Implementation of point cloud data structuring using k-d tree

Raivis Baltmanis

Forest faculty

Latvia University of Agriculture

Jelgava, Latvia

Raivis.baltmanis@gmail.com

Mārtiņš Krūmiņš

Latvian National Forest Competence Center

Riga, Latvia

martinskruminsh@gmail.com

Agris Zimelis

LSFRI Silava

Salaspils, Latvia

agris.zimelis@silava.lv

Nowadays, with the continuous rapid development of laser scanning technologies, not only development of solutions for tackling topical issues of various industries becomes significant. The high volume of information that has to be processed must be considered as well. Scanners with capacity to obtain more than a million of information units per second are already being used. Therefore it is crucial to develop data storage and structuring solutions in order to minimize computer resources and time spent for data processing. Use of k-d tree data structures could be considered as one of the most suitable solutions for storage of information at the time of data processing. This study contains also description of the method for conversion of structured data into binary files in order to speed up data processing when using the data repeatedly.

Keywords: LIDAR, k-d Tree, data structure

# Introduction

Three-dimensional (3D) laser scanning, known also as LIDAR (Light Detection and Ranging), is a 3D imaging technology that obtains information about the object in the form of a point cloud with the help of a laser ray and a rotating mirror. Using the 3D scanners, a three-dimensional point cloud depicting the surface of the scanned object with a very high precision is obtained. Nowadays, studies on utilization of 3-dimensional laser technologies in various industries are topical in the world.

Forestry is one of the potential fields for utilization of laser scanners. Use of laser scanning technology in this industry helps solving various tasks that require high precision measurements without the requirement to cut trees and without disturbing the processes of forest growth. Thus, using data obtained as a result of the scans, it is easier to adopt economically efficient decisions related to forest management

# LIDAR data

Speed at which latest 3D terrestrial laser scanning equipment is capable of obtaining data nowadays reaches over one million points per second. The speed of data collection and thus also the volume is expected to grow. Hence, research on effective techniques for collection and structuring of laser scanning data is becoming more and more crucial in order to ensure complete utilization of computer capacity.

The set of information gathered as a result of laser scanning is called a point cloud. Explained simply, it can be regarded as a list of the coordinates of points in three dimensional space. Number of points in such a cloud depends on the size of the scanned object as well as on the settings of laser scanning intensity. Considering the high volume of data in the point cloud obtained as a result of a high quality scanning, it is important to know data storage types not only for the time when no manipulations are done with the data (data archive, files etc.) but also during the time of data processing. Uploading high volumes of information onto the memory of computer burdens it heavily. Afterwards, performing calculations or otherwise using the point cloud, earlier uploaded onto the memory of computer, a walk-through of the unstructured point cloud is performed, searching the interested data area. Thus, every time going through the whole data set, both time is being consumed, as well as computer resources. In order to optimize data search from the whole point cloud making it faster and more effective, the data uploaded from the file should be sorted into a particular structure (it is a manner in which information is stored and organized in computer memory in order to make utilization of information more efficient). If modifications of utilized data structures are used in an operating programme, the software operating principle remains the same because generally one correct implementation is substituted by another correct implementation [1].

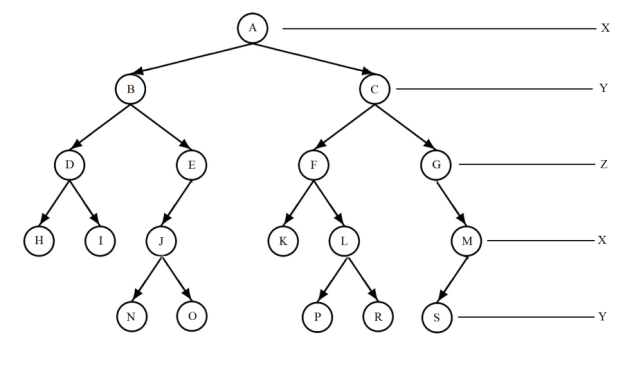
The difference between the various implementations of data structures lies in the speed they work at. It is possible to achieve a significant improvement in the functioning of the programme and solution of the algorithm by selecting a data structure suitable for the type of a problem. In order to achieve the best improvement in solving the algorithms, first of all it is necessary to identify two indicators – what kind of data needs to be structured and what kind of manipulations would be performed with the data. Justification for the choice of structure used for the chosen data set arises from these two points.

Answering the first question, scanning of standing trees using a Trimble FX 3D laser scanner was performed in this study (the mentioned equipment may generate up to 216000 data points per second operating at maximum capacity [2]). The data obtained after scanning was used to calculate tree trunk diameter at different levels of height. In order to obtain the results, algorithms for data filtering, selection, slicing and nearest neighbour search were applied. Considering the above mentioned information, a decision was made to apply the k-d tree structure for data organization.

# creation of k-d tree data structure

The typesetting k-d in the name of the structure describes a term k-dimensional, which means that it may be used to structure information of various numbers of dimensions. The data used in this study is 3-dimensional with points X, Y and Z as coordinates of the scanned data point. This data structure, similar to the other structures meant for spatial data, hierarchically divides the space into small areas, each of which contains some of the representatives of the total volume of data [1]. This way, knowing the coordinates, quick access to the point or area of points of interest is achieved. Usually, in order to implement the k-d tree data structure, the information space is sliced in one of the dimensions dividing the space into two equal parts. Each of the next steps breaks all previous data areas obtained in the previous step, taking into account that slicing takes place in another dimension. To ensure the created k-d tree structure achieves high performance surge, it is important for it to be well balanced. In this case, it means that almost all the data points must be equidistant from the root element of the structure. To achieve this effect, the data median of the corresponding dimension should be used as a splitting point when constructing a tree.

Construction of a tree is based on the use of unstructured scanned points in ranked order in which they appear in the data file. Initially, data is read from the file and arranged into an unstructured related list. Afterwards, a tree is constructed using data from the list. All data required for the tree is known before the construction of the tree. Hence, a divide and conquer method may be applied to form a k-d tree structure [3]. A principle that a tree consists of two elements – branches and leaves was chosen when creating the structure. Each object of a branch contains information on the dimension, based on which further data organization is performed. Another parameter is a median value of the data set in the corresponding tree position of the same dimension. Figure 1 gives an example of a schematic representation of a tree.



1. A priciple of dimensional separation in a k-d tree construction

Each branch of a tree contains data on its subordinate lower branches – the left and the right ones. It should be noted that branches of the tree do not store data points, but only supporting information helping to faster locate the searched point or data area.

The second type of objects the created tree structure is composed of are called leaves. Those are elements of the last level of the tree, and they do not have any sub-branches. The point cloud is being separated until there is left only one data point. At this point of time, a leaf of the tree is formed, in which the corresponding data point is saved and which has no subordinate elements. The recursive approach of the tree construction process is implemented using the data stack that implements a LIFO (last in, first out) processing method of stored elements. There was no information of branches and leaves stored in implementation of the algorithm in the data set. Instead, specific information, directly required in the construction process of a tree was used, however it is not utilized for further use. Components of such information are a data set that is conformable with the corresponding position of the tree structure, depth of the tree and an indicator that determines whether the data corresponds to the right or left sub-branch of the previous level branch. The latter is important in order to follow the principle suitable for the binary search tree, which states that values higher than the element are stored in the right sub-branch, while values lower - in the left sub-branch.

# search of the nearest neighbours

Designing nearest neighbour search algorithm in the k-d tree structure, it is requisite to be guided by the principle, which recursively examines the whole tree in order to find the data area or a leaf in which the current point is closest to the specified one. On the other hand, developing a version of the algorithm, which intends locating a fixed number of nearest neighbours, it should be taken into account that they will be located in different last level leaves of the tree. Implementing the nearest neighbour search algorithm, the distance between data points was determined as the Euclidean distance. Initially, it may seem that the main objective in the process of finding the nearest neighbour is to find such a leaf of the tree that contains the required point, and then search this leaf and all the nearest ones. However, the problem is that the nearest neighbour may be located in a completely different branch of the tree [4].

A term of a limiting parallelepiped was introduced in order to ensure the effectiveness of the algorithm. Such an object is linked to each of the branches of the tree structure. It describes the limits for such a smallest possible parallelepiped, which comprises all the data points of the particular branch. This object is being used in order to determine whether there is possible a data point on the particular tree branch that is closer to the provided point than the already found nearest neighbour. For instance, if distance from the provided point q till the nearest edge of the limiting parallelepiped of the branch is larger than the neighbour found so far, it is considered that there is no need to search the sub-elements of this branch.

Another condition that must be taken into account in order to obtain a more efficient algorithm is the order in which sub-branches of each branch are reviewed. Often, the first to deal with is left sub-branch and then right one, but in order for an algorithm to reach the “correct” point faster, a condition is introduced as per which the first to be reviewed is the sub-branch whose limiting parallelepiped is located closer to the given point q, thus ensuring that the first to inspect is the branch with the highest possibility to locate the required point [4].

It is considered that search for the nearest neighbour in a k-d tree data structure in a situation when data is located according to the random variable method, on average can be described with O(log(n)). However, in the worst scenario, it may correspond to O(n), where n is the number of points in dataset [5]. For comparison, execution time of nearest neighbour search for an unstructured data set corresponds to O(Nd), where N is the size of the data set and d - the number of data dimensions. However, in order to objectively assess the designed data structure performance particularly in the nearest neighbour search operation, a test was developed in which a point was chosen (let's call it a given point) from a one million large data set using a case method, and the nearest neighbour had to be found for the point. Afterwards, the closest neighbour of this point was sought both in k-d tree data structure, as well as in unstructured data or data form a simple list. Performance was measured in milliseconds, which is a variable measure, because it is directly dependent on the processor load at the time of algorithm. Consequently, each test was repeated ten times, setting the average nearest neighbour discovery time. The operation was repeated 100 times, each time choosing a different random point.

Nearest neighbour search algorithm execution time is dependent on which of the existing dataset points it is applied to. Therefore, the results may be volatile. To assess the impact of this volatility, a comparison of the obtained dispersion results was conducted, as the indicated value characterizes the process stability. An F-test was used for comparison of unstructured data and speed variance of nearest neighbour search within the points located in k-d tree structure. Total algorithm speed performance test results are summarized in Table 1.

1. comparison of the nearest neighbour algorithm performance speeds

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data structuring type | n | xˉ | S | V% |
| Unstrucured data | 100 | 101.827 | 2.752 | 2.702 |
| Kd-tree | 100 | 1.481 | 0.930 | 62.82 |

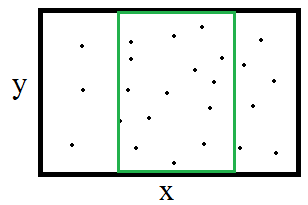
The test results determined that the developed algorithm adapted for k-d tree structure conducted nearest neighbour search operation in considerably shorter time than in the case of unstructured data. This is an average of 1481 milliseconds in k-d tree case, and 101,827 milliseconds for unstructured data after 100 repeated measurements. Comparing execution time dispersions of the two tested algorithms, it was determined that for unstructured data, the variance is significantly lower than for the k-d tree nearest neighbour search algorithm (p <0.05). It is observed also that, dispersion, expressed as a percentage, is lower (2.702%) in unstructured data compared to the data structure (62.82%). Also, dispersion, measured in percentage, is observed to be lower in unstructured data (2.702%), compared to the k-d tree data structure (62.82%). This is explained by the fact that no matter which point of the data set is chosen for nearest neighbour determination, the algorithm each time determines the distance between the given point and each one of the remaining data points from the date set together. So each nearest neighbour search operation takes place, searching the entire dataset. Consequently, the algorithm run-time variance is relatively low. On the other hand, nearest neighbour search algorithm run-time variance customized for k-d tree structure creates a condition that not all tree branches are screened. This is determined by a condition of the limiting parallelepiped of separate data areas. Thus, by choosing a point from the set randomly and then searching for the nearest neighbour, a different quantity of the tree branches is searched each time, resulting in different execution time of the algorithm.

# Range search

Another of the more common practices performing data processing of point clouds, is sorting of information in a given area. Since the point clouds are three dimensional data, data selection should be able to operate in all of them. So, when designing an algorithm for searching a particular area, it must be taken into account that it can be defined in all three dimensions. For example, provided that from the total point cloud some data has to be picked - with X dimension values from five to ten, Y dimension values ranging from six to seven, but the Z dimension from zero to three, regardless of the size (in this study millimetres are used for the point cloud). Such defined boundaries in 3-dimensional space create a parallelepiped with data required for the selection. However, the fact should be taken into account the that, working with the data obtained as a result of scanning, it is not always necessary to perform data selection in such a space limited from all sides. If we assume that coordinate Z describes height of the data points, then it may be necessary to select data, for example, at the height of at least two meters. In such a situation, the space meant for data selection is limited only on one side. A similar situation may happen when there is a need to select data based on the values of two or three dimensions. Therefore, developing an algorithm, it should be possible to select the data, even if space constraints are defined at one or two dimensions. However, considering the design features of k-d tree data structure, also performing selection of data at one or two dimensions, a parallelepiped must be defined in 3D space, where the required points are being searched. In order to make it happen, a parallelepiped described in the previous chapters is being used. While building a data structure, one of the parameters has to be defined in a way that it contains completely all tree data. Coordinates of such a parallelepiped are used as the default values in establishment of data selection criteria, and when it is required to select data by one or several criterion, the respective border values of all data containing parallelepipeds are changed to the selection criteria specified.

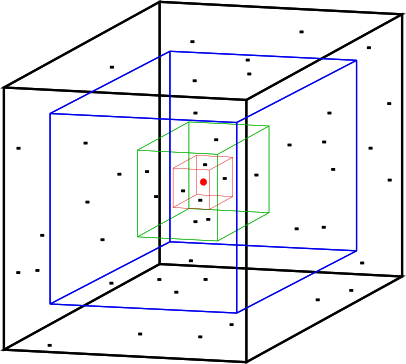
In order to explain the nature of the idea, a 2D schematic representation of the principle for data selection criteria was created (see image 2). In that, an all-data-containing figure (in a three dimension space, it means a data containing parallelepiped) was marked in the shape of a triangle. As it can be seen, the initial limits are specified such that it covers all the cloud data. A situation is imaged, when you need to perform data selection based only on the X dimension values of the points. Thus only rectangular boundaries are modified of the required dimension, while keeping the rest unchanged.

As a result, the data selection area is determined by the image shown in green rectangle (image 2). This way, indicating the data restrictions in one or two dimensions, a three-dimensional parallelepiped is constructed, which contains all the points matching with data selection criteria.



1. A schematic representation of data selection area if it is made only in X dimension

Taking into account the specificity of k-d tree structure, searching for data in a particular area, the entire tree is searched only in the worst case, or if the data selection criteria include a high proportion of the total point cloud. If the data selection criteria describes a relatively small share of the total cloud (in the context of this study it could be a fragment of a tree trunk), algorithm views only a part of the points. This is because whole structure branches of k-d tree may be considered unsuitable, thus significantly reducing the total volume of branches under review in the search of data. The data search algorithm adjusted for the k-d tree structure could provide good high-speed performance in situations when data has to be selected from a relatively small area (if compared with the size of the total point cloud). A similar situation is possible if the given search area has no data. Also when k-d tree algorithm can detect such a situation earlier, not performing search of all data. To confirm the hypotheses, a test algorithm was developed to compare the run-time in k-d tree structure and in a simple list of data or unstructured data. Evaluation of the results required a comparison of execution time of the two algorithms by asking different sizes of data sampling areas (see image 3). Since data is in a 3D space, the selected areas used for algorithm performance testing were measured in volume units. In total, 100 different size areas were defined for the data set, where volume of each area may be expressed as percentage of the total point cloud volume. This way an opportunity to assess the performance of data selection algorithms both in relatively small areas (up to 10% of the total cloud volume) and large areas (over 90% of the entire point cloud volume) was obtained.

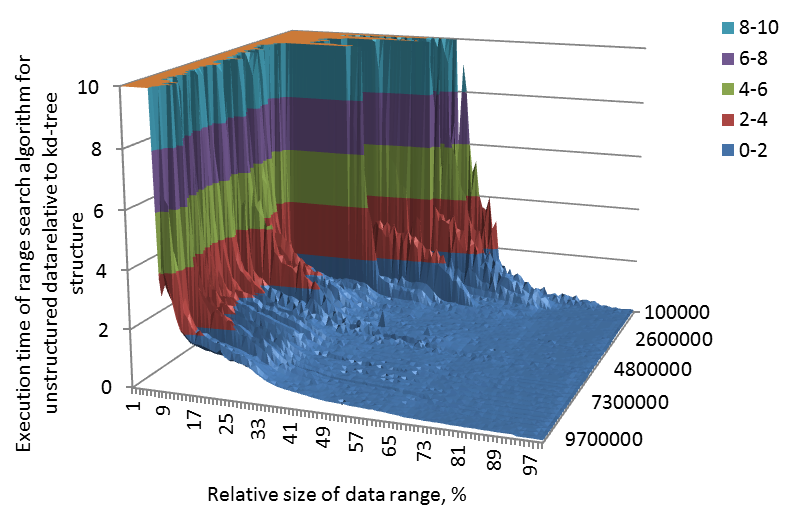


1. A principle test scheme for data serach in a given area

All areas defined for testing purposes have a common centre, which coincides with the centre of the parallelepiped including the point cloud. Each time, increasing the data area, the distance between the sides of the parallelepiped and the centre was increased. The increase was performed in all the dimensions by a fixed number of units, which was obtained by dividing the distance from the cloud hugging parallelepiped sides to the centre dividing into 100 similar parts. Compliance with these provisions ensured smooth data selection area increase in size relative to the entire data set.

Based on the quick-performance assessment principle for the algorithm described, the test was selected for different sized data sets, which contained from 100 000 to ten million points. The data set size was increased by 100 000 points in each new test, thus gaining 100 data sets of various sizes to perform the tests. Next, 100 areas of relatively different sizes were defined for the data set, and data selection was performed. Since data selection speed was measured in milliseconds, which may be a volatile measure depending on the workload of computer processor at the time of the test, three repetitions were performed, registering the average performance time. During the first high-speed tests, it was concluded that the run-time differences of the two algorithms, expressed in milliseconds, may not be sufficiently perceptible, given the complexity of the test and the number of repetitions by changing various parameters. Therefore, the test results were used for graphical display of both the algorithm execution time ratio. More precisely - the result of unstructured data in relation to the in k-d tree data selection algorithm execution time. Therefore, the graph displays the result as the value that determines how many times the algorithm of unstructured data for more than fulfilled in the case of structured data (see figure 4).

Thus, the chart shows values that indicate how many times longer the unstructured data algorithm lasted, compared to structured data. (see figure 4).



1. Performance time of a data selection algorithm for unstructured data compared to the k-d tree data structure

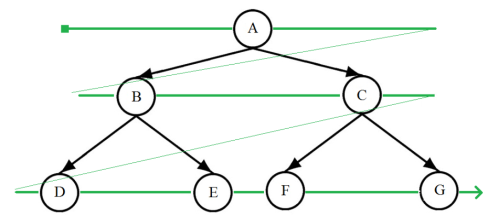
The horizontal lines of the figure (see figure 4) show relative size of the data selection area (against the whole data point cloud). The vertical line shows how many times longer an unstructured data algorithm lasted compared to k-d tree data selection. The other horizontal or depth line shows the size of the data set that was used to perform the tests.

During assessment of the results, it gets confirmed that performing data selection in a relatively small area, algorithms of k-d tree structure show better performance timing compared to a search through the entire set of unstructured data. In situations when it is require to select data from an area that contains more than a half of the point cloud space, k-d tree data structure algorithm implementations developed in this study do not provide a significant improvement of a high-speed performance, but when selection is underway in the area, having a capacity of less than 50% of the total volume of point cloud, k-d tree data structure algorithm shows faster performance, and the test results show that reduction in the area of data selection criteria, the developed data selection algorithm of k-d tree data structure operates ten and 100 times faster than in case of unstructured data. It must be remembered that vertical axis was limited to a value of ten in order to ease visual reception, although at 1% to 5% large data selection area, this value amounts to several hundreds.

# Preservation of data structure in a file

Considering that the volume of data obtained as a result of laser scanning may extend to several millions of points, it is wise to clarify how long time is spent to build the structure because every time unstructured data from a file is selected, a tree has to be build using the data. In this regard, a hypothesis was proposed stating that development of a method to preserve structured data in a file would make reading the file faster than reading information and preparation for work using unstructured information from a data set. In order to save a tree structure in a file and later read it obtaining the same structure, a uniform style is required to systematically view branches of k-d tree structure. There exist various algorithms for tree examination. In this study, it was decided to use a tree branch viewing through the levels, because after reading information from the file, the tree is being built again with the same levels.

The principle of a tree walk-thorough by its levels (Level-order traversal) is such that initially all branches are being reviewed at a particular depth and only then the algorithm switches to the next, deeper level (see figure 5). The data shown in the image would be saved in a file in the following sequence: ABCDEFG.

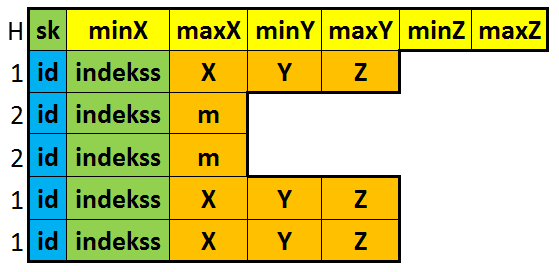


1. A binary tree level-order traversal

The benefit of the use of such a method for data structure walk-through is that sub-branches of each branch can be determined in a simple manner. In order to achieve that, we have to add a number indicating a serial number of the branch in the structure for each of the branches if it is viewed using a Level-order traversal algorithm. These numbers may be called indices. So, assuming that the branch index is n, then the sub-branch indices are calculated using the equations 2\*2\*n and n+1 [6]. Once an index is added to each of the branches while saving data structure on a file, it is possible to determine the branch, which will be considered in the downstream. Parent index is calculated using the equations n/2, if the branch is the left sub-branch, or n/2-1, if the branch is the right sub-branch. To determine whether the branch is parent’s left or right sub-branch, it is necessary to determine whether the index is an even number. Once indices for the branches of the tree structure are assigned in level-order traversal, branches with an even index are always left sub-branches of their parents, and branches with an odd index – right sub-branches.

When saving a tree structure to a file, it must be taken into account that it consists of two types of elements - branches, which are the organizers, and leaves that contain the data points. Each of these elements is characterized by different information. For branches, it is the median of the data in a certain dimension, while leaves contain the actual data points, which are basically three figures - the coordinates.

Since the tree structure is being recorded into the file in binary format, it is important to define what type of values, and in what order are being stored in order to enable reconstruction of the recorded information in the form of a k-d tree structure. Therefore, the total content of the file has been split into two parts - the header information and the contents of the file. Number of data points and coordinates of the parallelepiped boundary comprising the data points are recorded in the header information, while contents part of the file gets two kinds of information recorded in it – information on the branches and leaves of the tree (the last level of branches containing data points). Schematically, a fragment of data recording and reading is shown in Figure 6.



1. File structure for saving k-d tree data

Each colour area of the image represents a different type of information that is being recorded in the relevant position of the file:

* green – 32 bit integer,
* yellow and orange – 64 bit fractional in a floating point notation,
* blue – 8 bit integer.

The first line of the schematic representation of the file contains the main content. Thus the first value is an integer indicating the number of data points in the structure. This is followed by six fractions that in a particular order represent coordinates of data enclosing parallelepiped boundaries. Further in the file contents, the descriptive fragment of tree elements begins with an eight-bit row that describes the number one or two. This value is used as an indicator to determine what type of wood element is being reviewed:

* one – a branch,
* two – a leaf.

Based on these values, a further data reading scenario is set. Regardless of the value of indicator, it is followed by a 32-bit integer that describes the index of the tree element. It is used in order to determine, which one of the existing built tree branches would the reviewed element be added to. The next steps of file reading depend on the value of the indicator. If the value is one, reading of the following three fractional numbers, representing values X, Y and Z of the data point is done. On the other hand, if the index value is two, the next 64 bits are taken, which characterize the median value of the tree branch in fractional form. This way cyclically the entire contents of the file are read, adding each of the elements to the structure of the tree.

After saving data in a file, detection of file reading speed was performed and the process was repeated ten times. As a result, it was concluded that saving structured data in a file for re-use, the information gets prepared for use approximately three times faster than using unstructured files and then structuring the data.

# Conclusion

Utilization of the k-d tree structure, adjusted for three-dimensional data, in implementation developed in this study, significantly enhances performance of such algorithms as nearest neighbour search and range search. By using k-d tree structure speed characteristics of range search algorithm execution is much better than for unstructured data. Even more, in case of range search, if data selection area is relatively small, the execution time may be reduced to 100 or more times. In turn, the developed method for saving k-d tree data structure in a file, ensures at least three times faster reload of the data compared to structuring unsorted data after reading it from a file. This makes it possible saving data processing time when re-using the data files. Also the developed method provides data storage in binary format. That is giving better performance during data reading than point cloud storing in text format. This fact is worth mentioning because ASCII format is supported by most of laser scanner data processing software and it is most commonly used to export point cloud data.

##### Acknowledgment

The study is done within the scope of the project (No. L-KC-11-0004) of the National Forest Competence Centre.

##### References

1. Steven S. Skiena (2008) “The algorithm design manual.” London: Springer London. 730 p.
2. *Trimble FX 3D datasheet*. Available at: <http://trl.trimble.com/docushare/dsweb/Get/Document-397918/022504-105A_Trimble_FX_DS_0309_lr.pdf>
3. J. Michael. *kD Trees*. Available at http://ldots.org/kdtree/
4. M.M. Hugue. *More on kd-*trees. Available https://www.cs.umd.edu/users/meesh/420/Notes/MountNotes/lecture18-kd2.pdf
5. J.L. Bentley. “Multidimensional binary search trees used for associative searching”, 1975, Communications of the ACM, 18(9), p. 509-517.
6. J. Morris. *Data structures and algorithms: Heaps*. Available at https://www.cs.auckland.ac.nz/~jmor159/PLDS210/heaps.html