Heliyon

ELSEVIER

Received:
5 March 2017
Revised:
20 May 2017
Accepted:
15 June 2017

Cite as: Robert W. Youngren, Mikel D. Petty. A multi-resolution HEALPix data structure for spherically mapped point data. Heliyon 3 (2017) e00332. doi: 10.1016/j.heliyon.2017.e00332

A multi-resolution HEALPix data structure for spherically mapped point data

Robert W. Youngren [a,*], **Mikel D. Petty** [b]

[a] *Simulation Technologies, Inc., Huntsville, Alabama 35805 USA*

[b] *University of Alabama in Huntsville, Huntsville, Alabama 35899 USA*

* Corresponding author.

E-mail address: Robert.Youngren@simtechinc.com (R.W. Youngren).

Abstract

Data describing entities with locations that are points on a sphere are described as spherically mapped. Several data structures designed for spherically mapped data have been developed. One of them, known as Hierarchical Equal Area iso-Latitude Pixelization (HEALPix), partitions the sphere into twelve diamond-shaped equal-area base cells and then recursively subdivides each cell into four diamond-shaped subcells, continuing to the desired level of resolution. Twelve quadtrees, one associated with each base cell, store the data records associated with that cell and its subcells.

HEALPix has been used successfully for numerous applications, notably including cosmic microwave background data analysis. However, for applications involving sparse point data HEALPix has possible drawbacks, including inefficient memory utilization, overwriting of proximate points, and return of spurious points for certain queries.

A multi-resolution variant of HEALPix specifically optimized for sparse point data was developed. The new data structure allows different areas of the sphere to be subdivided at different levels of resolution. It combines HEALPix positive features with the advantages of multi-resolution, including reduced memory requirements and improved query performance.

An implementation of the new Multi-Resolution HEALPix (MRH) data structure was tested using spherically mapped data from four different scientific applications

http://dx.doi.org/10.1016/j.heliyon.2017.e00332

2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

(warhead fragmentation trajectories, weather station locations, galaxy locations, and synthetic locations). Four types of range queries were applied to each data structure for each dataset. Compared to HEALPix, MRH used two to four orders of magnitude less memory for the same data, and on average its queries executed 72% faster.

Keywords: Computer science

1. Introduction

Spherically mapped data describes objects or entities with locations that lie on or can be projected onto a real or conceptual sphere. Examples include geographic information system (GIS) data associated with geographic longitude and latitude locations and astronomical observation data associated with celestial right ascension and declination locations. Measurements in GIS and astronomical applications are often spherically mapped. The datasets considered in this work are point data, in that each data record includes a spherically mapped point location, but additional data may be present as well. For example, a data record describing an astronomical observation may include the celestial location as right ascension and declination as well as other information relating to the observation, such as redshift and apparent magnitude. This article will refer to the point itself, or to the data record associated with the point, as appropriate to the context.

A number of data structures specifically designed to store and access spherically mapped data have been developed. In general, they partition the surface of the sphere into "cells" (also known as "pixels") and associate the data records with the cells that contain the points' locations. They differ in a number of ways, including how the sphere's surface is partitioned into cells and how each cell may be recursively subdivided into smaller cells. Each has strengths and weaknesses specific to its design.

One of them, the Hierarchical Equal Area iso-Latitude Pixelization (HEALPix) data structure[1], was specifically designed to support fast numerical analysis of spherically mapped data. HEALPix has been successfully used for a number of applications, perhaps most notably for storing cosmic microwave background observation data. However, for applications involving relatively sparse spherically mapped point datasets, HEALPix has some possible drawbacks. Each HEALPix data structure has a single fixed resolution for the entire sphere, which can cause inefficient memory utilization for some datasets. Data points at locations that are closer together than the data structure's fixed resolution can cause overwriting of data. Finally, certain HEALPix range queries can return data points that are not actually within the query area.

[1] HEALPix can be found at http://healpix.sourceforge.net

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

A new data structure, Multi-Resolution HEALPix (MRH), was developed to address these issues. As its name implies, MRH is multi-resolution; different portions of the sphere may be represented and subdivided at different levels of resolution in the same data structure. The level of resolution, i.e., the depth of subdivision, of each area depends on the density of data points in that area. Multi-resolution has been shown to be advantageous in a wide range of data structures and applications, e.g., multi-resolution mapping of geospecific model databases [1] and modeling theory [2]. MRH is intended to combine the best aspects of HEALPix with the advantages of a multi-resolution data structure, including reduced memory requirements, improved query efficiency for some types of queries, and flexible handling of proximate points. The MRH design combines multiple quadtrees and the convenient Morton cell addressing scheme, which is inherently multi-resolution and very compact.

The remainder of article is structured as follows. Section 2 presents a brief survey of data structures for spherical mapping, especially HEALPix. Section 3 details MRH, the new multi-resolution variant of HEALPix. Section 4 defines the four types of range queries supported by MRH (disc, polygon, latitude strip, and neighbor) and describes the algorithms to perform them. Section 5 explains the benchmarking datasets, queries, and process used to compare HEALPix and MRH. Section 6 reports the benchmarking results. Finally, section 7 discusses the conclusions and proposes possible future work.

2. Background

This section presents a brief survey of data structures for spherically mapped data. The data structures are broadly categorized as Platonic or Igloo, depending on the base partition of the sphere. HEALPix is then described in more detail because of its importance to this work.

Many of the data structures described in this section use quadtrees to internally store the cells of the sphere's partition and the data within those cells. A full explanation of quadtrees is outside the scope of this article; introductory explanations may be found in [3] or [4]. Briefly, a quadtree is a hierarchical data structure based on the idea of recursive decomposition of space [5]. Simple quadtrees, for example, initially partition the plane into four quadrants and then subdivide each quadrant into four subquadrants, repeating the subdivision recursively until the desired resolution is reached. In a quadtree data structure, each quadrant corresponds to a node, and the subquadrants of a quadrant correspond to children nodes of that node. Quadtrees have been used for a wide range of applications, such as video compression [6] and query processing in sensor networks [7]. The quadtrees described in this section often use a partitioning shape other than quadrants, such as triangles, but the concept is quite similar.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

2.1. Platonic spherical data structures

Several different types of Platonic spherical data structures have been developed. They are characterized by the sphere's surface being partitioned or divided into areas or cells defined by the projection of the faces of the five regular polyhedral or Platonic solids (tetrahedron, hexahedron, octahedron, dodecahedron, or icosahedron; see Fig. 1) onto the surface of the sphere [8]. The resulting polyhedral projections are shown in Fig. 2. A geospatial indexing scheme for relational databases called the Hierarchical Triangular Mesh (HTM) was introduced in [9]. With HTM, the sphere is initially partitioned into a spherical octahedron aligned at the poles with four spherical triangular cells in the northern hemisphere and four in the southern hemisphere. (In this article, the necessarily curved projections of planar figures, such as the triangular faces of an octahedron, onto the surface of a sphere, will be indicated when needed by the adjective "spherical". Hence a "spherical triangle" is the projection of a triangle onto the surface of a sphere.) The spherical triangular cells can be recursively subdivided into four smaller spherical triangles down to the level of desired resolution. HTM is intended to index the sphere and provide complex trigonometric queries in spherical space [10]. HTM uses a spatial index to transform regions of the sphere into unique identifiers. HTM has been used extensively in various astronomical projects, including the Sloan Digital Sky Survey, ESA GAIA [10], Virtual Sky, Sky Query, SuperCOSMOS Sky Surveys, STScI Guide Star Catalog 2 [9], and SkyDOT [11]. HTM has been used to aid in cross-matching data across astronomical data catalogs [12] and in GIS projects, such as the Hadoop Distributed File System [13]. HTM (together with HEALPix) has also been applied to image edge analysis in the spherical domain [14].

The Quaternary Triangular Mesh (QTM) data structure was introduced in [15]. The base partition of the sphere in QTM is octahedral. Thereafter, QTM uses a triangular partitioning scheme in which each cell is recursively subdivided into four smaller, self-similar spherical triangular cells. Each of the eight base cells of the octahedron is represented internally as a quadtree. The nodes of the quadtree correspond to the cells.

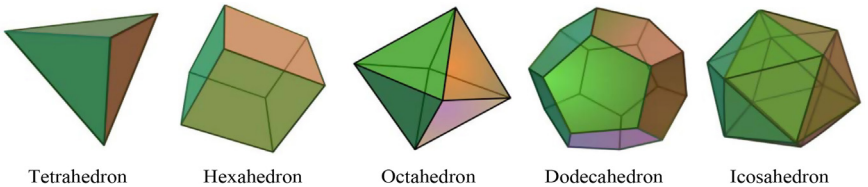

Fig. 1. The five regular polyhedra or Platonic solids. (Image Source: Wikipedia Commons. License: GNU Free Documentation License. Author: Cyp.).

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

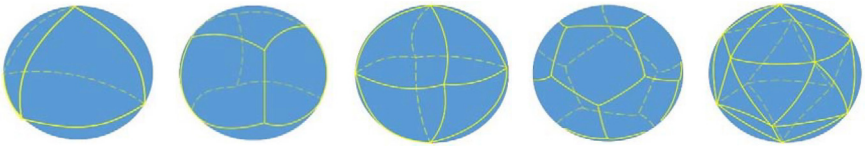

Fig. 2. Spherical partitions produced by projecting Platonic solids onto a sphere; adapted from [8].

A slightly different version of QTM, the Hierarchical Spatial Data Structure (HSDS), is described in [16]. As with the QTM and HTM data structures, in HSDS the initial partition of the sphere is octahedral, dividing the sphere into eight spherical triangular cells that intersect at the poles. The base cells are recursively subdivided into 4 smaller triangular cells until the desired level of resolution is achieved. The HSDS cell numbering method is somewhat different than the QTM method described in [17] in that it was designed to be easier to transform from latitude and longitude to cell identifier and vice versa.

Another variant of QTM, also utilizing an octahedral base partition of the sphere into spherical triangular cells that intersect at the poles, is the Semi-Quad Code QTM (SQC-QTM), which was originally proposed in [18] and described in [15]. SQC-QTM is optimized for efficient spatial access and dynamic display of data on the sphere. It uses a completely different cell numbering method called semi-quadcodes that is claimed to be more efficient in addressing locations on a sphere than the previous cell numbering methods utilized in QTM. SQC-QTM base cells can be recursively partitioned into four new triangular cells at successive levels of resolution and still be addressable using the Semi-Quad Codes discussed in [18].

The Sphere Quadtree (SQT) data structure described in [19] approximates the sphere by successively subdividing the faces of an icosahedron and projecting the new vertices onto the sphere, thereby defining cells on the sphere. The icosahedron was chosen because it has the largest number of faces of the Platonic solids, its faces are triangular, subdividing any face produces four new faces, and subdivision involves only division of faces at all resolutions. SQT has been utilized in numerous astronomical [20] and GIS projects [21].

The Truncated Icosahedron (TI) spatial data structure was proposed in [22]. TI uses a partition of the icosahedron's hexagonal cells into diamond- and triangular-shaped cells. In addition, a sampling grid based on the Lambert azimuthal equal-area map projection onto the faces of a truncated icosahedron is fit to the sphere. This geometrical model is claimed to have less deviation in area when hierarchically subdivided than any of the other Platonic solids and to produce less distortion in the shapes of the subdivided cells [22]. However, the details of the TI algorithms, including quadtree access and search, point location, and conversion between latitude and longitude values and cell addresses are not available in [22]. In addition, in the TI, the hexagonal cells are used only to orient

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

the original projection of the truncated icosahedron onto the sphere so as to cover the area of interest by the hexagonal cells.

The Quadrilateralized Spherical Cube (QS) is described in [23]. The celestial sphere is projected onto the six faces of a hexahedron (cube) in a distorted tangent plane projection such that equally spaced rows and columns of cells of equal area are formed. Although all the cells (at a given level of resolution) are equal area, they have a variety of shapes when projected back onto the celestial sphere. The QS was used in the Cosmic Background Explorer (COBE) mission [24]. The base cells of the QS can be recursively subdivided down to level of required resolution and addressed effectively.

A modification to the QS structure, proposed in [25], replaces the initial hexahedral partition with an icosahedral partition. The icosahedron was selected because it has more, and therefore smaller, faces than the other polyhedra, and consequently the projection of the faces onto the sphere are flatter and more equal in area, resulting in less overall distortion in cell shape. This modification does not recursively subdivide the base cells but instead uses a sampling grid at a pre-specified level of resolution that overlays the base cells and divides the base cell area.

The Linear Quadtree (LQ) data structure is presented in [26]. LQ uses a hierarchical triangular partitioning scheme, much like the SQT [19], QTM [15], and HTM [9] data structures. LQ employs straightforward bit-level arithmetic to perform neighbor finding at any depth in the linear quadtree.

The octahedron and icosahedron platonic solids are also utilized as the base partition for the diamond partitioning scheme described in [27]. This data structure is similar to those already described. However, instead of the base partition having spherical triangular cells, spherical diamond cells are used where the cells are formed by merging spherical triangular cells with a shared edge. The partitioning scheme recursively subdivides diamond-shaped cells into four new diamond-shaped cells down to the level of resolution required.

A scheme very similar to that described in [27] is presented in [28], called the Linear Quadtree QTM (LQ-QTM). The sphere is subdivided based on either an octahedral or icosahedral projection, and the base partition is composed of pairs of adjacent spherical triangular diamond-shaped cells that partition the surface, and thus creates a nested subdivision of the spherical surface by quadtree recursive partition. LQ-QTM uses the same Morton coding system as the index for addressing the diamond-shaped cells that was previously described in [27].

2.2. Igloo spherical data structures

Igloo spherical data structures do not use projections of the Platonic polyhedra to partition the sphere. Instead, a hybrid partition is used wherein different regions of

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

the sphere (northern hemisphere, southern hemisphere, and equator) may be subdivided in different ways. For example, the northern and southern hemispheres might be split into spherical triangular base cells and the equatorial region split into spherical quadrilateral cells, as shown in Fig. 3.

In an igloo partition, the sphere is divided into rows with edges of constant latitude. Each row is divided into identical cells by lines of constant longitude. The equatorial cells are roughly trapezoidal shaped, becoming nearly square away from the poles. Fig. 3 illustrates an example of an igloo partition. The advantages of igloo partitioning schemes include ease of implementation, naturally azimuthal cell spacing, possibly equal area cell shapes, and approximately square cells.

A multi-resolution igloo data structure designed to store high resolution global images is described in [29], where it is called the Ellipsoidal Quadtree (EQT). In EQT, the data structure takes the non-spherical ellipsoidal shape of the earth into consideration. The ellipsoidal surface is divided into bands of quadrangular cells. The bands run parallel to the meridian lines (lines of longitude) and parallel lines (lines of latitude) and consequently the sides of each cell are parallel to the meridians and the parallels. The EQT structure is hierarchical in the same manner as ordinary quadtrees, and the cells are constructed in such a way that they all have the same area at the same level in the quadtree.

A method of partitioning a sphere into cells for purposes of Cosmic Microwave Background data analysis, called the Gauss-Legendre Sky Pixelization (GLESP), is presented in [30]. GLESP is another igloo partitioning scheme that enforces strict orthogonal cell alignment along rough azimuthal lines using Gauss-Legendre polynomial zeros [31].

2.3. HEALPix

A unique spherical data structure, the Hierarchical Equal Area iso-Latitude Pixelization (HEALPix), was introduced in [32]. HEALPix was designed to support fast numerical analysis of spherically mapped data [10]. The HEALPix projection is a novel hybrid combination of the Lambert cylindrical equal area

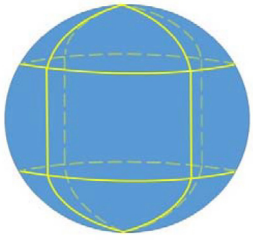

Fig. 3. Igloo partition of the sphere with different cell shapes in northern, southern, and equatorial regions.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

projection (introduced in 1772) and the Collignon pseudocylindrical equal area projection (introduced in 1865). As a software library, HEALPix not only contains useful functions to project three-dimensional (3D) spherical data points into two-dimensions (2D) but a data structure to store spherical data point references and perform various region queries [32]. The HEALPix spherical partitioning scheme is not based on a Platonic regular polyhedron. Instead the base partition of the sphere is into twelve diamond-shaped, equal area cells (four in the northern hemisphere, four along the equator, and four in the southern hemisphere) as shown in Fig. 4. These base cells are then recursively subdivided into four diamond-shaped subcells until the desired level of resolution is obtained. Internally, twelve quadtrees, one associated with each base cell, store the data records associated with that cell and its subcells.

In the current definition and implementation of HEALPix, each instance of a HEALPix data structure has a fixed single level of resolution that is set when the data structure is initialized based on the application's requirements. HEALPix data structures do not all have the same resolution, but in any single HEALPix data structure, all twelve base cells are recursively subdivided to the same resolution.

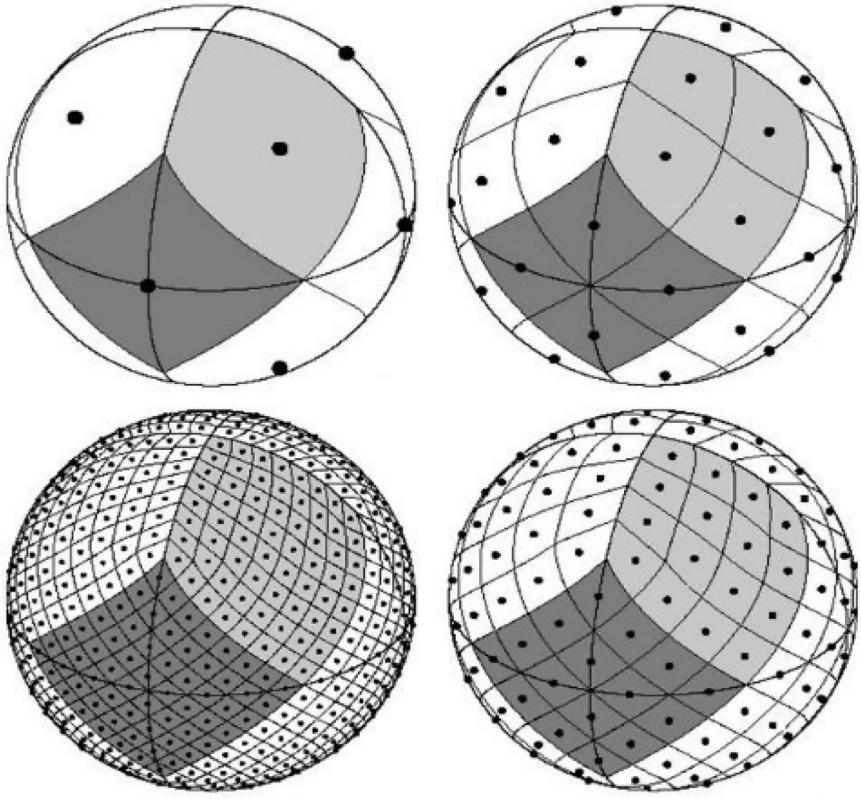

Fig. 4. HEALPix partitions of the sphere; clockwise from upper left $N_{cell} = 12$, $N_{cell} = 48$, $N_{cell} = 192$, and $N_{cell} = 769$. (Image Source: [32]. Used with permission of the author. Reproduced by permission of the AAS.).

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Thus the internal quadtrees are all completely expanded (every node has four subnodes) down to the selected level of resolution.

The total number of cells per level of resolution is given by Eq. (1) and (2):

$N_{side} \in \{1, 2, 4, 8, \ldots, 2^k\}$ (1)

$N_{cell} = 12(N_{side})^2$ (2)

where k is described as the order of resolution in HEALPix. Order of resolution, or simply "order", is also called level of resolution, or simply "level", in MRH. Both order and level refer to the number of times a base cell has been subdivided into equal area subcells. In this article the term "level" will be used hereinafter to avoid any confusion with other meanings of the term order, such as computational complexity and sequence.

The cells align on so-called iso-latitude rings, resulting in the alignment of neighboring cells along the same horizontal latitude line. The number of iso-latitude rings is given by Eq. (3).

$N_{ring} = 4(N_{side} - 1)$ (3)

For 32-bit signed integer addressing, there is a limit of $N_{side} = 2^{13}$ which produces $N_{cell} = 805{,}306{,}368$, each of which has a resolution of 25.8" of arc or approximately 827.3 meters on the Earth's surface at the equator (mean radius of 6,371 km). For 64-bit signed integer addressing, the limit is $N_{side} = 2^{29}$ which produces $N_{cell} = 3.46 \times 10^{18}$, each of which has a resolution of 3.93×10^{-4}" of arc or about 12.13 millimeters on the Earth's surface. (Cell resolution on the Earth's surface is given for illustration. Of course, HEALPix is often used to represent spheres other than the Earth's surface.)

In HEALPix, points on the sphere are located with longitude (azimuth) and latitude (elevation). Longitude (ϕ) values start at 0 at the Prime Meridian and increase in an easterly direction around the circumference of the sphere, ending at the starting point at 2π; thus $\phi \in [0, 2\pi]$. Latitude (θ) values start at 0 at the North Pole and end at π at the South Pole; thus $\theta \in [0, \pi]$. However, by convention, the spherical coordinates are scaled to 0.0 to 2.0 in ϕ, a variable denoted p, and -1.0 to 1.0 in θ, a variable denoted z; p and z can be calculated from ϕ and θ using Eqs. (4) and (5).

$$p = \frac{\phi}{\pi} \tag{4}$$

$$z = \cos(\theta) \tag{5}$$

HEALPix uses two different indexing schemes: RING and NESTED. In the RING scheme, the cell index winds down from North to South Pole through the consecutive iso-latitude rings. In the NESTED cell scheme, the cell index grows with consecutive hierarchical subdivisions on a tree structure seeded by the twelve

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

base-resolution cells. Two levels of NESTED cell partitions are shown in Fig. 5. Each part (a) and (b) of Fig. 5 show the sphere and cells that partition it projected onto a rectangle. In Fig. 5, the vertical scale is expressed in z, and the horizontal scale in p, and the large numbers are the HEALPix base cells. The HEALPix NESTED cell addressing scheme follows a Morton addressing methodology, also known as Z-ordering, which is a method of spatially labelling cells in a specific sequence to preserve multiple levels of quadtree resolution and spatial cell location [33]. For example, the first subdivision of a cell is illustrated in Fig. 6; note that the labeling sequence follows a reflected "Z" pattern.

Both the RING and the NESTED indexing schemes map the two dimensional distribution of discrete cells on a sphere into a one dimensional array (also known as a map) of size N_{cell}. The array is indexed by cell address to preserve the spatial relationships of the cells, which is essential for computations involving datasets with very large total cell numbers. The entries of the array can be of any user-defined data type.

2.4. HEALPix applications

HEALPix has been used successfully used for a large number of different scientific applications; a small subset of selected examples are mentioned here. HEALPix is often used for astronomical applications, including the study of the cosmic microwave background (CMB). Those applications include the Planck Integrated Data and Information System for analysis of CMB foreground data [34], the Balloon Observations Of Millimetric Extragalactic Radiation And Geophysics (BOOMERanG), the Microwave Anisotropy Probe (MAP), Planck Surveyor

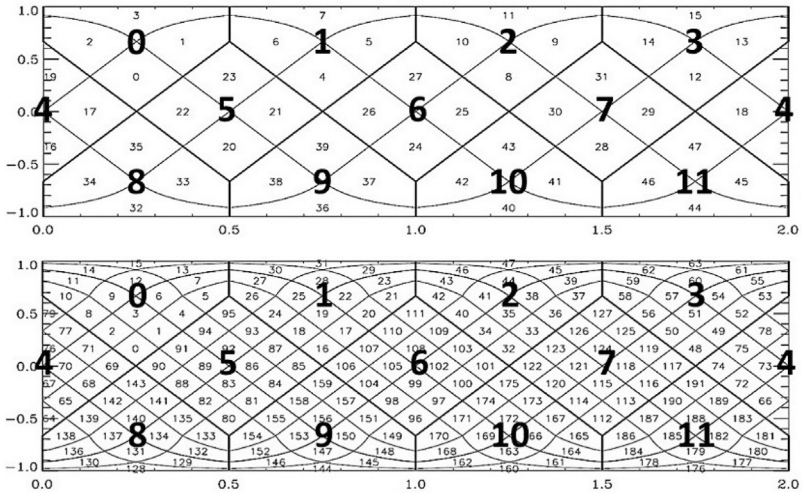

Fig. 5. Examples of HEALPix NESTED indexing: (a) $N_{cell} = 48$; (b)$N_{cell} = 192$. (Image Source: [32]. Used with permission of the author. Reproduced by permission of the AAS.).

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

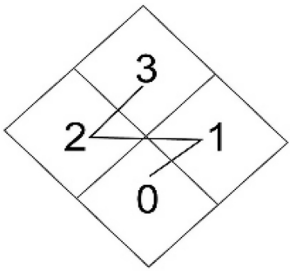

Fig. 6. HEALPix cell labeling using Z-order addressing.

(European Space Agency), Wilkinson Microwave Anisotropy Probe (WMAP), Herschel, Single Aperture Far-Infrared Observatory (SAFIR), and Beyond Einstein CMB [35, 36]. Another recent proposed application of HEALPix, called the Hierarchical Progressive Survey (HiPS), is to provide access to sky survey data at progressively higher or lower levels of resolution through a dedicated client/ browser interface [35]. HEALPix has been used to calculate wavelet transforms on the sphere for geopotential calculations [37].

In addition to astronomical applications, HEALPix has also been used for GIS applications, including the Cassandra project, where researchers are mapping different weather-related measurement data, including rainfall, solar radiation, and temperature [36]. HEALPix has been used to sample directly in the (hemi)spherical domain to solve illumination integrals (instead of following the regular approach of warping the samples from a square which can introduce distortions) while performing frequency analysis for variance in Monte Carlo integration [38]. There has been some work with HEALPix (together with HTM) involving image edge analysis in the spherical domain [14]. The HEALPix and HTM partitioning schemes have been adapted to provide efficient indexing of spherically mapped data stored in MySQL tables [39]. HEALPix has also been used for spherical terrain rendering using a level-of-detail scheme where native, high resolution terrain elevation measurements are stored at leaf nodes and internal quadtree nodes are populated based on hierarchically downsampling the native measurements in an iterative fashion [40]. HEALPix is available in many languages including C, C ++, Fortran90, IDL, GDL, Java, Julia, Matlab, Octave, Python, and Yorick [41].

2.5. HEALPix issues

Three issues with the HEALPix data structure arise when it is used for relatively sparse point data. The first is that, by design, HEALPix uses a fixed level of resolution. In any given HEALPix data structure, all of the base cells are recursively subdivided to the same level of resolution, and thus the internal quadtrees have the same depth. The level of resolution for a given HEALPix data structure is selected when the data structure is created, thereby setting the size of

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

the internal arrays as well as the number, locations, and sizes of the cells that partition the sphere. There is a reason for this; when the total number of cells is known, many of the spatial relationship computations are also established, which improves the computational efficiency of range queries. (Range queries are explained later.) When a range query is performed on a HEALPix data structure, a list of all of the cells (actually, their unique integer addresses) that overlap the query range is returned. This list is known as a HEALPix coverage map. The cells' addresses are spatial indexes into the HEALPix internal array that contains the stored data record for the point, if there is one, in each cell. The HEALPix internal array must be checked for each of the cells in the coverage map to determine whether or not it contains a point.

The primary reason for the fixed level of resolution in HEALPix is the nature of the data that the data structure was originally designed for. In the initial application of HEALPix, which was astrophysical and astronomical sky survey data storage and retrieval, the data was typically collected at a fixed level of resolution that was determined by the resolution of the measuring equipment (e.g., digital image collection). In such applications, every HEALPix cell would contain data, e.g., light magnitude values in various wavelengths, and there would be high cell-to-cell data variability. A data structure with multiple levels of resolution was not required.

However, when using HEALPix for storing data associated with discrete points, particularly when the data points are relative sparse and consequently many cells do not contain a point, a fixed-resolution data structure can be inefficient with respect to memory utilization and query performance. The data structure may be larger than necessary and queries may be (on average) slower than necessary because the depth of the internal quadtrees is (on average) deeper than necessary. For example, consider storing 5000 data points in a HEALPix data structure. To ensure that multiple data points do not map to the same HEALPix index (i.e., are located in the same cell), the data structure must be initialized at a sufficiently high resolution. The minimum distance between any two data points determines the level of resolution selected during initialization. Suppose that based on the minimum distance of the data points, a HEALPix data structure with 12 levels of subdivision is required. Instantiating a HEALPix data structure at that level of resolution would create an array with 201,326,592 elements. If it is used to store only 5000 data points, the data structure is only ~0.0000248% filled, which is inefficient in terms of memory utilization.

Continuing with this notional example, consider the results of a disc query. A disc query returns a list of HEALPix indexes or the addresses of all the HEALPix cells that overlap the query disc. Depending on the size of the disc and the resolution of the HEALPix data structure there could be many thousands of HEALPix indexes

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

returned. However, simply knowing all the HEALPix cells that overlap the query disc is not enough; the desired result is normally the actual data points that fall within the query disc. Therefore, the HEALPix map must be interrogated at each returned HEALPix index to see if the respective HEALPix map element is empty or if a data point is located in the cell. This data point search is computationally expensive because every returned HEALPix index must be checked.

The second HEALPix issue is that because it uses a fixed array (known as a map) to store the data records associated with each data point, it is possible that data records can be overwritten. As the HEALPix map is being loaded multiple data points may have spherical coordinates that translate to the same HEALPix map index, i.e., are located within the same cell; such points will be referred to as "proximate". This is especially true for low-resolution HEALPix maps, in which the cells are relatively large and consequently two data points' spherical coordinates could have substantial separation and still be proximate. When proximate points are loaded or inserted into the data structure, as each one is loaded its data record overwrites the previous data record stored for that cell, and thus only the data record for the last point located within particular cell to be loaded is retained; the previously inserted records that map to the same cell index are overwritten. The solution available in HEALPix is to use a higher level of resolution with the intent of avoiding duplicate HEALPix map indexes, i.e., avoiding proximate points. Of course, this means that there are more cells and a larger data array is required, causing larger memory requirements. Nevertheless, the risk of overwriting data records is never fully eliminated. It is always possible that multiple data points to be added to the HEALPix map have the exact same spherical coordinates, or coordinates separated by less than the new cell size, or even less than the minimum possible cell size. In these cases the points will map to the same HEALPix map index, and all but the last data record inserted will be overwritten.

The third HEALPix issue is that, even when a data point is found at a particular HEALPix map element corresponding to a cell that overlaps the query range, it is still not certain that the point is within the query range. It is possible that the data point is located in a HEALPix cell that overlaps the query range but is actually outside the query range's boundary. Consider the example disc query shown in Fig. 7. The heavy curved line is the boundary of the query disc. The alphabetically labeled cells are the HEALPix cells that overlap the query disc; .i.e, the cells in the HEALPix coverage map. The numerically labeled points represent the actual data points stored in the HEALPix data structure. Data points 37 and 239 are within the query disc boundaries and should be returned by the query. However, data points 124 and 1193 are within cells that overlap the query disc but are outside the disc, and thus should not be returned. To guarantee that all returned points are within the query range, an additional filter step must be used to screen out all data points that

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

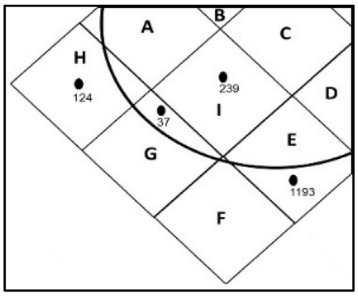

Fig. 7. Illustration of HEALPix query disc boundary (black curve), overlapping HEALPix cells (cells labeled with letters) and data point locations (circles labeled with numbers).

are in the coverage map but are not located within the range query boundary, which requires additional computation time.

3. Design

This section details the new Multi-resolution HEALPix (MRH) data structure, beginning with the motivation for developing it and continuing with the specifics of its structure and operation.

3.1. Motivation and overview

The purpose of the MRH data structure is to store data describing objects or entities whose locations lie on or can be projected onto a sphere. The development of a multi-resolution variant of the proven HEALPix data structure was motivated by the previously noted drawbacks of HEALPix when used for relatively sparse spherically mapped point data. The design of the MRH data structure is intended to combine the best aspects of HEALPix, including equal area cells and fast range queries, with the advantages of a multi-resolution data structure, specifically reduced memory utilization and improved query performance for some types of queries, while providing precise query results and flexible handling of duplicate points.

In contrast to the fixed resolution of any given HEALPix data structure, a MRH data structure may be multi-resolution; individual cells can be subdivided to any level independently of other cells. The basic concept behind MRH is to utilize the properties of the data to determine the cell resolutions and quadtree structure. A notional example MRH data structure is shown in Figs. 8 and 9 . In Fig. 8, the colors are used to illustrate that base cells can be subdivided to various levels of resolution; red is level 0 (base cell), orange is level 1, yellow is level 2, green is level 3, blue is level 4, and purple is level 5. In Fig. 9, different cell sizes and label lengths represent various levels of resolution; the base cells are level 0 and are outlined by the thick black lines, the cells labeled with single digit numbers are

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

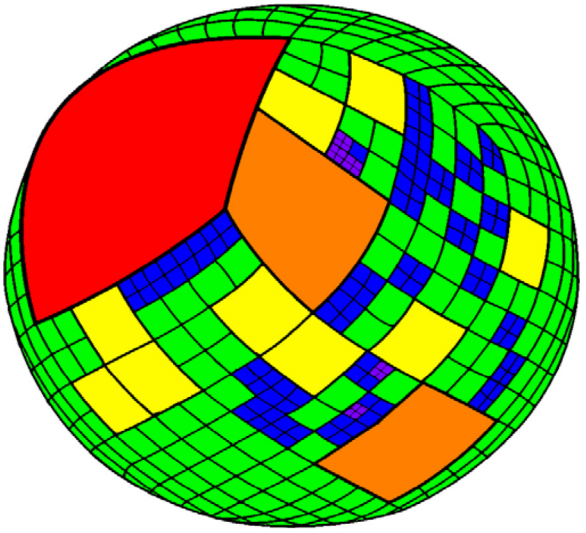

Fig. 8. Notional example Multi-Resolution HEALPix data structure showing six levels of cell resolution. Red is level 0 (base cell), orange is level 1, yellow is level 2, green is level 3, blue is level 4, and violet is level 5.

level 1 and the cells labeled with two digit numbers are level 2. As in the fixed resolution HEALPix, MRH cells at the same resolution have equal area. Internally, the MRH structure is represented by a quadtree forest, with one quadtree associated with each base cell. Depth variability is possible in each quadtree and sub-quadtree, depending on the level of cell subdivision. MRH uses as much resolution as is needed in each area and no more. The internal quadtrees are not expanded beyond the depth needed to store the data points, resulting in fewer internal quadtree nodes and leaves. The multi-resolution structure of the MRH can result in a smaller memory requirement compared to a HEALPix data structure storing the same data. Having a smaller quadtree forest can also mean improved query performance if there are fewer cells to interrogate compared to the same data stored in the HEALPix data structure.

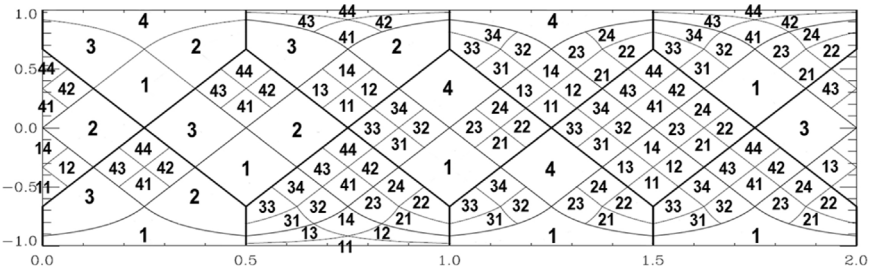

Fig. 9. Example Multi-Resolution HEALPix data structure showing two levels of cell addressing. Thick lines outline each of the twelve base cells. Single digit numbers are labels for level 1 cells; two digit numbers are labels for level 2 cells.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

3.2. MRH map description

Recall that a HEALPix map is a fixed array of a preset size depending on maximum resolution required. The elements of the map are the user-defined data records associated with the point. The map is indexed by unique HEALPix index values that follow the Z-ordering addressing scheme. The MRH data structure also includes a map of user-defined data records. However, the map is a list of variable length and each element of the map is non-empty and contains actual data point information. The map index of a data point is what is eventually stored inside a proper spatially correlated MortonNode within one of the twelve Morton Linear Quadtrees (MLQs) associated with the base cells. When a range query returns a list of Morton addresses, these addresses are searched in the proper MLQ and the leaf nodes found contain map indexes that indicate where in the MRH map to find the information for the data points within the query's range. The data type of the MRH map is user-defined and can be either simple data types or more complex structures or classes. The MRH map is an array of user defined data types or templates. The only requirement is that the user provide a spherical location (ϕ and θ in the HEALPix coordinate system) of the point associated with the data record to be added to the MRH map and data structure. The point's location will be used to assign the data record's MRH map index to the proper MLQ, which is discussed in the next section.

3.3. Morton cell addressing

Like HEALPix, the MRH data structure utilizes the Morton addressing scheme, but unlike HEALPix, MRH implements full multi-level Morton addressing. Fig. 10 shows what the first subdivision of a cell would look like following MRH's Morton addressing scheme. Also note in Fig. 10 that the cells are labeled starting at 1 rather than 0.

Fig. 11 shows an example of multi-level Morton addressing; note that the Z-ordering scheme is still preserved. The label of a child cell is determined simply by appending the child cell's Z-ordered label to the parent cell's label.

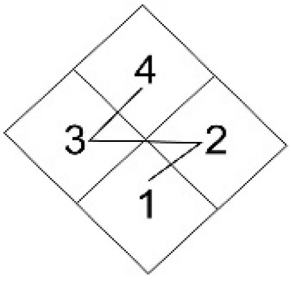

Fig. 10. Level 1 labeling of subdivided cell using Morton addressing.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

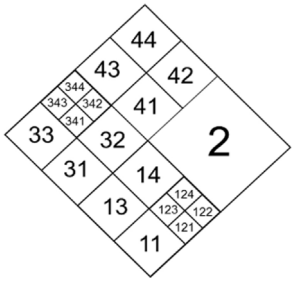

Fig. 11. Example Multi-level labeling of subdivided cell using Morton addressing.

Labeling recursively subdivided cells by appending a Z-ordered digit to the cell's parent's label makes it easy to determine a cell's spatial position and depth in the quadtree using its label. The cell depth is given by the length of the label. For example, cell 342's label is three digits long and so would be stored at level 3 of the quadtree (where the root is level 0). The cell's spatial position can be determined by inspecting the digits of its label from left to right. For example, cell 342's leading digit 3 means the level 1 spatial location of cell 342 is within the western quadrant of the original level 0 cell. The next digit 4 means the level 2 spatial location of cell 342 is in the northern quadrant of the parent level 1 cell 3. Finally, the last digit 2 means the level 3 spatial location of cell 342 is located in the eastern quadrant of the level 2 cell 4.

The cell subdivision shown in Fig. 11 would have the quadtree shown in Fig. 12. The MRH internal representation of the quadtree of Fig. 12 as an MLQ and the associated point data records are shown in Fig. 13. In Fig. 13, the MLQ (left) is a linear array of Morton nodes, one for each quadtree node, containing the cell's Morton address and possibly an index into the data array (map) that leads to the point data record stored at that node, if there is one. Morton nodes in the MLQ for quadtree leaf nodes that correspond to cells that contain a point contain indexes to the data array, whereas Morton nodes in the MLQ for quadtree internal nodes and leaf nodes that correspond to cells that do not contain a point do not have indexes. (In Fig. 13, dots represent indexes; for clarity only a subset of the indexes are also depicted as arrows.) Fig. 13 depicts only one MLQ; there is one MLQ per base cell. There is only one data array; the data records for all the points in the structure are stored in the same data array. Each entry in the array is an instance of a user-

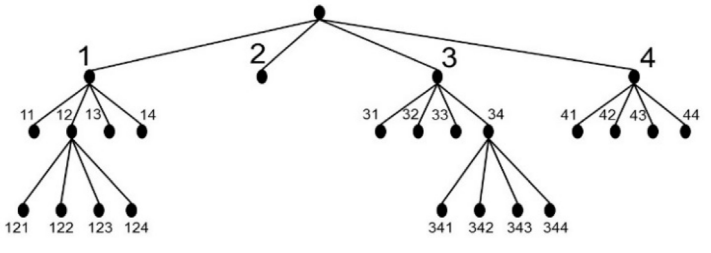

Fig. 12. Example quadtree representation of multi-level Morton addresses.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

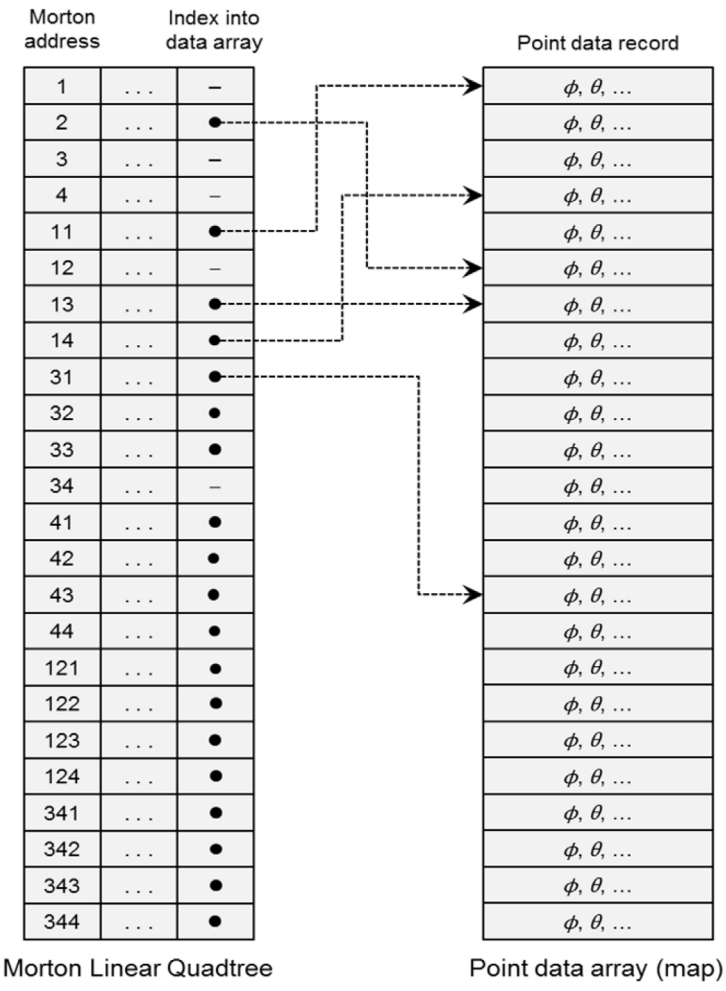

Fig. 13. Example MRH quadtree structure, showing the MLQ and the data map.

defined point datatype that stores all of the information associated with a point, including its spherical location coordinates (ϕ, θ).

The structure of the MLQ allows straightforward determination of the spatial relationships between cells such as child, sibling, and parent cells. For example, consider the Morton address 124. This cell is at level 3 in the quadtree. The parent cell is found by removing the last digit of the Morton address (4) giving 12, which is the Morton address of the parent cell of 124. Siblings of cell 124 are 121, 122, and 123. Likewise, the children of cell 12 are 121, 122, 123, and 124. To determine if a given cell has any siblings or descendants a binary search of the sorted array for the specific Morton address is performed.

3.4. Morton datatype definition

In MRH, the Morton address datatype is not simply a list of digits; in fact, it consists of 30 separate bit fields. Twenty nine of the fields are 2 bits in length and

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

one is 6 bits in length, for a total of 64 bits. Each 2-bit field encodes the Morton address at a specific level. Morton address 1 is encoded as binary 00, 2 as 01, 3 as 10, and 4 as 11. The Morton address datatype can encode up to 29 levels of Morton addresses. The 6-bit field is used to indicate the maximum encoding level of a Morton address. The maximum level encoder is necessary because it is otherwise impossible to tell if a particular level is encoded with Morton address 1 (00) or if that level is simply unset (also 00) because the Morton address is not that long.

To compare Morton addresses as needed for searching and sorting, the Morton address datatype can easily be cast into a 64-bit unsigned integer. Comparison between Morton addresses is made between the 64-bit unsigned integer representations by comparing their values. Alternatively, Morton addresses can be compared by inspecting the bit encodings at each level.

3.5. Morton and HEALPix addressing relationship

The key to MRH functionality is the close relationship between Morton addressing and the NESTED HEALPix addressing schemes. Morton addressing is the scheme by which cell numbers are assigned. From inspection of the base HEALPix NESTED addressing scheme (Fig. 14(a)) and the base Morton addressing scheme (Fig. 14(b)), it can be seen that the two are nearly identical. The difference is that the HEALPix addressing scheme starts at 0 and Morton addressing starts at 1.

However, this one-to-one HEALPix-Morton address relationship applies only to HEALPix addresses that have been "normalized". A normalized HEALPix address is a HEALPix address with a value that has been shifted into the range of addresses covered by the base HEALPix cell 0. Normalizing a HEALPix address preserves the original HEALPix address's relative spatial location with respect to its parent base HEALPix cell. For example, consider HEALPix address 121 in Fig. 5(b). This HEALPix cell is located in base HEALPix cell 7 at level 2. The normalized HEALPix address at level 2 of HEALPix address 121 is 9. Note that the relative spatial location of HEALPix address 121 within base HEALPix cell 7 is identical to the relative spatial location of HEALPix address 9 within base HEALPix cell 0.

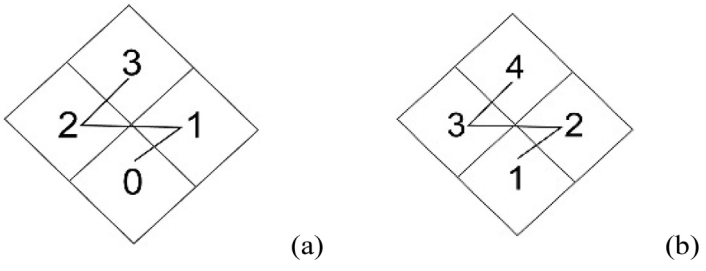

Fig. 14. Cell addressing schemes: (a) HEALPix; (b) Morton.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

The algorithm to compute the normalized HEALPix address is straightforward. For every level of resolution, the HEALPix nested addressing scheme always starts at address 0 inside HEALPix base cell 0 and labels subcells in Z-order up to N_{pix} in HEALPix base cell 11. The relationship between the number of subcells per base HEALPix cell is given by the specified HEALPix level and Eqs. (6) and (7).

$$N_{side} = 2^k \quad (6)$$

where k is the HEALPix level or number of cell subdivisions

$$N_{pix} = N_{side}^2 \quad (7)$$

For example, consider the level 1 partition of base HEALPix cell 0. The sub-cell addresses are: 0, 1, 2, 3. A level 2 partition of base HEALPix cell 0 has the following sub-cell addresses: 0, 1, 2, . . . , 15. There are twelve base HEALPix cells ($addr_{base}$) in the HEALPix data structure; the sub-cell Z-order addresses continue from base HEALPix cell 0 to base HEALPix cell 11 as shown in Fig. 5. The level 1 subcells of base HEALPix cell 0 (Fig. 5(a)) are labeled 0, 1, 2, 3 and the sub-cell labeling continues in base HEALPix cell 1 with subcells 4, 5, 6, 7. Finally, the level 1 subcells of base HEALPix cell 11 contains subcells 44, 45, 46, 47. Level 2 sub-cell labeling is shown in Fig. 5(b). Therefore, to normalize any HEALPix address at a given level Eq. (8) is used.

$$addr_{norm} = addr - \left(addr_{base} \times N_{pix}\right) \quad (8)$$

Once the HEALPix address has been normalized the algorithm in Fig. 15 can be used to compute the corresponding Morton address. The inputs to the algorithm are the HEALPix address and level. The algorithm constructs the Morton address level-by-level based on the level of the HEALPix address. At each level the HEALPix address is either 0, 1, 2, or 3 and so the respective Morton address at the same level will be 1, 2, 3, or 4. The algorithm can also be reversed to compute a HEALPix address from a Morton address at given level.

Lastly, if the Morton address is not from a base HEALPix cell 0 the reverse normalization algorithm shown in Eq. (9) is used to compute the correct, non-normalized HEALPix address, $addr$.

$$addr = addr_{norm} + \left(addr_{base} \times N_{pix}\right) \quad (9)$$

```
1. mask = GetBitMask(level);
2. for( int i = level; i > 0; i-- )
3.    nextBit = ( (hpxid & mask) >> ( (2*i) - 2) );
4.    SetMortonBit( m, nextBit, (level-i) + 1);
5.    mask = mask >> 2;
```

Fig. 15. Algorithm to compute Morton address given normalized HEALPix address and level.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

3.6. MRH quadtrees

Like HEALPix, MRH consists of a forest of 12 quadtrees or MLQs; one for each base cell. The MLQs are used for range queries and searches of data records stored in the MRH map. Internally, the MLQ is a list of MortonNodes sorted ascending by the 64-bit integer value of the Morton address. Each MLQ's addressing is based on base HEALPix cell 0 subcell addressing, or "normalized" HEALPix addressing, as discussed earlier. MLQs are constructed data record reference by data record reference based on the spatial location of the data points in each data record. Each MLQ initially consists of four MortonNodes at the lowest resolution Morton addresses 1, 2, 3, and 4. As data records are added to the MRH map, new MortonNodes are created and added to the MLQ with the property of a single data record reference per MortonNode. To insert the MortonNode, the MLQ is searched using the MortonNode's Morton address. The MortonNode is initially assigned a Morton address of the lowest level of resolution, level equal 1; therefore it will have the address 1, 2, 3, or 4 (Fig. 14(b)). If a previously inserted MortonNode is not found in the MLQ, then the new MortonNode is appended to the MLQ and the tree is re-sorted by ascending Morton address.

However, in general, MortonNode insertions can cause tree insertion collisions. A tree insertion collision occurs when a MortonNode is found in the MLQ (known as the collided MortonNode) with a Morton address that matches the Morton address of a MortonNode to be inserted into the MLQ (known as the colliding MortonNode). In this case the collided (existing) and colliding (new) MortonNode's respective spatial locations (φ, θ) are used to compute new Morton addresses at the next higher level of resolution (next deeper level of the MLQ) for both the collided and colliding MortonNode. These new Morton addresses will either be different or still match. If they are different, the collided and colliding MortonNodes are updated with the new Morton address and re-inserted into the MLQ. If the new Morton addresses still match, then the preceding process is repeated until either different Morton addresses are computed or the maximum user specified tree depth or the maximum physical tree depth (tree depth of 29 equivalent to HEALPix level equal 29) is reached. If unique Morton addresses are still unable to be computed and the maximum user specified tree depth or the maximum physical tree depth is reached, there are two ways to resolve the Morton address matching conflict.

The first method to resolve the Morton address matching conflict is used when the spatial coordinates of the collided and colliding MortonNodes are proximate i.e., their spatial coordinates do not match exactly (to machine epsilon). In this case, the same Morton address is assigned to both the collided and colliding MortonNodes at the highest allowable resolution. However, to distinguish between these MortonNodes in the MLQ, the colliding MortonNode has a subaddress value

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

that can be used to uniquely identify this duplicate. Therefore, in future searches of the MLQ involving this Morton address, both MortonNodes would be found.

The second method to resolve the Morton address matching conflict is used when the spatial coordinates of the collided and colliding MortonNodes are duplicate, i.e., their spatial coordinates do match exactly (to machine epsilon). In this case, instead of appending the colliding MortonNode to the MLQ, the collided MortonNode's list of MRH map indexes is appended with the colliding MortonNode's corresponding MRH map index. This scenario results from records with duplicate spatial coordinates being found in the data file being processed, i.e., measurement made from same location but different time. Therefore, in future searches of the MLQ involving this Morton address, this MortonNode would be returned, which references multiple MRH map indexes.

In the final step of MortonNode insertion, the MLQ is sorted by ascending Morton address. The updated MLQ storing the MRH map indexes would have the structure shown in Fig. 13. Note that the only MortonNodes stored in the MLQ are leaf nodes that reference a MRH map indices and the respective parent MortonNodes. A key design goal of the MLQ is to store only those MortonNodes that are absolutely essential in order to keep the overall memory requirement of each MLQ as small as possible.

4. Methods

This section defines the four types of range queries supported by MRH (disc, polygon, latitude strip, and neighbor), describes the algorithms to process them, and reports the theoretical worst case computational complexity for each.

In general, a range query returns the set of points in a data set that are located within a specified geometric area, which is referred to as the query range. In the context of spherical data structures, a range query retrieves and returns the points and associated data records that are located within a portion of the sphere's surface that is specified in the query. For example, in a disc query, the query range is a disc specified using a (ϕ, θ) location and a radius r on the sphere's surface. The points located within the disc centered at the given location having the given radius are retrieved, along with their data records.

4.1. Query types and algorithms

Disc queries return the set of data points located within a query disc. Polygon queries return the set of data points located within a query polygon. Latitude strip queries return the set of data points located within a query latitude range, given as minimum and maximum latitude values. Neighbor queries return the set of data points located in cells neighboring a given query cell. For all four query types, the

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

data records associated with the returned points are also returned. These four range queries were implemented in MRH because they are the range queries that HEALPix supports.

At the conceptual level, there are four steps to processing a disc or polygon query in MRH:

1. Select the HEALPix coverage map cell resolution based on the approximate area of the query disc or polygon. The query disc or polygon area is compared to a pre-computed Table of HEALPix cell areas for all supported levels of resolution. The HEALPix cell area that it closest to the query disc or polygon area is selected for the HEALPix coverage map generation.
2. Generate a HEALPix coverage map of all cells, at a level of resolution given in the query, that overlap the query range (either a disc or a polygon). The HEALPix coverage map algorithm starts by testing all the base HEALPix cells for overlap with the query area. Cells that overlap the query area but are not completely within the query area will continue to be subdivided and each subcell tested for overlap with the query area until the level of resolution given in the query is reached. If a particular cell is found to be completely contained within the query area, that cell is subdivided down to the query resolution and all the subdivided cells are added to the HEALPix coverage map without further overlap checking. Cells that overlap the query area but are not completely within the query area will continue to be subdivided and each subcell tested for overlap with the query area until the query resolution level is reached. Those cells that still overlap the query polygon at the query resolution are added to the HEALPix coverage map and returned to calling method (Fig. 16(a)).
3. Check each cell in the HEALPix coverage map to determine if it contains a data point. Fig. 16(b) shows an example of spatially correlated MRH MortonNodes

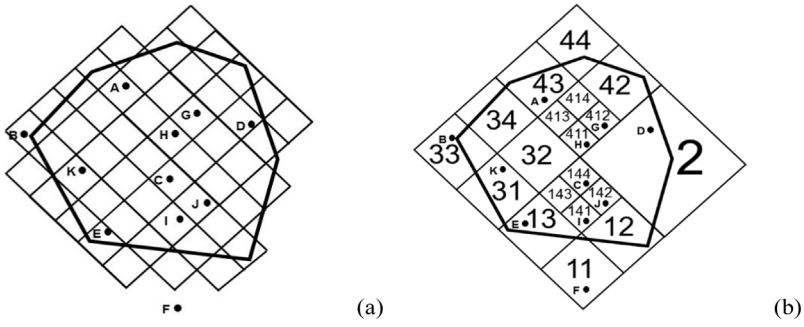

Fig. 16. (a) Example HEALPix query polygon and level 2 coverage map with data points shown for reference; (b) Spatially correlated MRH MortonNodes with mapped data points that correspond the same spatial area of the query polygon boundaries.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

with mapped data points that correspond to the HEALPix coverage map shown in Fig. 16(a). For example, searching the MRH data structure with each HEALPix coverage map (Fig. 16(a)) cell would result in searches of the MRH MortonNodes shown in Fig. 16(b). However, there are scenarios where searching up the MRH data structure is necessary. For example, consider the data points labeled "A", "B", "D", "E", and "K", in Fig. 16(b). The HEALPix coverage map from Fig. 16(a) has a query resolution that is at a higher resolution than many of the cells in the MRH data structure where data point locations are mapped to. Searching the MRH data structure at those HEALPix coverage map cells will fail to find spatially correlated MRH MortonNodes; i.e. the HEALPix coverage map cell in which data point "A" resides in has no corresponding MRH MortonNode of the same resolution. Therefore, it is necessary to search the lower resolution parent MRH MortonNode for the data point; e.g., parent MRH MortonNode would have the Morton address 43 and contain data point "A". This process is known as "up-search" and is triggered when a search of the MRH data structure, at a given cell resolution fails and up-searches have been enabled. Up-searches require additional processing and therefore add to the overall computation time.

4. Test each data point found to determine if it is actually within the query area, by performing either a point-in-polygon test or a point-in-disc test, depending on the type of query.

A latitude strip query returns the list of data points that fall within or on a latitude band or strip specified by the user. At an abstract level, there is only a single step to processing a latitude strip query in MRH: loop through all the MortonNodes of all MLQs and test if each the MortonNode data point reference falls within the specified latitude strip boundaries.

A neighbor query returns the list of data points that fall within the immediate vicinity of a specified location. The neighbor query is actually a special case of the polygon query where the polygon is diamond-shaped to match the shape of a HEALPix cell at a given level of resolution and centered at the specified query location. At the conceptual level there are three steps to processing a neighbor query:

1. Compute HEALPix neighbor query using the provided query center point. The HEALPix neighbor query returns a coverage map consisting of all HEALPix cells that contain the query center point and immediate neighboring cells.
2. Search the MRH data structure for data points using the list of HEALPix cell addresses contained in the HEALPix coverage map.
3. Test each data point found to determine if it is actually within the query area, by performing either a point-in-polygon test.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

4.2. Query computational complexity

Let N_{HPX} be the total number of cells in the HEALPix map which represents, in linear form, twelve full quadtrees with no internal nodes, only leaf nodes. Let N_{MRH} be the sum of all MortonNodes (internal and leaf) contained in all twelve MLQs that make up the MRH data structure. Both N_{HPX} and N_{MRH} count all nodes that can be searched or found in the respective data structures.

Let M_{HPX} be the total number of found HEALPix indices in the HEALPix coverage map produced by HEALPix range queries. Finally, let M_{MRH} be the total number of found HEALPix indices in the HEALPix coverage map produced by MRH range queries. For disc and polygon queries, typically $M_{MRH} \ll M_{HPX}$ because the query resolution setting for the HEALPix coverage map in MRH is based on comparing the area of the query to various HEALPix cell resolutions and choosing the closest match to set the query resolution.

In contrast, for disc and polygon queries in HEALPix, the query resolution is fixed, based on the resolution chosen when the HEALPix map was constructed. MRH sets the query resolution for the HEALPix coverage map to be as low as possible; therefore it is potentially of lower resolution than was used to create the corresponding HEALPix data structure.

Table 1 summarizes the worst case computational complexity of each of the range query types. Essentially, in the worst case every range query requires time on the order of the number of nodes (N_{HPX} or N_{MRH}), because in the worst case a query could overlap all of the cells and a data point could be located in every cell. The exception is the HEALPix neighbor query, which always runs in constant time in all cases.

5. Experimental

Although the theoretical computational complexity of HEALPix and MRH queries are of academic interest, the primary focus here is on the data structures' memory

Table 1. Summary of worst case computational complexity for various range queries.

Range query	HEALPix	MRH
Disc	$O(N_{HPX})$	$O(N_{MRH})$
Polygon	$O(N_{HPX})$	$O(N_{MRH})$
Latitude strip	$O(N_{HPX})$	$O(N_{MRH})$
Neighbor	$O(1)$	$O(N_{MRH})$

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

utilization and average query performance for practical applications. For many applications it is likely that $N_{MRH} \ll N_{HPX}$ because of MRH's multi-resolution structure. To assess the effect that will have on memory requirements and average query performance, the data structures and query algorithms were extensively tested and their performance compared empirically. This section describes the data sets, queries, and process used for the empirical performance testing.

5.1. Benchmarking datasets

Application datasets containing spherically mapped data were used for comparing, or benchmarking, the performance of HEALPix and MRH. Several criteria were used to select benchmarking datasets; the datasets were chosen to: (1) be from representative spherical mapping applications; (2) have point counts of increasing orders of magnitude, from $\sim 10^3$ to $\sim 10^6$; (3) be from different scientific application areas; (4) have points located in different portions of the sphere. Four benchmarking datasets were used:

- Fragmentation. The spatial locations of the 4798 fragments resulting from a warhead detonation as their flyout trajectories carried them past a specific radius from the detonation point. The locations were calculated by a high-fidelity physics-based model of the detonation of an explosive device fragmenting a metal cylinder designed to break up in a predictable way [42]. Fig. 17 shows the data points in the dataset.
- NOAA. The geographic locations (longitude, latitude) of the 9959 U. S. National Oceanic and Atmospheric Administration weather stations in the continental United States [43].

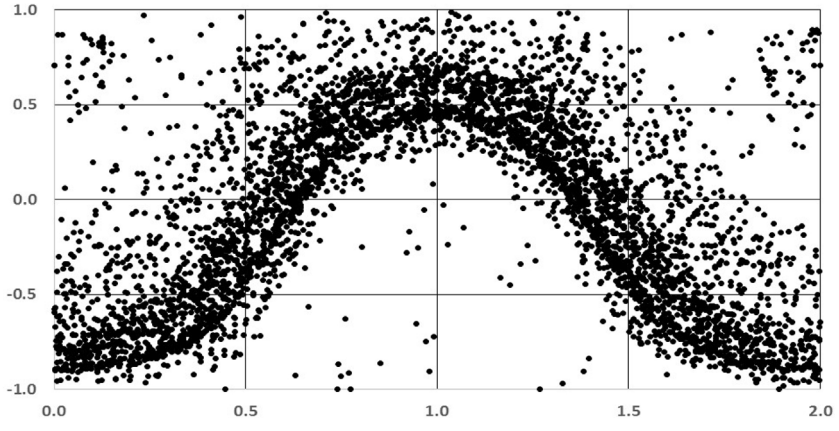

Fig. 17. Fragmentation dataset point locations; the surface of the sphere and the points have been projected onto a rectangle.

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

- SDSS. The celestial sphere locations (right ascension, declination) of 99,565 galaxies with a measured redshift of $z > 0.6619$ found in Sloan Digital Sky Survey catalog [44].
- Synthetic. The spherical locations (ϕ, θ) of 993,258 randomly generated uniformly distributed points on a sphere spanning the entire HEALPix coordinate system.

To allow direct comparison of the memory requirements and query performance of HEALPix and MRH, precisely the same data points were loaded into both data structures for each dataset. To ensure that, the test datasets were filtered to remove proximate points. As data is loaded into HEALPix, each data point's spherical coordinates are used to compute the proper index location in the HEALPix map. Proximate data points can map to the same cell, and thus the same HEALPix map index, and as data is loaded into the HEALPix map, previously loaded data may be overwritten. It was important that the HEALPix data structure fit within main memory of the computer used for the performance testing, thus a HEALPix map size of HEALPix level 12 was selected, which created a HEALPix map with 201,326,592 elements or 14,155,776 Kb (13.5 Gb) in size. The same resolution was used for all four test datasets. Each dataset was loaded into a HEALPix data structure, and during the loading some data overwriting occurred. After each dataset's data had been loaded into the HEALPix map, the data in the map was written back out into a new "filtered" dataset. The filtered datasets contained only the data for the last point that mapped to each cell. (For almost all cells, there was at most one point.) The number of data points removed from each dataset by the filtering process was less than 1% of the original number for all datasets; the specific number of points removed were: Fragmentation, 0 points, 0%; NOAA, 34 points, ~0.34%; SDSS, 367 points, ~0.37%; and Synthetic, 6742 points, ~0.67%. The number of data points reported earlier for each dataset is the final, filtered count.

These filtered datasets were used for the benchmarking. Checking of the results returned by each query used for the benchmarking confirmed that subsequent to this filtering process the query results always matched exactly between the HEALPix and MRH data structures.

Table 2 summarizes the number of MortonNodes (internal and leaf) of each of the twelve MLQs in an MRH data structure when constructed from the respective benchmarking datasets. Notice that, with a single exception, the NOAA dataset only spans three different MLQs. The unequal node distribution in the Synthetic dataset; in particular the number of nodes in MLQs 5–8 are significantly less than the number of nodes in the other MLQs. It is hypothesized that an artifact of the random data point generator was producing data points that disproportionately fell along the equatorial region between the maximum (north pole) and minimum

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Table 2. Summary of the number of MortonNodes in each MLQ for each MRH constructed with benchmarking datasets.

MLQ #	Fragmentation	NOAA	SDSS	Synthetic
1	122	9120	5982	165012
2	1050	4747	45744	164246
3	1165	0	44928	164696
4	131	1	3381	164658
5	196	0	36396	94765
6	1266	3320	1705	94330
7	379	0	30499	94797
8	1369	0	4312	95151
9	1286	0	904	165160
10	51	0	40	163978
11	32	0	181	164299
12	1253	0	48	164309
Average	692	5729	14510	141283

(south pole). This would lead to a denser clustering of data points in the MLQ 5–8 region and through the data point filtering process a large amount of those data points would likely have been removed.

Table 3 summarizes the average MLQ depth of each of the twelve MLQs, where the depth of each MortonNode (internal and leaf) is added to a sum total and divided by the total number of MortonNodes in the MLQ. Notice that despite the SDSS dataset having more data points, the NOAA dataset appears to be denser if average MLQ depth is used as an indicator of data point density; the Synthetic dataset is still the densest as predicted.

5.2. Benchmarking queries

To benchmark the query performance of the HEALPix and MRH data structures, test queries of each type (disc, polygon, latitude strip, and neighbor) were randomly generated using custom software. A total of 30 queries of each types were generated for each dataset. Each test dataset had unique ranges where points were located, so it was necessary to parameterize the queries to conform to those ranges. The ranges were given as minimum and maximum ϕ and θ values.

Generating the disc queries started with setting minimum and maximum values for the query disc center coordinates and query disc radii. For each query, the minimum and maximum ϕ and θ values were used to generate a random disc center that would fall within a bounding box defined by those values. Next, a random

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Table 3. Summary of the average MLQ depth of each MLQ for each MRH constructed with benchmarking datasets.

MLQ #	Fragmentation	NOAA	SDSS	Synthetic
1	3.70	8.21	8.36	8.82
2	5.27	8.36	8.28	8.82
3	5.36	0.00	8.17	8.82
4	3.85	1.00	7.66	8.82
5	4.11	0.00	8.05	8.27
6	5.34	8.12	7.63	8.26
7	5.23	0.00	7.97	8.27
8	5.52	0.00	7.83	8.26
9	5.42	0.00	7.97	8.81
10	3.63	0.00	7.10	8.82
11	3.22	0.00	7.86	8.81
12	5.44	0.00	6.29	8.81
Average	4.67	8.23	7.76	8.63

radius within the minimum and maximum radius was generated. Lastly, the disc was checked to ensure that it did not overlap the bounding box defined by the minimum and maximum ϕ and θ. If the disc was found to cross the bounding box, the query was not used and another was generated.

Generating the polygon queries also started with setting minimum and maximum query disc center coordinates and minimum and maximum query disc radii. This was because discs were used to create the query polygons. For each query, a valid disc was generated and used to define a polygon whose points fall along the perimeter of the disc, i.e., a cyclic polygon. Given a minimum and maximum arc length (in radians), a loop incrementally added sides to the polygon by randomly generating an arc length within those limits and adding the chord of an arc of that length, beginning from the endpoint of the last side, as a next side of the polygon. The polygon construction process continue until the total arc length exceeded 2π radians.

Generating the latitude strip queries started with setting minimum and maximum values for θ, which were specified in the HEALPix coordinate system in radians. The minimum and maximum θ values were then used to generate two random θ values, θ_1 and θ_2, that defined either one or two latitude strips based on their relative values. If $\theta_1 \leq \theta_2$, then a single latitude strip with θ_1 as the minimum and θ_2 as the maximum was defined. If $\theta_1 > \theta_2$, then two latitude strips were defined and were considered both part of the same query. The first latitude strip was

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

bounded by the North Pole as the maximum and θ_1 as the minimum, and the second latitude strip was bounded by θ_2 as the maximum and the South Pole as the minimum.

Generating the neighbor queries was quite simple. First, all the records of the benchmark dataset file were read into an array and the number of records counted. Next a random integer was generated in the interval $[0, n - 1]$ where n is the number of records. The random integer drawn is used as an index into the array and the spatial location (ϕ, θ) of the point at that array index was used as a neighbor query.

5.3. Benchmarking process

The benchmarking process was straightforward. For each of the four datasets, a HEALPix data structure and an MRH data structure were loaded with the test data and their memory requirements recorded. Then each of the 120 test queries (30 each of four types of queries) was executed 30 times on each data structure and the execution times recorded. (Although the test computer was not running any other user processes during the performance testing, the queries were repeated 30 times to smooth out any latencies due to non-benchmarking system processes.)

The same data was loaded and the same queries were applied to both HEALPix and MRH to allow a direct comparison of the data structures' memory requirements and query performance. To confirm the accuracy of the HEALPix and MRH queries, prior to the timed query executions, the actual data points returned by each query were compared. After performing the filtering described earlier, the MRH and HEALPix query results matched exactly; the same points were returned for each pair of corresponding HEALPix and MRH queries during the testing.

6. Results

This section reports the benchmarking results. The efficiency and performance of the two data structures, HEALPix and MRH, were compared on memory requirements and query performance.

6.1. Memory requirements

MRH was significantly better than HEALPix with respect to memory requirements; it required four orders of magnitude less memory in the best case (Fragmentation dataset) and two orders of magnitude in the worst case (Synthetic dataset). As explained earlier, for each of the four test datasets a HEALPix map of level 12 was created, which has a memory requirement of 14,155.78 Mb or 201,326,592 quadtree nodes. Even with the largest dataset, the HEALPix map was sparsely populated, so there was substantial unused space. In contrast, when the

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

MRH data structures were constructed, only populated leaf nodes and their respective parent nodes were instantiated, which resulted in a significantly smaller memory requirement.

A summary of the memory requirements of the four benchmarking datasets in the two data structures is shown in Table 4. Bold text in Table indicates smaller memory requirements. For MRH, the reported memory requirements include not only the MRH map but also all twelve of the MLQs. For HEALPix, the memory requirements are just for the HEALPix map which is organized as a single, full, quadtree whose elements represent leaf nodes.

The memory requirements of the complete MRH data structures increased linearly with dataset size. However, because the HEALPix data structures' resolution was set to 12 to minimize overwriting of data points, their memory requirement was the same for all four datasets. Similarly, for MRH the number of MLQ internal and leaf nodes also increased linearly with dataset size. HEALPix, on the other hand, because it is a fixed size at level 12 also had the same number of quadtree nodes for all four datasets.

6.2. Query performance

In addition to the MRH data structure's reduced memory utilization, MRH query performance was, on average, also substantially better than HEALPix query performance, although HEALPix was better for some query types on some datasets. Of the sixteen combinations of query type and dataset, MRH and HEALPix each were faster for eight combinations. More importantly, the average query time over all datasets and all query types of MRH was faster than HEALPix. The total execution time for all 14,400 (480 × 30) queries on HEALPix was 7,485,371 milliseconds for an average query time of 693.09 milliseconds. In comparison, the total execution time for the same queries on MRH was 2,127,948

Table 4. HEALPix and MRH memory requirements for the test datasets.

Metric	Dataset	HEALPix	MRH
Bytes (K)	Fragmentation	14,155,776	**920**
	NOAA	14,155,776	**1908**
	SDSS	14,155,776	**19,242**
	Synthetic	14,155,776	**189,045**
Quadtree size (nodes)	Fragmentation	201,326,592	**8300**
	NOAA	201,326,592	**17,188**
	SDSS	201,326,592	**174,120**
	Synthetic	201,326,592	**1,695,401**

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Table 5. HEALPix and MRH query performance results for the test queries on the test datasets (times are in milliseconds).

Query type	Dataset	Queries × reps	HEALPix			MRH		
			Sum	Mean	Std dev	Sum	Mean	Std dev
Disc	Fragmentation	30 × 30	80,638	89.60	58.15	1,030	1.14	0.92
	NOAA	30 × 30	5,554	6.17	4.43	3,744	4.16	3.28
	SDSS	30 × 30	110,916	123.24	25.81	59,046	**65.61**	33.02
	Synthetic	30 × 30	82,051	**91.17**	67.82	185,281	205.87	136.57
Polygon	Fragmentation	30 × 30	49,169	54.63	50.69	7,035	**7.82**	8.81
	NOAA	30 × 30	20,498	**22.78**	28.87	55,584	61.76	82.46
	SDSS	30 × 30	163,957	**182.17**	118.34	240,129	266.81	246.51
	Synthetic	30 × 30	403,754	**448.62**	630.11	1,217,455	1352.73	1544.36
Latitude strip	Fragmentation	30 × 30	736,228	818.03	484.91	1,200	**1.33**	0.60
	NOAA	30 × 30	406,271	451.41	593.89	2,323	**2.58**	1.23
	SDSS	30 × 30	768,036	853.37	483.97	30,406	**33.78**	14.31
	Synthetic	30 × 30	4,658,295	5,175.88	3398.76	324,636	**360.71**	145.92
Neighbor	Fragmentation	30 × 30	1	<**0.01**	0.0	1	<0.01	0.0
	NOAA	30 × 30	1	<**0.01**	0.0	16	0.02	0.10
	SDSS	30 × 30	1	<**0.01**	0.0	30	0.03	0.13
	Synthetic	30 × 30	1	<**0.01**	0.0	32	0.04	0.14
All	All	480 × 30	7,485,371	693.09	371.61	2,127,948	**197.03**	138.65

milliseconds for an average query time of 197.03 milliseconds. This 72% improvement in average query time is significant in that the benchmarking process was designed to simulate a wide range of dataset types and application queries that a user might perform.

Table 5 summarizes the benchmarking query performance results for all query types and datasets. Bold text in Table indicates faster query times. The last row of Table combines all query types and datasets.

7. Conclusions

This section reports the conclusions of this work and suggests some additional research to improve the performance and extend the utility of MRH.

7.1. Findings

MRH, a multi-resolution variant of the widely used fixed-resolution HEALPix data structure, was developed with the design goal of reducing memory requirements

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

without sacrificing range query capabilities or average query execution speed. Empirical testing using representative scientific data and queries showed that for relatively sparse point datasets the new MRH data structure has both significantly reduced memory utilization and improved average query times. The average query performance in the benchmarking is seen as a good indicator of typical performance of the data structures for scientific applications involving point data.

An objective of this research was to let the spatial properties of the data determine the construction of the data structure's quadtrees. MRH's use of Morton addressing to encode HEALPix (address, level) pairs in the construction of MLQs achieves this. MLQs exhibit multiple levels of resolution determined by the spatial properties of the data point references they are storing.

Moreover, the MRH MLQs have the capability to handle duplicate mapped data point references for proximate points, unlike the overwriting that can occur with HEALPix. Such points would be preserved during MLQ construction by appending the proximate points' data to a list at each quadtree node; later all such data would be retrieved in queries. (Although the correct functioning of this capability was verified during implementation, it was not used in the benchmarking reported earlier so as to provide a direct performance comparison.) This is a useful capability, in that some applications may use point data that intentionally maps to the exact same cell and Morton address for a given level of resolution; examples include different types of measurements at the same location or measurements recorded at the same location at different times. It should also be noted that in MRH, which has no fixed resolution limit, for datasets containing regions of closely spaced points the cells can be subdivided and the quadtrees extended as needed to avoid the issue of proximate points. Memory is available for this increase in resolution because it is only done where necessary, not throughout the entire data structure as with HEALPix.

The variability of query performance between HEALPix and MRH among the various data sets can partially be explained by the relative density of the data points of the data set and range query area triggering relatively expensive up-searches in the MRH data structure. As the range queries were randomly generated it is possible that the resultant range query resolution was higher than the MRH data structure resolution. As explained in Section 4.1, searches of the MRH data structure at resolutions higher than the underlying MLQ resolution will fail to find data point references even if there is a data point that maps to the area described by the query cell. In these cases, an up-search of the MLQ is triggered that checks the parent MortonNodes of the query cell for possible data point references. These up-searches require additional computation time.

As point density increases, MRH begins to lose its advantages in memory utilization and query performance, but HEALPix data structures must be built at

http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

higher levels of resolution to avoid excessive overwriting of proximate points. Currently there is no definite criteria that can be used to determine whether MRH is preferable to HEALPix, or vice versa. However, for relatively sparse data sets where the likelihood of data points being mapped to neighboring high resolution cells is low, MRH is recommended because its memory requirements will be much less than HEALPix. For high density data sets where the likelihood of data points being mapped to neighboring high resolution cells is high, HEALPix is recommended.

7.2. Future work

A baseline version of MRH including the most useful HEALPix range queries was implemented, but there are additional HEALPix methods that could be implemented in MRH. These include two-point correlation and complex analyses involving multiple disc, polygon, neighbor, or latitude strip queries.

Additionally, improvements to the computational efficiency of MRH may be possible. One possible optimization may be to limit the size of any particular MRH MLQ. Once a quadtree is considered full, any additional data points that map to the same spatial area covered by the full MLQ would simply be inserted into a new quadtree that covers the same area. Of course, the search operation would also have to be modified to search all MLQs that map to same area. This could improve efficiency by restricting the depth of any particular quadtree. The optimum size limit for an MLQ would be determined experimentally.

Another possible optimization lead to a more efficient way to construct MRH MLQs. For a given dataset, constructing a HEALPix data structure is substantially faster than constructing the corresponding MRH. For most applications, this is a minor issue, in that the data structure is ordinarily initialized offline and then used repeatedly. If initialization time is an issue, initializing a HEALPix data structure first and then working recursively to compute appropriate, minimal resolution MRH Morton addresses for each data point using the high resolution HEALPix addresses may take less time than initializing the MRH data structure from scratch.

Additional experimentation to clarify the specific dataset characteristics that indicate the use of MRH or standard HEALPix would be useful. This experimentation could be based on a sequence of synthetic datasets of increasing point density to determine the density at which MRH's advantages with sparse data diminish.

Also, the feasibility of adapting the MRH data structure to other applications of HEALPix, such as those mentioned earlier, or to applications that do not use spherically mapped point data could be studied. An example of latter is described in [45], in which spatially gridded point data representing ocean and atmosphere

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

information at three different levels of resolution are stored in a data structure that combines those levels.

Finally, progressive data is an approach to partitioning a large volume of data points that may map to the same region in order to facilitate viewing the data at different levels of zoom, similar to the approach taken in [35]. One way to adapt MRH to store such data would be to develop a "Multi-MRH" data structure, consisting of multiple MRH data structures, with each data point assigned to one of the MRH structures based on the value range of a specific data attribute. Multi-MRH queries could be modified to not only specify region of interest location but also value range of interest.

Declarations

Author contribution statement

Robert W. Youngren: Conceived and designed the experiments; Performed the experiments; Analyzed and interpreted the data; Contributed reagents, materials, analysis tools or data; Wrote the paper.

Mikel D. Petty: Conceived and designed the experiments; Analyzed and interpreted the data; Wrote the paper.

Funding statement

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Competing interest statement

The authors declare no conflict of interest.

Additional information

No additional information is available for this paper.

Acknowledgements

Some of the results in this paper have been derived using the HEALPix package [32]. Krzysztof M. Górski, Ph.D. (Jet Propulsion Laboratory, Pasadena California USA) granted permission to reuse two figures from his publication [32]. Wesley N. Colley, Ph.D. (Torch Technologies, Huntsville Alabama USA) suggested the HEALPix data structure for this research. Their support is gratefully acknowledged.

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332
2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

References

[1] D. Craciun, J. Deschaud, F. Goulette, Automatic Ground Surface Reconstruction from mobile laser systems for driving simulation engines, SIMULATION: Transactions of the Society for Modeling and Simulation International 93 (4) (2017) 201–211.

[2] B. Liu, A. Verbraeck, Multi-resolution modeling based on quotient space and DEVS, Simulation Modelling Practice and Theory 70 (2017) 36–51.

[3] M.J. Laszlo, Computational Geometry and Computer Graphics in C++, Prentice Hall, Upper Saddle River NJ, 1996.

[4] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Second revised ed., Springer-Verlag, Berlin Germany, 2000.

[5] H. Samet, The Quadtree and Related Hierarchical Data Structures, ACM Computing Surveys 16 (2) (1984) 187–260.

[6] R.E. Chaudhari, S.B. Dhok, Fractal Video Coding Using Fast Normalized Covariance Based Similarity Measure, Mathematical Problems in Engineering (2016) 11 1725051.

[7] J. Kim, I. Shin, Y. Zhang, D. Kim, K. Han, Aggregate Queries in Wireless Sensor Networks, International Journal of Distributed Sensor Networks (2012) 15 625798.

[8] A.J. Kimerling, K. Sahr, D. White, L. Song, Comparing Geometrical Properties of Global Grids, Cartography and Geographic Information Science 26 (4) (1999) 271–287.

[9] A.S. Szalay, J. Gray, A. Thakar, P. Kunszt, T. Malik, J. Raddick, C. Stoughton, J. vandenBerg, The SDSS SkyServer -Public Access to the Sloan Digital Sky Server Data, Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison Wisconsin, June 3-6, 2002, pp. 570–581.

[10] W. O'Mullane, A.J. Banday, K.M. Gorski, P. Kunszt, A.S. Szalay, Splitting the Sky -HTM and HEALPix, Mining the Sky, Proceedings of the MPA/ESO/MPE Workshop, Garching Germany, July 31-August 4, 2000, pp. 638–648.

[11] P. Wozniak, K. Borozdin, M. Galassi, W. Priedhorsky, D. Starr, W.T. Vestrand, R. White, J. Wren, October, SkyDOT (Sky Database for Objects in the Time Domain): A Virtual Observatory for Variability Studies at LANL, (2002) October 23.

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332

2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

[12] A.S. Szalay, Large Databases in Astronomy, Mining the Sky Proceedings of the MPA/ESO/MPE Workshop, Garching Germany July 31-August 4, 2000.

[13] L. Zhenhua, H. Yingjie, Z. Haidong, Y. Bailang, W. Jianping, L. Bo, Z. Hui, Spatial indexing of global geographical data with HTM, Proceedings of 18th International Conference on Geoinformatics, Beijing, June 18-20, 2010, pp. 1–6.

[14] A. Lorbert, P.J. Ramadge, Level set estimation on the sphere, IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP), Dallas Texas, March 14-19, 2010, pp. 2202–2205.

[15] G. Dutton, A Hierarchical Coordinate System for Geoprocessing and Cartography, Lecture Notes in Earth Sciences 79 (1999).

[16] M.F. Goodchild, S. Yang, A hierarchical spatial data structure for global geographic information systems, CVGIP: Graphical Models and Image Processing 54 (1) (1992) 31–44.

[17] G. Dutton, Encoding and handling geospatial data with hierarchical triangular meshes, In: M.J. Kraak, M. Molenaar (Eds.), Advances in GIS Research II (Proc. SDH7, Delft Holland), 2, Taylor & Francis, London UK, 1996, pp. 505–518.

[18] E.J. Otoo, H. Zhu, Indexing on spherical Surfaces using semi-quadcodes, Proceedings of 3rd International Symposium of Advances in Spatial Databases, Singapore (1993) 510–529.

[19] G. Fekete, L.A. Treinish, Sphere quadtrees: a new data structure to support the visualization of spherically distributed data, Proceedings of SPIE 1259, Extracting Meaning from Complex Data: Processing, Display, Interaction, Santa Clara California, August 1, 1990.

[20] G. Fekete, Rendering and managing spherical data with sphere quadtrees, Proceedings of the First IEEE Conference on Visualization, San Francisco CA, October 23-26, 1990, pp. 176–186.

[21] N.M. Short, R.F. Cromp, W.J. Campbell, J.C. Tilton, J.J. LeMoigne, G. Fekete, N.S. Netanyahu, K. Wichmann, W.B. Ligon, Mission to Planet Earth: AI views the world, IEEE Expert 10 (6) (1995) 24–34.

[22] D. White, J.A. Kimerling, S. Overton, Cartographic and Geometric Components of a Global Sampling Design for Environmental Monitoring, Cartography and Geographic Information Systems 19 (1) (1992) 5–22.

[23] R.A. White, S.W. Stemwedel, In: D.M. Worrall, C. Biemesderfer, J. Barnes (Eds.), The Quadrilateralized Spherical Cube and Quad-Tree for All Sky

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332

2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Data, 25, Proceedings of Astronomical Data Analysis Software and Systems I A.S.P. Conference Series, San Francisco, 1992, pp. 379–381.

[24] National Aeronautics and Space Administration, November 20 2003, Accessed, COBE Quadrilateralized Spherical Cube, (2017) . November 20 2003, Accessed January 11 2017 http://lambda.gsfc.nasa.gov/product/cobe/skymap_info_new.cfm.

[25] M. Tegmark, An icosahedron-based method for pixelizing the celestial sphere, The Astrophysical Journal 470 (2) (1996) 81–84.

[26] M. Lee, H. Samet, Navigating through triangle meshes implemented as linear quadtrees, ACM Transactions on Graphics 19 (2) (2000) 79–121.

[27] D. White, Global Grids From Recursive Diamond Subdivisions Of The Surface Of An Octahedron Or Icosahedron, Environmental Monitoring and Assessment 64 (1) (2000) 93–103.

[28] J. Bai, X. Zhao, J. Chen, Indexing of the discrete global grid using linear quadtree, Proceedings of ISPRS Workshop on Service and Application of Spatial Data Infrastructure, Hangzhou China, October 14-16, 2005, pp. 267–270.

[29] P. Ottoson, Retrieval of geographic data using ellipsoidal quadtrees, In: Jan Terje Bjørke, Håvard Tveite (Eds.), Proceedings of the 8th Scandinavian Research Conference on Geographical Information Science, Ås Norway, 2001, pp. 89–98 June 25-27.

[30] A.G. Doroshkevich, P.D. Naselsky, O.V. Verkhodanov, D.I. Novikov, V.I. Turchaninov, I.D. Novikov, P.R. Christensen, L.Y. Chiang, Gauss—Legendre Sky Pixelization (GLESP) for CMB maps, (2005) arXiv:astro-ph/0305537v4, May 27.

[31] A.N. Lowan, N. Davids, A. Levenson, Table of the zeros of the Legendre polynomials of order 1-16 and the weight coefficients for Gauss' mechanical quadrature formula, Bulletin of the American Mathematical Society 48 (10) (1942) 739–743.

[32] K.M. Górski, E. Hivon, A.J. Banday, B.D. Wandelt, F.K. Hansen, M. Reinecke, M. Bartelman, HEALPix −a Framework for High Resolution Discretization, and Fast Analysis of Data Distributed on the Sphere, The Astrophysical Journal 622 (2) (2005) 759–771.

[33] G.M. Morton, A computer Oriented Geodetic Database; and a New Technique in File Sequencing, Technical Report IBM Ltd., International Business Machines Co. Ltd., Ontario Ottawa, 1966.

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332

2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

[34] G. Giardino, A.J. Banday, K. Bennett, P. Fosalba, K.M. Górski, W. O'Mullane, J. Tauber, C. Vuerli, Analysis of CMB foregrounds using a database for Planck, Proceedings of Mining the sky MPA/ESO/MPE Workshop, Garching Germany, July 31-August 4, 2000.

[35] P. Fernique, M.G. Allen, T. Boch, A. Oberto, F. Pineau, D. Durand, C. Bot, L. Cambresy, S. Derriere, F. Genova, F. Bonnarel, Hierarchical progressive surveys. Multi-resolution HEALPix data structures for astronomical images, catalogues, and 3-dimensional data cubes, (2015) May 9.

[36] B. Dowis, B. Singh, Summer of eResearch Projects -Geospatial Lattices with HEALPix, (2011) . March 3 2011, Accessed October 25 2011 http://www.eresearch.org.nz/content/geospatial-lattices-healpix.

[37] M. Shahram, D. Donoho, J. Starck, Multiscale representation for data on the sphere and applications to geopotential data, Proceedings of SPIE 6701, San Diego California, August 26 2007, 2007.

[38] G. Singh, Sampling and Variance Analysis for Monte Carlo Integration in Spherical Domain, Ph.D. Thesis, Université Claude Bernard Lyon 1, 2015 September 8 2015.

[39] L. Nicastro, G. Calderone, Multiple Depth DB Tables Indexing on the Sphere, Advances in Astronomy (2010) 11 524534.

[40] R. Westerteiger, A. Gerndt, B. Hamann, Spherical Terrain Rendering using the hierarchical HEALPix grid, Proceedings of IRTG 1131–Visualization of Large and Unstructured Data Sets Workshop 2011, Kaiserslautern Germany, June 10-11, 2011, pp. 13–23.

[41] E. Hivon, K.M. Gorski, M. Reinecke, HEALPix -Data Analysis, Simulations and Visualization on the Sphere, (2017) . Accessed April 26, 2017 https://sourceforge.net/projects/healpix/.

[42] A.W. Pike, J.M. Steen, R.W. Youngren, MOUT End Game Tool Enhancements for System Level Personnel Lethality Assessment, Proceedings of 2016 Alabama Simulation Conference and Exposition, Huntsville AL, May 3-5, 2016.

[43] National Oceanic and Atmospheric Administration Cooperative Observer Network, (2015) . Retrieved October 1 2015 https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/cooperative-observer-network-coop/.

[44] Sloan Digital Sky Survey, SDSS Query/CasJobs, (2015) . Accessed October 1 2015 http://skyserver.sdss.org/CasJobs/.

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332

2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

[45] R.A. Reynolds, H. Iskenderian, S.O. Ouzts, Using Multiple Resolutions and Representations to Compose Simulated METOC Environments, Proceedings of the Spring 2002 Simulation Interoperability Workshop, Orlando FL, March 10-15, 2002, pp. 215–220.

 http://dx.doi.org/10.1016/j.heliyon.2017.e00332

2405-8440/© 2017 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).