RIT Department of Computer Science

Thesis Proposal

Real-Time Continuous Surface Modeling of Massive Geospatial Point Clouds with User-Selectable Entities

Student: Stephen Ranger, Advisor: Professor Reynold Bailey

Abstract

Massive point cloud data sets are currently being created and studied in academia, the private sector, and the military. Many previous attempts at rendering point clouds have allowed the user to visualize the data in a three-dimensional way but did not allow them to interact with the data and would require all data to be in memory at runtime. Recently, a few systems have cropped up that deal with real-time rendering of massive point clouds with on-the-fly level of detail modification that handles out-of-core processing but these systems have their own limitations. With the size and scale of massive point cloud data coming from LiDAR (Light Detection and Ranging) systems, being able to visualize the data as well as interact and transform the data is needed.

Previous work in out-of-core rendering [3] [4] [5] showed that using octrees and kd-trees can increase the availability of data as well as allow a user to visualize the information in a much more useful manner. However, viewing the data isn't enough; applying work in context-aware selection [2] and surface creation [1] the visualization system would greatly benefit in usability and functionality.

Chapter 1: Background

The basis for this continued work stems from multi-dimensional data structures and how they are leveraged in a geospatial environment. These data structures, the quadtree, octree, and kd-tree, are used to partition a (usually) cartesian coordinate system into a tree structure.

Quadtrees

The Quadtree separates a two-dimensional coordinate system into a tree structure where each node in the tree is split along the center of both axes within its axis-aligned bounding volume creating four child nodes. This allows for much more efficient searching than a simple data structure like an array or list would. Below, Illustration 1 shows a set of points in a two-dimensional cartesian coordinate system stored in a quadtree where each node that has only a single point remaining does not split again; thus saving computational time and storage space. The Quadtree structure is used in geospatial visualizations as it aligns to the longitude and latitude coordinate system, however, the disconnect along the 180-degree meridian and the difficult-to-handle north and south border meeting at a single point cause more issues.

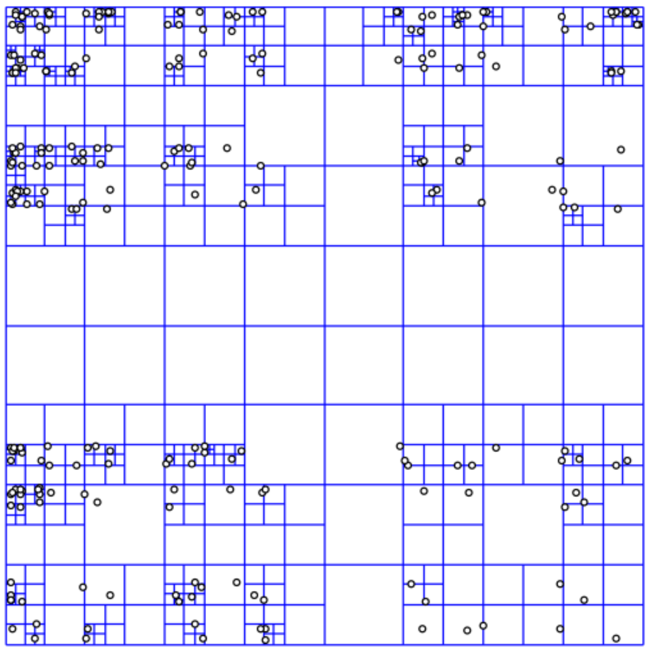

Illustration 1: Quadtree Structure [10]

Octrees

The Octree structure is identical to the quadtree structure except that it is applied to a three-dimensional coordinate system. Each node is split into potentially eight child nodes but it still uses a uniform split location by bisecting the span along each axis of its axis-aligned bounding volume. The octree is used quite often in Computer Graphics as it fiots well to an XYZ cartesian coordinate system and allows culling, bounds intersection, and mouse picking for many objects quickly and efficiently. However, in a geospatial visualization, this structure does not align well to an elliptical surface such as the Earth. Below, Illustration 2 is an image of an octree in 3D as well as the flattened version of the tree.

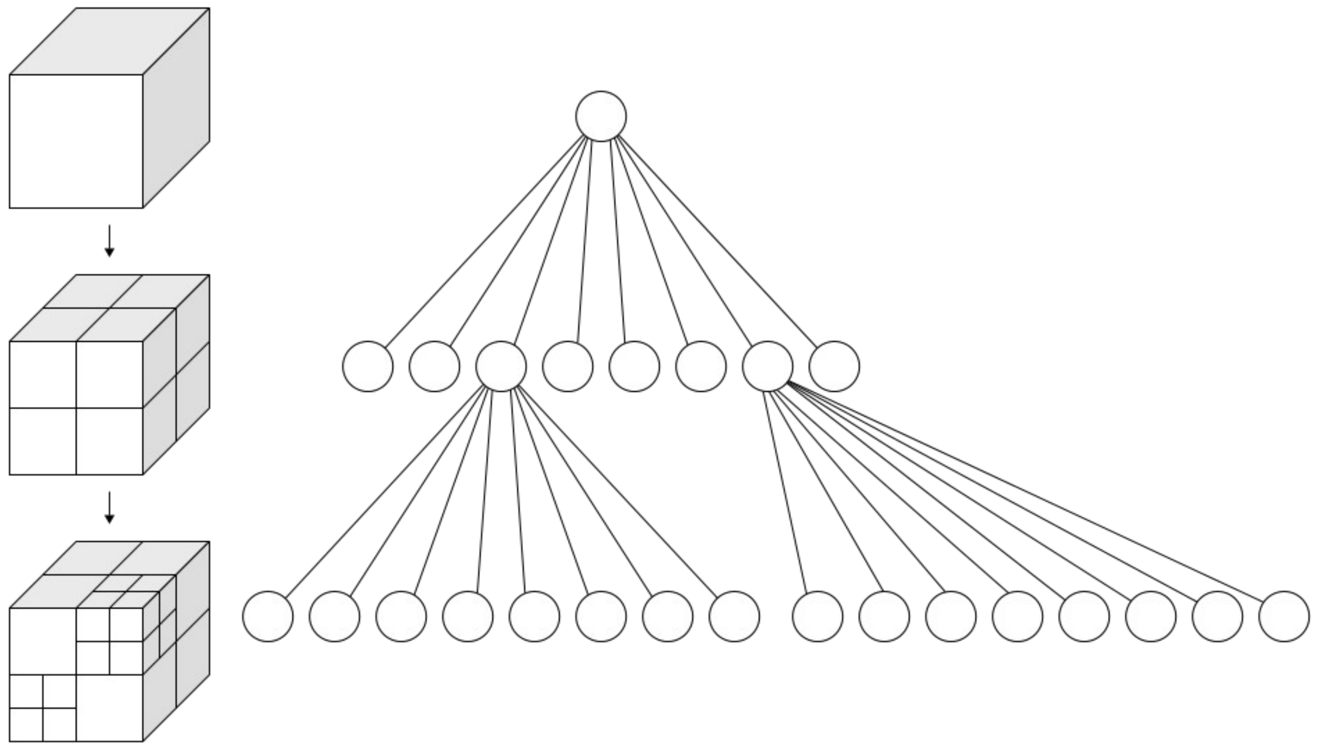
Illustration 2: Octree Structure and Flattened Tree [11]

KD-Trees

The KD-Tree is a specialized binary tree that allows for more control over how the tree is structured as well as lending itself to being more uniform and balanced than previously mentioned tree data structures. The KD-Tree, at each level, chooses a single axis to split along. The axis chosen is defined by a set of criteria such as which axis has the longest span between its min and max values. Then, the split location is also defined by a set of criteria such as splitting each side into containing a uniform number of points. The actual split location, unlike the quadtree and octree, is a single point that acts as the node in the tree at that depth. This continues until each node has a single point or until some threshold is reached based on tree depth or point count. The KD-Tree is a binary tree but can be used with any dimensionality which allows it to be applied to a two-dimensional longitude-latitude coordinate system as with the quadtree and an XYZ cartesian coordinate system like the octree. It has many of the same drawbacks when it comes to applying it to a geospatial coordinate system. However, one upside is that it can be built in such a way that each node has a relatively uniform number of points which can increase storage efficiency and render times by allowing us to store nodes in chunks without wasting padded space but with the drawback that level-of-detail algorithms make the scene seem non-uniform as each node is rendered and the node dimensions are not consistent. Below, Illustration 3 shows a two-dimensional KD-Tree partitioning algorithm with red and blue lines as alternating split locations. Illustration 4 shows the flattened tree structure for part of the previous illustration and which axis was used at each level.

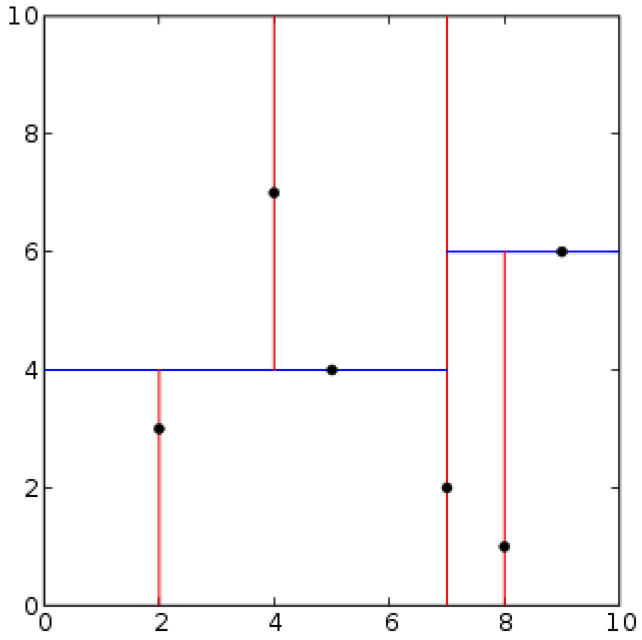

Illustration 3: Two-Dimensional KD-Tree Structure [12]

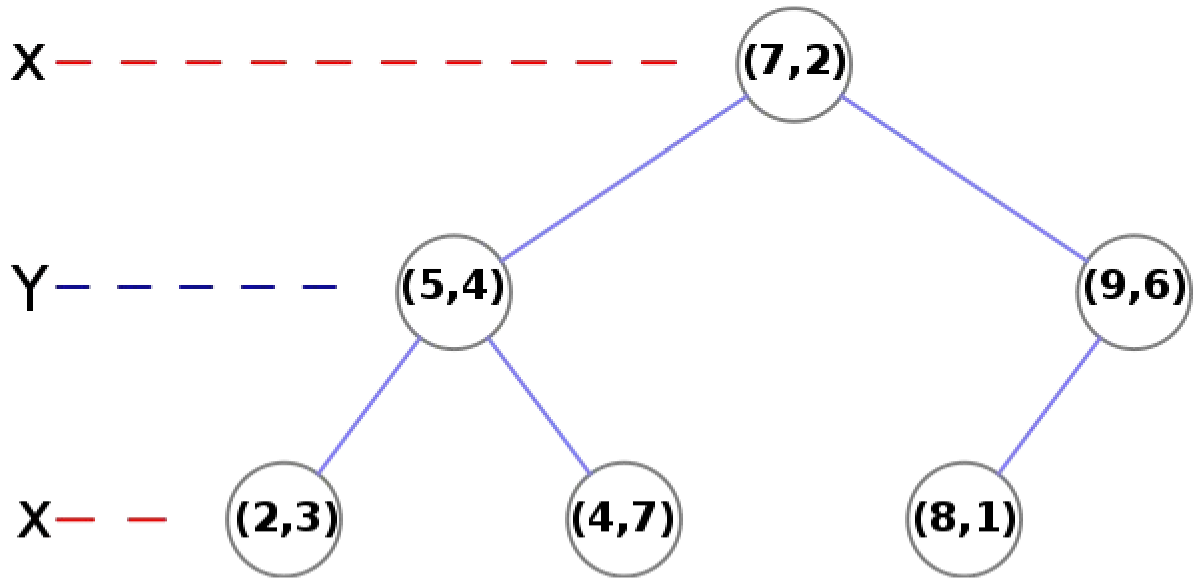

Illustration 4: KD-Tree Partitioning Example [13]

Previously, quadtree, octree [3] and kd-tree partitioning systems have been applied to a Cartesian system in order to add on-demand searching for view-dependent slices of data. However, once the data set is converted into a geospatial positioning system, either the Cartesian coordinates are now not surface aligned, or issues arise along the partitioning system boundaries where values are not continuous (such as with latitude 90 != latitude -90 but longitude 180 == longitude -180). Using a Cartesian projection from geospatial coordinates solves this issue but adds complexity when deciding what to render as it is no longer surface aligned. I will look into applying a surface-aligned data structure to a global data set such as a tetrahedral mesh. By using this structure to search for nodes within my view frustum I will be able to render progressively deeper nodes as the visualization's viewpoint moves closer or further away from the target and as the level of detail increases and pull these nodes from an on-demand point server.

Also, in recent work, real-time rendering of depth culling and surface representation [1] has been used to hide unseen data points as well as to fill surface information in on a massive unstructured point cloud. I propose to use this information and apply it to a sparse context-aware selection algorithm [2] in order to adapt it to a more uniformly dense data set. I believe that this on-the-fly surface creation can be used to augment the context-aware selection algorithm in order to give it another avenue for object separation within a uniformly-dense data set.

Chapter 2: Methodology

Previous work in the field of massive point cloud creation and processing has moved toward data systems [6] such as OpenTopography.org as well as research in leveraging octree and kd-tree data structures for taking massive point cloud data and partitioning it into manageable pieces for the visualization system. Separate work in selection and surface generation have increased the user interaction and utility of these types of datasets. However, tree structures in cartesian or geodetic coordinate systems each have their own drawbacks and limiting the user to visualization alone gives them a powerful tool with nothing to use it for.

The first hurdle is to look at the state of data partitioning and access. Currently, octrees and kd-trees work in a cartesian coordinate system. With point cloud data that is relatively limited in size and scope, using an arbitrary coordinate system can work well. Moving towards larger data sets and rendering them on a geospatial projected surface and now your data no longer follows an axis-aligned format. This causes more issues with rendering such as data structures that clip the projected surface and do not fill the partitioned sections well. The first goal of this research is to modify the data structure used to store the massive point cloud data from an octree/kd-tree structure where the XYZ cartesian grid doesn't align well with a geospatial projected surface (WGS84) into an icosahedral grid using previous work in spherical self organizing grids [7] and traversal of triangle mesh elements [8] [9]. Allowing the data to be split into surface aligned data structures should look more pleasing to the end user as the level of detail is modified onthe-fly compared to an octree/kd-tree structure without the need of overdrawing or forcing the user to wait until all data is available before rendering.

The second hurdle is to add functionality to the system instead of just being a visualization system. Previous work in the area consisted of interacting with unstructured point clouds [2] by allowing the user the ability to select more-dense regions of data using a screen-space masking system. However, this implementation is limited by requiring the data be unstructured, or containing a very heterogeneous density throughout. Unfortunately, LiDAR data is rarely sparse; it is rarely pruned or processed at all before being accessed by researchers. In another area, on-the-fly surface creation has been applied to point cloud data in order to show physical structures [1] in the input data; this shows off actual objects in the virtual world without having extensive up-front processing of the data. This surface modeling can be leveraged against the contextaware selection algorithm in a uniformly dense point cloud to allow the user to select via screen-space masking controls objects in the LiDAR point cloud data. From this, the point data can be displayed separately for further study or exported for use in other applications.

In order to evaluate these assumptions, the author proposes a three-tiered visualization system. Using knowledge from other on-the-fly data access systems such as OpenTopography.org [6] and the author's work in the private sector, an on-demand point cloud data server will be designed that will allow the visualization system to access point data from a centralized location without needing to store large amounts of data locally. This server will be extensible in that is will support any data structure format that is necessary for evaluating the utility of the icosahedral data structure (icosatree) against the more common octree structure. This server will be developed using two separate technologies. First, the majority of data access requests will be done through regular web server front end; the data will be stored in globally accessible directories via an HTTP URL. Each octree/icosatree node will be a separate file on the server and can be downloaded with a simple HTTP request. Second, a separate server can be (optionally) started for accessing detailed subsets of the point cloud via a simple bounding box; this will allow the visualization to ask for the most detailed data available for a specific region without having to download the entire tree structure beforehand. This will be useful for viewing selection regions or exporting data for external use. In order to handle level of detail in a real-time system, the octree/icosatree data structures will be split into virtual cells. Each parent node will be allowed n-points per virtual cell before the remainder are sent to child nodes. This will allow the visualization to be uniformly dense any any specific level of the tree structure.

Once the data server is operational, a Java/OpenGL visualization application will be developed for viewing the massive point cloud data. This work will use a graphics library built upon JOGL (Java OpenGL library; jogamp.org) and written by the author of this paper for use throughout his graduate studies. The visualization will also be built upon his extensive experience in developing octree-based massive point cloud systems in the private sector. The visualization system will support both octree-based and icosatreebased data structure input and the ability to enable and leverage surface modeling [1] and context-aware selection [2] modes. This augmented depth culling surface selection algorithm will be the main portion of research and development in the visualization system. The visualization system will also have the ability to collect performance metrics and will export for future analysis.

Chapter 3: Design

I intend to implement a system containing four pieces: A Partitioning application, A Visualization tool, The Rendering Component, and the Selection Algorithm. Below is the initial framework for each portion of my implementation, how they will be developed, a rough workflow of each, and its intended purpose.

Partitioning Application

The partitioning application will be a standalone system that will read the point data from either CSV or LAS formatted files. It will create the different partitioning systems needed to evaluate my thesis and store them in a custom binary format for ease of access at runtime. The files will be designed so that they can be accessed via the local file system or through a web server.

The data structure will be partitioned such that each level in the tree structure, for both the octree and icosatree, points are left in each node. The nodes will be split into a grid structure so all nodes at a specific depth are uniformly dense in order for the level of detail algorithm to only have to deal with screen area and not point densities within the visualization tool when determining how deep to traverse. The data will also contain information defining the number of points in the node, the number of child nodes, and the layout of the binary data.

Visualization Tool

The visualization tool will consist of a Java/OpenGL rendering system based off a rendering toolkit the author has developed. Geospatial navigation and terrestrial terrain will need to be added as needed for the visualization. The visualization will render a simple WGS84 projected globe, a spherical orbit navigator, the point cloud renderer that will support the different data structures implemented by the partitioning application, and the selection algorithm used by the user that will allow them to select objects, as the points that comprise them, from the scene.

The visualization tool will render each item in the scene as its own scene element. The camera navigation will be based on a geospatial anchor point and a azimuth/elevation/distance offset from the anchor. A local origin will offset the scene component's local coordinate system for floating point precision reasons and each scene element will update its own position based on this local origin.

Rendering Component

The rendering component will read the root node from the dataset (locally or over the web). It will then use the bounding volume of the node to determine if it should be rendered or not and if its children should be queried. It will then use a screen-space level of detail algorithm to determine how deep to traverse; this depth will be tunable based user preference.

The octree and icosatree will be supported initially by the rendering component. The octree will be as defined in the background section. Each node will contain a set number of cells where points will be stored. Any points that overflow a cell will be passed down to a child node. The same will be true for the icosatree, however, the tree will handle an altitude dimension to split nodes into smaller pieces in order to handle vertical data better. The icosatree will initially have twenty sides and each side will split into four triangles. Therefore, the initial level of the tree will need to be handled slightly different than the subsequent levels. I plan on testing out a few different ideas for how best to split the tree by altitude but two initial thoughts were to initially split the tree into a number of larger layers, each with its own icosatree or split each starting triangle of the root node into four triangles along the face and two slices along its altitude. The first would simplify culling a large portion of the tree if any layer were empty but has the downside in the layers would need to be relatively thin to give a useful cell size and would not scale well as the tree went deeper. The second would allow the triangles to stay relatively uniform in all directions but would make traversing it a bit more involved.

Selection Algorithm

The selection algorithm will attempt to apply two separate point cloud selection algorithms to the same set of data in order to achieve a better result. The Screen-Space Operator Algorithm [1] is used to define surfaces inside the point cloud; this is useful for visualizing the walls of buildings and such. However, the algorithm itself doesn't handle selection of objects. The CAST algorithm [2] allows selection of more dense portions of a point cloud via two-dimensional mouse selection but it requires a dense cloud of points to handle selection. I propose that applying the former algorithm's surface creation with the selection capabilities of the CAST algorithm will allow the selection of objects in a scene by a user for manipulation or export.

Chapter 4: Evaluation

In order to evaluate my designs, the visualization system will have the ability to collect, analyze, and export runtime and rendering metrics for the application. The partitioning system used for the data will initially be a Cartesian octree, storing a random subset of points per octet and passing down the remainder to each of its eight child nodes. The system will then render each parent until a specified level of detail is reached. The visualization client will be written in Java and OpenGL (jogamp) using math and graphics libraries written by the author throughout his academic career. This visualization will display a portion of the point cloud by using frustum culling on the octree itself and applying the augmented depth culling surface selection algorithm on the visible point data. Then, the icosatree structure will be developed in the same manner and tested against the same selection algorithm. This testing will be done on a number of data inputs and pre-defined selection masks.

In order to measure the success of the algorithm, I will be comparing the accuracy of the original selection algorithm with my augmented algorithm; I will also compare both to a brute-force approach for accuracy. The computation time and rendering time for the visualization and selection algorithms will be compared. I will also compare the octreestored data to the icosatree-stored data in node access delay, bounding box retrieval, and average point weight of nodes to determine whether either is more or less sparse than the other for the same data set.

Chapter 5: Evaluation Outcomes

I will measure success or failure when I feel that the augmented algorithm has been implemented to the best of my ability and it either succeeds or fails in surpassing the original algorithms accuracy. The possible outcomes of this proposal are that my augmented algorithm works as intended and exceeds the original, my algorithm does not exceed the original, or my algorithm is much slower as to not be realistic in a real-time scenario.

Chapter 6: Deliverables

Deliverables will include a final report detailing the outcome of the algorithm developed along with any source code developed when testing its validity. The report will also include examples and comparisons with the baseline algorithm. Documentation for the source code will also be provided.

Chapter 7: Schedule

2016-06-01: Initial website posted
2016-06-08: Massive Point Cloud server complete
2016-06-15: 3D Visualization complete
2016-06-29: Original algorithms implemented
2016-07-13: Early algorithm implemented (first draft)
2016-07-20: Intermediate algorithm implemented
2016-08-03: Algorithm compelte
2016-08-10: Technical Report Complete
2016-08-17: Tentative Defense Date

Chapter 8: Current Status

The current status of my work is that I have done some research looking into previous research in the area as well as much of my own work as a full-time software developer on visualizing massive point clouds. I have also begun to write math and graphics libraries that I have been using over the past few years in academia.

References

[1] Ruggero Pintus, Enrico Gobbetti, and Marco Agus. 2011. Real-time rendering of massive unstructured raw point clouds using screen-space operators. In Proceedings of the 12th International conference on Virtual Reality, Archaeology and Cultural Heritage (VAST'11), Matteo Dellepiane, Sebastian Pena Serna, Holly Rushmeier, Luc Van Gool, and Franco Niccolucci (Eds.). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 105-112. DOI=http://dx.doi.org/10.2312/VAST/VAST11/105-112

[2] Lingyun Yu, Konstantinos Efstathiou, Petra Isenberg, Tobias Isenberg. CAST: Effective and Efficient User Interaction for Context-Aware Selection in 3D Particle Clouds. IEEE Transactions on Visualization and Computer Graphics, Institute of Electrical and Electronics Engineers, 2016, 22 (1), pp.886-895.<10.1109/TVCG.2015.2467202>.

[3] Wenzel K, Rothermel M, Fritsch D, Haala N. An out-of-core octree for massive point cloud processing. In: Workshop on processing large geospatial data Cardiff, UK, July 8th, 2014, {http://rs.tudelft.nl/~rlindenbergh/workshop/WenzelIQmulus.pdf}; 2014.

[4] Goswami, P.; Yanci Zhang; Pajarola, Renato; Gobbetti, E., "High Quality Interactive Rendering of Massive Point Models Using Multi-way kd-Trees," in Computer Graphics and Applications (PG), 2010 18th Pacific Conference on , vol., no., pp.93-100, 25-27 Sept. 2010 doi: 10.1109/PacificGraphics.2010.20

[5] Rico Richter and Jürgen Döllner. 2010. Out-of-core real-time visualization of massive 3D point clouds. In Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH '10). ACM, New York, NY, USA, 121-128. DOI=http://dx.doi.org/10.1145/1811158.1811178

[6] Sriram Krishnan, Christopher Crosby, Viswanath Nandigam, Minh Phan, Charles Cowart, Chaitanya Baru, and Ramon Arrowsmith. 2011. OpenTopography: a services oriented architecture for community access to LIDAR topography. In Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications (COM.Geo '11). ACM, New York, NY, USA, , Article 7 , 8 pages. DOI=http://dx.doi.org/10.1145/1999320.1999327

[7] Yingxin Wu, Masahiro Takatsuka, Spherical self-organizing map using efficient indexed geodesic data structure, Neural Networks, Volume 19, Issues 6–7, July–August 2006, Pages 900-910, ISSN 0893-6080, http://dx.doi.org/10.1016/j.neunet.2006.05.021.

[8] Michael Lee, Hanan Samet, Traversing the Triangle Elements of an Icosahedral Spherical Representation in Constant-Time, In Proc. 8th Intl. Symp. on Spatial Data Handling, Vancouver, Canada, July 1998, pp. 22–33

[9] Denis White, Global Grids from Recursive Diamond Subdivisions of the Surface of an Octahedron or Icosahedron, Environmental Monitoring and Assessment September 2000, Volume 64, Issue 1, pp 93-103

[10] By David Eppstein - self-made; originally for a talk at the 21st ACM Symp. on Computational Geometry, Pisa, June 2005, Public Domain, https://commons.wikimedia.org/w/index.php?curid=2489019

[11] By WhiteTimberwolf, PNG version: Nü - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=9851485

[12] By KiwiSunset at the English language Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=16242249

[13] By MYguel - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4535483

[14] By User:DTR - Vectorisation of Image:Icosahedron.jpg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2231553