Natural Resources Canada
Ressources naturelles Canada

The GeoHashTree: a multiresolution data structure for the management of point clouds

N. Sabo, A. Beaulieu, D. Bélanger, Y. Belzile, and B. Piché

Geomatics Canada

Technical Note 4

2014

Canada

Geomatics Canada

Technical Note 4

The GeoHashTree: a multiresolution data structure for the management of point clouds

N. Sabo, A. Beaulieu, D. Bélanger, Y. Belzile, and B. Piché

2014

ISSN 1914-4229
Catalogue No. M103-1/4-2014E-PDF
ISBN 978-1-100-23098-6
doi:10.4095/293383

A copy of this publication is also available for reference in depository libraries across Canada through access to the Depository Services Program's Web site at http://dsp-psd.pwgsc.gc.ca

This publication is available for free download through GEOSCAN (http://geoscan.ess.nrcan.gc.ca).

Cette publication est aussi disponible en français.

Recommended citation

Sabo, N., Beaulieu, A., Bélanger, D., Belzile, Y., and Piché, B., 2014. The GeoHashTree: a multiresolution data structure for the management of point clouds; Geomatics Canada, Technical Note 4, 12 p. doi:10.4095/293383

Critical reviewer

J. Brodeur

Authors

N. Sabo (Nouri.Sabo@RNCan-NRCan.gc.ca)
A. Beaulieu (Alexandre.Beaulieu@RNCan-NRCan.gc.ca)
D. Bélanger (David.Belanger@RNCan-NRCan.gc.ca)
Y. Belzile (Yves.Belzile@RNCan-NRCan.gc.ca)
B. Piché (Benoit.Piche@RNCan-NRCan.gc.ca)
Centre for Topographic Information – Sherbrooke
2144, rue King Ouest
Sherbrooke, Quebec
J1J 2E8

Correction date:

The GeoHashTree: a multiresolution data structure for the management of point clouds

N. Sabo, A. Beaulieu, D. Bélanger, Y. Belzile, and B. Piché

Sabo, N., Beaulieu, A., Bélanger, D., Belzile, Y., and Piché, B., 2014. The GeoHashTree: a multiresolution data structure for the management of point clouds; Geomatics Canada, Technical Note 4, 12 p. doi:10.4095/293383

Abstract: Over a number of years, lidar has become one of the major elevation-data acquisition technologies. However, the management of lidar data is extremely complex due to the phenomenal amount of data generated by this technology. To facilitate lidar-data management, this article proposes a GeoHashTree, which is a multiresolution data structure for managing different types of point clouds. The GeoHashTree is a hierarchical structure which can present data of regular or irregular distribution with various levels of abstraction. In addition to facilitating the management of point clouds, this structure reduces data storage space considerably, while also facilitating data access and handling. This article introduces the GeoHashTree and describes a prototype based on it.

Résumé : Depuis un certain nombre d'années, le lidar est devenu une des importantes technologies d'acquisition des données altimétriques. Cependant, la gestion des données lidar est très complexe, compte tenu de la quantité phénoménale de données que génère cette technologie. Pour faciliter la gestion des données lidar, cet article propose le GeoHashTree, une structure de données qui permet de gérer différents types de nuages de points, à de multiples résolutions. Le GeoHashTree est une structure hiérarchique qui permet de présenter les données de distribution régulière ou irrégulière sous différents niveaux d'abstraction. En plus de faciliter la gestion des nuages de points, cette structure permet de réduire considérablement l'espace de stockage de données tout en facilitant l'accès à ces données, ainsi que leur manipulation. Cet article présente le GeoHashTree, ainsi qu'un prototype basé sur cette structure.

INTRODUCTION

The ever growing popularity of elevation data and the use of these data in many fields demonstrate the need for a good knowledge of topography. In several countries, elevation data are among the data in greatest demand. This is the case in Canada, where elevation data account for 73% of all data downloads from the GeoBase portal (http://www.geobase.ca/). This growing popularity of elevation data is largely attributable to the impressive number of applications requiring this type of data. Nowadays, many applications used in flood-risk management, telecommunications, regional and urban planning, and various other fields cannot function without elevation data. We can also foresee a greater need in the future. For example, road-elevation data will soon be incorporated into intelligent-transportation and automobile-driving–assistance systems to reduce fuel consumption and greenhouse-gas emissions by 4 to 12% (L. Sugarbaker, G. Snyder, and D. Maune, 2012, presentation titled 'Results of the National Enhanced Elevation Assessment (NEEA)', given at the 12th International LiDAR Mapping Forum, Denver, Colorado, January 23–25, 2012).

In many organizations, elevation data from various sources, of varying degrees of precision, and from various eras coexist. There are data gathered several decades ago using conventional methods, and more recent and precise data gathered by means of modern-day technologies, such as lidar. Even if, at first glance, it might seem that only the most recent and most precise data should be kept, the reality is very different. In many regions and countries around the world, elevation-data coverage is often a patchwork of data of varying degrees of resolution and quality, gathered in various eras. Moreover, even where the coverage is complete and homogeneous, there is always a need for a variety of elevation models (e.g. surface models, terrain models, canopy models, etc.) and resolutions, because of the wide range of applications. This is because, for one thing, some applications do not need very high-resolution data; and for another, there is a growing demand for historical data, especially in the climate-change field. This coexistence of data from various sources leads to major integration problems, especially since in most applications, these data must be integrated with data other than elevation data (e.g. land-cover data).

In fact, the need for elevation data continues to grow and become more specialized, and the acquisition technologies for this type of data are becoming more and more numerous and accessible, as well as better performing. Although technological development in the past few years has resulted in new sensors that can gather very high-precision and high-resolution data, the generated data are becoming so increasingly voluminous and complex that serious problems are arising in terms of management and utilization of the data. For example, the current lidar sensors are able to acquire up to one million points in a single second, with centimetric vertical and horizontal precision. With one million points per second, one can imagine the quantity of data that can be gathered during major acquisition campaigns such as country-wide campaigns. The management and utilization of such a large quantity of data requires tools other than those traditionally used to manage vector and raster data.

To facilitate the management, utilization, and integration of elevation data of various types and resolutions, we propose in this article a new data structure called the GeoHashTree (GHT). The GHT is a multiresolution hierarchical structure that can manage regularly and irregularly distributed data (e.g. lidar data) at various levels of abstraction. Moreover, given its capacity to index all of the data, this structure can easily be integrated into database-management systems.

This paper provides an introduction to lidar data and their management, followed by a description of this new structure and the presentation of a functional prototype based on it. Test results are presented and discussed before the final conclusions.

LIDAR DATA AND THEIR MANAGEMENT

Lidar data

In the past few decades, lidar (**li**ght **d**etection **a**nd **r**anging) is one of the technologies that have radically changed the method used to acquire elevation data. Although it dates back to the 1960s, lidar technology is developing at breakneck speed. The types of utilization of lidar data are increasingly numerous and the sensors are becoming increasingly better performing, which continually gives rise to new challenges in terms of management and utilization of these data. Unlike traditional photogrammetry, where the elevation must be extracted from stereoscopic models, lidar provides the elevation directly, thus saving time. In fact, lidar is more than about elevation, because some of its attributes are used for purposes other than elevation. For example, through the use of lidargrammetry, some topographic features, such as infrastructure, can be extracted by using pseudostereo pairs created from intensity images and elevation.

Although on the one hand, this technology makes it possible to acquire very high-precision data, the storage, management, and utilization of these data nonetheless present considerable challenges because of their irregular distribution, their density, and the quantity of information they contain. Unlike raster data, lidar data are irregularly distributed data in the form of point clouds that are not organized logically, thus making it impossible to create a mathematical function like that of the raster grid that can predict the coordinates of the points. On the other hand, lidar produces a phenomenal quantity of points, each one of which has no fewer than a dozen attributes, often in different formats. For example, the time required to acquire one million points has decreased from more than 15 years (using surveying techniques) to a few seconds using lidar technology (S. Daniel,

2011, notes for the course 'LiDAR terrestre et aéroporté : principes et applications' given at Laval University, Québec, Quebec, on November 21, 2011). In addition, lidar is a mix of several models (e.g. surface and terrain models) owing to the multiplicity of returns. Although most often, two main models are managed (i.e. surface and terrain models), it should be taken into consideration that each return may represent a model that may be used for other applications. As mentioned above, the volume of data generated by these technologies, combined with the complexity of these types of data, make management and utilization of the data very difficult.

For a long time, a file-management approach has been used for storing and managing lidar data. However, a new approach has emerged over a number of years that makes it possible to manage lidar data in a database-management system. In the next sections, we will discuss the two main approaches to lidar-data management.

Lidar-data management using files

As stated above, lidar data are very complex because of the many attributes they have and the huge volume of data generated by this technology. Because their complexity makes them difficult to manage and manipulate, lidar data are usually managed differently depending on the users. For example, companies that conduct the surveys manage the data in the form of point clouds, while the end users often make do with digital elevation models that are an interpolation of the point clouds. The two types of data, i.e. point clouds and digital models, are usually managed by using files of different formats.

To store point clouds, ASCII or LAS file formats are generally used (ASPRS, 2012; Huber, 2011). The advantage of the ASCII format is that people can read it, that it can be read and edited by any text editor, and that it presents fewer problems of compatibility between platforms. However, this format is very voluminous and nonperforming when data are manipulated, which makes it sometimes unusable. Given the astronomical volume of point clouds, the use of the ASCII format is becoming less attractive as a method of managing a large quantity of data, which is why binary formats such as LAS have made their appearance.

Over a number of years, LAS has become the de facto standard for storing point clouds. LAS is a binary format that uses a predetermined set of fixed-size attributes. It is a high-performance format for data exchanges; unfortunately, in native LAS format, the points are not ordered, which makes arbitrary access to the data impossible (Graham, 2009). But arbitrary access to data is vitally important in the utilization of elevation data (e.g. to easily find the elevation of a point). The LAS format is therefore a compromise between efficient transmission of point clouds and flexibility (Graham, 2009). So, in order to fully utilize point clouds by means of spatial and attribute queries, they have to be indexed in order to facilitate arbitrary access.

Although some software editors offer their own indexing system, this approach poses serious problems.For one thing, the indexing of billions of points may require a huge amount of storage space, and for another, the index must be updated each time a change is made to the data. With respect to LAS format, the current specification does not allow the addition of new attributes to an existing file, because the format has a standard, fixed-size attribute structure intended solely for lidar-data management, which makes it less flexible, if not unusable, for the integration of data of various types and sources.

Furthermore, the LAS format does not have any data-generalization mechanism. The capacity to generalize lidar data is crucial for data visualization. Without this capacity, all of the data must be displayed regardless of the visualization scale, which is unimaginable for a vast spatial area (e.g. country-wide or region-wide scale), given the quantity of points to be displayed. For all of these reasons, although the LAS format performs well for production operations and in terms of its level of compression, it is not well suited to the data analysis and utilization process in circumstances where the data must be arbitrarily accessed, integrated, and updated.

The difficulty in managing and utilizing point clouds in ASCII or LAS format has often prompted some organizations to convert lidar points into regularly distributed data (e.g. raster data) in the form of digital models in order to facilitate their manipulation. Although regularly distributed data have their place in elevation-data management, such conversions are irreversible and do not allow the richness of the lidar data (i.e. precision and range of attributes) to be retained. In fact, the digital models only take elevation into account and leave out all of the other lidar attributes, despite their usefulness in other applications (e.g. RGB for visualization or intensity for the extraction of topographic features).

In brief, the current methods for storing lidar data in files have serious limitations in terms of managing, integrating, and utilizing various types of data. With a file method, it would be very difficult to set up an effective system for managing data covering a large territory because, on the one hand, many operating systems are limited in terms of the size of files and, on the other hand, partitioning the data into several files would require setting up a system that would be able to index the data, ensure continuity and integrity of the data, avoid data overlap, provide multiple, concurrent access to data, and so on.

To facilitate the management and utilization of point clouds and their integration with other types of data, one solution is to store them in a database-management system.

Modern databases already integrate the main types of spatial data (vector and raster), which would facilitate the integration and interaction of point clouds with these types of data.

Lidar-data management in a database-management system

In the past few years, considerable effort has gone into making it easier to integrate point clouds into a database in the same way as other types of existing data, such as vector and raster data, which are currently managed in a variety of databases (Nandigam et al., 2010; Arias Prado, 2011; Ott, 2012). There are several advantages to managing point clouds in a database: (1) the benefits that a database provides in terms of security, concurrent access, user management, scalability, management of updates, quick access through high-performance indexing and partitioning systems, cloud computing, and version control; (2) easier utilization due to the SQL language; (3) easier interaction with other types of data already stored in databases (e.g. vector and raster data); and (4) seamless integration between disparate data sets, combined with the possible use of an abundant range of operating tools (Graham, 2009). The use of files to manage point clouds is viable for projects that involve limited space coverage. However, in the case of large-scale coverage on a regional or national scale (for example, Canada's landmass of more than 10 million km^2), the use of a database is more appropriate, given the above-mentioned flexibility and advantages, and also considering that modern-day databases already provide the basic infrastructure to facilitate large-scale coverage (e.g. extension mechanisms).

To store point clouds in a database, three principal methods may be considered: the single-point method, the multipoint method and the tile method.

The single-point method consists of storing each point in vector form (i.e. one record per point) and indexing them. Each point attribute constitutes a separate field. This is a very easy solution to implement because the point type and the tools capable of manipulating it are found in the majority of database-management systems. However, this method has serious limitations: the indexing of billions of points requires a commensurate amount of storage space, and updating of the index after changes are made is time-consuming.

The multipoint method consists of storing sets of points in blocks (in the form of binary large objects, or BLOBs) instead of storing each point individually. With this method, attributes and geometries are stored in different fields: a field for storing the geometries of the set of points in the block, in addition to as many fields as there are attributes. This method may be considered an optimization of the single-point method because it requires fewer saving operations and is consequently easier to index, resulting in better performance and enhanced storage space. However, this method requires the addition of a spatial-extent field to each block. This field makes it possible to spatially index the blocks in order to facilitate spatial queries. Moreover, the fact that the attributes are in blocks and dissociated from the geometry can make some operations more difficult, such as a spatial query combined with an attribute query. Since the points inside a block are not indexed, arbitrary access to the points inside a block is impossible, which may create performance problems in the case of large-size blocks.

The tile method is used to subdivide data into smaller size tiles and to store each tile in BLOB form. Unlike in the preceding method, the attributes and geometry form a whole. The tiles can be stored by making partial or complete use of an existing structure or format (e.g. LAS) in order to encompass the points and their attributes. To facilitate access to the data, the extent of each tile can be spatially indexed. This method has more advantages in terms of storage than the preceding two; however, it inherits all of the advantages and deficiencies of the tile format. For example, in the case of a native LAS format, although the tiles can be indexed, the points inside the tiles will not be indexed. Therefore, in order to facilitate access to the data, the size of the tiles must be kept to a strict minimum, and consequently there is an increase in the number of tiles, which brings us back to the problem of indexing.

Technically, some existing database-management systems have mechanisms for adding new types of data with which it is possible to interact directly. A good method of storing point-cloud data in a database should include the following features:

- allow easy updating of existing data;
- allow the integration of disparate data (different sources, resolutions, models, etc.);
- allow arbitrary access to data through spatial and attribute queries;
- facilitate analysis and utilization without incurring excessive cost in terms of storage and performance;
- allow data to be presented at several levels of abstraction (several resolutions). This characteristic is very important when visualizing huge volumes of data, such as lidar data;
- facilitate the conversion of one type to another (e.g. convert point clouds into raster data and vice-versa).

Data-file management is one method that can accommodate a situation where the area covered by the data is quite small and interaction is limited. To manage data covering a large expanse where there is much interaction and many operating constraints (e.g. multiple access), a database-management method is more appropriate. However, given the nature of lidar data and the need to integrate them with other types of data (which may have lower resolution and

precision), the current database-management methods are limited (an indexing problem can result from the high number of points, there is no data structure facilitating data integration, etc.); hence the need to develop a more suitable data structure.

To facilitate the management and utilization of point clouds of various types and resolutions (including lidar data), we developed a new structure called the GeoHashTree.

GEOHASHTREE (GHT) STRUCTURE

The GeoHashTree (GHT) is a generic data structure that is used to store, access, and manipulate point clouds, as well as to integrate data of various types (irregularly distributed data, such as point clouds, and regularly distributed data, such as raster data) and various resolutions into the same structure, and allow interaction between these data. The GHT is based on the Geohash (Wikipedia, 2012), a comprehensive geocoding system in the public domain based on a hashing function that subdivides the Earth's surface into a hierarchical grid. It is used to encode the co-ordinates (longitude, latitude) into a chain of characters. The GHT therefore provides a geospatial-data structure to which can be attached an unlimited number of attributes from various types of data with different resolutions without duplicating the co-ordinates of each source. For example, lidar data can be integrated with other elevation data that have lower resolution. This can be done without adding undue extra load because, although the two types of data are different, only the co-ordinates of the higher resolution data are retained in Geohash format, whereas in the case of the lower resolution data, only the attributes are used. So regardless of the types of data that are mixed together, a single geospatial-data structure in Geohash format is used, which results in a huge gain in terms of storage space when several types of data are integrated. Geohashes are generated only where there are data. Moreover, because the geocoding in Geohash format is also a spatial index, no spatial index for the data is required, while arbitrary access to all of the points is permitted.

One of the great advantages of the GHT is its flexibility. It makes it possible to convert point clouds or any other 'XY-attribute' type of data into a hierarchical structure where each point is indexed as a result of the GHT's tree structure and is presented in a multilevel format. The number of levels varies according to the density of the information and the precision of the co-ordinates to be stored. Thus, even irregularly distributed data such as lidar data are represented in pyramid format, similar to that of raster data. This pyramid representation of the data combined with the indexing of all the points of a GHT helps to facilitate the interpolation of the point clouds in order to convert them into raster data. The graphic in Figure 1 is a multilevel representation of a single attribute with variable resolution in a GHT (each attribute of a GHT can be multiresolution and represented in a multilevel format). In general, three main steps are required to convert points into raster data: (1) creation of the grid; (2) selection of points inside each cell; and (3) generalization of the selected data within each cell by applying a mathematical function (e.g. inverse distance weighting). However, in a GHT, the structure itself is a grid and we know all of the points within each cell. Furthermore, in each node of a GHT, several statistics related to each attribute, such as minimum, maximum, and average values, are present. And these statistics are often the same as those used to generalize the values of a cell during interpolation of the lidar data.

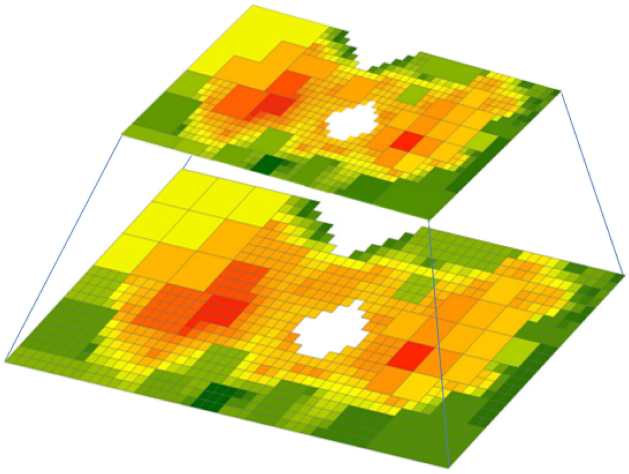


Figure 1. Multilevel representation of a variable-resolution attribute in a GeoHashTree.

Moreover, this same pyramidal GHT structure will help make it easier to visualize point clouds. Given the phenomenal number of points in lidar data, being able to visualize all of the points at all scales cannot be considered in the case of a large area. With the GHT, it is possible to envision displaying point clouds at the most appropriate structure level for each scale. Thus, at smaller scales, generalized data (e.g. minimum, maximum, or average) will be shown and as the user zooms into the data, more detailed levels will be displayed.

There are four main stages in storing data in a database in GHT format: encoding of the co-ordinates, creation of the tree, optimization of the tree, and storage of the tree in a database.

Encoding of co-ordinates in Geohash

The first stage in creating a GHT starts with the encoding of the geographic co-ordinates (longitude and latitude) of the points as Geohash. To create the Geohash for a point (e.g. −73.5, 45.4), the space is subdivided iteratively and bits are allocated in accordance with the quadrant in which the point is located. The subdividing begins with the planetary co-ordinates ([−180, 180] and [−90, 90]). For example, the result for the first subdivision of our point will be 01,

because it is located in the upper-left quadrant (*see* Fig. 2). The result of encoding our point in binary code will be 011100001000101010110010111. This number is then converted into a decimal number, which, in turn, is converted into characters by using a base 32 conversion table. Thus the final result will be f25dr.

Geohash is a comprehensive system that uses arbitrary precision mechanisms. The number of characters determines the precision of the co-ordinates, and the gradual elimination of characters from the end of the character chain allows the precision to be reduced (e.g. 6gkzwgjzn820 for −25.382708, −49.265506 and 6gkzwgjz for −25.383, −49.266). The precision of such an encoding system can be greater than a femtometre (10^{-15} m). The lidar data can be amply represented by a Geohash with 15 character resolution, which corresponds to a metric precision of about 10^{-7} m.

Usually, in a Geohash-based system, the more the Geohash prefixes for two locations resemble one another, the more they are spatially close to each other; for example, 6gkzwgjz (−25.383, −49.266) and 6gkzmg1w (−25.427, −49.315). This property makes it possible to use the Geohash itself as a spatial-data indexing system (simple sort). However, this property is not always complied with (two spatially close points located on either side of a subdivision line during the hashing process can have different Geohashes); so it is necessary to implement neighbourhood-based strategies to support the selection of Geohashes. Thus, the specific selection must always be based on nine Geohashes, i.e. the Geohash concerned and the eight others around it. This strategy is easy to implement, especially when it is known that the Geohash sort in alphabetical order follows a Z pattern.

There are other similar encoding systems, such as the HHCode (Helical Hyperspatial Code) developed by the Canadian Hydrographic Service, which is used to encode data in binary form (Varma et al., 1990). Geohash was selected because it is in the public domain and has already been implemented in several databases (e.g. PostgreSQL and MySQL) and Open Source libraries in various existing languages (C, Java, Python, JavaScript, etc.). Although Geohash is a system in which the co-ordinates themselves can be used as an index, several disadvantages result when it is used that way, such as high storage costs and poor performance. To minimize these disadvantages and obtain other advantages, such as the multilevel aspect or the capacity to integrate several attributes in a multisource data structure, we developed the GeoHashTree (GHT) from Geohash.

GeoHashTree construction

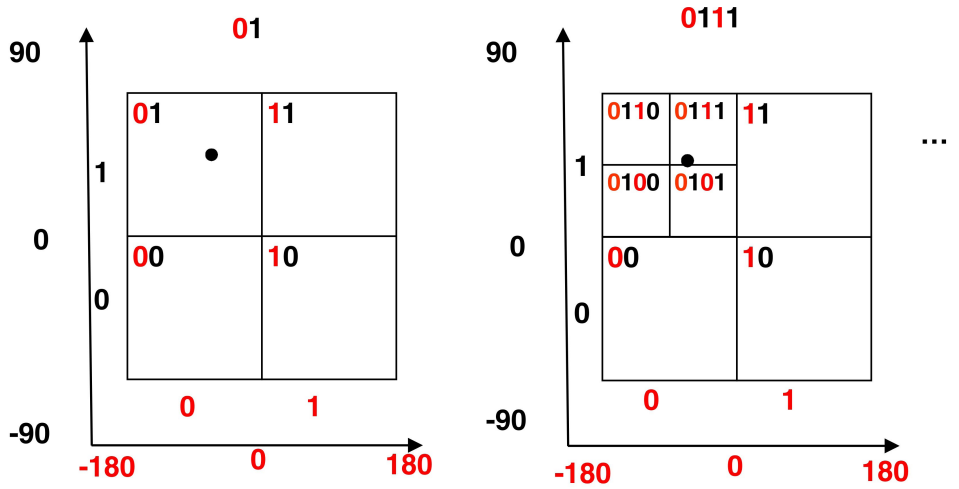
The GHT is an inverted tree in which all of the children of a node share the same Geohash prefix and the leaves contain a list of attributes (Fig. 3). Thus, the points that are spatially close to one another will share the same parent. The GHT is created from Geohash generated in the preceding step. Statistics such as the minimum, average, and maximum of certain relevant attributes can be attached to each node, as necessary. The tree structure supplemented by these statistics gives the GHT a multiresolution character (comparable to raster-data pyramids). When a new point is inserted into the structure, the Geohash of the new point is compared with the Geohashes of the existing points. Based on the result of this comparison (proximity), the new point is inserted in the right location in the structure. In a GHT, each attribute has a name and a type, which allows an almost unlimited number of attributes for each point. For example, several elevations from various sources can be stored on the same point (e.g. lidar, SRTM (Shuttle Radar Topography Mission), spot elevations, etc.), and data other than elevation (e.g. rainfall, multiband imagery) can even be inserted into the same structure.

When a point with lower resolution is added to a GHT, it inserts at a level in the tree corresponding to its resolution (which depends on the length of its Geohash string), because each level of a GHT corresponds to a given resolution. The correlation between the length of a Geohash string and the resolution is attributable to the fact that the longer the Geohash string, the greater the number of subdivisions made to create it. Therefore, the choice of the length of the Geohash string of a point determines the resolution at which it is stored. For example, a Geohash string 15 characters long for a point located at the Equator represents a resolution of about 1 mm, while for a point 10 characters long, the resolution is about 1 m.

GeoHashTree optimization

Although the preceding step involves a certain degree of optimization because of the tree structure that allows partial sharing of Geohash prefixes, the amount of storage space is still high, so it is necessary to optimize further. To do that, the attributes of each point are restructured. When freshly created, each GHT leaf has all of the attributes of the point; however, it is clear that within a GHT, some attributes are recurring and others are stored in unreasonably sized types (e.g. 123.5 stored in double precision). These two situations are used during optimization. Depending on the attributes, two types of optimization are applied (the two methods are not exclusive): migration of attributes and change of type.

- ***Migration of attributes*** is a form of optimization applied to recurring attributes, such as the number of returns or the classification, in the case of lidar data. When the value for a given attribute in a node is the same for all of the child nodes, this attribute is stored solely at the level of the parent node. For example, when all of the leaves of a GHT have the same value for an attribute, this attribute will not be stored in the leaves. It will be represented at the level of the parents or perhaps even at the root level. In the specific case of lidar data, many attributes lend themselves well to this form of optimization.



Result in binary form:
01110001000101011001011 1

In decimal form:
01110-00010-00101- 01100-10111 = 14-2-5-12-23

In characters (using the conversion table):
14-2-5-12-23 = f25dr

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Base 32	0	1	2	3	4	5	6	7	8	9	b	c	d	e	f	g
Decimal	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Base 32	h	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z

Figure 2. Geohash coding.

Geohash	Z	C	T	A	I	N	R	P	E	D	U	r	g	b
f2j 2prt 49pjz	251.15	2	78494.953266	12	20	2	2	1039	0	1	0	0	0	0
f2j 2prt 031tc	251.78	1	78494.953293	12	253	1	1	1039	0	1	0	0	0	0
f2j 8025 pr1p2	255.98	1	78494.953358	13	255	1	1	1039	0	0	0	0	0	0
f2j 8025 1pnz6	255.31	1	78494.953463	13	76	1	1	1039	0	0	0	0	0	0

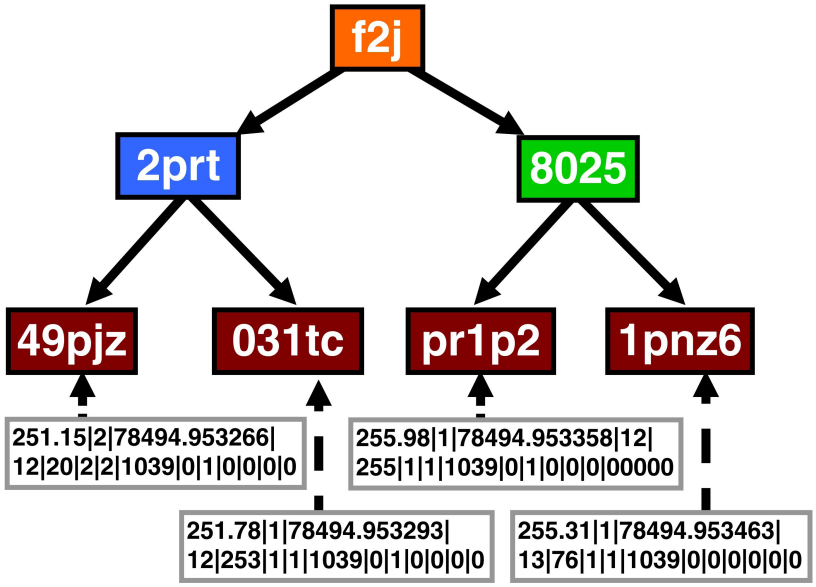


Figure 3. GeoHashTree structuring.

- ***Change of type*** is a form of optimization applied to attributes stored in large-size data types (e.g. floating point, double precision, etc.). This optimization makes it possible to change an attribute type in order to minimize the amount of storage space it takes up, if necessary. For floating-point–type attributes, a resolution is used to convert them into integers. For each attribute, a single resolution is used for all points in the GHT. (For example, the value of 123.56 will be 12356, using a resolution of 0.01.) Then, for each attribute to be optimized, minimums and maximums are calculated. For each point, the difference between the value of its attribute and the minimum for this attribute is calculated. By analyzing this difference, the most appropriate type for representing the attribute for this point is determined. For example, if the value of an attribute of a point is 123.56 in double precision, and if the resolution of this attribute is 0.01 and the minimum value for this attribute is 122.35, the interval will be (123.56 – 122.35)/0.01 = 121. In this case, the interval may be stored as an unsigned-character–type attribute that can take on a value of 0 to 255 and has a size of one byte. Therefore, this attribute will be stored in the form of one byte instead of eight for the initial data in double precision. The initial type and statistics (minimum and maximum) of an attribute are stored at the level of the tree — therefore only once — and used for all points in the GHT. The use of the resolution also makes it possible to control the degree of optimization where a high degree of precision is not required (optimization with loss). Figure 4 demonstrates the principle of GHT optimization.

Storage of GeoHashTrees in a database

Once created and optimized, the GHT can be stored in a binary-string field in a database, in which case it must be serialized beforehand. When storing the GHT in a database, the root Geohash can be used as a spatial index. Thus, an additional field may be added to receive the root Geohash of each GHT in order to avoid searching unnecessarily in the binary fields during queries. Because of the Geohash property according to which the more the prefixes in the index for two locations resemble one another, the closer they are spatially, this column can become the spatial index of the table once the content is sorted. Moreover, this same Geohash property is used to partition the database when necessary (horizontal partitioning). To facilitate queries, the root of each GHT contains general statistics for each attribute (minimum and maximum). When attribute queries are made, these statistics help to decide whether or not the GHT should be considered and deserialized.

PROTOTYPE

Prototype architecture

Based on the structure outlined above, a functional prototype was developed in C++. This prototype can be used to create GHTs from data in various formats and to store them in a PostgreSQL database (PostgreSQL, 2012). This prototype has three main components: a ‘reader/writer’ module, a GHT-creation module, and a ‘dumper/loader’ module.

- ***The ‘reader/writer’ module*** serves as the interface between the GHT creation module and the existing data formats (e.g. LAS and CSV) that is used to read and write in those formats. It therefore makes it possible to manipulate lidar data (in LAS format), raster data (in any format supported by the Geospatial Data Abstraction Library (GDAL)), and any XY-attribute–type data in text format. The ‘reader’ imports the data to be used to create GHTs, and the ‘writer’ exports GHTs in LAS or CSV format. The ‘reader/writer’ module uses the libLAS libraries (libLAS, 2012) to read and write LAS format, and GDAL for raster data format.
- ***The GHT-creation module*** is used to take data imported from the ‘reader’, reproject them if necessary, convert the XY co-ordinates of each point into Geohash, sort the points on the basis of the Geohash, and create the GeoHashTrees. The sorting groups the points together spatially in order to make point packets. Each packet will make up a GeoHashTree. To create the Geohashes, the Geohash library developed by Kato (2012) under an MIT licence was used. In addition to making it possible to encode and decode the Geohashes, this library is used to create the neighbourhood of any Geohash, thus making it easier to implement a data-selection strategy.
- ***The ‘dumper/loader’ module*** serves as the interface between the GHT-creation module and the database. It is used on the one hand to serialize GHTs created using the above module and to store them in the database, and on the other hand to select GHTs stored in the database (in response to a user query) and deserialize them. During the selection process, it uses an SQL query specifying both the extent of the area and the attributes to be selected. This module is based on the Libpq library included in PostgreSQL. Libpq is a C library used as an interface for the PostgreSQL database. A client program can thus use this library to connect to the database and submit SQL queries.

In addition to creating GHTs from point clouds and storing them in a PostgreSQL database, the prototype also makes it possible to interact with this database using spatial and attribute queries. For example, users can select all of the points included in a bounding box that have an elevation greater than 150 m, and store the XY co-ordinates, the elevation, and the number of returns in a CSV file. Of course, the attribute query can be made for any attribute of the data

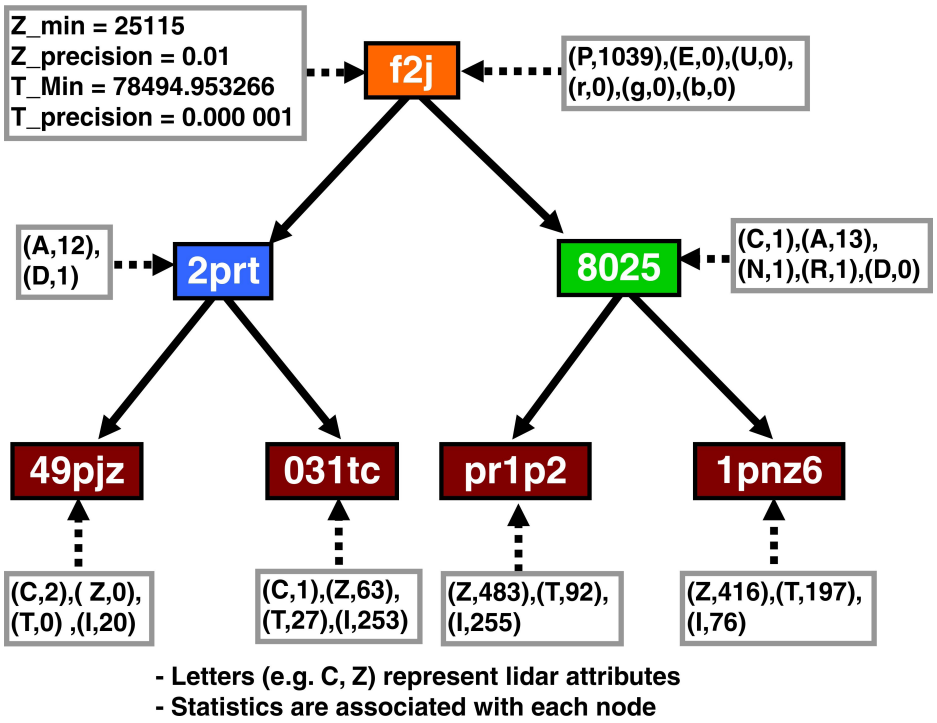


Figure 4. GeoHashTree optimization.

and the result may contain any attribute requested by the user. The simplified architecture of the prototype is shown in Figure 5.

Test data and materials

To test the prototype, lidar data in LAS format from the state of Indiana (U.S.A.) were used. The test data covered an area of about 23 km^2 (based on the bounding box) and contained 21 364 937 points in a 570.5 MB LAS file. Each point in the data set had 11 attributes. These data were converted into a GHT and stored in the PostgreSQL database. To make it easier to create the GHTs — in order to avoid a memory problem — the LAS file was split into six smaller files (about 100 MB each). All of the tests were conducted on a computer equipped with an i5-3570@CPU3.4GHz processor and a random access memory of 8 GB in Windows.

Selecting the size of GHTs and creating GHTs

In each LAS file, the XY co-ordinates of each point were converted into Geohash strings 14 characters long. This length made it possible to guarantee precision to the millimetre, which is amply sufficient because the initial lidar data had a centimetric precision. The result at this stage was a list in which each item was composed of the Geohash and all of the attributes of a point. Based on this list, we created a GeoHashTree for each set of points sharing the first seven Geohash characters (the prefix). The length of the prefix has a big impact on the number and size of the GHTs created, as well as on the GHT compression rate and performance. The choice of a prefix seven characters in length was based on the tests outlined in the graph in Figure 6. Based on these tests, it may be said that prefixes with a length of seven characters or less provide the best compression rate. It can also be said, based on the graph in Figure 7, that when the prefix is too short, the size of the GHT increases drastically, resulting in considerable use of memory (high memory footprint) that can lead to memory problems. However, when the GHT size is too small, a deterioration in the compression rate has been observed, because the optimization that was used was based on the sharing of attributes between various points, and therefore it is effective only where the number of points in a GHT is relatively high. For all of these reasons, we decided to choose a seven-character prefix. Of course, the prefix length and the number of Geohash characters are specific to our test data. For each type of data and type of use, these parameters must be selected on the basis of the data resolution and the desired performance and compression rate.

Because the initial data are in UTM co-ordinates, the procedure for creating GHTs also includes a previous conversion of the UTM co-ordinates into geographic co-ordinates, while preserving the same reference system (WGS84). Thus, 1687 GHTs were created and stored in the 9.0.1 PostgreSQL database, in bytea format, in the GHT column of the table. In addition, for each GHT, the root Geohash (seven characters long) was stored in a second field of the same record in order to facilitate spatial queries. The average time required to create, optimize, serialize, and store the 1687 GHTs in the database was about four minutes. This included the network

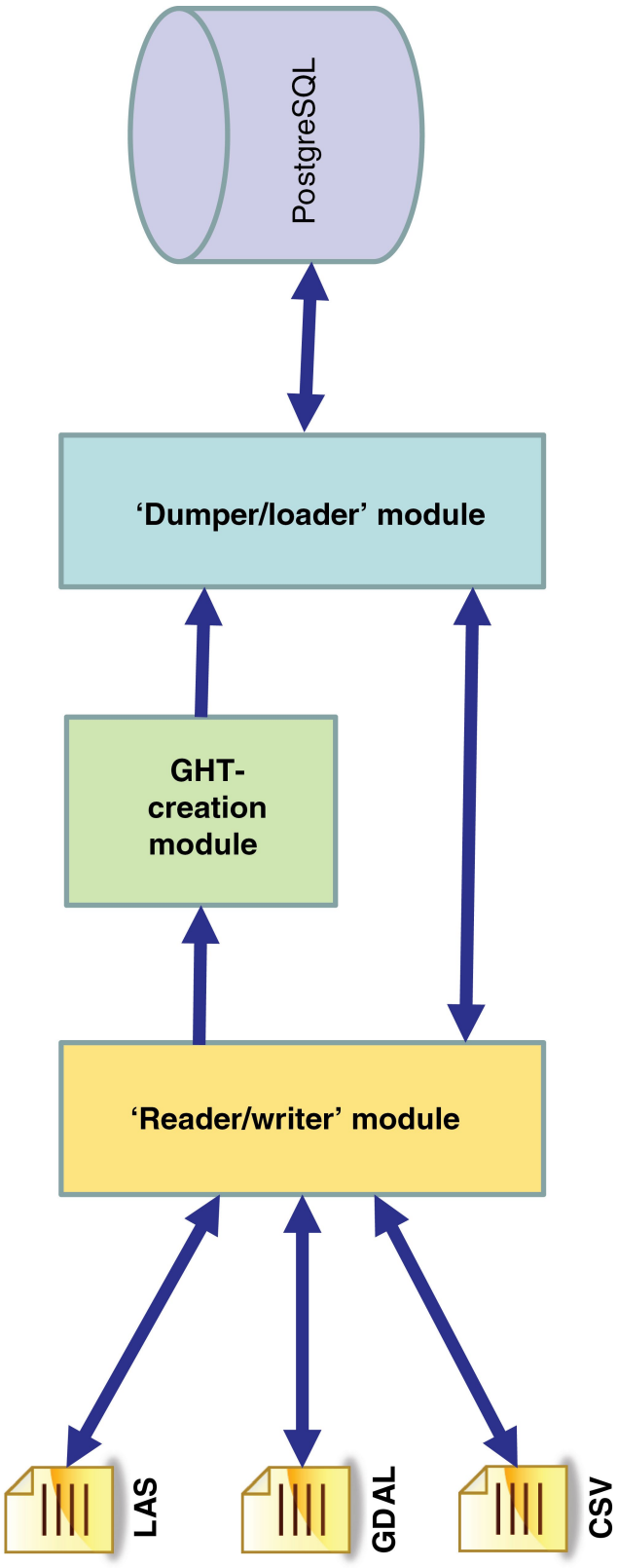


Figure 5. Simplified architecture of the prototype.

transfer time. Once stored in the database, the newly created GHTs took up only 321 MB, for a compression rate of 44% compared with the initial LAS file. As stated above, GHTs are not only a means to store lidar data; they can also be used to store point clouds of various types and resolutions and to have data at several levels of abstraction within the same structure. In such a structure, each point is indexed, thus facilitating arbitrary access to the data.

Operating tests

To test the utilization of GHTs stored in the database, spatial and attribute queries were made. Two spatial queries were used: (1) selection of data included in a large spatial extent (100% of the test data) and (2) selection of data included in a limited spatial extent (covering about 3% of the test data). Subsequently, each spatial query was combined with the attribute queries (number of returns R = 2 and elevation Z ≥ 309). For each combined query, the selected GHTs were converted into points and stored in a CSV file in X,Y,Z format. Table 1 shows the results of the tests. For example, the amount of time required to select and convert all the points with a number of returns equal to 2 into a CSV file (X,Y,Z) was 17 seconds. This is a shorter amount of time than that required to make the same query for a LAS file stored in the same network using the las2las application (27 seconds to select the data using las2las plus 5 seconds to convert selected points into CSV using the las2txt application). The objective of these tests was not to compare the performance of GHTs in relation to LAS; it was simply to obtain an idea of GHT performance.

CONCLUSIONS AND DISCUSSION

In this article, we have described the GeoHashTree, an innovative structure for multiresolution and multisource data that makes it possible to store point clouds or any other type of point data in a database, while making it easier to manage and utilize them and to integrate them with other types of data. Although the key characteristic sought during the development of this structure was flexibility, the GeoHashTree also optimizes storage space and performs very well when interacting with the database. The current set-up is a compromise between storage-space optimization, performance optimization and structure flexibility. Despite this compromise, GHTs provide acceptable performance and compression rates, especially since a large quantity of data must transit through the network with the usual latency. One of the advantages of this approach is the flexibility and versatility of the solution. This flexibility helps to manage data from various sources within the same structure.

Given the results of the tests and the performance of existing systems for storing point clouds in files (e.g. LAS files), we believe that the developed method provides excellent performance, especially given that this method (in terms

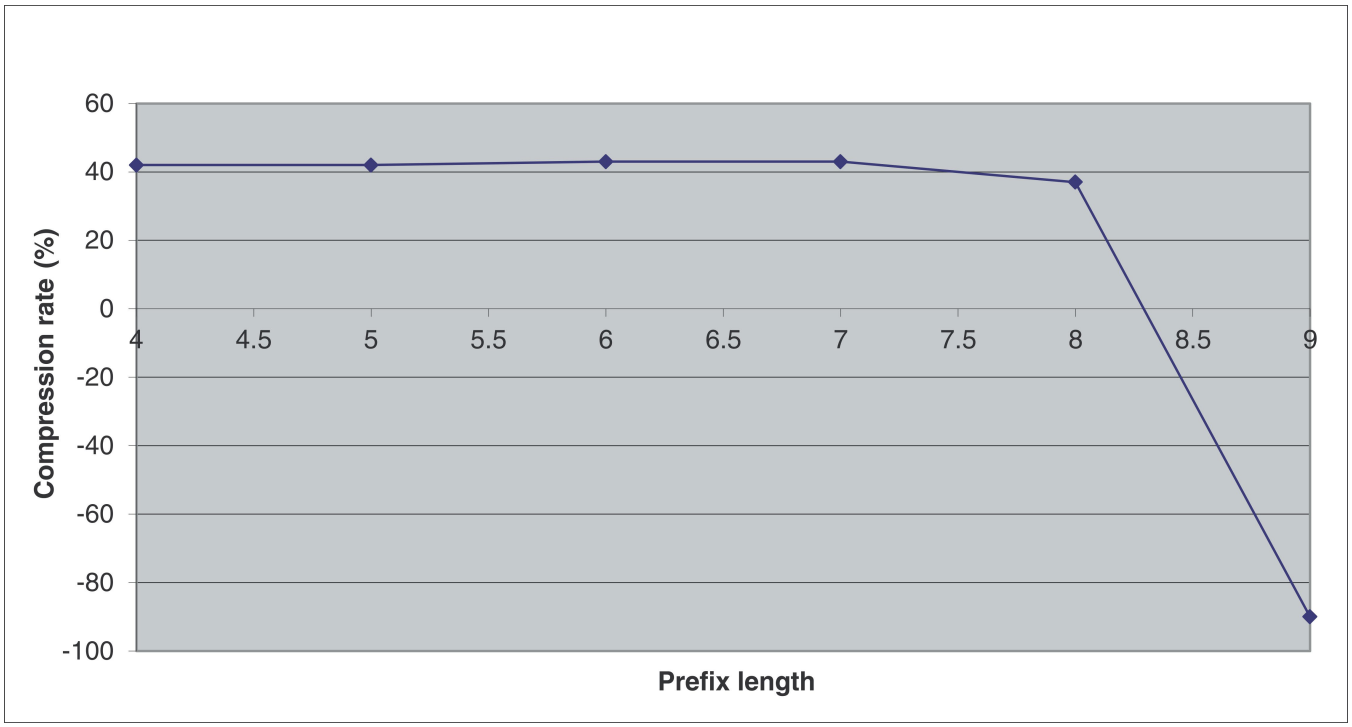


Figure 6. Compression rate (relative to LAS) versus prefix length in a GeoHashTree.

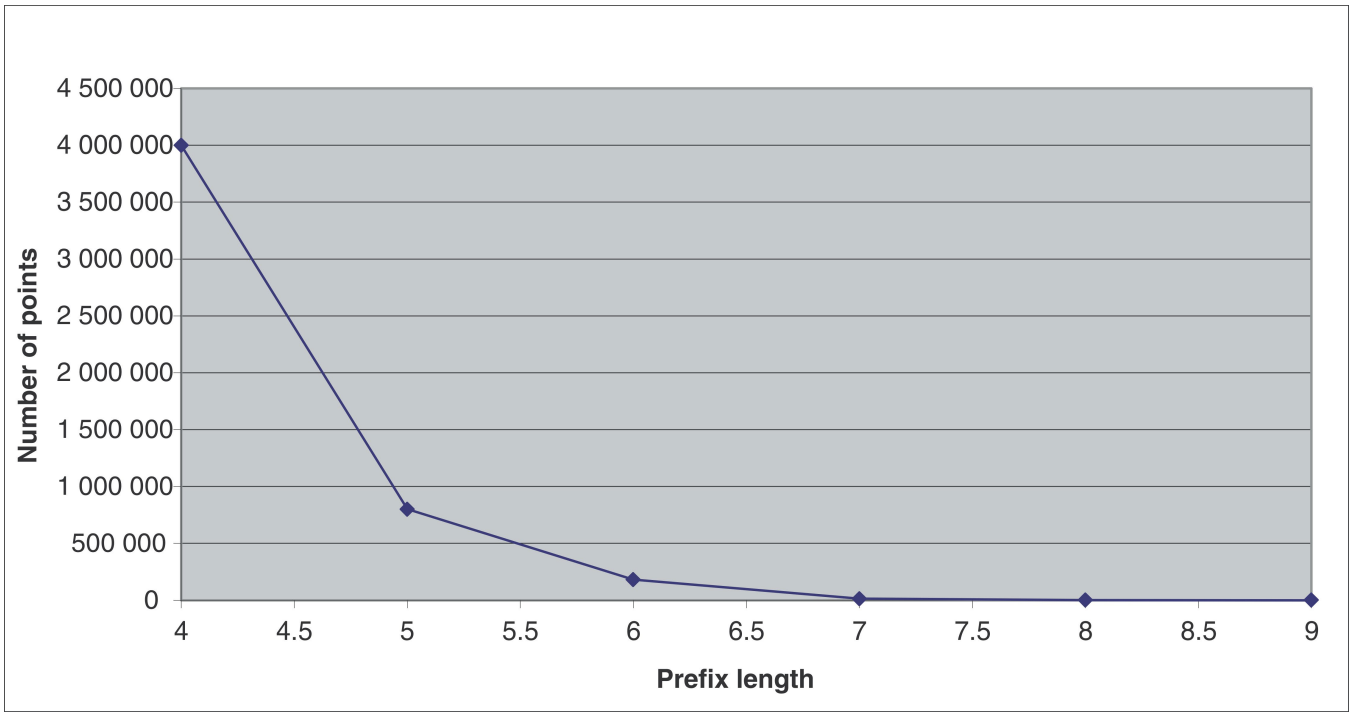


Figure 7. Average number of points in a GeoHashTree.

Table 1. Results of performance tests

Select data included in whole data set			Select data included in small area		
'Where' condition	Number of points	Processing time (s)	'Where' condition	Number of points	Processing time (s)
All	21 364 937	73	All	580 150	2.1
R = 2	617 214	17	R = 2	5 079	0.4
Z≥309	1 072	20	Z≥309	2	0.3

of storage and access speed) can still be improved. In fact, in the way it is currently implemented, the database is only used to store data. There is no mechanism for prefiltering data attributes when submitting queries because the GHTs are not integrated as a type in the database. The consequence of such a situation is that for each combined spatial and attribute query, all of the GHTs included in the selection area must be transferred to the client side, even if many of these GHTs do not meet the attribute criteria and if the GHTs have header metadata that help us determine whether or not they meet the selection condition without their having to be completely deserialized. For example, for the selection of data included in a vast spatial extent (100% of the test data) where elevation $Z \geq 309$, all of the GHTs (1687) had to be transferred to the client side even if, in reality, only four of them contained points responding to the attribute query. Storage improvements are still possible because a base 32 table is currently used to encode GeoHashes, which results in an overall cost of 3 bits per Geohash character, or up to 37% per Geohash of a point. This overall cost can be minimized by encoding the Geohashes using a base 64 table or by storing them in binary form (without converting into characters). In addition to improving storage capacity, this approach would also improve performance.

Currently, the prototype uses only data expressed using geographic co-ordinates. However, the Geohash encoding principle may be viewed as a quadtree, which would make it possible to easily use nongeographic co-ordinates. To do this, all that would be necessary would be to store in each GHT or in a separate table some metadata concerning the limits of the spatial extent of the mapped territory (e.g. xmin, xmax, ymin, ymax of the UTM zone). To facilitate interaction between various GHTs, they should be in the same system, which means that their Geohashes must be generated from the same spatial extent. Because GHTs present the data in a multiresolution, multilevel grid format in which all of the points are indexed, this structure would make it possible to facilitate the conversion of irregularly distributed data into regularly distributed data (e.g. raster data). Moreover, this multilevel aspect of GHTs could make it easier to visualize point clouds. An M.Sc. thesis project is underway in collaboration with Laval University to develop such a capacity to visualize GHTs by making use of the structure's multilevel aspect. The GHT structure code is currently available in OpenSource (https://github.com/pramsey/libght) and in pointcloud (https://github.com/pramsey/pointcloud), a PostgreSQL extension for the management of point clouds.

ACKNOWLEDGMENTS

The authors wish to thank the GeoConnections program for its support of the National Elevation Project, within which the GeoHashTree structure was developed. The authors would also like to thank Herman Varma, formerly with the Canadian Hydrographic Service, for his wise advice.

REFERENCES

Arias Prado, D.A., 2011. Efficient LiDAR data bulk load in a PostGIS database; <http://ariasprado.name/2011/02/06/lidar-bulk-data-load.html> [accessed May 20, 2013].

ASPRS, 2012. LAS Specification, version 1.4–R12; available at <http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html> [accessed May 15, 2013].

Graham, L., 2009. Management of LiDAR data; Chapter 10 *in* Topographic laser ranging and scanning: principles and processing, (ed.) J. Shan and C.K. Toth; CRC Press, Florida, p. 295–306.

Huber, D., 2011. The ASTM E57 file format for 3D imaging data exchange; *in* Proceedings, SPIE 7864, Three-dimensional imaging, interaction, and measurement, 78640A, January 27, 2011. doi:10.1117/12.876555

Kato, L., 2012. Geohash library; <https://github.com/lyokato/objc-geohash> [accessed May 16, 2013].

libLAS, 2012. <http://www.liblas.org> [accessed May 16, 2013].

Nandigam, V., Baru, C., and Crosby, C., 2010. Database design for high-resolution LIDAR topography data; *in* Proceedings, Scientific and statistical database management, 22nd International Conference, SSDBM 2010, (ed.) M. Gertz and B. Ludäscher; Lecture notes in Computer Science, v. 6187, p. 151–159. doi:10.1007/978-3-642-13818-8_12

Ott, M., 2012. Towards storing point clouds in PostgreSQL; HSR Hochschule für Technik Rapperswil, Rapperswil, Switzerland, 29 p.

PostgreSQL, 2012. <http://www.postgresql.org> [accessed May 15, 2013].

Varma, H., Boudreau, H., and Prime, W., 1990. A data structure for spatio-temporal databases; International Hydrographic Review, Monaco, v. 67, p. 71–92.

Wikipedia, 2012. Geohash; <http://en.wikipedia.org/wiki/Geohash> [accessed May 14, 2013].

Geomatics Canada Project 362304NP23