ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume I-3, 2012
XXII ISPRS Congress, 25 August – 01 September 2012, Melbourne, Australia

THE OPALS DATA MANAGER – EFFICIENT DATA MANAGEMENT FOR PROCESSING LARGE AIRBORNE LASER SCANNING PROJECTS

J. Otepka [a, *], G. Mandlburger [a], W. Karel [a]

[a] Institute of Photogrammertry and Remote Sensing, Vienna University of Technology Vienna, Karlsplatz 13, 1040 Wien, Austria - (jo, gm, wk)@ipf.tuwien.ac.at

Commission III, WG III/2

KEY WORDS: Airborne Laser Scanning, LIDAR, data management, software concept, spatial indices, point clouds

ABSTRACT:

The fast measurement rate of today's Airborne Laser Scanners results in billions of points for single ALS projects. Efficient algorithms and data management methods are, therefore, a precondition for successful project handling. The software package OPALS (Orientation and Processing of Airborne Laser Scanning data) was especially designed to meet those criteria. Central core of the package is the OPALS Data Manager (ODM). It provides both, fast spatial access to huge point clouds, as well as a flexible attribute schema to store additional point related quantities.
Concepts of the spatial data organization and implementation details about the attribute handling are presented. Additionally, design rationales of the ODM, its file format and the system performance are described.

1. INTRODUCTION

Point clouds from Airborne Laser Scanning (ALS), also termed airborne LiDAR (Light Detection And Ranging), has proven to be an efficient acquisition method for a wide range of topographic applications. Nowadays, forestry, geology, hydrology, cartography, and other geo-disciplines and applications, including natural hazard studies, make use of digital models or specific parameter sets derived from laser scanning point clouds. The fast technological advances within the last two decades have opened up new areas of research and applications of LiDAR data.
A typical density of 10 laser echoes per m² covering a project area of 100km² results in 10^9 points. As ALS data sets may contain billions of points in a single project, efficient algorithms and data management methods are a precondition for successful project handling.

Modern LiDAR sensors acquire range data at an enormous rate but also provide valuable attributes for each detected echo. (Axelsson, 1999) argues that the full potential of airborne laser scanning can only be exploited, if conversions like vector to raster, or the omission of additional attributes (e.g., echo number, recording time) are avoided. Especially full-waveform scanners, which record the emitted and backscattered signals as a function of time, allow applying more sophisticated detection and extraction routines (Wagner et al., 2006). Extracted waveform quantities such as amplitude or echo width are useful for all sorts of applications (Doneus and Briese, 2006; Alexander et. al., 2010). Hence, it is an absolute requirement for state-of-the-art LiDAR data management to support a flexible schema in order to preserve these valuable point attributes.

ALS data are acquired by a series of distinct, overlapping flight strips. Several processing steps like quality control and documentation or georeferencing of the raw point clouds (Ressl et. al., 2008) rely on strip-wise data handling. On the other hand, when computing a feature for each LiDAR point (surface normal vector, surface roughness, etc…), information from all overlapping strips in the vicinity of the point should be taken into account. This argument favors a project or block-wise data handling in which the original strip ID may be an attribute. Obviously, access to the data in strip-wise or in a compound manner depends on the processing stage. There is no optimal single state. Therefore, the data management tools should support both handling strategies.

1.1 State-of-the-Art Point Cloud Handling

Driven by the demand for high-resolution data, various strategies, new file formats, and point cloud libraries have been developed in the last couple of years.

Isenburg (2011) is the developer of LAStools, a set of tools for handling and processing LiDAR data based on the LAS file format. A fork of this project is libLAS, a LiDAR data translation toolset (libLAS, 2012). A comprehensive library for processing 3D point data is the Point Cloud Library (PCL) (Rusu and Cousins, 2011) which has a strong computer vision background. The Point Data Abstraction Library (PDAL, 2012) is focusing on import, export and filtering point data. However, it has not been released yet.

David et. al. (2008) proposed a library concept that stores LiDAR scan lines as pixel lines within multi layer raster files providing spatial access based on the natural acquisition topology. It has been implemented within the FullAnalyse project. Bunting et. al. (2011) proposed a new file format for discrete and full-waveform data sets implemented in the Sorted Pulse Data Library (SPDLib). Another interesting new development is the E57 file format, a compact, vendor-neutral format for storing point clouds, images, and metadata produced by any 3D imaging system defined within the ASTM E2807 standard. An implementation is available through the libE57 (2012) Library.

[*] Corresponding author

Beside the fact that most of the aforementioned developments are quite new, OPALS favors its own data handling strategies because of supporting:

- database features like dynamic and extendable attribute schema, data views and null values
- point, line, and polygon data

1.2 OPALS

The software package OPALS is developed at the Institute of Photogrammetry and Remote Sensing at the Vienna University of Technology. It is designed for Orientation and Processing of Airborne Laser Scanning point clouds providing tools for the entire processing chain starting from analysis of the full-waveform signal, quality control, geo-referencing, structure line extraction and terrain model derivation, to subsequent applications (forestry, city modeling, etc.).

Important concepts of OPALS are flexible automatic work flows for processing huge data sets. This was achieved by a modular design based on small, well-defined components and extensive scripting support. Each component (module) can be accessed as command line executable, as python module or via a C++ API. This gives the user full control to compose arbitrary processing chains. More details about OPALS and its framework can be found in (Mandlburger et. al., 2009) and (OPALS, 2012)

Central core of most OPALS modules is the OPALS Data Manager (ODM). Main feature of the ODM is to support, both, efficient spatial queries and a dynamically extendable attribute schema. The ODM comes into play after raw georeferencing of the point cloud. Hence, data processing is enabled on a per strip basis as well as for the entire flight block. OPALS modules potentially compute new point features, which are then appended to the existing point attributes. In most cases, the neigbhourhood of the points based on metric measures are analysed. E.g. *opalsNormals* estimates the three components of the surface normal vector (nx, ny, nz) by fitting a robust regression plane through the k-nearest points; *opalsAddInfo* allows computing new arbitrary attributes by combining existing attributes and (potentially) raster data sets in a generic mathematic formula. Furthermore, OPALS modules support powerful filters to sub-select the input data based on attributes. Obviously, attributes play an important role in the flexible and modular concept of OPALS.

2. ODM CONCEPT

One of the major issues for processing ALS projects is to tackle the huge amount of data. Memory limits usually prevent the entire point cloud from being loaded into random access memory. Hence, it is necessary to use strategies that split the overall data set in appropriate chunks and process them automatically considering a certain overlap. In case of topographic data, the horizontal extent exceeds the vertical range by far. Thus, indexing the ground plane domain is of higher importance than indexing the elevation.

Within OPALS LiDAR data are imported into the ODM in an initial step. Using *opalsImport* one or multiple input files can be imported into a single ODM. Thereby, the full information of the input data including all attributes is preserved. For each point, the correspondence to the original file is stored such that the original file structure can be restored from the ODM after processing.

To save disk space the ODM uses a global reduction point (double precision) which allows storing the actual geometry objects with single precision.

2.1 Spatial indices

Laser scanning data, as well as data from automatic image matching (digital photogrammetry) consist of point data only. However, the support of polygonal data is crucial for certain projects. E.g. terrain structure lines, water bodies in the form of shore and coast lines, river axes or boundaries of vegetation classes may either be measured manually or result from (semi-)automatic extraction processes (Briese, 2006; Mandlburger et. al., 2011). Therefore, a versatile data handling concept has to support, both, point and polygonal data in a unified manner.

Over the last three decades a variety of different spatial indices have be developed, which all have one thing in common; they subdivide the index space domain in a regular or irregular manner. Depending on data types (point, line, polygon, etc.), data distribution, query types and potential update operations different indexing methods should be used. There is no optimal index for all situations. E.g. the k-d tree (Bentley, 1975) is a very fast indexing method for nearest neighbor queries but does not support line data. Additionally, the k-d tree rapidly gets unbalanced in case of update operations (insert or delete points). Quad- and octtree structures are well suited for update operations and support point and polygonal data. However, they cannot compete in terms of speed regarding nearest neighbor queries.

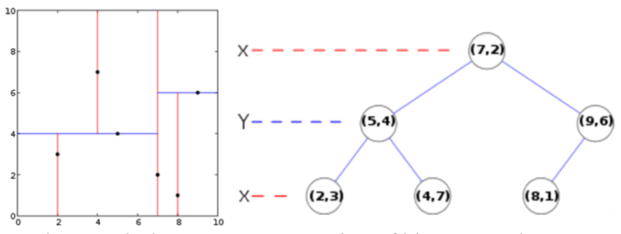

Figure 1. k-d trees are an extension of binary search trees

Whereas quad- and octrees are limited in adapting to inhomogeneous data distributions, R-trees group nearby objects using minimum bounding rectangles or boxes in the next higher level of the tree. The key difficulty of R-trees is to build balanced trees and to minimize, both, coverage and overlap of node bounding hyperboxes (Beckmann et al., 1990).

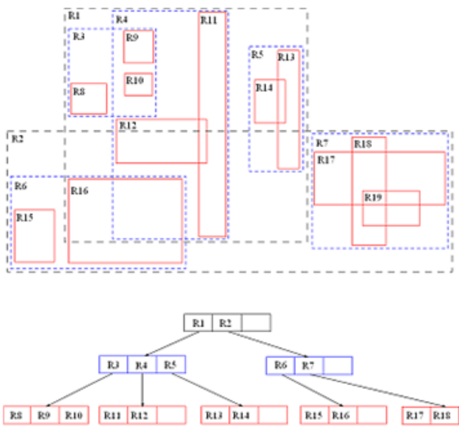

Figure 2. R-trees partition space based on bounding rectangles/boxes (hyperboxes)

However, R-trees tend to outperform quad- or octtrees (Kanth and Kothuri, 2002; Kim and Patel, 2010) since it is possible to build balanced trees even for inhomogeneous data distributions. Heavy update activity can lead to unbalanced trees which may outweigh the aforementioned advantage.

2.2 Spatial ODM concept

There are three requirements that a considerable spatial indexing structure for ALS data has to meet (in priority order):

1. Support of swapping strategies since memory limitations prevent the entire point cloud being loaded into random access memory
2. Maximum performance for point data on spatial queries, such as k nearest neighbor and range queries
3. Support of point, line, and polygonal data
4. Thread safety to support Multi-Core CPUs

Combining those requirements with the background knowledge of spatial indices from the previous section, it becomes evident that maximum performance for spatial queries of point data and the support of polygonal data cannot be provided within a single spatial index. Within typical ALS projects the point data exceed the amount of line and polygon data by far, justifying the usage of the optimal index for point data. This is why the ODM uses different spatial indices for point and polygon data.

2.2.1 Point Index of the ODM

Point data are spatially indexed in two levels within the ODM. First, the data are sorted into a regular grid of quadratic tiles allowing the implementation of a straight forward swapping strategy. The tile size can be manually set or is, else, automatically computed based on point density estimations. Currently, OPALS uses relative small tiles containing 150.000 to 200.000 points in average. Hence, the ODM can keep a decent number of tiles in memory at once. Several OPALS modules (*opalsNormals*, *opalsEchoRatio*, etc.) use tiles as processing units. Considering a certain data overlap (which requires neighboring tiles to be kept in memory as well) and multiple processing threads (potentially assigning a new tile to each thread) the memory limit is easily exceeded if the tiles are too big. Furthermore, the ODM uses tile caching based on a list of recently used tiles to minimize IO operations.

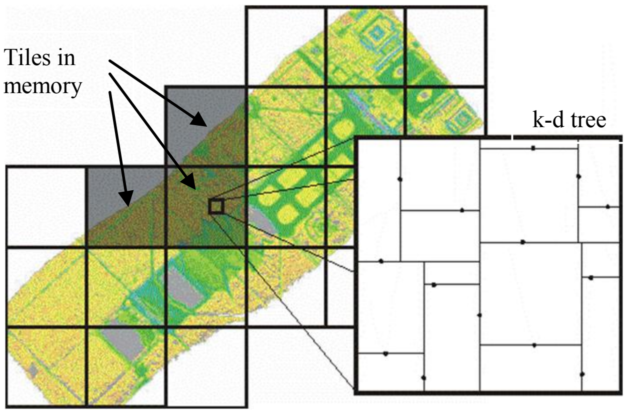

Figure 3. Two level point index structure of the ODM

In the second level, the points of a single tile are indexed using a k-d tree. Whereas the first level tile index is made persistent, the k-d trees are build on the fly after reading the tile points into memory. This has several advantages:

- Less disk space is required since no second level index information is stored
- Different indices can be used depending on processing needs (e.g., 2D or 3D k-d trees)
- During data import new points are always appended at the end of the tile. Therefore, it is never necessary to reorganize the point data within a tile which saves additional IO operations.

The implementation of the k-d tree is based on the respective implementation of the Computational Geometry Algorithms Library (CGAL, 2012). K-d trees use coordinate axis aligned hyperplanes for space-partitioning. The original k-d tree algorithm uses the median of all x coordinates to split the root node. Then, the median of the y coordinates is used for partitioning the sub nodes; and so on. In case of a 2D k-d tree the median search utilizes the x, y coordinates in a circular manner (c.f. Figure 1). This leads to a balanced tree (all leaf nodes are at the same tree level). However, depending on the data distribution and application a balanced tree does not necessarily result in the best performance for certain operations. Within a tile with strongly varying extents (e.g. at the edge of a flight strip), the distance between splitting k-d tree hyperplanes will reflect the specific variation of the data extents. Hence, the non-directional nearest neighbor search has to investigate more tree nodes as in situation with homogeneous hyperplane distances. This is why the CGAL k-d tree implementation has a flexible splitting concept and comes along with a set of predefined splitting strategies.

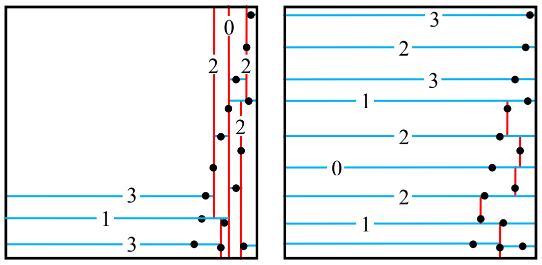

Figure 4: Splitting lines using the original k-d tree algorithm (left) and a median of maximum spread split (right)

The ODM uses the CGAL median of maximum spread splitting strategy in two and three dimensions with some improvements to increase the robustness in case of odd point arrangements. Although median splitters are expensive to compute, tests have shown that the limited number of points per tile result in negligible differences regarding tree building times.
The selected implementation does not support dynamic insertion or deletion. However, this is in line with the overall OPALS concept as certain point measurements are rather disregarded for processing (using attribute filters) than permanently deleted from the data set.

2.2.2 Polygonal data index of the ODM

The ODM manages all non point data in a second independent index. Typical linear data of interest are terrain structure lines or river networks, but also 2-dimensional polygons, such as catchment polygons, parcels, buiding outlines or vegetation boundaries. The ODM uses an R*-tree index implementation of the Spatial Index Library developed by (Hadjieleftheriou et al., 2002). As mentioned in 2.1 minimal coverage and overlap of the bounding hyperboxes is crucial to the performance of R-trees. For this reason different R-tree variants have been developed using heuristics to minimize overlap and coverage.
The R*-tree combines a revised node split algorithm and the concept of forced reinsertion at node overflow. Deletion and

reinsertion of entries allow "finding" a new position that may be more appropriate than their original location within the tree (Wikipedia, 2012).

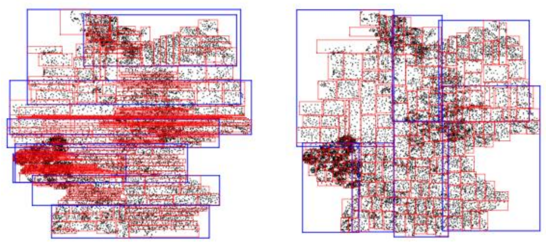

Figure 5: Examples of visualized indices of different splitting heuristics; left: R-tree quadratic split; right: R*-tree topological split (Wikipedia, 2012).

In contrast to the CGAL k-d tree implementation, the Spatial Index Library already contains thread-safety support and swapping strategies to handle big data set. However, the amount of polygonal data in typical ALS application is noncritical.

In order to query point and line related data in a uniform way, the ODM provides a common interface for the separated indices, but it is possible to access both indices separately as well.

2.3 Attribute schema

As pointed out in 1.2, additional point attributes are of crucial importance for ALS data processing. The implemented ODM concepts are comparable to tables of relational databases and support an unlimited number of attributes and a variety of different data types for efficient storage usage.

Data type	Memory consumption
Byte	1 byte
unsigned byte	1 byte
Short	2 byte
unsigned short	2 byte
Integer	4 byte
unsigned integer	4 byte
64bit integer	8 byte
float	4 byte
double	8 byte
fixed length string	n bytes
variable length string	4 + n bytes
Boolean	1 byte in memory 1 bit on disk

Table 1. Supported data attribute types and their memory consumption

ODM attributes can be dynamically added while processing, thus, the attribute layout does not have to be defined in advance. As a consequence, each geometry object (point and polygonal object) can have a different set of attributes. This feature helps to save storage space if data within an ODM are only partly processed or data from different acquisition methods (e.g. laser scanning and photogrammetric image matching) are combined within one ODM. Additionally, a null flag is provided for each attribute value; hence, a specific no-data indicator is not required.
The ODM also supports the database concept of views. They are used by OPALS modules to inspect a specific set of attributes. A view guarantees access to the sought-after attributes in the corresponding order. The ODM is not limited to one view. Hence, a module can use multiple views on the same data set at once.

ODM attributes are differentiated into two groups:

- Predefined attributes (with semantic)
- User-defined attributes (without semantic)

Predefined attributes have fixed names and data types, as well as a well-defined semantic and unit. Examples are attributes that are defined in the LAS format (ASPRS, 2012), or attributes computed by specific OPALS modules (e.g. *opalsNormals*).
On the other hand, the name and type of user-defined attributes are freely selectable. Those attributes can come from the data import or from the generic attribute generation module *opalsAddInfo*. Although user-defined attributes have an unknown semantic, it is well possible to use them in filters or as processing attributes in most OPALS modules. This powerful functionality allows investigating and establishing new processing concepts on a user level.
A special ODM feature is the on-the-fly statistics computation (count, minium, maximum, average and sigma) of all attribute columns. The statistics give a good overview about the attribute values and help to speed up certain processing tasks. The actual computation is carried out in a background thread to secure a minimal effect on the overall performance.

With the flexible attribute schema and the spatial indices the ODM is comparable to geo-relational database systems. So why not using mature database systems like PostgreSQL/PostGIS (Refraction Research, 2012) or Oracle Spatial (Oracle, 2012) offering full administration capabilities, concerning both, geometry and attributes? The answer is performance and dynamic extendabilty. The ODM is an efficient light-weight tool tuned for high-speed processing and flexibility but not for transactions control or user security functionality. The strength of database systems is the long-term archiving and persistence, rather than a fast and optimized data access as necessary for ALS projects.

2.4 Persistent form of the ODM

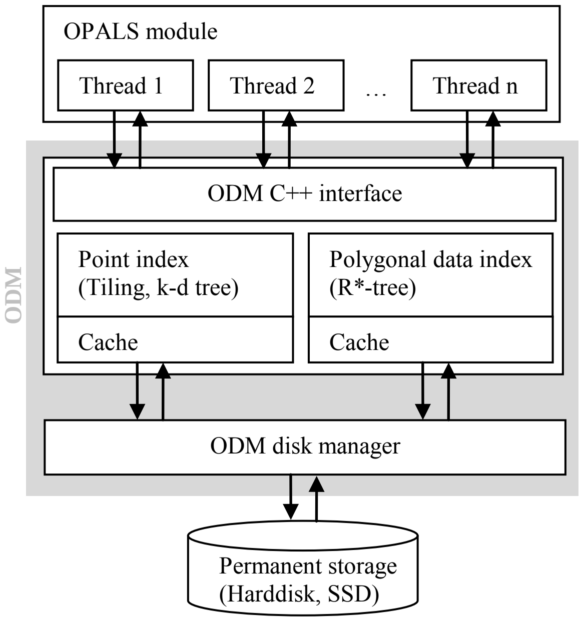
OPALS module
Thread 1
Thread 2
…
Thread n
ODM
ODM C++ interface
Point index (Tiling, k-d tree)
Polygonal data index (R*-tree)
Cache
Cache
ODM disk manager
Permanent storage (Harddisk, SSD)

Figure 6. Internal structure of the ODM

So far design rationales of the general data organization and the used spatial indices have been discussed. This leads to a few basic conditions on how the data have to be made persistent. However, the format of the point tiles and attributes is not defined by the index implementation. All IO operations within the ODM are carried out by a specific disk manager component. High-level objects (e.g. full data tiles) are passed to the ODM disk manager storing the data in the corresponding format. The disk manager is decoupled from the actual managing structure (see). Hence, from a conceptional point of view, it is straight forward to support a different persistent form. At the current state, all data are stored in a single file as described in the following section. However, new possibilities in LAS 1.4 to describe attributes in a generic way, will lead to an additional ODM disk manager implementations in the near future (see section 5)

2.4.1 Single file disk manager

The current disk manager implementation stores a full ODM in a single file making ALS project management flexible and straight forward. Since the disk manager has to support dynamic IO operation of independent data streams (e.g. one stream for each point tile), a simple file system structure was used for realization. Such a file is always structured in chunks (or sectors). The ODM uses IDs to identify different streams. A stream itself consists of a list of chunks that can grow or shrink dynamically. If a stream is deleted, all corresponding chunks are added to a simple garbage collection. In case a new chunk is required, it is either requested from the garbage collection or appended at the end of the file. Beside the actual data chunks, the administration information (ID chunk lists and garbage collection) must be stored within the file as well. It is most common to store this information at the beginning or at the end of file (header/trailer). Both options have different advantages and disadvantages. The ODM disk manager stores the administration information in a trailer at the end of the file.

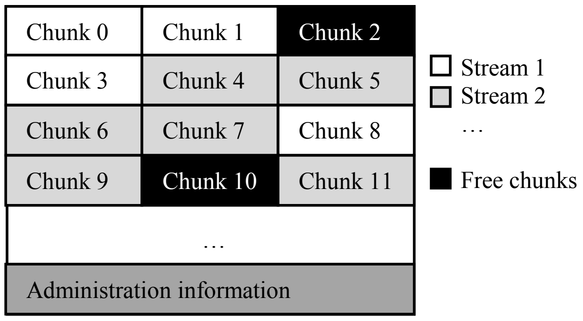

Figure 7. Internal structure of the ODM file

A unique ID is assigned to each high level object that is passed to the disk manager. Hence, if such an object is updated, the entire object has to be re-written by the disk manager. Each tile is represented by two independent streams; a coordinate stream and an attribute stream. This allows reducing write operations and, thus, improving the overall performance.

2.5 Processing Strategies

Nearly all OPALS Modules process the input data in rectangular chunks from the upper left to lower right corner. The size of the processing window either corresponds to the native ODM tiling structure (e.g. *opalsNormals*, *opalsEchoRatio*, etc.) or is adapted to the output format (e.g. titled GeoTiff). Most modules use an additional overlap for data selection (e.g. *opalsNormals*, *opalsGrid*, etc.) to avoid artifacts at the border of the processing window. In case of multiple processing threads, each thread processes a different chunk to minimize the synchronization effort. Each thread holds a shared pointer (with reference counting) to its corresponding ODM point tile object. This allows an easy detection of tiles that can be dropped from memory without explicitly releasing them.

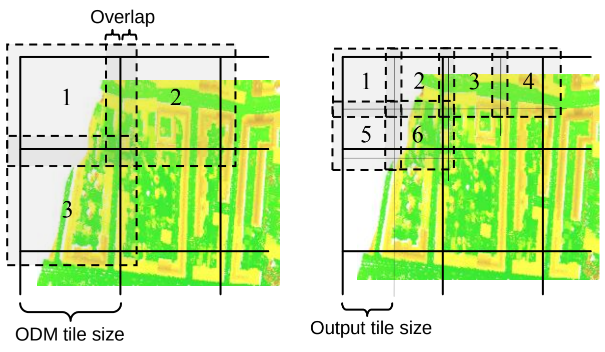

Figure 8. Processing strategy of *opalsNormals* (left) and *opalsGrid* (right)

3. PROCESSING PERFORMANCE

Although a detailed discussion of the processing performance is beyond the scope of this paper, a few computation times are presented in the following since the performance is crucial to efficient LiDAR software.

The test procedure consists of typical tasks performed within most LiDAR projects:

- Derivation of a digital surface model (DSM) using moving least squares interpolation with a tilted plane model (*opalsGrid)*
- Computation of a point density image to secure that the tendering specifications are met (*opalsCell*)
- Estimation of a local surface plane for each point (*opalsNormals*) for subsequent use in radiometric calibration, segmentation, classification, etc..
- Computation of the echo ratio (ER) for each point (*opalsEchoRatio*). The ER is a local measure of transparency and roughness useful for building detection (Höfle et. al., 2009) and forest delineation (Eysn et. al., 2012)

The processing times of two data sets with 11 and 36 million points (see Figure 9) given in LAS format (274 MB and 979 MB) are listed in Table 2. The data sets were processed on an Intel Core i7 Computer with 8 GB Ram running on Windows 7 64 bit.

	Data set 1 11 million pts	**Data set 2 36 million pts**
opalsImport	26.3 [s]	105.2 [s]
opalsGrid (1m grid size)	24.5 [s]	67.6 [s]
opalsCell (5m cell size)	30.0 [s]	58.5 [s]
opalsNormals	69.4 [s]	216.1 [s]
opalsEchoRatio	60.0 [s]	192.6 [s]

Table 2. Processing times of different OPALS modules on a Core i7 64bit Computer

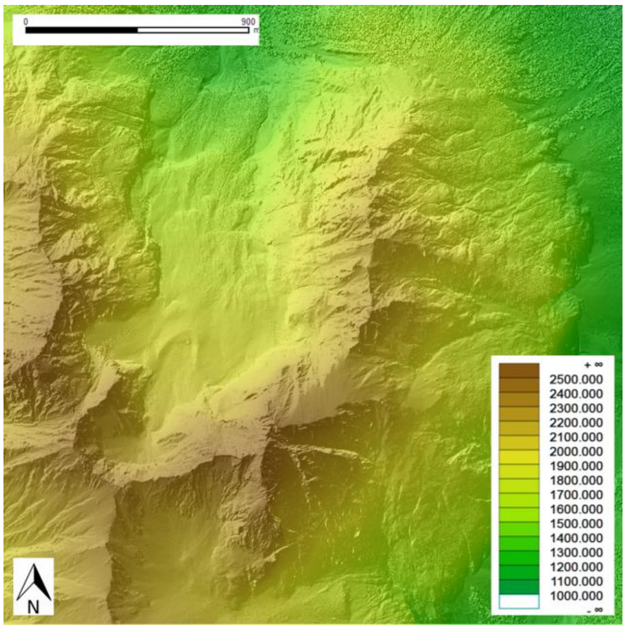

Figure 9. DSM of data set 2; elevation coding overlayed with hill shading; data: Province of Tyrol

4. IMPLEMENTATION DETAILS

The ODM is programmed in ISO C++ using a set of modern standard open-source libraries:

- Standard Template Library (STL)
- (Boost, 2012) Libraries (Exception, Filesystem, Serialization, Smart Ptr, Thread, etc.)
- CGAL
- Geospatial Data Abstraction Library (GDAL, 2012)

Like most of the aforementioned libraries, the ODM uses templates to a high degree. Templates allow programming in a generic way that is far beyond the possibilities of classic C++. Additionally, the compiler can optimize the code to a higher degree, since less virtual function calls are usually needed. The downside of template programming is the harder readability of the code and, especially, the slower compile times in case of a larger software project. This is why the C++ Interface of the ODM does not contain templates, but only uses abstract interfaces favoring the loose coupling principle.
Parallelization of computationally intensive tasks is state-of-the-art to support modern multi-core CPUs. OPALS and the ODM use (OpenMP, 2012) and the Boost Thread Library to fulfill this requirement.

Although OPALS and the ODM are programmed in ISO C++ and multi-platform libraries are used, at the moment only a Windows version is available. However, efforts are currently made to port the code to Linux.

5. SUMMARY AND OUTLOOK

Processing large ALS projects requires efficient algorithms and data handling strategies. In this paper we presented the ODM as a central core of the software packages OPALS providing, both, efficient spatial access to billions of points and a flexible attribute schema to store arbitrary quantities along with each point. Combined with the extensive scripting functionality of OPALS individual processing strategies can be established.

The ODM uses two different indices to optimize the performance of spatial queries. One index manages all point data whereas the second index organizes polygonal data. The point index is structured in two levels. First, the point data are sorted into regular tiles whereas k-d trees are built on the fly as a second level for each tile.
The persistent format of the ODM is controlled by the ODM disk manager. Currently, a single file in a proprietary format is used due to the lack of an appropriate open standard format. The situation has changed with the draft specification of LAS 1.4 which has been released in October 2011. Whereas LAS has always supported an additional memory block per point (extra bytes), there was no possibility to generically describe its content. The new specification now provides an appropriate way of describing additional point attributes. Based on this new feature, it would be possible to organize each tile in a separate LAS file containing all attributes. The ODM would, consequently, change from a single file to a directory containing all point tiles as LAS files and some additional metadata files. As a result, the user would have direct access to the ODM data structure. This additional disk manager is a medium-term goal of the ODM developments.
Although the ODM supports a variety of simple data types, compound data types like vectors, matrices, and lists are interesting for certain attributes (e.g. normal vector, beam vector, etc.). Virtual attributes, combining existing attributes by a specific formula, are another planed feature (e.g. beam vector length, incident angle, etc.) This reduces disk consumption and avoids redundancy.
The static first-level tiling concept of the ODM turned out to be reliable for ALS because of the homogenous point distribution over the ground plane domain. Only in very rare cases with exotic point arrangements, OPALS modules ran into problems because the tile size was selected inappropriately. At the Institute of Photogammetry and Remote Sensing, OPALS and the ODM was already successfully used to process Terrestrial Laser Scanning (TLS) data. As expected, the tile size selection is the crucial point for processing such data. For TLS (or combined ALS/TLS) processing within OPALS, it is planned to replace the first level index by a quad- or octtree structure to support more heterogeneous data distributions.
Another interesting topic of research is loss-less compression. (Isenburg, 2011) has developed an efficient LAS file compressor. The compressed files are about 10% of the original file size. In the face of such high compression rates, it is tempting to use compression within the ODM. Though, the IO of compressed data is usually slower than uncompressed IO, it is foreseen to add compression to the ODM while at the same time preserving the key features of performance and flexibility.

References

Alexander, C., Tansey, K., Kaduk, J., Holland, D., Tate, N. J., 2010. Backscatter coefficient as an attribute for the classification of full-waveform airborne laser scanning data in urban areas. ISPRS Journal of Photogrammetry and Remote Sensing 65 (5), pp. 423-432.

ASPRS, 2011. LAS file format exchange activities, http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html (15.1.2011).

Axelsson, P., 1999. Processing of laser scanner data – algorithms and applications. ISPRS Journal of Photogrammetry and Remote Sensing 54, 138-147.

Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B., 1990. The r*-tree: An efficient and robust access method for points and rectangles. Garcia-Molina, H., Jagadish, H. V. (Eds.), Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990. ACM Press, pp. 322-331.

Bentley, J. L., 1975. Multidimensional binary search trees used for associative searching. Communications of the ACM 18 (9), pp. 509-517.

Boost, 2012, Homepage of Boost C++ Libraries
http://www.boost.org (15.1.2012)

Briese C., 2006. Structure line modelling based on terrestrial laserscanner data, Symposium of ISPRS Commission V - Image Engineering And Vision Metrology, H. Maas, D. Schneider (Ed.); XXXVI/5 (2006), 1682-1750

Bunting P., Armston, J., Clewley, D., Lucas, R., 2011. Sorted Pulse Data (SPD) Format: A new file structure for storing and processing LiDAR data, SilviLaser 2011

CGAL, 2012. Homepage of the Computational Geometry Algorithms Library
http://www.cgal.org (12.1.2012)

David, N., Mallet, C. and Bretar, F., 2008. Library concept and design for lidar data processing. In: GEOgraphic Object Based Image Analysis (GEOBIA) Conference, Calgary, Canada.

Doneus, M., Briese, C., 2006. Digital terrain modelling for archaeological interpretation within forested areas using full-waveform laserscanning. In: The 7th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST. Cyprus.

Eysn, L., Hollaus, M., Schadauer, K., Pfeifer, N., 2012, Forest Delineation Based on Airborne LIDAR Data, Remote Sensing, 4 (2012), 3, pp. 762 - 783

GDAL, 2012. Homepage of the Geospatial Data Abstraction Library
http://www.gdal.org (15.1.2012)

Hadjieleftheriou, M., Kollios, G., Tsotras, V., Gunopulos, D., 2002. Efficient indexing of spatiotemporal objects. 8th International Conference on Extending Database Technology (EDBT). Prague, Czech Republic.

Höfle, B., Mücke, W., Dutter, M., Rutzinger, M. and Dorninger, P., 2009. Detection of building regions using airborne LiDAR – A new combination of raster and point cloud based GIS methods. GI_Forum 2009 - International Conference on Applied Geoinformatics, Salzburg, pp. 66-75.

Isenburg, M. 2011, LASzip: lossless compression of LiDAR data, European LiDAR Mapping Forum 2011 (ELMF)

Kanth, R., Kothuri, V., 2002. Quadtree and r-tree indexes in oracle spatial: a comparison using gis data. In: Proceedings of ACM SIGMOD Conference, 546-557.

Kim, Y. J., Patel, J. M., 2010. Performance comparison of the r*-tree and the quadtree for knn and distance join queries. IEEE Transactions on Knowledge and Data Engineering 22 (7), 1014-1027.

libE57, 2012. Homepage of libE57: Software Tools for Managing E57 Files
http://www.libe57.org/index.html (19.4.2012)

libLAS, 2012. Homepage of libLAS
http://liblas.org// (19.4.2012)

Mandlburger, G., Otepka, J., Karel, W., Wagner, W., Pfeifer, N., 2009. Orientation and processing of airborne laser scanning data (opals) - concept and first results of a comprehensive als software. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences38 (Part 3/W7). Paris, France, pp. 55-60.

Mandlburger, G., Vetter, M., Milenkovic, M., Pfeifer, N., 2011. Derivation of a countrywide river network based on Airborne Laser Scanning DEMs - results of a pilot study, 19th International Congress on Modelling and Simulation. Modelling and Simulation Society of Australia and New Zealand, 2423 - 2429.

OPALS, 2012. Homepage of OPALS
http://www.ipf.tuwien.ac.at/opals (12.1.2012)

OpenMP, 2012, Homepage of the OpenMP Architecture Review Board
http://openmp.org/wp/ (15.1.2012)

Oracle, 2012. Homepage of Oracle Spatial
http://www.oracle.com/technology/products/spatial (15.1.2012).

PDAL, 2012. Homepage of Point Data Abstraction Library
http://pointcloud.org/ (19.4.2012)

Refraction Research, 2012. Homepage of PostGIS
http://postgis.refractions.net (15.1.2012)

Ressl, C., Kager, H., Mandlburger, G., 2008. Quality Checking of ALS Projects Using Statistics of Strip Differences. In: International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 37 (Part B3b). pp. 253-260.

Rusu, R. B., Cousins, R., 2011. 3D is here: Point Cloud Library (PCL), International Conference on Robotics and Automation 2011 (ICRA)

Wagner, W., Ullrich, A., Ducic, V., Melzer, T., Studnicka, N., 2006. Gaussian decomposition and calibration of a novel small-footprint full-waveform digitising airborne laser scanner. ISPRS Journal of Photogrammetry and Remote Sensing 60 (2), pp. 100-112.

Wikipedia, 2012, R*-tree article in wikipedia
http://en.wikipedia.org/wiki/R*_tree (15.1.2012)