergence, and pulse repetition frequency on laser pulse return intensity and canopy frequency distribution. *Canadian Journal Of Remote Sensing*, 33:312–324, 2007.

ICSM. Datums 2: Datums explained in more detail, December 2016. URL http://www.icsm.gov.au/mapping/datums2.html.

INSPIRE TWG CRS/GGS. Inspire specification on coordinate reference systems - guidelines, 2017. URL https://inspire.ec.europa.eu/documents/Data_Specifications/INSPIRE_Specification_CRS_v3.0.pdf.

Nick Johnson. Damn cool algorithms: Spatial indexing with quadtrees and hilbert curves, November 2009. URL http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves.

Mathias Lemmens. Point clouds (1), 2014. URL https://www.gim-international.com/content/article/point-clouds-1.

Charles Loop and Jim Blinn. Rendering vector art on the gpu, October 2017. URL https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch25.html.

Roger Lott. Lats and longs, wgs84 or dggs?, 2017. URL http://www.ee.co.za/article/lats-longs-wgs84-dggs.html.

D. Lu, P. Mausel, E. Brondizio, and E. Moran. Change detection techniques. *International Journal Of Remote Sensing*, 25(12):2365–2401, 2004.

O. Martinez-Rubi, S. Verhoeven, M. Van Meersbergen, M. Schûtz, P. Van Oosterom, R. Gonçalves, and T. Tijssen. Taming the beast: Free and open-source massive point cloud web visualization. *Capturing Reality Forum 2015, 23-25 November 2015, Salzburg, Austria*, 2015. URL http://resolver.tudelft.nl/uuid:0472e0d1-ec75-465a-840e-fd53d427c177.

Martijn Meijers. *Variable-scale Geo-information*. PhD thesis, Delft University of Technology, 2011.

Harvey Miller. Tobler's first law and spatial analysis. *Annals of the Association of American Geographers*, 94(2):284–289, 2004.

Daniel G. Montaut, Michael Roux, Raphael Marc, and Guillaume Thibault. Change detection on point cloud data acquired with a ground laser scanner. In *ISPRS Workshop - Laser Scanning*, pages 1–6, September 2005.

Mircea Neacsu. Wgs84 or grs80, 2011. URL https://www.uvm.edu/giv/resources/WGS84_NAD83.pdf.

Pascal Neis and Alexander Zipf. Analyzing the contributor activity of a volunteered geographic information project — the case of openstreetmap. *ISPRS International Journal of Geo-Information*, 1(2):146–165, 2012.

NPS. Time systems and dates - gps time, April 2016. URL www.oc.nps.edu/oc2902w/gps/timsys.html.

Judy M. Olson. Map projections and the visual detective: How to tell if a map is equal-area, conformal, or neither. *Journal of Geography*, 105(1):13–32, 2006.

Open Data Diliman. Open data diliman, October 2017. URL https://opendata.upd.edu.ph/doku.php?id=exemptions.

Open Geospatial Consortium. Discrete global grid systems domain working group, October 2017a. URL http://www.opengeospatial.org/projects/groups/dggsdwg.

Open Geospatial Consortium. Ogc announces a new standard that improves the way information is referenced to the earth, October 2017b. URL http://www.opengeospatial.org/pressroom/pressreleases/2656.

Open Knowledge International. What is open data?, October 2017. URL http://opendatahandbook.org/guide/en/what-is-open-data/?_sm_au_=iHVsN7PqnnTDftJR.

Patrik Ottoson and Hans Hauska. Ellipsoidal quadtrees for indexing of global geographic data. *International Journal of Geographical Information Science*, 16(3):213–226, 2002.

Perry Peterson. Digitalearth canada. In *International Conference on Cyberworlds (CW)*. PYXIS, March 2018.

Norbert Pfeifer. Point clouds from calibration to classification, 2018. URL https://3d.bk.tudelft.nl/pdfs/pcp2018/pcp2018_NorbertPfeifer.pdf.

D. Pfoser, Y. Tao, K. Mouratidis, M. Nascimento, M.F. Mokbel, S. Shekhar, and Y. Huang. In *Advances in Spatial and Temporal Databases*, page 25. Springer, August 2011.

Stella Psomadaki, Peter van Oosterom, Theo Tijssen, and Fedor Baart. Using a space filling curve approach for the management of dynamic point clouds. *ISPRS Annals Of The Photogrammetry, Remote Sensing, And Spatial Information Sciences*, pages 107–112, 2016.

Matthew Purss, Robert Gibb, Faramarz Samavati, Perry Peterson, J Andrew Rogers, Jin Ben, and Clinton Dow. Discrete global grid systems abstract specification, October 2017. URL http://www.opengeospatial.org/docs/as.

Matthew B.J. Purss. Discrete global grid systems a new way to manage 'big earth data', 2016.

Python Software Foundation. psycopg2 2.7.3.2, 2018. URL https://pypi.python.org/pypi/psycopg2.

QPS. Utc to gps time correction, January 2018. URL https://confluence.qps.nl/qinsy/en/utc-to-gps-time-correction-32245263.html.

Felix Rohrbach. Lidar footprint diameter, October 2015a. URL http://felix.rohrba.ch/en/2015/lidar-footprint-diameter/.

Felix Rohrbach. Point density and point spacing, October 2015b. URL http://felix.rohrba.ch/en/2015/point-density-and-point-spacing/.

Nathan Rooy. Calculate the distance between two gps points with python (vincenty's inverse formula), December 2016. URL https://nathanrooy.github.io/posts/2016-12-18/vincenty-formula-with-python/.

Sacred Geometry. Archimedean solids, 2017. URL http://www.sacred-geometry.es/?q=en/content/archimedean-solids.

Kevin Sahr. Location coding on icosahedral aperture 3 hexagon discrete global grids. *Computers, Environment and Urban Systems*, 32(3):174–187, 2008.

Kevin Sahr, Denis White, and A. Kimerling. Geodesic discrete global grid systems. *Cartography and Geographic Information Science*, 30(2):121–134, 2003.

Markus Schûtz. Potree: Rendering Large Point Clouds in Web Browsers, 2016. URL https://www.cg.tuwien.ac.at/research/publications/2016/SCHUETZ-2016-POT/.

John Snyder. An equal-area map projection for polyhedral globes. *Cartographica*, 29:10–21, 1992.

John Snyder. *Flattening the earth : two thousand years of map projections*. University of Chicago Press, 1993.

John Snyder and Philip Voxland. An album of map projections, 1989. URL https://pubs.er.usgs.gov/publication/pp1453.

Christian Strobl. Dimensionally Extended Nine-Intersection Model (DE-9im). In *Encyclopedia of GIS*, pages 240–245. Springer, 2008.

Mike Tully. Just how accurate is lidar?, December 2012. URL https://aerialservicesinc.com/2012/12/just-how-accurate-is-lidar/.

Hans van der Marel. *Reference Systems For Surveying And Mapping*. Faculty of Civil Engineering and Geosciences, Delft University Of Technology, 2016.

Peter van Oosterom. Variable-scale topological data structures suitable for progressive data transfer: The gap-face tree and gap-edge forest. *Cartography and Geographic Information Science*, 32(4):331–346, 2005.

Peter van Oosterom, Oscar Martinez-Rubi, Milena Ivanova, Mike Horhammer, Siva Ravada, Theo Tijssen, Mike Horhammer, and Martin Kodde. Point cloud data management. 2014.

Peter van Oosterom, Oscar Martinez-Rubi, Milena Ivanova, Mike Horhammer, Daniel Geringer, Siva Ravada, Theo Tijssen, Martin Kodde, and Romulo Goncalves. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. *Computers and Graphics*, 49:92–125, 2015.

Edward Verbree, Martin Kodde, and Peter van Oosterom. Ogc 1703 open point cloud map, October 2017.

Pierre Voet. Geodetic infrastructure in belgium, June 2011. URL http://www.clge.eu/documents/events/93/infrastructure_belgium.pdf.

Ruisheng Wang, Jiju Peethambaran, and Dong Chen. Lidar point clouds to 3-d urban models: A review. *IEEE Journal Of Selected Topics In Applied Earth Observations And Remote Sensing*, 11(2):606–627, 2018.

Denis White. Global grids from recursive diamond subdivisions of the surface of an octahedron or icosahedron. *Environmental Monitoring And Assessment*, 64(1):93–103, 2000.

Denis White, Kevin Sahr, A.J. Kimerling, and L. Song. Comparing area and shape distortion on polyhedral-based recursive partitions of the sphere. *International Journal Of Geographical Information Science*, 12(8):805–827, 1998.

Wen Xiao, Bruno Vallet, and Nicolas Paparoditis. Change detection in 3d point clouds acquired by a mobile mapping system. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 64:331–336, 2013.

Xuesheng Zhao, Jianjun Bai, Jun Chen, and Zhilin Li. A seamless visualizaton model of the global terrain based on the qtm. In *Advances In Artificial Reality And Tele-Existence*, pages 1136–1145. Springer, November 2006.

A APPENDIX

A.1 DGGS STATISTICS TABLE

Resolution	Number Of Cells	Area (Sq.m.)
0	10	51,006,562,172,409
1	40	12,751,640,543,102
2	160	3,187,910,135,776
3	640	796,977,533,944
4	2,560	199,244,383,486
5	10,240	49,811,095,871
6	40,960	12,452,773,968
7	163,840	3,113,193,492
8	655,360	778,298,373
9	2,621,440	194,574,593.2
10	10,485,760	48,643,648.31
11	41,943,040	12,160,912.08
12	167,772,160	3,040,228.02
13	671,088,640	760,057.0049
14	2,684,354,560	190,014.2512
15	10,737,418,240	47,503.5628
16	42,949,672,960	11,875.8907
17	171,798,691,840	2,968.972675
18	687,194,767,360	742.2431688
19	2,748,779,069,440	185.5607922
20	10,995,116,277,760	46.39019805
21	43,980,465,111,040	11.59754951
22	175,921,860,444,160	2.899387378
23	703,687,441,776,640	0.724846845
24	2,814,749,767,106,560	0.181211711
25	11,258,999,068,426,200	0.045302928
26	45,035,996,273,705,000	0.011325732
27	180,143,985,094,820,000	0.002831433
28	720,575,940,379,279,000	0.000707858
29	2,882,303,761,517,120,000	0.000176965
30	11,529,215,046,068,500,000	0.000044241
31	46,116,860,184,274,000,000	0.000011060
32	184,467,440,737,096,000,000	0.000002765

Table A.1: Whole-Earth statistics about the first 32 resolutions of an icosahedral rhombus 2D DGGS.