A Novel Compression Algorithm for LiDAR Data

Ruoyu Du
Division of Computer Science and Engineering
Chonbuk National University
Jeonju, Korea
dury@chonbuk.ac.kr

Hyo Jong Lee
Division of Computer Science and Engineering
Center for Advanced Image & Information Technology
Chonbuk National University
Jeonju, Korea
hlee@chonbuk.ac.kr

Abstract—**Light Detection and Ranging (LiDAR) data compression is a critical research field for data processing during past few years because of the large storage size. When LiDAR has small number of data points, the surface generation represented by interpolation methods may be inefficient. This paper presents a compression algorithm to simplify the LiDAR data set, which is constructed based on the quad-tree structure, for fast preprocessing step. The related theory and the methods to make it reality are discussed in detail respectively. Some conclusions were induced from tests: the method presented in this paper can provide multiple compression ratios based on needs, and guarantee the accuracy of LiDAR data for each case.**

Keywords- LiDAR data; quad-tree; data compression.

I. INTRODUCTION

LiDAR is a new type of active light sensing device, which has developed rapidly in last few years. Because of its high frequency sampling, excellent performance at low altitude detection and strong anti-interference capability, it has been widely applied in many high-tech fields such as environment monitoring, aerospace, communication, navigation and orientation. Low flying aircrafts equipped with modern laser-range scanning technology collect precise elevation information for entire cities, counties, or even states [1-3]. One of the major drawbacks for a wider application of LiDAR used to be the high cost of data acquisition. However, this problem has been greatly alleviated by thrilling developments in hardware [4]. Although LiDAR data has become more popular for average users, how to effectively process the raw data and extract useful information is still a big challenge. Compared to image processing, LiDAR data processing is gaining importance in the field of remote sensing and geographical information system (GIS) for generating DEM (Digital Elevation Model) [5], TIN (Triangulated Inverse Network), DTM (Digital Terrain Model) and many other surface models. LiDAR systems or technologies are able to collect dense and accurate information from surfaces or terrain. Light Distance and Ranging, along with airborne laser swath mapping data, is quickly becoming one of the hottest tools in the geosciences for studying the earth’s surface. LiDAR data more accurately create DEM than any other software.

LiDAR data processing is divided into two steps: First of all, Data pre-processing: After the data acquisition it is necessary to process data for further analysis. In data pre-processing techniques, data must be filtered for noise. The data must be corrected through the global positioning system survey. Pre-processing computes the collected point coordinates from independent parameters such as scanner position, orientation parameters, scanner angular deflection, and laser pulse time. Most LiDAR file formats are ASCII. ASCII file formats represent x, y and z values. X, y and z values contain latitude, longitude and elevation data. For data pre-processing, latitude, longitude and elevation data must be converted into a local referenced coordinate system. Second, Data post-processing: After pre-processing, LiDAR data is ready for post processing. To derive final products like DEM (Digital Elevation Model), DTM (Digital Terrain Model), and TIN (Triangulated Irregular Network) for further analysis, post processing is important. With the help of LiDAR data surface modeling can be generated [5-8]. These surfaces are derived using skilled technical staff, ArcView and 3D Analyst. For generating these 3D models current aerial photography, imagery, and existing maps are required with high levels of accuracy. Current images or photography should be geo-referenced. If the current images are not geo-referenced then geo-referenced 4k digital CCD imagery collected by a calibrated camera or lens must be acquired. Post processing processes are most important and have a vast influence in fields like 3D modeling, etc.

As we know, LiDAR data contain a greater amount of redundant data (such as the building of a certain facade, there only have the edge information is sufficient, but the LiDAR data based upon the setting sampling interval to provide elevation of the edge and internal information on all the sample points, which redundant information to the model or model feature extraction is almost no help). Therefore, in the pre-processing of LiDAR data, to select or design the appropriate compression method [9, 10] to simplify them is necessary. The current methods of data simplification mainly include two categories: 1) A simple re-sampling of the original data set according to a certain sampling interval (e.g., Interlaced, every other column sampling), there are obvious disadvantages to this method such as the fact that redundant information and the same non-redundant information are weakened, losing some of the useful information; 2) Homogeneous region of the merger and re-sampling. The original data set within a homogeneous area (such as a the building facade) is obviously recognized, there is a huge data redundancy through a set of algorithms, the merger of the region, with a small number of sample points (or synthetic sampling points) to replace the original data come from a number of sampled data in order to achieve data reduction purposes [7, 8]. This approach has a clear advantage, that is, data re-sampling according to the actual situation of the object adaptive changes, not only achieving the purpose of data reduction, but also effectively preserving the useful features of information.

This paper proposes a data compression system for LiDAR based on quad-tree structure, which can be used in the LiDAR data pre-processing procedure. Section II introduces the quad-tree structure. Section III describes our proposed method. Experimental results and discussion are described in Section IV. Finally, a conclusion is summarized in Section V.

II. THE RELATED THEORY

Quad-tree is a class of hierarchical data structures which contains two types of nodes: non-leaf node and leaf node, or we can call them internal node and external node [11-14]. It is most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions. Specifically, quad-tree is a good data structure for organizing pixels in a photo and for organizing computer graphics. The picture can be divided into quadrants, and each quadrant can be divided into four more. This can be repeated again and again until you reach the level of individual pixels. Although data stored in a quad-tree structure can require a lot of storage space compared to other methods of organizing data for computer graphics, the quad-tree structure has several advantages. First, you can delete the entire photograph or graphic in a single step by clearing the root node, which clears all of its children nodes, too. Second, you can quickly reduce the resolution in a photograph by simply clearing the final level of children nodes. This will thereby reduce the amount of storage space it requires. Finally, finding a particular area of the photograph for image manipulation is easier with the quad tree structure.

Nowadays, in the fields of computer graphics and image processing, a quad-tree is always regarded as a way of encoding data that enables one to reduce storage requirements by avoiding sub-dividing the same areas rather than storing values for each pixel [15]. At this point, a quad-tree is considered as a spatial index which recursively decomposes an image into square cells of different sizes until each cell has the same value. Currently, quad-trees can be classified according to the type of data they represent, including areas, points, lines and curves. More generally, quad-trees have been categorized by whether the shape of the tree is independent of the order in which the data is processed [16]. Some common types of quad-trees in this kind of classification are: region quad-tree, point quad-tree, edge quad-tree, and PR quad-tree. In the following section, we will outline these four types of quad-tree.

A. Region Quad-tree

The most studied quad-tree approach to region representations, called a region quad-tree, is based on the successive subdivision of a bounded image array into four equal-sized quadrants. If the array does not consist entirely of 1s or entirely of 0s (i.e. the region does not cover the entire array), then it is subdivided into quadrants, the sub-quadrants, etc., until blocks are obtained that consist entirely of 1s or entirely of 0s; i.e., each block is entirely contained in the region or entirely disjoint from it. The region quad-tree can be characterized as a variable resolution data structure. Fig.1 shows an example of region quad-tree structure. The right part of this figure is the tree representation of the spatial distribution, and was formed by starting at the top-left square in left part.

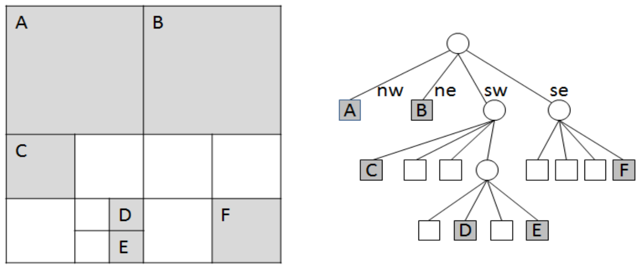


Fig. 1. Representations of a region quad-tree. Left part: Spatial, Right part: Tree.

B. Point Quad-tree

The point quad-tree is a marriage between a uniform grid and a binary search tree. Decomposition occurs whenever a bucket overflows. In the current implementation the bucket size is 4. The decomposition is done by splitting the bucket into 4 buckets along the x and y coordinates of the lowest order point in this bucket (the point that was inserted first to the bucket). The fields of the record type for this node comprise an attribute for information, two co-ordinates, and four compass points that can point to four children. The tree resulting from such decomposition is called a point quad-tree, which is shown in Fig.2.

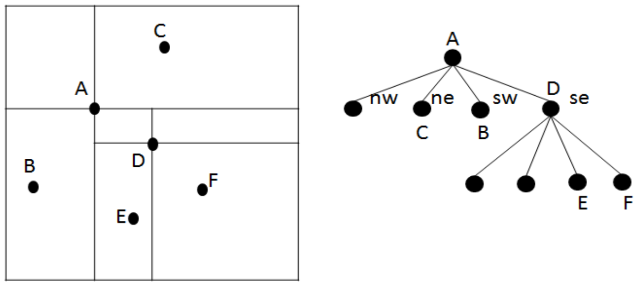


Fig. 2. Representations of a point quad-tree. Left part: Spatial, Right part: Tree.

C. Edge Quad-tree

The edge quad-tree [17] is based on the observation that the number of squares in the decomposition can be decreased by terminating the subdivision whenever each square contains no information or nothing more than a single curve which can be approximated by a single straight line. Applying this process leads to quad-trees in which long edges are represented by large blocks or a sequence of large blocks, however, small blocks are required in the vicinity of comers or intersecting edges. Of course, many blocks will contain no edge information at all.

D. *PR Quad-tree*

The PR quad-tree is a full four-way branching (4-ary) tree in shape. The PR quad-tree represents a collection of data points in two dimensions by decomposing the region containing the data points into four equal quadrants, sub-quadrants, and so on, until no leaf node contains more than a single point. The P in its name stands for "point" and the R stands for "region" (Samet 87) [18]. In other words, if a region contains zero or one data points, then it is represented by a PR quad-tree consisting of a single leaf node. If the region contains more than a single data point, then the region is split into four equal quadrants. The corresponding PR quad-tree then contains an internal node and four sub-trees, each sub-tree representing a single quadrant of the region, which might in turn be split into sub-quadrants. Each internal node of a PR quad-tree represents a single split of the two-dimensional region [18-20]. The four quadrants of the region (or equivalently, the corresponding sub-trees) are designated (in order) NW, NE, SW, and SE. Each quadrant containing more than a single point would in turn be recursively divided into sub-quadrants until each leaf of the corresponding PR quad-tree contains at most one point.

The function of the PR quad-tree is to contain the data over a finite plane. The PR quad-tree's function is constrained by two rules:

1. A cell with more than n data points must be divided into equal quadrants.
2. A divided cell must contain more than n data points to justify division.

This leads to the pattern of recursive divisions of cells shown in Fig. 3. These divisions change with the movements of the data points. As points cross the boundaries of the quadrants, divisions are either created (if a point moves into a quadrant containing n points) or eliminated (if a point moves out of a quadrant causing a divided cell to contain less than n points).

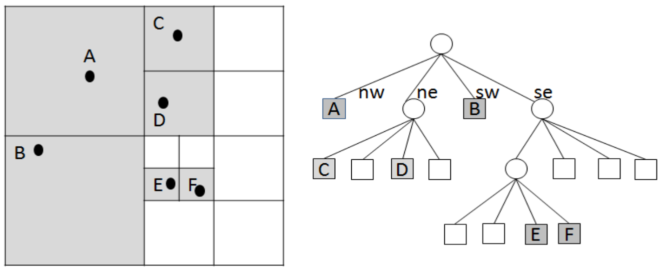


Fig. 3. Representations of a PR quad-tree. Left part: Spatial, Right part: Tree.

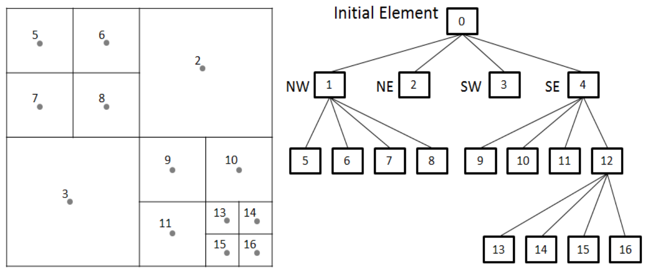
III. COMPRESSION METHOD

The LiDAR data is relatively huge and irregular in the dense pattern. The first goal is to compress point data without losing valuable information, as described before. It is necessary to reduce the size of the quad-tree before processing the data. Based on the point region quad-tree, we proposed a method to reduce the size while retaining accuracy. A detailed description follows.

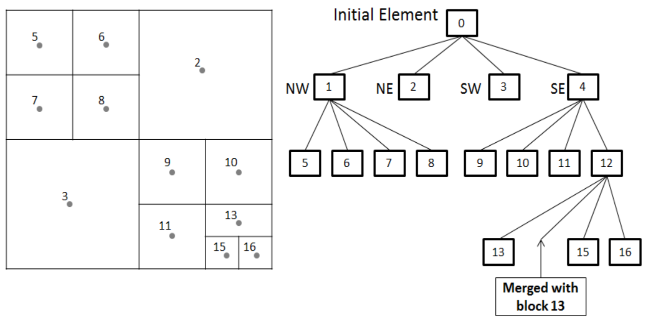
In the creating PR quad-tree structure (point region quad-tree), data is stored in leaves only. Internal nodes have four pointers which are labeled as NW, NE, SE, and SW. If during insertion, a leaf node is found and the coordinates of the leaf node's data differ from the data that is trying to be inserted, then a new internal node is created and both pieces of data are inserted. As with building a quad-tree, it requires creation of both internal and leaf nodes. This process is recursive. We could put an insert method into the node class. LiDAR data could be called insertion on a node and they would "know" which insertion to call. The internal node would ask the data their coordinates and determine where they do. In order to do this, the method needs to know some additional information. It needs to know the corner of the world and the width of the world. The leaf node insertion would have to do some more work. First, it would need to check to make sure that the data that is trying to be inserted isn't already in the tree. If it isn't, then it needs to create a new internal node, and then ask the new node to insert both the data that is already in the tree and the data that is trying to be inserted.

This quad-tree with a threshold allows us to reduce the size of the tree at the cost of prediction accuracy, as it is no longer an exact copy of the original data. Furthermore, setting an accurate threshold value of height is important to simplify a quad-tree. A threshold is chosen carefully to merge any adjacent nodes into a single node. At the same time, each node has its own region. These regions are merged when a criterion is met, i.e., if the height difference for the adjacent nodes is smaller than the predefined threshold.

According to Fig. 4(a), suppose the difference of height between Node 13 and 14 falls into the threshold, and Node 13 is higher than Node 14. We can consider them as a single node and merge them. Owing to the fact that Node 14 has a smaller height value it has to merge with Node 13 and give an average height value. Average location value in the Node 13 and 14 can be redefined as the x, y coordinate value of Node 13. Similarly, we could assume the difference of height between Node 13 and 15 is bigger than the threshold. Then, these two nodes would remain as they are and then check their adjacent nodes. Here, a limited distance range can be referred to find adjacent nodes. After nodes are merged, we use the average height as a new value to represent the height information in such a node. Fig. 4(b) illustrates the reduced quad-tree structure based on our method from Fig. 4(a). A pseudo code is shown in Fig. 5.



(a)



(b)

Fig. 4. (a) Example of 2D space and its quad-tree representation, and (b) The corresponding compressed quad-tree structure for Fig. 4(a)

```
01 procedure Data compression
02   Input: LiDAR data
03   Output: a compressed quad-tree (I)
04 begin
05   Create a normal quad-tree
06     Insert all the leaf nodes of quad-tree (I) into
         a linear list, L.
07   while L is not empty do begin
08      Pick a node μ, from L.
09      Find the edge-neighbors of μ.
10        Calculate the height difference with the
          neighbors of μ.
11      if the abs( height difference ) < threshold
          then
12            Average the location coordinate and
              height value between μ and
                  the neighbor of μ.
13            Store the new node into L.
14            Discard this node and merged the
                neighbor of μ.
15      else pass the node
16   end while
17 return quad-tree (I).
```

Fig. 5. A pseudo code for the compression step of quad-tree

Although each sub-quad-tree can be tested to determine if the height difference is less than the threshold and can be merged, this method is still incapable of being justified as a practicable or feasible method due to its accuracy. To prove that our method is practical, a series of experiments was performed with different thresholds. The errors were measured from those experiments to prove the validity numerically.

Though each sub-quad-tree following threshold merge, this method is still incapable of being defended or justified if this is practicable or feasible. To prove our viewpoint, we made a simple experiment to get a series of compression ratios with different thresholds and accuracy evaluation.

IV. Experiment

LiDAR data are converted into a set of 4-tuple data in text format, which consist of latitude, longitude, height, and reflection ratio. Each row denotes a single data point. The resolution of data points was about 3/m^2. These data points were constructed into a quad-tree structure as described in the previous section. A threshold will be decided to reduce the size of the quad-tree before transforming irregular data into regular data. In this experiment we attained the reduced number of nodes in the quad-tree structure by setting different thresholds. According to the reduced number, we could calculate the Compression ratio and RMS, as shown in Table I.

TABLE I. Number of Reduced Nodes and Evaluation with Different Threshold

Threshold	Total number	Reduced number	Compression ratio (%)	RMS (%)
0.09	4833568	327571	93.223	0.201
0.10	4833568	336078	93.047	0.217
0.11	4833568	377357	92.193	0.245
0.15	4833568	534351	88.945	0.289
0.19	4833568	651227	86.527	0.303
0.20	4833568	767280	84.126	0.318
0.25	4833568	777334	83.918	0.322

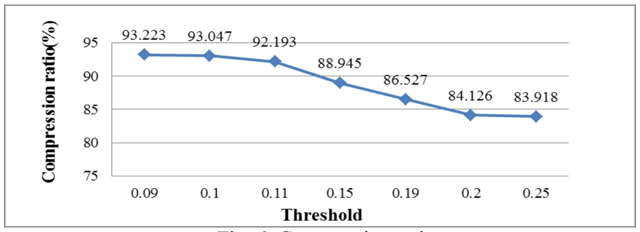


Fig. 6. Compression ratio

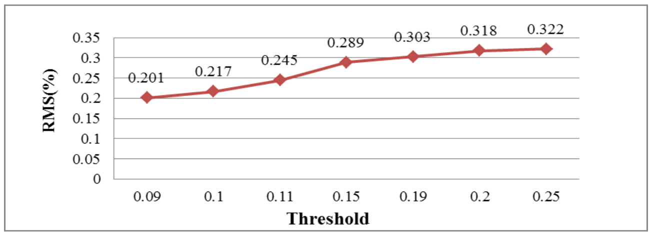


Fig. 7. Root Mean Square Error

Here, Compression ratio tells us the size of the simplified quad-tree structure. Compression ratio = (total number - reduced number)/total number. In mathematics, the root mean square (abbreviated RMS), also known as the quadratic mean, is a statistical measure of the magnitude of a varying quantity. It is especially useful when varieties are positive and negative. The smaller the RMS we have, the more positive our accuracy. In Table I, on the one hand, we can find the least RMS when the threshold is 0.09. That means that high accuracy varies inversely with threshold growth. On the other hand, we can draw a conclusion that the Compression ratio was decreased by an increasing threshold. Fig. 6 and Fig. 7 show us another modality based on Table I.

V. Conclusions

LiDAR data require heavy computational processing. The proposed data compression system is begun by building a point region quad-tree and compressing this structure with a threshold. In this paper, we proposed a preprocessing method to compress the data set based on a quad-tree structure. We introduced a series of basic knowledge about quad-trees in detail and discussed how to give the compression process for LiDAR data. Through making an experiment, we analyzed how to simplify data under the premise of guaranteeing the accuracy of LiDAR data.

The contributions of this paper are accurate collection of boundary points and possibility provided to adjust thresholds. If more sophisticated object extraction algorithms are provided, it will generate more precise objects. We proved our system can simplify some pre-processing tasks well for a data set with a large amount of data and guarantee the accuracy of building shapes with our experiment. Therefore, it is suitable for LiDAR data of object extraction applications.

References

[1] Andersen, H.E., McGaughey, R.J. and Reutebuch, S.E. "Estimating canopy fuel parameters using airborne LIDAR data," Remote Sensing of Environment, vol. 94, 2005, pp. 441-449.

[2] Andersen, H.-E.; Reutebuch, S.E. and McGaughey, R.J. "A rigorous assessment of tree height measurements obtained using airborne LIDAR and conventional field methods," Canadian Journal of Remote Sensing, vol. 32(5), 2006, pp. 355-366.

[3] Maas and H.G. "Methods for measuring height and planimetry discrepancies in airborne laserscanner data," Photogrammetric Engineering and Remote Sensing, vol. 68(9), 2002, pp. 933-940.

[4] Qing Yin, JinHai He and Hua Zhang. "Application of laser radar in monitoring meteorological and atmospheric environment," Journal of Meteorology and Environment, vol. 1025(5), 2009, pp. 48-56.

[5] P. Axelsson, "DEM generation from laser scanner data using adaptive tin models,"International Archives of Photogrammetry and Remote Sensing, XXXIII, Part B3, 2000, pp.85-92.

[6] P.A. Burrough, "Principles of Geographical Information Systems for Land Resources Assessment," Oxford University Press, Oxford, pp. 194.

[7] B. Alexander and H. Christian, "Aspects of generating precise digital terrain models in the Wadden Sea from lidar-water classification and structure line extraction," ISPRS Journal of Photogrammetry & Remote Sensing, vol.63, 2008, pp.510-528.

[8] L. Cheng, J. Gong, X. Chen and P. Han, "Building boundary extraction from high resolution imagery and LIDAR data," International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences , vol.37 (Part B3), 2008, pp. 693-698.

[9] J.Cleary, I. Witten. Data compression using adaptive coding and partial string matching, in IEEE Transactions on Communications, vol. 32(4), 1984, pp.396-402.

[10] ChangBin Shi. "Comression experiment for weather radar reflective data," Electronic Test, (1), 2010, pp.24-28.

[11] Chunqiang Qian and Jicheng Wang. "Application of Quadtree Theory in Fractal Image Coding," Comptuer Engineering and Application, vol. 43(23), 2007, pp.61-63.

[12] Yusheng Jiang,Chao Wei,Lanxin Ding. "Based on Quadtree Image Compression Algorithm for Research and Implementation," Comptuer Knowledge and Technology. vol.4, No.8, December 2008, pp.2220-2222.

[13] C.R. Dyer. "The space efficiency of quadtrees," Computer Graphics and Image Processing, vol.19(4), 1982, pp. 335-348.

[14] I.Gargantini. "An effective way to represent quadtrees," Commun. ACM, vol. 25(12) , December 1982, pp.905-910.

[15] C. L. Wang, S. C. Wu, Y. K. Chan, and R. F. Chang. "Quadtree and statistical model-based lossless binary image compression method," Imaging Science Journal, vol. 53(2), January 2005, pp.95-103.

[16] X. Yin, I. Diintsch, and G. Gediga. "Choosing the root node of a quadtree," IEEE Granular Computing Conference, August 2009.

[17] H. Samet, "The Quadtree and Related Hierarchical Data Structures," ACM Computing Surveys, Vol. 16, No. 2, June 1984, pp. 187-260.

[18] H. Samet, "Foundations of Multidimensional and Metric Data Structures". Morgan-Kaufmann, 2006.

[19] Samet, Hanan. "Design and Analysis of Spatial Data Structures". College Park, MD: Media Express, 1999.

[20] Winder, Ransom. "Examples of a Point-Region Quadtree with moving points or a Dynamic PR Quadtree". 29 September 2000.