in: Proceedings 20th Congress International Society for Photogrammetry and Remote Sensing (ISPRS) 2004

SPATIAL ACCESS METHODS FOR ORGANIZING LASERSCANNER DATA

Thomas Brinkhoff

Institute for Applied Photogrammetry and Geoinformatics (IAPG)
FH Oldenburg/Ostfriesland/Wilhelmshaven (University of Applied Sciences)
D-26121 Oldenburg, Germany
Thomas.Brinkhoff@fh-oldenburg.de

Commission IV, WG IV/1

KEY WORDS: Laser scanning, Data Structures, Database, Performance, Processing

ABSTRACT:

Laserscanning produces large sets of multidimensional point data, which demand for an effective and efficient organization and storage. Adequate data structures must perform specific spatial queries and operations in order to support the computation and / or the construction of surface models. Because of their increasing size, it is not advisable to organize the clouds of points by main-memory data structures. Such an approach would lead to long loading times and misses scalability. Instead, persistent data structures are desirable. In this paper, the usage of multidimensional spatial access methods are investigated for organizing laserscanner data. Such access methods have originally been developed for storing and indexing geographic data in spatial database systems and Geographical Information Systems. Point access methods based on hierarchical hash trees are one important class of such access methods. Typical examples for hash trees are the BANG file and the buddy tree. Rectangle access methods are another class of relevant access methods. The R-tree and its variants are the most important representative of this class. R-trees are typically used by commercial spatial database systems. All data structures mentioned before are fully dynamic, i.e. they support arbitrary sequences of insertions, modifications and deletions. They allow a persistent storage of multidimensional points and preserve spatial proximity locally, i.e. within (database or file) blocks. The performance of the above point and rectangle access methods is investigated and compared for storing and querying large clouds of points representing buildings. The examination identifies those access methods that allow a fast construction of the data structure as well as an efficient support of relevant spatial queries. For some queries, however, a local preservation of spatial proximity is not sufficient. The extraction of points for overview purposes is an example for such a query. Therefore, different approaches for a global preservation of spatial proximity are introduced and experimentally investigated.

1. INTRODUCTION

Laserscanning has gained more and more importance in the last few years. It allows the simple and inexpensive measurement of spatial objects like façades or the interior of buildings. Laserscanning produces large sets of multidimensional points, which demand for an effective and efficient organization and storage (Niemeier & Kern, 2001). The measurements provide immediately Cartesian coordinate values (x,y,z) and – for some laserscanners – the intensity of the received signal. Therefore, the result of a measurement is a set of three- or four-dimensional points.

Because of the large data volume – several millions of points with increasing tendency – it is not advisable to store the points as conventional points in a CAD program (Schwermann & Effkemann, 2002). In general, the approach to maintain the cloud of points in main memory has several disadvantages:

- Such an approach requires a long time for loading the data from secondary storage (like from a hard disk).
- Main memory storage shows a bad scalability because it swaps (after exceeding a threshold) parts of the memory onto the slow secondary storage.

An alternative is the usage of *persistent data structures*. Such data structures store the data on secondary storage and allow reading only (the required) parts of the data. Such data structures have been developed to be used in (relational, object-relational and object-oriented) database systems as an index. Therefore, they are also called *index structures*. However, conventional index structures are optimized for one-dimensional data types like numbers and character strings. They cannot be used (without modification) for spatial data. For this purpose, *spatial index structures* (also called *spatial access methods*) have been developed for spatial database systems and Geographical Information Systems (GIS) (Rigaux et al., 2002). One category of spatial index structures are *point access methods*, which allow the dynamic organization of multidimensional points on secondary storage. *Rectangle access methods* are another class of spatial access methods supporting extended objects, especially multidimensional rectangles, but also non-extended objects, i.e. multidimensional points. All types of spatial index structures support the efficient processing of spatial queries.

In this paper, the question is investigated whether spatial access methods are suitable for storing point clouds measured by laserscanners. Section 2 presents different spatial access methods. The main focus is on so-called hash trees and R-trees. In section 3, we consider the use of such index structures for data produced by laserscanners. The paper concludes with a short summary and an outlook to future work.

2. SPATIAL ACCESS METHODS

2.1 Indexing in Database Systems

The main task of a database system (DBS) is to store large sets of data persistently. The database management system (DBMS) must support the insertion, modification and deletion of arbitrary data in arbitrary sequences. For this reason, the DBMS organizes the data in *database blocks*. The access to secondary storage (i.e. typically to hard disks) is performed blockwise, i.e., the access to a data record requires the transmission of (at least) one complete block that may store also non-required records.

An index dynamically organizes the database blocks in order to accelerate the access to blocks containing records that fulfill the query condition(s) (e.g., all persons born in Istanbul). The data structures that are used for building and maintaining an index are called *index structures*. In current database systems, two types of index structures are most often used: B-trees and hashing.

A *B-tree* is a dynamic balanced tree. Each of its nodes corresponds to a database block. B-trees store the data sorted according to a selected attribute. For processing a query, the tree is traversed starting at the most upper node ("root node"); only subtrees are accessed that potentially refer to queried data. Figure 1 illustrates a B-tree.

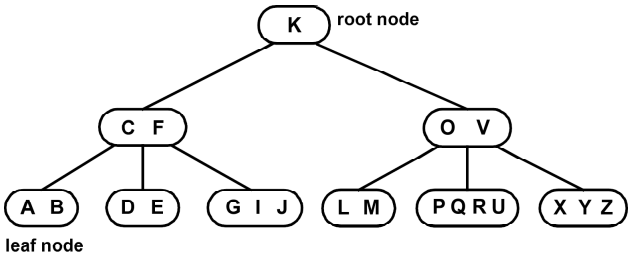

Figure 1. Example for a B-tree.

Hashing computes the location of a block on secondary storage (i.e. the block address) using one or more selected attribute(s) of a record. This computation is done by a *hash function*. Figure 2 depicts the hashing approach. Hashing supports efficiently exact match queries, i.e. the search for records with attribute value(s) that exactly match to the query condition (like in the above Istanbul example). However, hashing has efficiency problems either with handling uneven data distributions or with range queries (like finding all persons born in a city whose name starts with I).

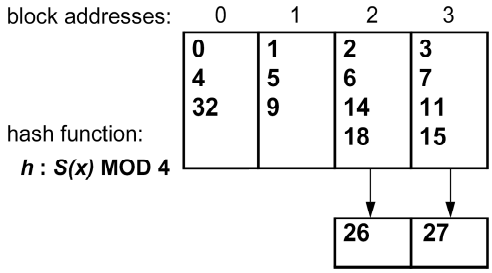

Figure 2. Example for hashing.

2.2 Indexing Spatial Data and Point Data

B-trees require a linear ordering of the data and hashing has – as mentioned before – problems with uneven data distributions or with range queries. Because of this reasons, conventional index structures cannot be used – without extensions – for organizing spatial data. Therefore, special *spatial access methods* have been developed for spatial database systems and Geographical Information Systems. *Point access methods* allow organizing multidimensional points and *rectangle access methods* – in addition – the storage of extended multidimensional objects like rectangles, cuboids, and (in approximation) of polygons, arcs and solids.

The *grid file* (Nievergelt et al., 1984) is an example for a multidimensional point access method. It is based on hashing. However, the hash function is replaced by a grid directory. This directory stores block addresses in its cells (see Figure 3). Grid files have performance deficits storing uneven or correlated distributed points.

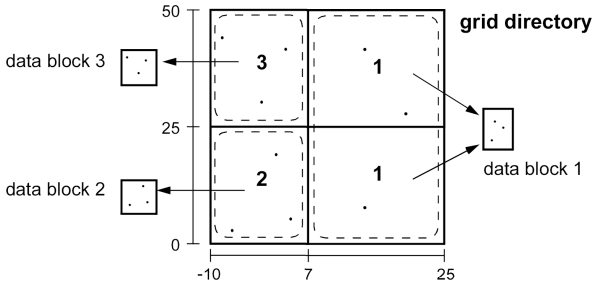

Figure 3. Example for a grid file.

The partitioning of the data space by grid files has following properties:

- The region described by a database block (the so-called *block region*) is rectangular.
- The data space is completely covered by the block regions.
- The block regions do not overlap.

However, for designing efficient spatial access methods, at least one of these three properties must not hold (Seeger, 1989).

2.3 Hash Trees

Hash trees are multidimensional point trees that combine hashing with data structures derived from trees. A typical example of a hash tree is the BANG file (Balanced and Nested Grid File) developed by M. Freeston (1987). The BANG file is a hierarchical tree. The upper part of the tree is the directory and the leaf nodes store the real data ("data nodes"). The block regions of the directory nodes are based on a grid structure and represented by a (multidimensional) rectangle. In contrast to conventional grid files, a block region does not represent the complete area of this rectangle. Instead, the included rectangles of the smaller block regions in the same node are removed from the rectangle. In consequence, the shape of block regions is irregular and may consist of several, unconnected areas. Figure 4 shows a set of points organized by a BANG file. The points are distributed on an area having the shape of a sinus curve. The figure depicts the partitioning of all nodes of the BANG file having the same height in the tree.

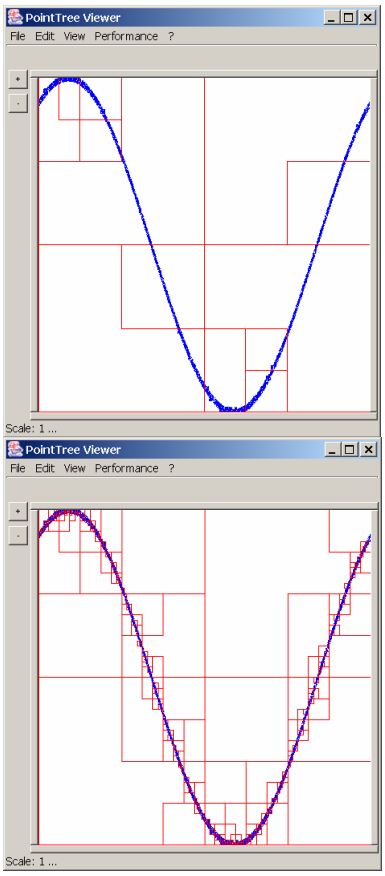

Figure 4. Example for the partitioning of a BANG file.

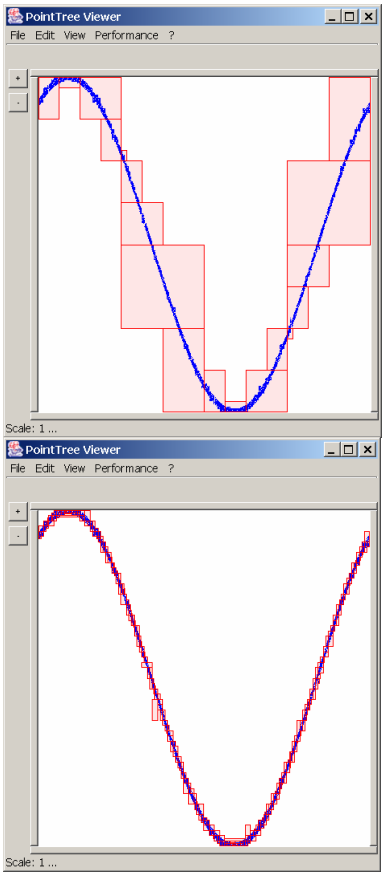

Figure 5. Example for the partitioning of a buddy tree.

Another example of a hash tree is the *buddy tree* (Seeger & Kriegel, 1990). The buddy tree is also a hierarchical tree with directory nodes containing rectangular block regions. In contrast to the grid file and the BANG file, however, the regions do not need to cover the complete data space. Figure 5 illustrates the partitioning of the buddy tree using the sinus data again.

2.4 R-Trees

The *R-tree* (Guttman, 1984) is a spatial access method organizing multidimensional points as well as rectangles. The R-tree has similar properties as the B-tree but it does not require a linear ordering. The block regions are minimum-bounding rectangles of all regions or data in the corresponding subtree. These block regions may overlap and do not need to cover the whole data space.

There exist several variants of R-trees. They differ in the insertion strategy (i.e., which subtree is chosen for storing a new object) and the criteria used for splitting a node if an overflow occurs. Figure 6 illustrates an R-tree: it shows the block region of the root node and (a part) of the partitioning of a node pointing to a data node. This illustrates the overlap between the block regions.

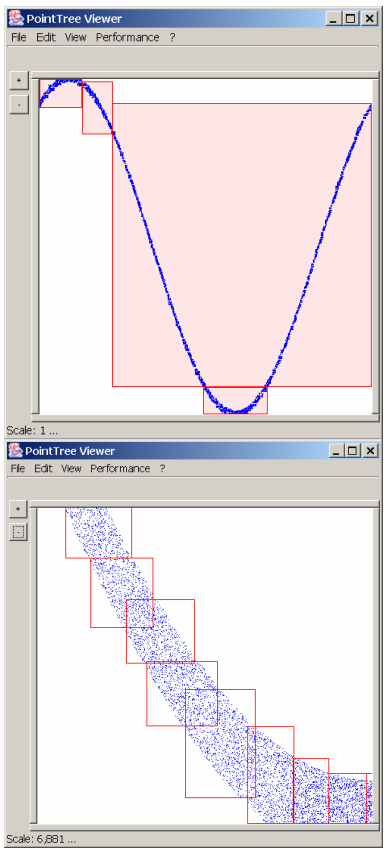

Figure 6. Example for the partitioning of an R-tree.

2.5 First Conclusions

The presented spatial access methods are dynamic index structures supporting the insertion, the modification and the deletion of points. They support the persistent storage of data on secondary storage like hard disks. All presented spatial access methods are suitable of two-, three- or more dimensional

points. They support the efficient processing of basic *spatial queries*. Such spatial queries are:

- spatial selection queries like the point query and the window query,
- the computation of k nearest neighbors (nearest neighbor query, see e.g. (Hjaltason & Samet, 1999)), and
- the spatial join (see e.g. (Brinkhoff et al., 1993)).

Index structures typically preserve the ordering of data locally. In the case of spatial access methods, spatially close objects are stored with high probability in the same database block. This is essential for processing spatial queries because one query accesses typically many spatially neighbored objects and because reading a database block from secondary storage is a very costly (i.e. slow) operation in comparison to other computer operations. Storing near objects in the same block reduces the number of block accesses and increases the probability to find the block in the cache of the database system or of the operating system.

Other techniques try to store blocks, which are described by spatially close block regions, physically close on the secondary storage ("global order"). The objective is to reduce the cost for reading sets of blocks required by one spatial query. The presented spatial access methods do not preserve the global order. This would require the usage of additional techniques like the approaches proposed by (Hutflesz et al., 1988) or by (Brinkhoff, 2001).

3. USAGE FOR DATA FROM LASERSCANNING

3.1 Preparation

The investigation of the spatial access methods presented in the sections 2.3 and 2.4 for data produced by laserscanning required some preparations:

- Modula-2 implementations of the BANG file and the buddy tree, both implemented for old Motorola processors, were adapted to current Intel processors. A unified API was designed and implemented.
- Besides the well-known *R*-tree* (Beckmann et al., 1990) – a very efficient variant of the R-tree – the new *Revised R*-tree (RR*-tree)* of Beckmann & Seeger (2004) was implemented in Java.
- The two Modula-2 implementations and the Java implementation of the R-tree variants were integrated under a unified Java user interface.

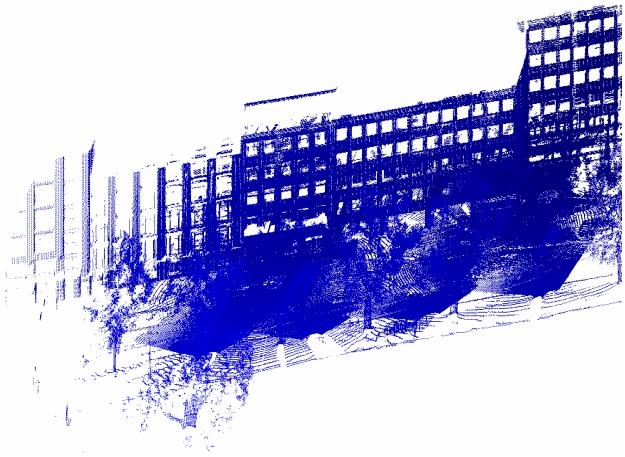

Figure 7. Illustration of the test data.

The data structures were tested by a cloud of points that originates from the measurement of the façade of a building. The data set consists of about 3.66 million points. It is illustrated in Figure 7.

3.2 Investigation

The following experiments were performed on a Pentium IV-PC with 2 GHz and 256 MB main memory.

The aspect investigated first was the construction of the spatial access methods. The points were inserted into the index structures in the order they have been measured. The observed storage overhead is – independently of the data structure – about 68%. The reasons are the storage requirements for the directory and – more significant – empty space in the data nodes. The empty space is due to fact that all investigated spatial access methods are dynamic index structures allowing insertions and deletions without worsening their performance. The empty space can be reduced by using special bulk-loading algorithms.

The insert throughput is depicted in Figure 8. The performance of the most spatial access methods is rather high. One exception can be observed: the R*-tree achieved only a third of the throughput compared to the other trees.

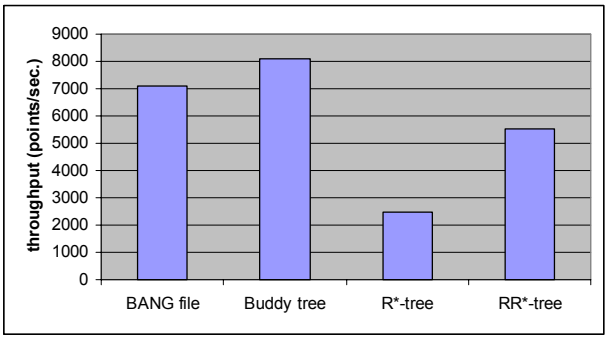

Figure 8. Throughput of the insertion of points.

All spatial access methods showed an excellent performance for extracting data points queried by small query cuboids. The access to the data does not require a long loading phase. The queried points were accessed by few block accesses.

Many applications require extracting an overview about the distribution and/or location of the points. One example is a rough visualization which is often sufficient for getting an impression of the data. A reasonable approach fulfilling such a requirement is to stop the traversal of the spatial access method as soon as the size of a block region falls under a given threshold. Such an approach reduces the number of block accesses significantly. However, the tests have shown that the query time was not reduced significantly. The reason is that the investigated spatial access methods preserve only the local order. In consequence, block accesses on the secondary storage had often led to a slow seek operation on disk. Therefore, the standard access methods were compared to globally reordered variants. For the globally reordered variants, the blocks were sequentially arranged on secondary storage according to a depth first traversal through the respective tree. This rearrangement was performed in a post-processing step after the insertion of the points. Figure 9 shows the results of this comparison. The query time of the standard version is set to 100% for each

spatial access method. We can observe a performance improvement of factors between 6 and 10.

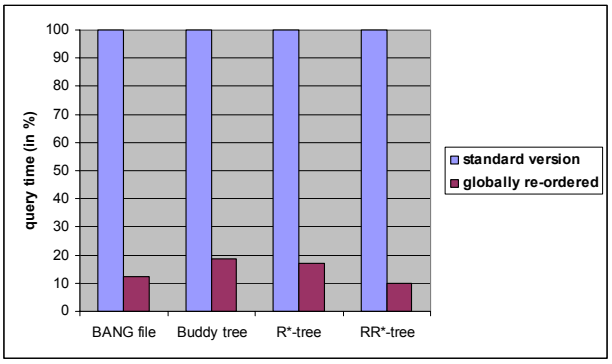

Figure 8. Comparison of query time.

4. CONCLUSIONS

In this paper, we discussed the usage of spatial access methods, which have been originally developed for organizing spatial data in spatial database systems and in GIS, for the persistent storage of point clouds produced by laserscanning. As potential data structures, the BANG file and the buddy tree as representatives of hash trees and the R*-tree and the RR*-tree as R-trees have been selected, implemented and experimentally investigated using real laserscanner data. The first results show that both types of index structures have the potential for organizing point clouds originating from laserscanning.

Two important tasks for future work can be identified: 1. The definition of typical query profiles. Such profiles would allow a more detailed investigation and comparison of index structures. 2. The order preserving properties and spatial hierarchies of spatial access methods may be used for analysing the clouds of points measured by laserscanners. Especially the extraction and approximation of surfaces and edges (e.g. like in (Niemeier & Kern, 2001) should be considered.

5. REFERENCES

Beckmann, N., H.-P. Kriegel, R. Schneider & B. Seeger, 1990. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: *Proceedings ACM SIGMOD International Conference on Management of Data*, Atlantic City, NJ, pp. 322-331.

Beckmann, N. & B. Seeger, 2004. Ready for System Integration: A Revised R*-tree with Improved Insertion and Search Performance. Technical Report of the University of Marburg.

Brinkhoff, T., 2001. Using a Cluster Manager in a Spatial Database System. In: *Proceedings 9th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS)*, Atlanta, GA, pp. 136-141.

Brinkhoff, T., H.-P. Kriegel & B. Seeger, 1993. Efficient Processing of Spatial Joins Using R-trees. *Proceedings ACM SIGMOD International Conference on Management of Data*, Washington, DC, pp. 237-246.

Freeston, M., 1987. The BANG file: A new kind of grid file. In: *Proceedings ACM SIGMOD International Conference on Management of Data*, San Francisco, CA, pp. 260-269.

Guttman, A., 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In: *Proceedings ACM SIGMOD International Conference on Management of Data*, Boston, pp. 47-57.

Hjaltason, G.R. & H. Samet, 1999. Distance Browsing in Spatial Databases. *ACM Transactions on Database Systems*, (24)2, pp. 265-318.

Hutflesz, A., H.-W. Six & P. Widmayer, 1988. Globally Order Preserving Multidimensional Linear Hashing. In: *Proceedings 4th International Conference on Data Engineering*, Los Angeles, CA, pp. 572-579.

Niemeier, W. & F. Kern, 2001. Anwendungspotentiale von scannenden Messverfahren. In: U. Weferling et al. (eds..): *Von Handaufmaß bis High Tech*, Verlag Philipp von Zabern, pp. 134-140.

Nievergelt, J., H. Hinterberger & K.C. Sevcik, 1984. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems*, (9)1, pp. 38-71.

Rigaux, P., M. Scholl & A. Voisard, 2002. Spatial Databases With Application To GIS. Morgan Kaufmann Publishers, San Francisco.

Schwermann, R. & C. Effkemann, 2002.: Kombiniertes Monoplotting in Laserscanner- und Bilddaten mit PHIDIAS. In: Luhmann T. (ed.), *Photogrammetrie und Laserscanning, Anwendung für As-Built-Dokumentation und Facility Management*. Herbert Wichmann Verlag. pp. 57-70.

Seeger, B., 1989. *Entwurf und Implementierung mehrdimensionaler Zugriffsstrukturen*. Dissertation of the University of Bremen.

Seeger, B. & H.-P. Kriegel, 1990. The Buddy Tree: An Efficient and Robust Access Method for Spatial Databases. In: *Proceedings 16th International Conference on Very Large Data Bases*, Brisbane, Australia, pp. 590-601.