ORIENTATION AND PROCESSING OF AIRBORNE LASER SCANNING DATA (OPALS) - CONCEPT AND FIRST RESULTS OF A COMPREHENSIVE ALS SOFTWARE

Gottfried Mandlburger[a], Johannes Otepka[ab], Wilfried Karel[ab], Wolfgang Wagner[ab] and Norbert Pfeifer[a]

[a] Institute of Photogrammetry and Remote Sensing, Vienna University of Technology,
[b] Christian Doppler Laboratory "Spatial Data from Laser Scanning and Remote Sensing"
Gusshausstrasse 27-29/E122, 1040, Vienna, Austria,
gm{jo,wk,ww,np}@ipf.tuwien.ac.at

Commission III/2

KEY WORDS: LIDAR, ALS processing software, automatic workflow, data administration

ABSTRACT:

Since the mid-1990s, the Institute of Photogrammetry and Remote Sensing (I.P.F.) is engaged in Airborne Laser Scanning (ALS) in research and development. Scientific contributions have been made in a wide field of related topics like full waveform signal analysis, georeferencing and filtering of ALS point clouds, automatic breakline modelling, DTM generation, quality control, etc. Apart from that, converting research ideas into software solutions is an enduring tradition at the I.P.F. for which the DTM program SCOP++ is an example. Partial solutions of ALS-related issues have been implemented in SCOP++, but a complete processing chain is missing, as the development cycles for this highly interactive program are long.
Thus, the objectives of the new OPALS program system are to provide a complete processing chain for large ALS projects and to shorten development cycles significantly. OPALS is designed as a collection of small well-defined modules which can be accessed in three different ways: (i) from DOS/Unix shells as executables, (ii) from Python shells as full-featured, platform-independent Python modules or (iii) from custom C++ programs by dynamic linkage (DLL) for fastest module calls. Sophisticated custom processing chains can be established by freely combining the OPALS modules using shell or Python scripts. To reduce development times, a lightweight framework is introduced. It allows non-expert programmers to implement their own modules, concentrating on the implementation of their latest research outcomes, whereas the framework deals with general issues like validation of user inputs, error handling, logging, etc. In this way, new research outcomes get available more rapidly for the scientific community. OPALS does not only target researchers, but also ALS service providers dealing with large ALS projects. Efficient data handling is a precondition for this purpose. Thus, the OPALS data manager (ODM) is one of the core units, allowing administration of data volumes in the order of 10^9 points. The ODM acts as spatial cache and provides high-performance spatial queries.
Currently, a quality control package (opalsQC) is in progress and first results (point density maps, strip difference maps, 3D-strip shifts) are presented in the paper.

1 INTRODUCTION

Airborne laser scanning (ALS) has been a main research topic at the Institute of Photogrammetry and Remote Sensing (I.P.F.) for more than a decade. Scientific contributions have been made in a wide field of ALS-related topics like filtering of point clouds (Kraus and Pfeifer, 1998), derivation of digital terrain models (Kraus and Pfeifer, 2001), (Pfeifer et al., 2001), georeferencing of flight strips (Kager, 2004), quality control of ALS data (Ressl et al., 2008), automatic modelling of breaklines (Briese, 2004), and many fields of application like building modelling (Rottensteiner and Briese, 2002), (Dorninger and Pfeifer, 2008), hydraulic modelling (Mandlburger et al., 2008), forestry (Hollaus et al., 2007) and geomorphology (Székely et al., 2008). Currently, the research activities also focus on full waveform laser scanning (Wagner et al., 2004) comprising the decomposition and calibration of the laser echoes (Wagner et al., 2006), as well as the improvement of digital terrain models (DTM) using additional full waveform echo attributes (Doneus and Briese, 2006), (Mandlburger et al., 2007).

ALS is a highly automated data capturing technique. Today, the sensor observations from global navigation satellite systems (GNSS), from inertial measurement units (IMU) and laser scanners are processed online and are simultaneously stored on hard disc arrays even during the flight mission. Thus, a first inspection of the raw point cloud can be done while still airborne. However, more sophisticated data processing is typically done in postprocessing back in the office. The processing chain comprises analysis of the full waveform signal, direct georeferencing, strip-wise checking of relative and absolute accuracy of the point cloud, improvement of the georeferencing if necessary, data organisation and administration, and finally, filtering of the point cloud, and derivation of digital terrain or other models. In this paper we present a comprehensive framework for these steps. While relying on the algorithms for data processing mentioned above, especially data administration in suitable structures is a challenging task. Another question is at which stages quality control shall be incorporated into the process.

The article is structured as follows. Section 2 describes a best practice workflow for processing ALS projects representing the basis for our new OPALS software. In section 3, the general concepts of OPALS are highlighted, and section 4 describes an example module in more detail. The current status is presented in section 5, and the article concludes with the major findings and addresses future work in the final section 6.

2 ALS DATA PROCESSING

Fig. 1 shows the proposed processing chain based on full waveform ALS data. The first section deals with the derivation of the 3D point cloud starting with the raw observations. On the one hand, the flight path is determined combining the observations of

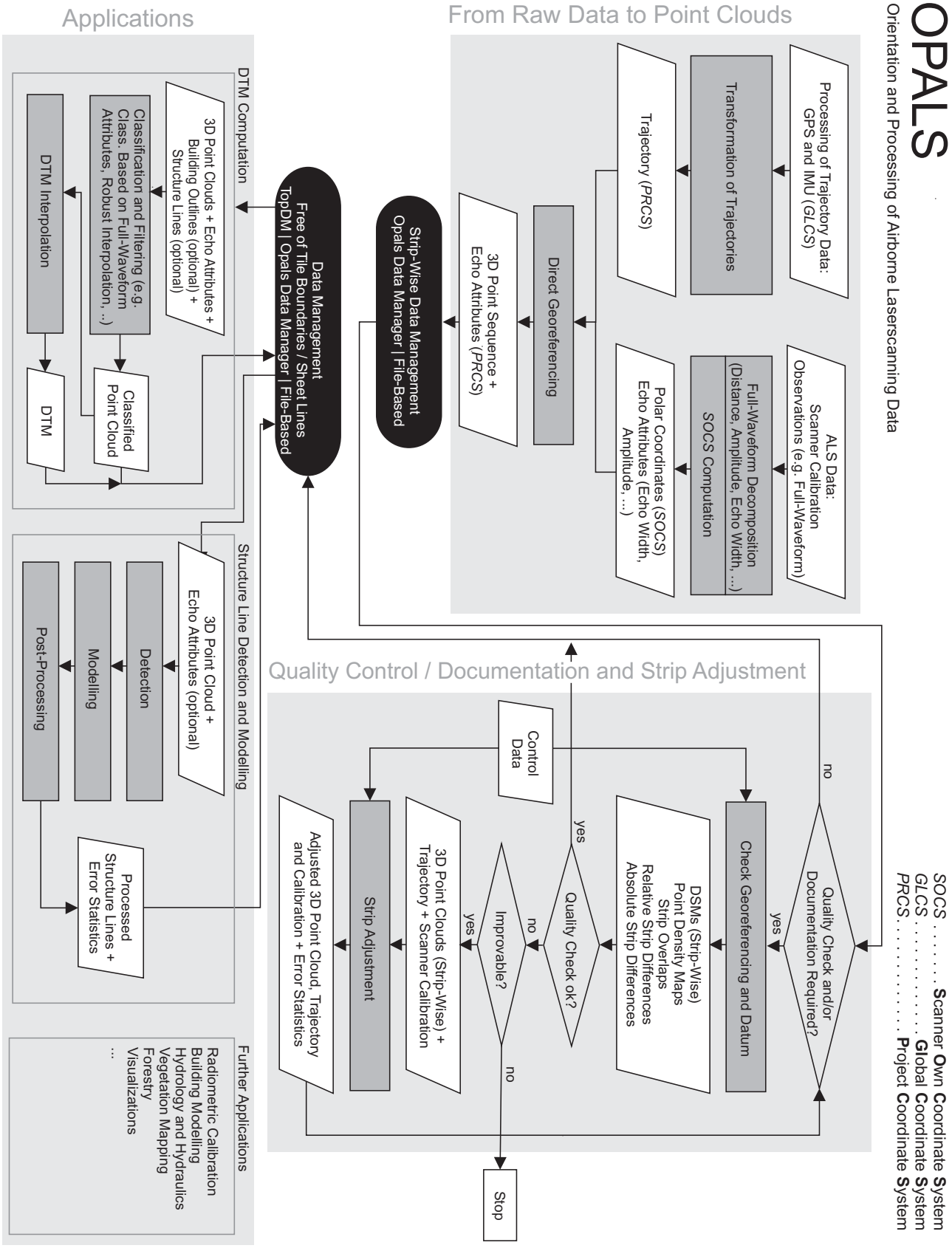

Figure 1: Proposed ALS flowchart representing the basis for the OPALS software

GNSS and IMU, and on the other hand, analysis and radiometric calibration of the echo waveform is performed, resulting in additional echo attributes like echo width, amplitude, and backscatter cross-section. For each strip, a 3D point cloud is derived by means of direct georeferencing. At this stage of the workflow, high-level data adminstration becomes relevant the first time, as subsequent quality control steps require efficient point data access mainly on a per-strip basis. Quality checks include the verification of full data coverage and the compliance with minimum point densities based on density maps as well as the examination of the strip registration precision using e.g. colour coded strip difference maps. For the latter, digital surface models (DSM) of the ALS strips and the list of all overlapping strips are required, again advocating for a strip-based data management as mentioned above. For checking the absolute planar and vertical accuracy, external reference data is necessary. As a result of quality control, calibration of the measurement system and/or strip adjustment may become inevitable, resulting in new 3D point clouds. Based on them, the quality control cycle is repeated until the desired quality criterions are met.

Once the final 3D point cloud is processed, a tile based or seamless data management supersedes the strip-wise administration, combining the point data (coordinates and echo attributes) of multiple ALS strips. The next major steps in the processing chain are the classification of each echo of the point cloud into either terrain or (different classes of) off-terrain points - often referred to as filtering - and the delineation of natural and artificial lineaments (breaklines and other structure lines). These processes mainly rely on geometric criterions (point height distribution of neighbouring points, angles between planar patches, etc.) but, additionally, the echo attributes derived from the full waveform backscatter signal help improving the precision and reliability of the results. Thus, the data administration must provide both efficient spatial access (nearest neighbour and range queries) as well as flexible administration of arbitrary attributes. Finally, the DSM and the DTM - both among the most important products of ALS - are interpolated either as regular grids, hybrid grids considering breaklines, or triangular irregular networks (TIN), serving as basis for many subsequent fields of application like building modelling, vegetation mapping and other forestry applications, hydraulic modelling for simulation of flood boundaries, and many more. A second quality control cycle (not shown in Fig. 1) for model compliance marks the last step of an ALS project.

Modern ALS sensors provide high point densities in the range of dozens of points per m^2 and, hence, the ALS processing software has to deal with billions of points even in a single ALS project. An efficient data management is, therefore, a precondition for successful project handling. File based data management, even using standard binary file formats like LAS (ASPRS, 2009) or shapefile (ESRI, 1998), only partially allows administration of additional attributes and doesn't provide geometry indices essential for fast spatial data access. On the other hand, geo-relational database systems like PostgreSQL/PostGIS (Refraction Research, 2009) or Oracle Spatial (Oracle, 2009) offer full administration capabilities concerning both geometry and attributes, but the strength of these systems is the long-term archiving and persistence aspect rather than near-realtime data access as necessary for ALS projects.

3 SOFTWARE CONCEPT

The main objectives of the OPALS software can be formulated as follows:

- complete processing chain from raw data to various products, e.g. DTM
- automatic work flow for huge data volumes
- rapid availability of recent research outcomes as software modules
- platform for sustainable scientific development beyond the duration of a PhD

To achieve these objectives, the basic concepts for the OPALS software are:

- modular design based on small components
- accessibility of modules as command line executables, Python modules, and via C++ API
- individual process control via scripts
- data administration based on the OPALS data manager
- interfaces for efficient data exchange with DTM, GIS and visualisation software
- abdication of interactivity as far as possible

The OPALS program system is mainly designed for automatic processing. Thus, a sophisticated graphical user interface and interactive editing steps are omitted deliberately. This may seem disadvantageous at first glance. However, OPALS is split into small, well-defined modules that may be combined freely, resulting in flexible, custom processing chains. For instance, the derivation of a hypsometric map of a single ALS flight strip is achieved using three different modules: import of strip point data, DTM grid interpolation and derivation of the colour coded rastermap, facilitating the re-usability of the respective components in different application environments. To further ease this combination, OPALS modules can be accessed in three different ways: (i) from command prompts / Unix or Linux shells as executables, (ii) from Python shells using platform-independent Python code, and (iii) from custom C++ programs by dynamic linkage (DLL). The latter allows experienced users direct embedding of OPALS components in their own C++ programming environment. By contrast, the former two options (stand-alone executable and Python API) allow combining OPALS modules in either Unix/Batch or Python scripts. Scripting is a powerful instrument, as it enables the construction of complex, custom processing chains by freely combining OPALS modules.

As pointed out before, efficient data management is regarded to be of crucial importance. Thus, the OPALS Data Manager (ODM) was developed, featuring high-performance spatial queries of point and line data even for large project areas. The ODM acts as a spatial cache combining the simplicity and efficiency of file based processing and the flexibility and expandability of database systems. An independent import module is provided to read data in arbitrary data formats or even to extract data from spatial databases, and to build up the ODM data structure. Subsequent application modules take an ODM file as input and have access to the coordinate and echo attribute information via an internal ODM library. Analogously, an export module is available for converting data stored in the ODM back to a series of supported file formats and databases. Due to the omission of interactivity, high-performance interfaces to external program systems like DTM, GIS, editing or visualisation programs are provided. In this regard, OPALS makes intensive use of open source solutions like the geodata abstraction library GDAL (GDAL, 2009) for accessing grid data and the OGR simple feature library (subset of

GDAL) for interchange of point and line related data. This provides conformance of OPALS to the specifications of the Open Geospatial Consortium (OGC, 2009) on the one hand and facilitates data exchange with software systems like GRASS (GRASS GIS, 2009) on the other hand which use the same technology.

Apart form a complete processing chain and the ability to handle huge data volumes, another main goal of OPALS is to shorten the time span for transforming research results into software modules. This is mainly achieved by using a light-weight framework allowing non expert programmers to concentrate on the actual research problem whereas the framework deals with general programming issues like validation of user inputs, error handling, logging, etc.

In the following subsections the OPALS framework and the basic concepts of the OPALS data manager are explained in more detail. Alternative software concepts for processing ALS data are e.g. described in (David et al., 2008).

3.1 OPALS framework

Every OPALS module is thought to take some input data, apply a set of algorithms considering certain parameters, and finally produce some output. This involves numerous recurring, common tasks like interface definition, user input validation, error handling, logging, progress control, licensing, and the like. Using a machine-oriented programming language like C++, these parts become voluminous, while the actual algorithm to be implemented may represent only a small fraction of the entire code. To disburden module programmers from all these matters, the light-weight OPALS software framework was set up.

The implementation of research outcomes by the different researchers themselves facilitates rapid public applicability on the one hand. On the other hand, these compact implementations are prone to reflect peculiar programming styles concerning naming conventions, log file layout, etc. in their interface. However, a uniform behaviour and look-and-feel of the individual modules is essential in order to ensure module interoperability and short training periods on behalf of users, which is therefore encouraged by the framework.

As mentioned above, the definition of interfaces and the validation of user inputs are core features of the OPALS framework. For each option, module programmers only need to specify an option descriptor, the respective value type, the optionality, and a help text. Options belong to one of four classes of optionality: (i) mandatory, (ii) estimable, (iii) ignorable, and (iv) aborting (e.g. 'help'). An option for a general input file may serve as a representative example:

```
( inputfile, std::string,   0, "input file name" )
```

This option is specified by an intuitive name (*inputfile*), it accepts string-type values (C++ standard template library string *std::string*), it is mandatory, as indicated by the optionality value *0*, and its meaning is further explained by a help text. Based on this generic option description, the framework performs a series of uniform tasks. First, separate get-, set-, and isSet- functions are created for each option, considering its name and data type. This is achieved using preprocessor macros, which result in the following, automatically generated code:

```
std::string get_inputfile() const;
void set_inputfile ( const std::string &infile );
bool isSet_inputfile() const;
```

Furthermore, the implementations as executable, Python module and shared library with C++ API are defined by the framework, again based on macros. Finally, the framework provides uniform functions for writing log files in a clear XML structure and an error handling system based on exceptions. An example module is described in more detail in section 4.

3.2 OPALS data manager

Compared to the flexibility and the generality of spatial databases, applications covered by OPALS require only a limited set of spatial operations onto the original data. For example, window and nearest neighbour queries are key operations for DTM interpolation, normal estimation, segmentation and similar tasks. Considering the properties of the source data and queries which have to be supported, the OPALS data manager (ODM) was designed and implemented to achieve maximum performance. The ODM operates as a spatial cache on top of either a low-level file based data administration or a spatial geodatabase running in the background.

The ODM stores point data in a K-d tree, a generalisation of a binary search tree (Bentley, 1975), and more complex geometries in an R*-tree (Beckmann et al., 1990). The K-d tree is an extremely fast spatial indexing method. This static indexing structure only supports point data, but its speed is outstanding which is why the disadvantage of two separated spatial indices and the limited support for insertion/ deletion was accepted. Both indices are wrapped in the ODM, such as if all geometry data were managed in a single spatial index structure. Both indexing methods have to be thread safe as multiple processing threads may access and modify the data manager simultaneously. Huge ALS projects can easily exceed the memory of todays computers. Hence, it was necessary to develop an extended K-d tree which swaps unneeded data to disk in an efficient manner. Therefore, the overall data area is spilt into tiles. The point data of each tile are then indexed by one K-d tree. An intelligent stacking system guarantees a low degree of data swapping which is crucial for the overall system performance. More details about point data administration using multiple K-d trees can be found in (Otepka et al., 2006).

Apart from fast spatial queries, the ODM also provides an administration scheme for storing attributes of arbitrary number and data type on a per-point basis. The additional point attributes may either stem from the initial analysis of the full waveform signal (e.g. echo width, amplitude, etc.) but may also be calculated by one of the OPALS modules. The three components (nx, ny, nz) of the surface normal vector, for instance, may be calculated for each ALS point in a separate module, stored as additional information to be used by a different module dealing with segmentation of surface elements. Thus, the additional information system is highly dynamic and can, therefore, be used to communicate information between different modules without the need for external storage of attributes.

4 MODULE EXAMPLE

In this section, the concrete module opalsGrid is explained in more detail. The scope of opalsGrid is to derive a regular grid in GDAL-supported format, based on an ODM file. Simple interpolation techniques like moving-planes are applied on the basis of *n* nearest neighbours. For the sake of clarity, only the five most important options are shown in this example, whereas the real opalsGrid module features some more.

As pointed out in section 3.1, the module programmer basically has to provide the generic option description and the implementation of the runModule() function. The following option list is used:

```
(( infile,        Path,      0, "input ODM file name"     )) \
(( outfile,vector<Path>,     1, "output gridfile name"    )) \
(( gridsize,      float,     2, "model grid width"        )) \
(( interpolation, IplMethod, 2, "interpolation method"    )) \
(( neighbours,    int,       2, "nr of nearest neighbours")) \
```

The OPALS framework automatically creates the C++ code for the command line executable as well as the C++/Python API for module opalsGrid. The following usage screen appears when entering `opalsGrid` within the command prompt without any further options:

```
Usage opalsGrid:
  --infile arg              input ODM file name
  --outfile arg (=estim)    output gridfile name
  --gridsize arg (=1)       model grid width
  --interpolation arg (=p) interpolation method
  --neighbours arg (=8)     nr of nearest neighbours
```

Below the header, one line is printed for each option. Each line starts with the option descriptor, followed by the keyword 'arg', if a value may be assigned. Estimable option values are indicated by a following '(=estim)', while constant default values are shown in round brackets. Mandatory options lack these indications. At the end of each option-line, an explanatory text is output. OPALS makes extensive use of the open source boost C++ libraries (Boost, 2009). For the executables, boost::program_options is applied to parse and store the command line options. In the following code snippet, showing the C++ class declaration of class ModuleGrid, boost::filesystem is deployed providing the data type for input and output file paths (already used in the generic options description above).

```
class ModuleGrid : virtual public ModuleBase
{
  typedef boost::filesystem:path Path;
  typedef opals::GridInterpolationMethod IplMethod;
public:
  // Constructors and Destructor
  ModuleGrid();
  ModuleGrid(const ModuleGrid &ref);
  virtual ~ModuleGrid() {};
  // set parameters
  void set_infile ( const Path &infile );
  void set_outfile ( const std::vector<Path> &outfile );
  void set_gridsize ( const float &gridsize );
  void set_interpolation ( const IplMethod &interpol );
  void set_neighbours ( const int &neighbours );
  // query if parameters are set
  bool isSet_infile() const;
  bool isSet_outfile() const;
  ...
  // get parameter value functions
  Path get_infile() const;
  ...
  int get_neighbours() const;
protected:
  virtual void checkModuleParameters();
  virtual void runModule();
  void estimate_outfile();
};
```

This class declaration is also the basis for the C++ API DLL and Python interfaces, where the latter is exported using boost::python. Please note, that the declaration and the definition (not shown here) of all access and query functions (set_option, get_option) were created automatically by the OPALS framework. The module programmer is only responsible for implementing the checkModuleParameters() function to verify the integrity of the parameter settings (cross dependencies of options) and the actual runModule() function. An estimate_outfile() function also appears in the declaration due to the optionality value *1* of option *outfile*, which indicates an option whose value may be estimated (in this case based on the input file name). For all remaining options with optionality value *2*, an appropriate, constant default value exists (c.f. usage screen). The following snippets demonstrate how OPALS modules can be applied in scripts, beginning with a simple Batch-file:

```
@echo off
rem +++ Simple opalsGrid Batch Example
echo Running opalsGrid...
call opalsGrid -inf=strip1.odm -out=strip1-dtm.tif -grid=0.5
echo Done!
```

In this example, a regular 0.5m grid is derived, and stored in GeoTiff file format (due to the output file extension .tif). The example shows that command line options can be abbreviated (e.g. grid=0.5), and options may be omitted if appropriate default values are available as for options *interpolation* (default: moving planes) and *neighbours* (default: 8). The final code snippet shows the same example embedded in a Python script:

```
#Simple opalsGrid Python Example
#++++++++++++++++++++++++++++++
from opalsGrid import *
mygrid = Grid()
print "Running opalsGrid..."
mygrid.set_infile('strip1.odm')
mygrid.set_outfile('strip1-dtm.tif')
mygrid.set_gridsize(5.0)
mygrid.run()
print "Done!"
#++++++++++++++++++++++++++++++
```

5 FIRST RESULTS

As a fist package, oaplsQC (quality control) is currently work in progress. opalsQC is a collection of modules for checking the quality of ALS point clouds. The aim of this package is to get a simple and fast overview about the captured terrain data and their quality. Among the features of interest are: DSMs, point density maps, point distance maps, 3D-strip difference measures, and the like. The final goal is to provide a professional basis of decision-making whether or not further pre-processing steps like strip adjustment are required. Table 1 shows a list of all modules designed for opalsQC along with a short description for each module.

module	description
opalsImport	importing point cloud data into ODM
opalsExport	exporting point could data from ODM
opalsGrid	simple and fast grid interpolation based on point cloud data
opalsCell	derivation of raster models via aggregation of point data information
opalsBounds	derivation of ODM outlines
opalsOverlap	derivation of a list of overlapping strip pairs
opalsMask	computation of grid masks
opalsDiff	calculation of difference grid models
opalsShade	derivation of shaded relief raster maps
opalsZColour	derivation of colour coded raster maps
opalsStitch	mosaicing of multiple grids/raster images
opalsLSM	derivation of 3D-shifts of overlapping strips via least squares matching (LSM)

Table 1: List of modules designed for opalsQC

Apart from the listed modules, opalsQC will also contain a set of standard scripts e.g. for automatically deriving colour coded strip difference maps or point density maps for a set of ALS strips. Fig. 2 shows an example of an image mosaic containing colour coded relative height differences of nine overlapping ALS strips. For the derivation of the illustrated map, the following OPALS modules were involved: opalsImport to import the point cloud data into separate ODM files, opalsGrid to calculate last-echo DSMs, opalsBounds to derive the strip outlines, opalsOverlap to determine overlapping ALS strips, opalsDiff to calculate the difference models, opalsZColour to derive the distinct colour coded raster maps and, finally, opalsStich to create the image mosaic.

6 CONCLUSIONS AND FUTURE WORK

In this article we presented the concept and first results of the new comprehensive ALS software OPALS. OPALS mainly fo-

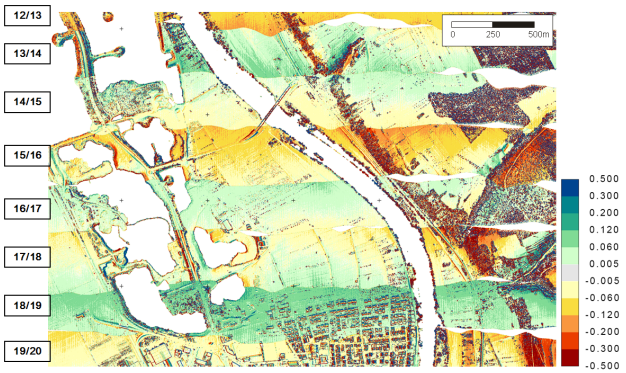

Figure 2: Raster image mosaic containing colour coded relative height differences of nine ALS flight strips (12...20) constituting 8 overlapping strip pairs (12/13...19/20), strip overlap: approx. 50%, flight direction: east-west, white areas: data voids due to water bodies or single strip coverage

cusses on automatic project workflows and processing of huge ALS projects, for which an efficient data administration is of crucial importance. The OPALS data manager was introduced, allowing fast spatial data access even for billions of points, and furthermore featuring a flexible administration of additional point attributes. In addition, OPALS provides a light-weight software framework covering general software development issues and, thus, supporting programmers to concentrate on their actual research problems. The entire scientific community benefits, since researchers can adopt new algorithms in their own research work more rapidly. Licensing issues are not covered in this paper, but the aim is to apply a moderate pricing policy. In this sense, OPALS is an initiative to advance the scientific progress in the field of laser scanning.

First experiences concerning a package dealing with quality control of ALS point data were gathered, and first results were presented. To achieve a complete processing chain from the raw ALS point cloud to a precise and reliable DTM, many steps are ahead, which we are willing to take since the first experiences with the OPALS system give cause for optimism.

REFERENCES

ASPRS, 2009. www.lasformat.org. Homepage of ASPRS LAS file format.

Beckmann, N., Kriegel, H.-P., Schneider, R. and Seeger, B., 1990. The r*-tree: An efficient and robust access method for points and rectangles. In: H. Garcia-Molina and H. V. Jagadish (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, 2009. www.boost.org. Homepage of boost C++ libraries.

Briese, C., 2004. Three-dimensional modelling of breaklines from airborne laser scanner data. In: IAPRS, Vol.XXXV, B3, Istanbul, Turky.

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. and 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.

Dorninger, P. and Pfeifer, N., 2008. A comprehensive automated 3d approach for building extraction, reconstruction, and regularization from airborne laser scanning point clouds. Sensors 8(11), pp. 7323–7343.

ESRI, 1998. www.esri.com/library/whitepapers/pdfs/shapefile.pdf. ESRI Shapefile Technical Description.

GDAL, 2009. www.gdal.org. Homepage of Geospatial Data Abstraction Library.

GRASS GIS, 2009. grass.osgeo.org. Homepage of Grass GIS.

Hollaus, M., Wagner, W., Maier, B. and Schadauer, K., 2007. Airborne laser scanning of forest stem volume in a mountainous environment. Sensors 7(8), pp. 1559–1577.

Kager, H., 2004. Discrepancies between overlapping laser scanning strips - simultaneous fitting of aerial laser scanner strips. In: IAPRS, XXXV, B/1, Istanbul, Turkey, pp. 555–560.

Kraus, K. and Pfeifer, N., 1998. Determination of terrain models in wooded areas with airborne laser scanner data. ISPRS Journal of Photogrammetry and Remote Sensing 53, pp. 193–203.

Kraus, K. and Pfeifer, N., 2001. Advanced DTM generation from LIDAR data. In: IAPRS, XXXIV, 3/W4, Annapolis, MD, USA, pp. 23–30.

Mandlburger, G., Briese, C. and Pfeifer, N., 2007. Progress in LiDAR sensor technology - chance and challenge for DTM generation and data administration. In: Proceedings of the 51th Photogrammetric Week, D. Fritsch (ed.), Herbert Wichmann Verlag, Heidelberg, Germany, pp. 159–169.

Mandlburger, G., Hauer, C., Höfle, B., Habersack, H. and Pfeifer, N., 2008. Optimisation of lidar derived terrain models for river flow modelling. Hydrology and Earth System Sciences Discussions 5(6), pp. 3605–3638.

OGC, 2009. www.opengeospatial.org. Homepage of Open Geospatial Consortium.

Oracle, 2009. www.oracle.com/technology/products/spatial. Oracle Spatial Homepage.

Otepka, J., Briese, C. and Nothegger, C., 2006. First steps to a topographic information system of the next generation. In: Symposium of ISPRS Commission IV - Geo Spatial Databases for Sustainable Development, Goa, India.

Pfeifer, N., Stadler, P. and Briese, C., 2001. Derivation of digital terrain models in the SCOP++ environment. In: Proceedings of OEEPE Workshop on Airborne Laserscanning and Interferometric SAR for Detailed Digital Terrain Models, Stockholm, Sweden.

Refraction Research, 2009. postgis.refractions.net. PostGIS Homepage.

Ressl, C., Kager, H. and Mandlburger, G., 2008. Quality checking of als projects using statistics of strip differences. In: IAPRS, XXXVII, pp. 253 – 260.

Rottensteiner, F. and Briese, C., 2002. A new method for building extraction in urban areas from high-resolution LIDAR data. In: IAPRS, XXXIV, 3A, Graz, Austria, pp. 295 – 301.

Székely, B., Hollaus, M., Zámolyi, A., Draganits, E., Roncat, A. and Pfeifer, N., 2008. Some geoscientific applications of airborne laser scanning dtms in austria. Journal of Alpine Geology 49, pp. 109–110.

Wagner, W., Ullrich, A., Ducic, V., Melzer, T. and 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.

Wagner, W., Ullrich, A., Melzer, T., Briese, C. and Kraus, K., 2004. From single-pulse to full-waveform airborne laser scanners: potential and practical challenges. In: IAPRS, XXXV, Istanbul, Turkey.