ARTICLE

Utilizing a Discrete Global Grid System for Handling Point Clouds with Varying Locations, Times, and Levels of Detail

Neeraj Sirdeshmukh
MSc Geomatics / Delft University of Technology / Delft / Zuid-Holland / Netherlands

Edward Verbree
Section GIS Technology / Delft University of Technology / Delft / Zuid-Holland / Netherlands

Peter Van Oosterom
Section GIS Technology / Delft University of Technology / Delft / Zuid-Holland / Netherlands

Stella Psomadaki
Regional Innovation Centre Europe / Fugro / Leidschendam / Zuid-Holland / Netherlands

Martin Kodde
Ingenieursbureau Geodelta B.V. / Delft / Zuid-Holland / Netherlands

ABSTRACT

Discrete global grid systems (DGGSs) have emerged in recent years as a new specification for working with global heterogeneous data sets in a Digital Earth framework. Point clouds originating from different sources usually have varying initial characteristics. This research aims to analyze to what extent a DGGS can be used to handle point clouds having varying coordinate systems, acquired at different levels of detail (densities), and at different times in the creation of a global map of point clouds. DGGSs, which are currently limited to a 2D (surface) space, are extended into 3D and 4D spaces to fully harness the multidimensional nature of point clouds. A continuous spatial indexing strategy, based on a space-filling curve, is then developed on an ellipsoidal model of the Earth and used to efficiently cluster and retrieve DGGS-based point clouds stored in a database. Finally, the queried points are visualized in a Web browser. The hierarchical, multi-resolution nature of a DGGS is exploited to achieve a variable-scale smooth-zoom visualization. The challenges and opportunities of point cloud integration in a DGGS are presented.

Keywords: DGGS, point cloud, database, indexing, LIDAR, projection, integration, visualization

RÉSUMÉ

Les systèmes de grilles globales discrètes (*discrete global grid systems* — DGGS) ont récemment fait leur apparition comme nouvel outil technique appliqué au traitement d'ensembles de données mondiales hétérogènes dans le cadre d'un modèle de planète virtuelle. Des nuages de points provenant de différentes sources ont habituellement des caractéristiques initiales variables. Les auteurs ont pour but d'analyser dans quelle mesure un DGGS peut être utilisé pour traiter des nuages de points dont les systèmes de coordonnées varient et qui se présentent selon différents niveaux de détail (densités) et à différentes étapes de la création d'une carte globale de nuages de points. Ils déploient la représentation spatiale des DGGS, actuellement limitée à deux dimensions (surface), sur trois et quatre dimensions pour bien saisir le caractère multidimensionnel des nuages de points. Une stratégie d'indexation spatiale continue, reposant sur une courbe de remplissage spatial, est ensuite élaborée sur un modèle ellipsoïdal de la Terre et utilisée pour procéder de manière efficace à l'agglomération et à la consultation des nuages de points, issus du DGGS, stockés dans une base de données. Pour finir, les points consultés sont visualisés à l'aide d'un navigateur Web. Les auteurs exploitent les propriétés du DGGS — hiérarchisation et multirésolution — afin d'obtenir une visualisation zoom lisse à diverses échelles. Ils recensent les difficultés que suppose l'intégration des nuages de points dans un DGGS et les possibilités qu'elle offre.

Mots clés : base de données, DGGS, indexation, intégration, nuage de points, LIDAR, projection, visualisation

Introduction

The incorporation of massive point clouds into a global point cloud "map" necessitates the addressing of three of the most common challenges faced when trying to integrate point clouds coming from different origins. These challenges pertain to the aspects of location, time, and level of detail (LOD) of point clouds (Verbree, Kodde, and Van Oosterom 2017). The first aspect pertains to point cloud geometry. Point clouds from different regions of the world will likely be in distinct reference systems, each of which may have its own unique origin, orientation, and scale. A problem arises in trying to integrate such heterogeneous data sets. Second, point clouds coming from different sources could have varying LODs; a point cloud collected using a terrestrial laser scanner could have a point density of 1000 points/square metre, whereas another point cloud of the same area but collected using an airborne scanner could have a much lower point density of 10 points/square metre. If one naively combines such point clouds of the same area with significantly varying LODs into one data set, sharp jumps between the densities will be seen at the boundaries between the two point clouds during visualization (Van Oosterom and others 2015). This is yet another problem that needs to be addressed. Finally, the time component is invaluable if point clouds of the same area but acquired at two different times need to be analyzed for the purposes of change identification. 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, a common underlying data structure would be ideal. A discrete global grid system (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 and others 2017). A DGGS provides rapid spatial data integration, storage, and analytics at scales from local to global in a single consistent framework. Although at present DGGSs are limited to 2D spaces (surfaces), we have extended them into 3D and 4D for analysis of multi-dimensional data such as point clouds. Therefore, a DGGS-based approach to handling massive global point clouds is investigated in this research.

The main research question addressed is the following:

> *To what extent can a DGGS be used to handle point clouds with varying locations, times, and densities/levels of detail?*

Relevant subquestions include how a variable-scale structure can be implemented to support smooth zoom in DGGS and how a DGGS can be constructed to store the time component associated with spatiotemporal point clouds for their analysis.

The next section provides a theoretical background for understanding the rest of this research and describes the 2D DGGS utilized. The following sections introduce the construction of 3D and 4D DGGSs and elucidate how a higher-dimensional DGGS was used for indexing, clustering, and querying point clouds in this research. The final section provides conclusions.

Background to DGGSs

DGGSs are polyhedral reference systems on the surface of a base polyhedron's circumscribed ellipsoid (Purss and others 2017) and consist of a base polyhedron, a polyhedron orientation, a subdivision shape, a refinement ratio, and an inverse projection method. The faces of a polyhedron are subdivided into a series of finer-resolution cells, which are then projected using an inverse equal-area projection onto the surface of an ellipsoid that represents the Earth. The icosahedron was chosen because it has smaller face sizes than all of the other Platonic solids, and therefore the resulting shape and area distortions when it is mapped to the ellipsoid are minimized. A classic pole-aligned orientation provides ease of use and computations. A variety of shapes can be used to subdivide the planar faces of the icosahedron into cells of finer resolution, including the hexagon, pentagon, rhombus, and triangle. With hexagons, a significant disadvantage lies in the fact that they are incongruent: it is impossible to decompose a parent hexagon perfectly into smaller child hexagons, complicating the implementation of a hierarchical indexing scheme. Triangles suffer from nonuniform orientation at successive resolutions. Squares, although they have a simple geometry, cannot be used on triangle-faced polyhedrons such as the icosahedron (Sahr, White, and Kimerling 2003), which are better approximations of the Earth than other polyhedrons such as the cube. Rhombuses have several advantages. The geometry of a rhombus is simpler than that of a triangle or a hexagon (Bai, Zhao, and Chen 2005). Rhombuses also have the same size, shape, and orientation at each level in a DGGS (White 2000). Quadrant-recursive orderings, such as Morton or Hilbert space-filling curves (SFCs), can be used to index and cluster the cells of a rhombus-based DGGS. The rhombus hierarchy is nested, so that each parent rhombus completely contains all of its four child rhombuses (White 2000), facilitating the implementation of a hierarchical indexing approach. The icosahedral Snyder equal-area (ISEA) inverse projection can be used to inverse-project the cells created on the polyhedron onto the ellipsoid. The cells on the polyhedron are all of equal area; thus after they are mapped to an ellipsoid using this projection, their areas remain equal. Every cell represents an equal portion of the Earth's surface, and has an equal probability of contributing to an analysis.

An ellipsoidal SFC provides a 1–1 mapping from a one-dimensional location on the curve to a point on the ellipsoid (Bartholdi and Goldsman 2001) and vice versa. The SFC to be used in this research is the Morton curve,

particularly because the cells of a 2D rhombus DGGS form a quadtree and it is quadrant-recursive. Quadtrees are tree data structures in which, at each level, each internal node forks into four children nodes (Ottoson and Hauska 2002), and they greatly simplify the implementation of hierarchical indexing algorithms on spatial data. Not all shapes can be indexed as a quadtree in a DGGS; for example, pentagons and hexagons do not lend themselves easily to hierarchy-based quadtree algorithms. A host of other benefits are offered. Using a Morton curve, the cells can be uniquely indexed in a hierarchical and space-filling manner; not all of the SFCs can be redefined as hierarchy-based indexing systems. Each cell can be uniquely identified using its ellipsoidal Morton SFC code. The congruent, nested nature of a rhombus DGGS allows a path-based ordering, in which each additional digit in the Morton code indicates the child of the parent rhombus in which an observation is located, thereby simplifying hierarchical algorithms. Path addresses provide several advantages: the lengths of the address indicate the discrete resolution and precision level in a DGGS, eliminating the need to store the precision separately as metadata; they do not store redundant information (Sahr 2008), and they can reduce data size (Bartholdi and Goldsman 2001), as 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). Consecutive numbers in the complete path address identify subregions within the parent rhombus, and thus all observations (points) that are not located within the corresponding child rhombus can be ignored instantaneously in a spatial query, thereby significantly speeding query execution time. Each path address location code for a rhombus also automatically encodes the number of cells in the whole-Earth discrete global grid and the spacing between cells at its respective resolution. Path addressing is favoured over pyramid addressing, which assigns a unique location code to a cell only within its containing single-resolution discrete global grid. There is no hierarchical structure available and there is a lot of data redundancy, as the location of the same point is stored once for every resolution of a DGGS. Therefore, relationships between cells at different levels of the hierarchy cannot be known based only upon a single pyramid address.

A SFC ordering is said to be continuous when cells with sequential indices on the curve share at least 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 a continuous ordering, where all of the cell centroids at any one resolution are connected on the SFC without any discontinuities. 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, and 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) – and is also adjacency-preserving, because the relationships with neighbouring rhombus cells can easily be found using their Morton codes. The cells of a DGGS can be thought of as "buckets" to which data are assigned, and the area of the assigned cell as an indicator of the spatial uncertainty of an observation, turning a DGGS into a discrete precision-encoding system. Furthermore, this DGGS-based SFC encoding can be extended into n dimensions so that not only the spatial but also the temporal dimension is integrated into a unique cell ID. Implementing a proper indexing approach turns a DGGS into a spatial database that can be queried and analyzed efficiently.

Real-world observations will move across a DGGS over a course of time due to geophysical forces such as tectonic motion. Given that a DGGS is an Earth-centred, Earth-fixed (ECEF) system, the cells are fixed onto the surface of the ellipsoid that represents the Earth, once the orientation of the polyhedron is defined. One can think of them as lying in an immobile position "below" the mobile tectonic plates above. This is ideal for using a DGGS to identify changes between two moments in time, as static cells are queried for changes.

The ISEA forward projection, on which the icosahedral rhombus DGGS is based, can be used to generate a unique ID for each cell at every resolution on the ellipsoid. The projection outputs coordinates on a flattened icosahedron given coordinates on an ellipsoid. The 20 triangular faces of the icosahedron are flattened onto a plane. Neighbouring faces of the flattened icosahedron (the triangles) are connected to form rhombuses, and these are then recursively subdivided into finer and finer cells until the desired resolution in the DGGS hierarchy is reached. Forward and inverse formulas for the ISEA projection are well documented (Snyder 1992). These can be used to convert latitude and longitude coordinates into coordinates on the projection. As the planar rhombuses are subdivided, a set of discrete resolutions is generated. In this way, the entire surface of the Earth can be partitioned into rhombus cells. Every parent rhombus has four smaller child rhombuses, thereby discretizing the Earth into a set of linear quadtrees on top of which Morton indexing can be applied.

As there are 10 total rhombuses, each can be assigned a label number from 0 to 9 that indicates its location on the Earth. To maintain a continuous ordering (with no sharp jumps in the IDs), these numbers have been assigned as shown in Figure 1. Then, to generate each cell at every

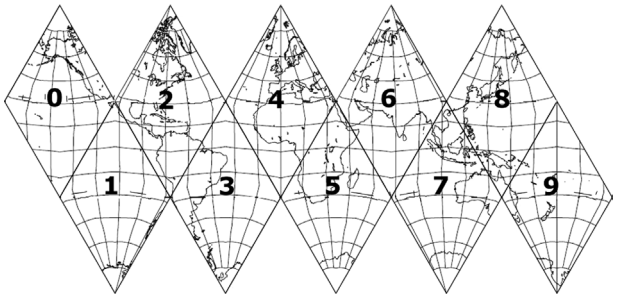

Figure 1. The continuous labelling of the rhombus faces of an icosahedron; the icosahedron is flattened out onto a plane

resolution within a face, each rhombus is subdivided into finer and finer cells, and there is a separate SFC at each resolution of a DGGS, as shown in Figure 2. It is important to state that all SFCs are based on hypercubes (n-dimensional analogues of squares, $n = 2$, and cubes, $n = 3$), and there is a separate hypercube on every rhombus face of the icosahedron. 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.

The SFC traverses the cells at every resolution on the flattened icosahedron, following a zigzag path indicated by Figure 3. No matter how a DGGS is constructed, either on a polyhedron or on an ellipsoid, a SFC will traverse the cells at a given resolution following a similar path. 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. Therefore, all points on any one face can easily be retrieved by selecting all Morton codes having the same first digit. The length of the code indicates the resolution of a DGGS in which a point is contained,

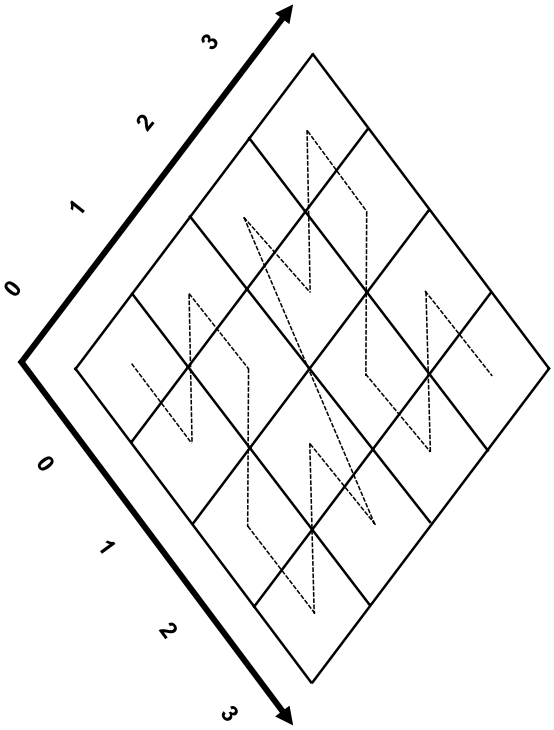

Figure 3. Construction of a DGGS at resolution 2, with four rhombuses on every side within the parent rhombus
Note: A Morton SFC traverses all of the cells at each resolution in a zigzag order. Its hierarchical, quadrant-recursive nature provides many beneficial properties.

and also an assigned 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.

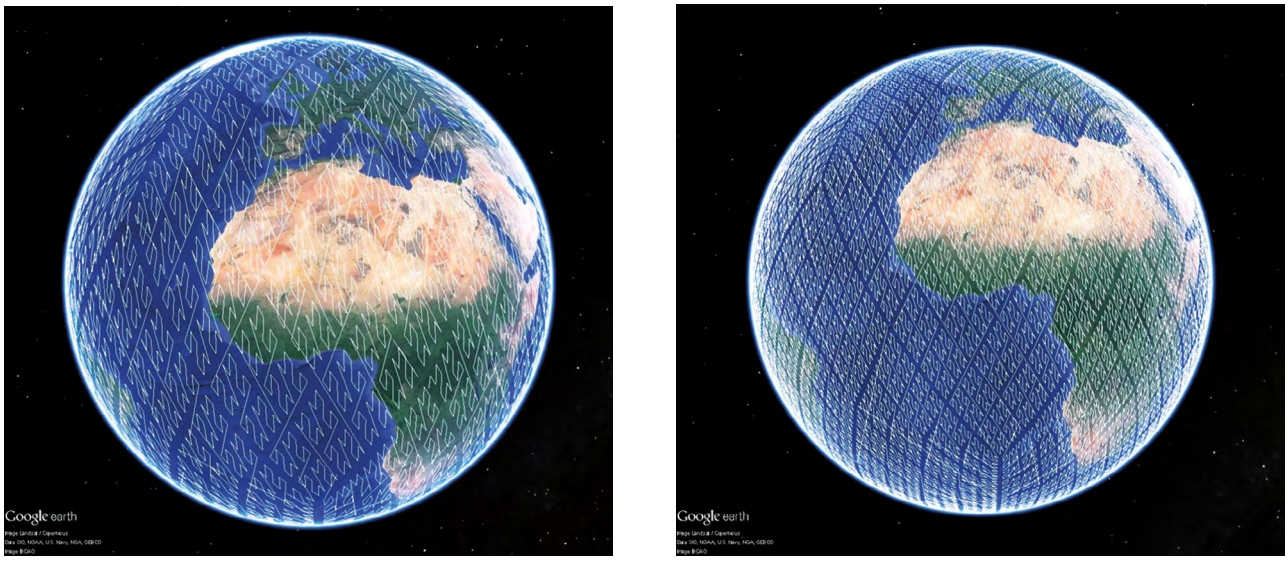

Figure 2. Ellipsoidal SFCs at resolutions 5 (left) and 6 (right) of an icosahedral rhombus 2D DGGS
Source: © 2019 Google Earth; data from SIO, NOAA, U.S. Navy, NGA, GEBCO.

*n*D Point Clouds

A point in a point cloud can not only have a unique *X*, *Y*, *Z*, and time value, but also be assigned an additional dimension that indicates its importance, that is, its significance as compared with other points. A variety of methods can be used to compute this importance value, including a random function (Van Oosterom and others 2015), minimum distance to the nearest point, or distance from the data collection platform (i.e., the scanner exit point where the pulse originated). This added dimension can then be used to visualize point clouds that have varying LODs efficiently in a smooth, continuous manner. The hierarchical nature of a DGGS can be utilized to create such a variable-scale representation of massive numbers of points. A highly regular global grid should document the precision and location of spatial data on the globe (Kimerling and others 1999). Every point in a point cloud has a spatial uncertainty that can be represented as an area around that point. A DGGSs 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. The smaller a DGGS cell, the higher the precision, because the smaller cell area indicates a low spatial uncertainty. Conversely, the larger the size of a DGGS cell, the lower the precision and the higher the spatial uncertainty of an assigned point. By assigning a point observation to a DGGS cell that has approximately the same area as the precision of that point, computing its importance, and visualizing the points based upon this continuous dimension, a true variable-scale visualization can be created. That is, the points are not stored for any pre-defined scales (the discrete levels of a DGGS), but can rather provide levels of detail for an entire scale range, at any scale (Meijers 2011).

To compute point precisions, a proper scientific approach is needed. A laser beam signal scatters more the further away it is from the source of emission (i.e., the scanner exit point). It occupies a cone-shaped volumetric region between the scanner and the target. The amount of dispersion is indicated by the beam divergence angle, expressed in milliradians. The spatial uncertainty of each point is indicated by the size of the laser beam footprint at that point, which can be inferred by examining the distance to the scanner; the greater this distance, the larger the footprint due to beam divergence, and the lower the precision of the point. By calculating the beam footprint size at each point in a point cloud, point precision can be determined. Then the point can be assigned to a DGGS cell covering the same location as the point and that has approximately the same area as the point's precision. 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 range (distance) from the scanner. A higher-precision point is contained within a lower precision point, and the containment is a property of their cell IDs. The added continuous dimension can be used to implement a perspective-view query, in which only the higher importance points are shown far from the observer position, and both higher- and lower-importance points are displayed close to the observer (Van Oosterom and others 2015). Points that 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 at higher levels of the structure.

At present, the OGC DGGS abstract specification is 2D on the surface of the globe. The cells are not volumetric; they do not inherently allow the storage and representation of 3Dvalues, such as those that include elevations, and much less 4D ones, such as those that include time. However, point clouds are multi-dimensional and it would be ideal to encode this information into a spatial location code also. This speeds up the processing of spatiotemporal queries on the data, as the query does not have to perform yet another sequential search on the third or fourth dimension attributes after narrowing the database records based on two attributes. The third dimension is assumed to be the distance of a point above or below the Earth's surface and the fourth to be the time at which a point was captured. In this research, for every point, the spatial dimensions (*X*, *Y*, and *Z*) were integrated into a single 3D DGGS code and an attempt was made to construct a 3D DGGS tessellation. It has also been demonstrated how, in addition to the spatial dimensions, time (*T*) can also be incorporated into the DGGS, turning it into a 4D structure.

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. It would be invaluable if DGGS could be extended into higher dimensions to take advantage of the multidimensional nature of point clouds.

The 2D cells can be extended into 3D either on the base polyhedron or directly on the sphere/ellipsoid. This is ideal because DGGSs, although usually constructed using a polyhedron, can also be built on the sphere/ellipsoid directly. 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 an SFC is created that traverses all of the cells of every hypercube. Then the cell vertices are inverse-projected onto the Earth model using an equal-area projection, and the cells appear on a spherical Earth as shown in Figure 4. The altitude above the face of the polyhedron (i.e., above the plane of the DGGS projection) is a scaled version of the altitude above the surface of the Earth model. If a DGGS is built on a spherical Earth directly, the

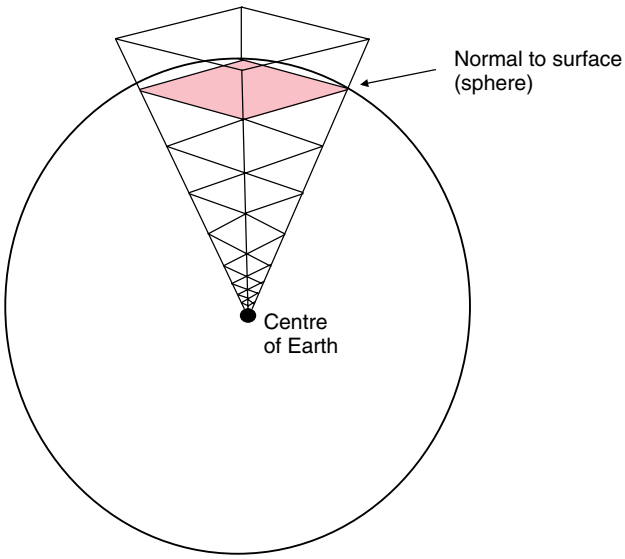

Figure 4. Visualization of how a 3D DGGS would appear on a sphere
Note: Only a single resolution is shown. The red cell is a 2D cell on the surface of the Earth and is provided for reference.

2D cells can be extended into 3D by computing the normal to the surface of the Earth at each of their vertices, which also automatically passes through the Earth's centre. For an ellipsoid, however, this is not always true. The key idea is to connect each of the vertices of each of the cells of a DGGS to the centre of the Earth model (a sphere or an ellipsoid) in an earth-centred system.

Unfortunately, 3D cells at any one resolution in the hierarchy will not be of the exact same volume, as the amount of space expands outward from the Earth's centre. The volume difference varies with respect to the location on the Earth and the Earth model (ellipsoid, sphere, etc.) and/or base polyhedron chosen to construct the DGGS. For point clouds, however, an elevation range of ±5000 m is sufficient, as even high-altitude airborne LIDAR acquisition surveys are usually performed at around 3000 m (Hopkinson 2007). In this elevation range, the volumes of the cells are roughly identical and the differences in volume can be ignored. Therefore, for all practical purposes, equal-volume cells are being used when it comes to handling point clouds. If the elevation range extends below the Earth's surface, negative elevation values are also automatically encoded, invaluable for bathymetric point clouds: the range of elevation values (maximum – minimum), always a positive number, is computed and hierarchically split into four smaller elevation ranges at every resolution of an aperture-4 DGGS.

Although the faces of the 3D cells are not planar, in practice one can think of them in terms of simple primitives such as 3D planes. Just as a parent cell is subdivided into four children on the flat 2D icosahedral plane, a volumetric 3D cell can be subdivided into eight children by splitting the 3D hypercube on the polyhedron by three splitting planes. A 3D SFC can be defined that traverses all of the cells at each resolution in a hierarchical manner, thus allowing the storage and representation of the 3D location in space of the point that was generated by the laser pulse. Furthermore, this approach allows cells of *any* shape and on the surface of *any* polyhedron to be extended into 3D in the same manner.

This approach of extending a 2D DGGS into 3D works for point clouds in any region of the Earth, both above or below the surface, as a base polyhedron or ellipsoid representing Earth is used as a model on top of which to extend a DGGS. A two-dimensional cell on the surface of the Earth becomes a three-dimensional volume. The 3D volumetric cells are bounded by the surface of the Earth; the 3D cells at ground level have the curved surface as one of their sides, and are "stacked" in layers on top of one another, extending upward or downward from the surface. As points in a point cloud also have elevation values, they can be "assigned" to the 3D cells in which they lie based upon these values, either directly or based upon their spatial uncertainties. Just as they can be integrated into a cell-based common coordinate system at various LODs and how the cells of a 2D DGGS can be used to study change over time, points can also be integrated into a common cell-based coordinate system and change detection can be performed in 3D. Once defined, the cells are fixed in 3D space and do not move. This provides the ability to answer one of the three most fundamental questions of spatial analytics, "How has it changed?" referring to how a spatial phenomenon might have changed over time within a 3D cell or set of cells. A higher-precision 3D cell is contained within a lower-precision 3D cell, and the containment is a property of their cell IDs along the SFC. The hierarchical nature of a DGGS is preserved, and utilizing a SFC through the cells allows efficient indexing and clustering of point data. Using a DGGS makes it unnecessary to use separate map projections, data, or coordinate systems for every region of the Earth, thus readily facilitating integration of point clouds with respect to locations, times, and LODs.

4D DGGS

All points in a point cloud are captured at a specific moment in time, and this additional dimension can also be encoded into a DGGS, turning it into a 4D cell structure that is fine enough in temporal resolution to encode not only the date but also the exact observation time of each point. The fineness in time (hours, minutes, or seconds) at which to encode the points can be set as appropriate. In 4D DGGS, the behaviour of the temporal dimension is similar to the behaviour of the spatial dimensions, in which at each resolution the range of time is hierarchically split into a certain number of finer parts. Just as space is subdivided into a hierarchy of nested spatial areas/volumes

in 2D/3D DGGS, in 4D DGGS time is discretized as a hierarchy of nested temporal ranges. This means that although time is inherently a continuous variable, it is represented as a series of discrete ranges in which, in each range, it takes only one value. For example, a large temporal range is subdivided into exactly four smaller temporal ranges at every resolution in any aperture-4 DGGS. With other apertures, the refinement ratio would be different. A 4D SFC will traverse all four dimensions and allow efficient indexing, clustering, and querying of 4D point cloud data. Each value taken by a discrete range in a 4D DGGS is a 4D SFC code, and this value is consistent throughout the range. Before an observation's attributes are encoded into an SFC, all dimensions should be scaled to a common range. For example, if X takes the range 1–1000, then Y, Z, and T should also take the range 1–1000. In most cases, X and Y will refer to longitude and latitude, respectively, and their values will come from the same domain and have similar units (degrees), whereas Z (likely metres) and T (minutes or seconds) will come from different domains and units. Scaling every dimension's values to a common unitless range eliminates any problems with units and ensures that the SFC traverses the hypercube in equal increments in all dimensions. For those dimensions without a clear start and end value, hypothetical values applicable to the task at hand can be chosen; for example, a domain from −5000 to 5000 m is sufficient for Z and the current time can be chosen as the end value for T. Then these scaled values are converted into binary form with the same number of bits as the DGGS resolution whose code needs to be computed. For example, if a code at resolution 8 needs to be found, then the scaled values for all dimensions should be converted into an 8-bit binary form. Then the following formula can be used to generate a four-dimensional hierarchical Morton code (MC) for a point:

$$\mathrm{MC} = 8 \times (T_{\mathrm{bin}}) + 4 \times (Z_{\mathrm{bin}}) + 2 \times (Y_{\mathrm{bin}}) + (X_{\mathrm{bin}}).$$

The 2D, 3D, and 4D Morton codes can also be decoded into their original values using a reverse encoding procedure.

Any general time-varying geospatial data set, and not only a point cloud, can be encoded into a 4D DGGS cell structure using the above method. The fixed set of cells of a 4D DGGS can then be used to perform change analysis on these data. The benefit of a time-discretized approach to change analysis is that the analysis can be performed at any arbitrary discrete LOD, the resolutions of a DGGS. Thus, the user is allowed much flexibility in choosing a resolution that is most suitable for the application at hand. For example, change analysis can be performed at resolution 8 (coarse), 17, or 32 (fine). As time has been discretized and each separate range of time has the same value (i.e., the SFC code), all of the points assigned to that range share the same SFC code, and they can be quickly filtered and processed using these codes. One might argue that discretizing time could lead to a loss of precision, because two points belonging to the same range that were captured at different times can still share the same DGGS SFC code. However, the true power of a DGGS-discretized code is that any sufficiently fine resolution can be chosen to encode time, and as resolution increases so does precision. Therefore, the same two points could fall into different time ranges at a higher level of resolution/precision.

Implementation and Results

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 RAM, and a 64-bit operating system. The data sets used include two mobile LIDAR scans of the Forepark area in the Hague, the Netherlands, one taken in 2010 and the other in 2016. The area of the Forepark is approximately 0.83 square kilometres. The 2010 scan contains 162 million points, and the 2016 one contains 198 million. As there was only one laser scanner at the time, the first scan had 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 two different moments in time and can also be used for the purpose of variable-scale visualization.

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 point clouds are captured is the Dutch RD system, with units of metres. Each point is stamped with its acquisition time, given in GPS time. Furthermore, the trajectory of the car that collected the points is known, along with the time when the car was at each point on the trajectory. The scanner-point distances can be computed based on matching the time component, and using the known direct linear relationship between the distance from the scanner's exit aperture and the size of the beam footprint, the footprint diameter can be calculated at the exact location where the beam hit the point. This gives an indication of the "true" location of the point (i.e., the accuracy), which could be anywhere within the "block" of spatial uncertainty given by the footprint; furthermore, any repeated measurements of the same point will likely fall somewhere in the same area (i.e., the precision).

With some testing, it can be determined that tiling the entire point cloud into smaller chunks for subsequent processing yields significantly shorter execution times than operating on the entire point cloud as one LAS file. A tile size of 5000 points per tile was found to be optimal. 2D and 3D Morton codes were computed for every point. Only the full-resolution codes were computed, which originate from the whole Earth.

Point clouds are usually massive datasets containing millions or billions of points. Therefore, the operations that are performed on point clouds need to be scalable in order

not to fail or deliver long response times for large point clouds. A DGGS is ultimately a large spatial, or in this case spatiotemporal, database. Moreover, a DGGS can be used not only for handling point clouds, but also for incorporating vector and raster data sources into a single framework. If users want to combine different types of data in their queries, a standardized and generic database solution is preferable to file-based solutions (Van Oosterom and others 2014). Therefore, it is essential to utilize a database for the storage and querying of DGGS data. A method that allows the shortest insertion times possible is needed, and this can be achieved using bulk loading instead of loading a single point in every insertion query. 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. The 2D and 3D Morton codes, along with the precision, are stored for every point. The bulk loading is performed in memory and the observed loading time is only 6 s per 1,000,000 points. This means that the loading time for the entire 2010 dataset (of 162 million points) would be only 17 min. This is currently the fastest existing implementation, given the limitations of Psycopg2, Python, PostgreSQL, and the machine on which the loading has been tested.

The full-resolution (resolution 32, millimetre-level precision) Morton codes can be stored, where the cells are so very small that they can essentially be considered as points. Storing coarser resolution codes does not allow unique identification of and access to the spatial and temporal properties of each point, unless information regarding the differences between the discrete and continuous representations of space and time is stored as additional attributes for every point. As resolution increases, the Morton code slowly converges to the true latitude, longitude, altitude, and time values of a point, and the cells converge to infinitesimally small points. Therefore, although these are all continuous variables, storing a single full-resolution code allows 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. Table 1 shows an example of convergence of all dimensions to their true values in a 4D DGGS.

An example of a point observation and its 4D Morton code is as follows:

Latitude: 36° N

Longitude: 25° E

Elevation: 44 m (WGS84)

Time: February 1, 2008 12 AM UTC = 885,859,218 s GPS time

4D Morton code (hexadecimal): 4F38AAA1D2369908 1A69D5B67D79A2928B

Table 1 illustrates the convergence of this 4D Morton code to the original values of its 4 dimensions as full-resolution is approached.

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 to 7, and with 4D DGGS this requires 4 bits per level, with values from 0 to F (with two groups of hexadecimal digits, 10 with hex numbers from 0 to 9 and 6 with hex numbers from A to 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. The maximum number of bits needed at resolution k is $4 + (n \times k)$ for n dimensions, whereas the maximum number of digits is $k + 1$ for 2D, 3D, and 4D (if hexadecimal is used). Unfortunately, as an initial icosahedral rhombus tessellation has 10 faces, storing the face number itself necessitates up to 4 bits of storage, thereby wasting six 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.

Once the points are stored in separate flat tables, a B-tree index can be created on the 3D Morton code to allow faster retrieval. Making an index on the full-resolution Morton code, however, does not make much sense, because the cells at resolution 32 are so very small that they reduce to a 0D point geometry. Querying these "fake" cells at such

Table 1. The convergence of a decoded 4D DGGS Morton code to the true values of all dimensions for a hypothetical point observation as full resolution is approached

Resolution	Latitude	Longitude	Elevation (m)	GPS time (s)	Equivalent UTC
2	30.2809	18.7822	0.0	599,616,027	Jan 6, 1999 00:00:09 a.m.
5	35.7917	23.7038	0.0	861,948,031	Apr 30, 2007 06:00:13 a.m.
10	35.9975	24.9653	39.0625	885,370,531	Jan 26, 2008 08:15:13 a.m.
15	36.0006	24.9989	43.9453	885,846,301	Jan 31, 2008 20:24:43 p.m.
25	36.0000	24.9999	43.9999	885,859,203	Jan 31, 2008 11:59:45 p.m.
32	36.0000	25.0000	44.0000	885,859,218	Feb 1, 2008 00:00:00 p.m.

a high resolution would essentially mean that we were querying the points directly. In DGGS, however, the *cell* is the fundamental unit of spatial analytics, and the operations that are performed using a DGGS 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 34 digits long, for the purpose of querying an index can be added on only a truncated full-resolution Morton code. This can be an arbitrarily defined number of digits long and is application-specific and dependent on the resolution 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 (as the length of the Morton code indicates the resolution). A resolution of 17 was used, at which the cells have a mean inter-cell spacing of 53 m and an approximate area of 2969 square metres. This was chosen 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 (i.e., including one digit for the face number). The B-tree index creation took approximately 22 min for a table of 162 million points, and only slightly longer for the 2016 table. Storing the Morton codes as text allows easy truncation and concatenation operations. Next, the flat tables were also clustered based upon the substring and not the full-resolution Morton code. This physically reorders the data in the table based on the index information, so that the number of disk pages to be fetched is reduced. This entire process results in two separate index-organized tables (IOTs), or tables with clustered indexes. Once an index has been created on a truncated Morton code, spatiotemporal queries can be performed at the particular level to which the codes have been truncated. Coarser-resolution cells can be used as filters 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 intersected. This can be speeded up by adding an R-tree index on the cell geometries so that only bounding boxes are compared and not geometries. However, the comparison of cell geometries, much less their construction and storage in a database, is a costly procedure, and can be avoided completely. Moreover, an SFC-encoded DGGS 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 in this research was performed in 3D using Python, is illustrated in Figure 5, and is as follows:

(1) A query region is defined by minimum and maximum latitude, longitude, and altitude coordinates on the WGS84 ellipsoid.

(2) An equally spaced grid of points in latitude/longitude (2D) and/or altitude (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 chosen to be not so small as to result in too many points, but also not so large 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. As there can be more than one point inside a single cell in the created grid, duplicate codes are removed.

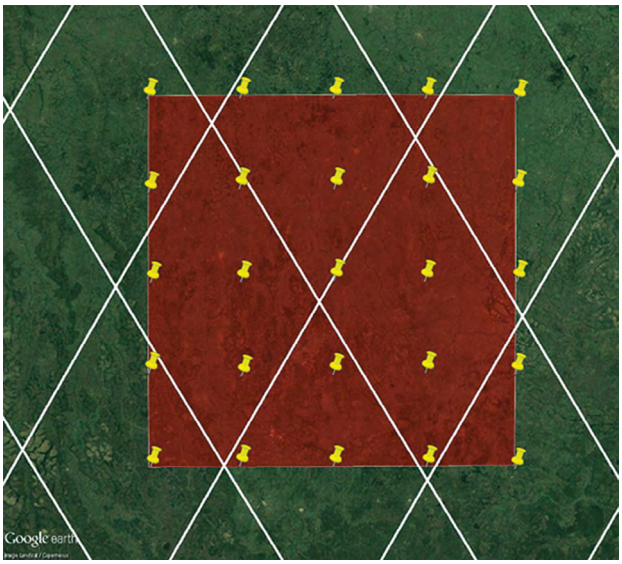

Figure 5. A sample query procedure for a DGGS-based point cloud (shown in 2D)
Note: A query region with minimum and maximum latitude, longitude, and altitude coordinates on the ellipsoid is created, and a regularly spaced grid of points (in degrees and meters) is created that intersects all of the cells inside the query region. Then the SFC codes of these points are found, duplicate codes are removed, and points having the same Morton codes as these are retrieved from a database where *n*D DGGS-encoded point clouds are stored. If faster retrieval is desired, it can be performed at a coarser resolution.
Source: © 2019 Google Earth; image from Landsat/Copernicus.

(4) All the points having the same Morton codes as these cells are then retrieved from the database at a coarser resolution. This retrieval is relatively fast, as the flat tables have been clustered and indexed with a B-tree on a truncated Morton code. Full-resolution codes are retrieved, but only using a decimated version of the codes. The attributes retrieved include the 3D Morton code and the continuous precision (used for variable-scale visualization).

(5) The retrieved full-resolution Morton codes are decoded into their latitude, longitude, and altitude values on the WGS84 ellipsoid for visualization.

This yields a set of all of the points falling into all the cells that intersect the query region. If an exact match is desired, then it is a simple matter of identifying the subset of points falling perfectly within the query region through comparison of the latitude, longitude, and altitude values of the retrieved points with the minimum and maximum latitude, longitude, and altitude 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 numeric values. However, this additional step is application-dependent and might not be required at all times. The above query procedure will work only if the query region is a latitude-, longitude-, and/or altitude-aligned 2D/3D polygon/volume.

DGGSs' hierarchical, multi-resolution nature is ideal for analyzing spatiotemporal phenomena at arbitrary discrete LODs. An example of change analysis between two moments in time pertains to point density. A particular single DGGS resolution (for example, 17) can be chosen (in *n*D), and the point density of every cell can be computed by dividing the total number of points that fall into a cell by the cell's area. The areas of the cells are fixed and known beforehand for every resolution. The cells can then be symbolized based upon their changes in point densities, yielding a visual spatial indication of change as shown in Figure 6. As the cells are equal-area, this spatial analysis is more meaningful than using non-equal-area cells such as those formed by the latitude/longitude graticule. 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 using a simple truncation/substring operation on the full-resolution codes to acquire the Morton code only up to a certain length (i.e., resolution). The truncated code indicates that the point could be located anywhere within the *n*D DGGS cell that shares the same code. If all of the points are truncated to the same length, the point density within each cell at that resolution can be 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.

A prototype Web viewer for point clouds was developed using the Cesium frontend JavaScript API along with Node.JS, allowing execution of JavaScript code server-side.

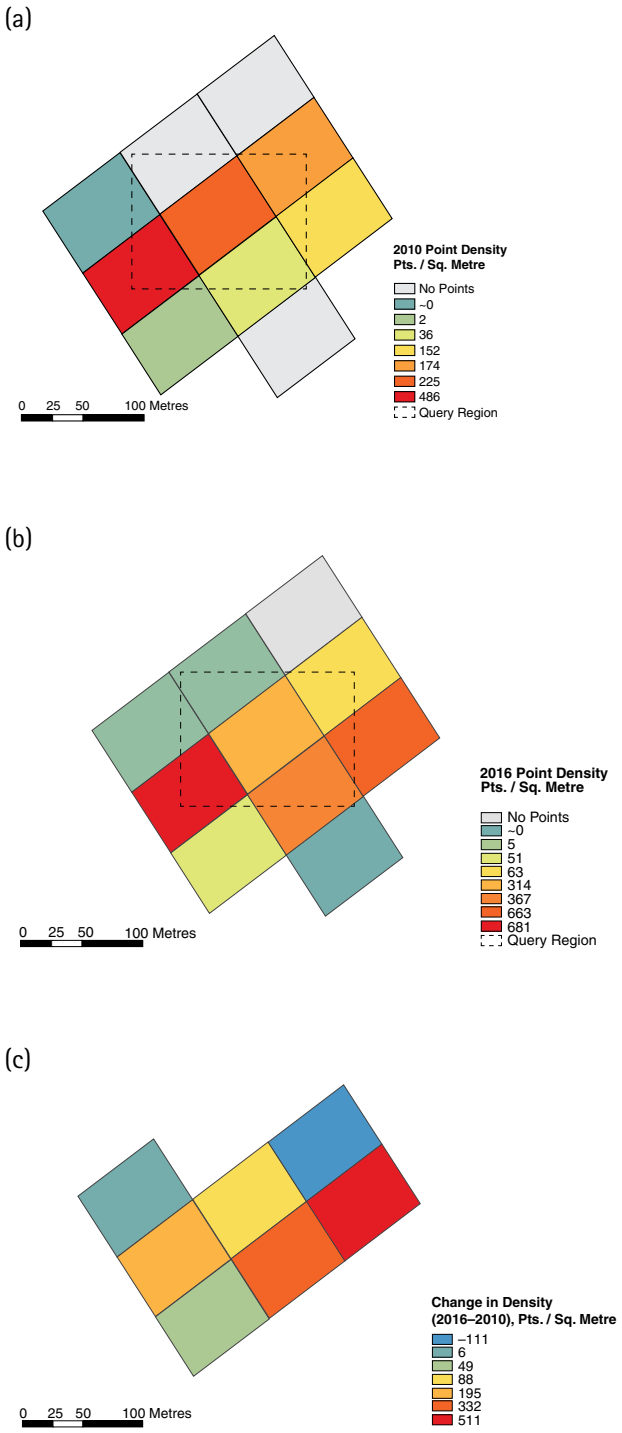

Figure 6. Example of a change analysis conducted on the cells of a 2D DGGS where the cells are fixed on the Earth's surface once the orientation of the DGGS is defined: (a) and (b) show point densities in a small subset of the study area in the years 2010 and 2016, respectively; (c) shows the difference in point densities between the two years *Note:* Only cells containing points in both data sets are shown. The use of equal-area cells makes such a spatial analysis more meaningful, as each cell has an equal probability of contributing to the analysis.

The user can draw a 3D bounding box in WGS84 coordinates and make a request to the server to retrieve all of the DGGS-encoded points from the database that are physically contained within the box. As the user can draw any arbitrary box, the algorithm first aligns it to the latitude, longitude, altitude grid. It then sends these bounding coordinates to the server, which feeds them as arguments into a Python script. This connects to the PostgreSQL database server, where the points are stored as a flat table and queried on the fly.

The full-resolution Morton codes retrieved at the end of the query procedure are then decoded into their latitude, longitude, and altitude values (above or below the surface) on the WGS84 ellipsoid. The full-resolution code allows accessing the 3D position of a point at an extremely fine LOD. These points can then be used for Web visualization in the 3D tiles format. To allow smooth zoom in/out, the *geometric error* property of a 3D tile can be used, and pre-processing performed to put the high-importance points into tiles with lower geometric error and the low-importance points into tiles with higher geometric error. The points retrieved are sorted by importance (precision) in ascending order, grouped into tiles, and put into a tile set that is streamed to Cesium. The greater the number of tiles, the greater the smooth zoom effect.

Conclusion

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 LODs, and acquired at different times. Indubitably, DGGSs transcend map projections and can be used as a seamless common global reference frame to integrate data from different reference systems. Their static cell structure can be exploited to index, cluster, and query global point clouds and conduct change analysis. They allow encoding observations at any discrete level of detail without introducing additional error. Extending them into n dimensions allows fully harnessing the multi-dimensional nature of point clouds.

Author Information

Neeraj Sirdeshmukh is a GIS specialist and has an extensive background in geospatial science and remote sensing. He holds a master's degree in geomatics from the Delft University of Technology in the Netherlands and a master's and a bachelor's degree in geospatial information sciences from the University of Texas at Dallas in the United States. His professional and research interests include point clouds, global navigation satellite systems, spatial databases, geodesy, and big geospatial data analytics and he has been avidly passionate for geography since a young age. E-mail: sirneeraj@gmail.com.

Edward Verbree obtained his MSc in geodesy in 1992 at the Delft University of Technology. Since 1997 he has been employed as an assistant professor at the GIS Technology Department at the Delft University of Technology. His research field include tetrahedralizations, tessellations, surface reconstruction, explorative point clouds, location awareness, and indoor positioning. He is currently teaching the course GEO1003: Positioning and Location Awareness for the MSc in geomatics. E-mail: e.verbree@tudelft.nl.

Peter van Oosterom obtained an MSc in technical computer science in 1985 from the Delft University of Technology. In 1990, he received a PhD from Leiden University. From 1985 until 1995, he worked as a computer scientist at the TNO-FEL laboratory in the Hague. From 1995 until 2000, he was senior information manager at the Dutch Cadastre. Since 2000, he has been a professor at the Delft University of Technology and the head of the GIS Technology Section. His research interests include point cloud data management/dissemination, 3D cadastre/land administration standardization, and map generalization/varioscale data structures. E-mail: P.J.M.vanOosterom@tudelft.nl.

Stella Psomadaki is a software engineer at Fugro N.V. in the Netherlands. She holds an MSc in geomatics from the Delft University of Technology and a BSc in rural and surveying engineering from the National Technical University of Athens. Her work focuses on full-stack Web development of Web–GIS applications utilizing cloud solutions. E-mail: stpsomad@gmail.com.

Martin Kodde obtained an MSc in geomatic engineering from the Delft University of Technology in 2006. He has subsequently worked as a geodetic consultant and the head of innovation at a large international survey and geotechnical engineering company in the Netherlands. In 2018, he became the director of Geodelta, a small software and consulting firm in Delft, specializing in photogrammetry, laser scanning, and geodesy. E-mail: martin@geodelta.com.

References

Bai, J., X. Zhao, and J. Chen. 2005. "Indexing of the Discrete Global Grid Using Linear Quadtree." Paper read at the ISPRS Workshop on Service and Application of Spatial Data Infrastructure, 14–16 October, Hangzhou, China. 267–70.

Bartholdi, J.J., III, and P. Goldsman. 2001. "Continuous Indexing of Hierarchical Subdivisions of the Globe." *International Journal of Geographical Information Science* 15(6): 489–522. https://doi.org/10.1080/13658810110043603.

Dutton, G. 1996. "Encoding and Handling Geospatial Data with Hierarchical Triangular Meshes." Paper read at the 7th International Symposium On Spatial Data Handling, 12–16 August, Delft, Netherlands. 8B.15–8B.28.

Hopkinson, C. 2007. "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(4): 312–24. https://doi.org/10.5589/m07-029.

Kimerling, J., K. Sahr, D. White, and L. Song. 1999. "Comparing Geometrical Properties of Global Grids." *Cartography and Geographic Information Science* 26(4): 271–88. https://doi.org/10.1559/152304099782294186.

Meijers, M. 2011. "Variable-Scale Geo-information." PhD dissertation, Delft University Of Technology, Delft.

Ottoson, P., and H. Hauska. 2002. "Ellipsoidal Quadtrees for Indexing of Global Geographic Data." *International Journal of Geographical Information Science* 16(3): 213–26. https://doi.org/10.1080/13658810110095075.

Purss, M., R. Gibb, F. Samavati, P. Peterson, J. Rogers, J. Ben, and C. Dow. 2017. "Topic 21: Discrete Global Grid Systems Abstract Specification." Available at http://docs.opengeospatial.org/as/15-104r5/15-104r5.html.

Sahr, K. 2008. "Location Coding on Icosahedral Aperture 3 Hexagon Discrete Global Grids." *Computers, Environment and Urban Systems* 32(3): 174–87. https://doi.org/10.1016/j.compenvurbsys.2007.11.005.

Sahr, K., D. White, and A. Kimerling. 2003. "Geodesic Discrete Global Grid Systems." *Cartography and Geographic Information Science* 30(2): 121–34. https://doi.org/10.1559/152304003100011090.

Snyder, J.P. 1992. "An Equal-Area Map Projection for Polyhedral Globes." *Cartographica* 29(1): 10–21. https://doi.org/10.3138/27h7-8k88-4882-1752.

Van Oosterom, P., O. Martinez-Rubi, M. Ivanova, M. Horhammer, D. Geringer, S. Ravada, T. Tijssen, M. Kodde, and R. Goncalves. 2015. "Massive Point Cloud Data Management: Design, Implementation and Execution of a Point Cloud Benchmark." *Computers and Graphics* 49: 92–125. https://doi.org/10.1016/j.cag.2015.01.007.

Van Oosterom, P., S. Ravada, M. Horhammer, O. Martinez-Rubi, M. Ivanova, M. Kodde, and T. Tijssen. 2014. "Point Cloud Data Management." Paper read at the IQmulus Workshop on Processing Large Geospatial Data, 8 July, Cardiff, UK.

Verbree, E., M. Kodde, and P. Van Oosterom. 2017. "Open Point Cloud Map." Paper read at the OGC Technical And Planning Committee Meeting, 20–24 March, Delft, the Netherlands.

White, D. 2000. "Global Grids from Recursive Diamond Subdivisions of the Surface of an Octahedron or Icosahedron." *Environmental Monitoring and Assessment* 64: 93–103. https://doi.org/10.1023/A:1006407023786.