Sorted pulse data (SPD) library—Part II: A processing framework for LiDAR data from pulsed laser systems in terrestrial environments

Peter Bunting a,*, John Armstrong b,c, Daniel Clewley a, Richard M. Lucas a

a Institute of Geography and Earth Sciences, Aberystwyth University, United Kingdom
b Joint Remote Sensing Research Program, School of Geography, Planning and Environmental Management, University of Queensland, St Lucia, QLD 4072, Australia
c Remote Sensing Centre, Science Delivery, Department of Science, Innovation Technology, Innovation and the Arts, 41 Boggo Road, QLD 4102, Australia

ARTICLE INFO

Article history:
Received 24 March 2012
Received in revised form 12 November 2012
Accepted 10 January 2013
Available online 8 February 2013

Keywords:
LiDAR
Software
Pulse
Waveform
Sorted
SPD

ABSTRACT

The management and spatial-temporal integration of LiDAR data from different sensors and platforms has been impeded by lack of generic open source tools and standards. This paper presents a new open source software system, the sorted pulse data software library (SPDLib), that provides a processing framework based on an implementation of a new file format for the storage of discrete-return and waveform LiDAR data from terrestrial, airborne and space borne platforms. A python binding and a visualisation tool (SPD Points Viewer), which build on top of the SPDLib and SPD file format have also been provided. The software and source code have recently been made freely available and can be accessed online through an open source code repository. Future developments will focus on the development of advanced waveform processing functionality and optimising IO performance. The software and documentation can be obtained from http://www.spdlib.org.

Crown Copyright © 2013 Published by Elsevier Ltd. All rights reserved.

1. Introduction

For many environmental management organisations the need for rapid and cost effective LiDAR data processing capacity is growing as acquisition costs decrease and availability increases. Airborne laser scanners (ALS) are the most commonly available LiDAR data source and provide coverage often exceeding hundreds or even thousands of square kilometres. Terrestrial laser scanners (TLS) are also becoming more widely used for environmental remote sensing (e.g., Maas et al., 2008; Brodou and Lague, 2012). One of the keys to the successful use of these data and technologies are the availability of suitable software tools for storing and analysing the data to gain the required results. In a collaborative research environment, such software should ideally be free (i.e., available without financial outlay) and open (i.e., source code available for modification). However, it also needs to provide support and tools for the latest technologies (e.g., waveform data) and common tasks (e.g., ground return classification) while providing a platform on which new techniques and ideas can be built.

The development of software for LiDAR processing is not trivial, particularly when dealing with large datasets that cannot fit into the random access memory (RAM) of many computers used for processing. Whilst a number of commercial software products are available, such as Terrasolid's Terra product line (Terrasolid, 2012), Pintools (Bentley Systems, Inc, 2012) and ESRI’s ArcGIS (ESRI, 2012), these are limited to processing discrete return data and generally do not support the ingestion or analysis of waveform data. Sensor manufacturers do provide tools for processing the data from their systems, such as Riegl's RiAnalyze, RiProcess and RiWorld (Reigl, 2012) or Leica's Cyclone (Lecia, 2012) software. However, these are proprietary and are mainly concerned with the pre-processing steps directly following acquisition, such as registration to world coordinates or translation of waveform data into discrete return data. A large body of software has also been generated through academic research, some of which has been released into the public domain under a range of licensing terms. Such software includes, OPALS (from the Vienna University of Technology; Mandlburger et al., 2009; Otepka et al., 2012), which supports a wide range of data processing tasks including the processing of waveform data and includes options for extending the existing functionality but it is not free or open source. The OPALS data manager provides a highly organised spatial view of the data where data are broken down into tiles of 150 000–200 000 points and R-trees are constructed within each tile. R-trees allow very fast nearest neighbour (NN) queries to be performed and the tiles minimise the data loaded into RAM. BCAL LiDAR tools (Idaho State University, 2012) and RSC LAS Tools...
(University of Queensland; Armstrong, 2012) provide utilities that are built on top of IDL/ENVI and allow processing and visualisation of point cloud LiDAR data. LASTools (Isenburg, 2012a) has functionality for processing very large discrete return LiDAR datasets in the LAS file format, focusing particularly on filtering and producing elevation models. More specialised tools include MCC-LiDAR (US Forest Service, 2012), which only provides an implementation of the ground return classification algorithm of Evans and Hudak (2007), and libLAS which provides a general purpose software library for reading and writing the ASPRS LAS file format data. LibLAS also provides the lasblocks utility, which provides a method for segmenting the data into blocks with similar numbers of returns, the algorithm attempts to provide blocks that are as square as possible. Splitting the dataset into blocks in this way significantly improves the processing performance where no spatial index is available. More recently, the Point Data Abstraction Library (PDAL, 2012) has provided a streaming base architecture with the aim of providing an abstractions layer from the specific file format in the same way as the Geospatial Data Abstraction Library (GDAL, 2012) does for image processing. Efforts have also been put into making LiDAR processing more accessible to the community through the use of online services, such as the OpenTopography (Krishnan et al., 2011) service.

For waveform data, Chauve et al. (2009a) published the Full-Analysis software, which provides a mechanism for the storage, analysis and visualisation of waveform LiDAR, but unfortunately further development of this platform has not been forthcoming. Following the development of the SPD format (Bunting et al., this issue), the PulseWaves format (Isenburg, 2012b) has also been proposed and follows the ideas set out in the SPD format. Although the specification has yet to be agreed upon, PulseWaves may form the basis of the standard by which waveform data will be delivered by vendors.

Therefore, the aim of this research was to provide a set of tools which support the processing of ALS and TLS datasets for environmental remote sensing, particularly for vegetation metrics. Common tasks include derivation of Digital Terrain Models (DTM), Digital Surface Models (DSM), Canopy Height Models (CHM) and structural measures such as gap fraction, canopy openness, leaf area index, height percentiles, apparent foliage profiles, and stem/crown locations and counts (Coops et al., 2007; Lee and Lucas, 2007; Zhang, 2008; Suratno et al., 2009; Evans et al., 2009). The paper first provides a brief overview of the sorted pulse data (SPD) file format (Bunting et al., this issue) on which this software is built and the recommend workflow with the software, including ground return classification and the decomposition of waveform data. An overview is also provided of the C++ software API, python binding and visualisation tool (SPD Points Viewer).

2. The sorted pulse data software library

Continuing from Part I (Bunting et al., this issue), the SPD software library (SPDLib) has been implemented within C++ and provides support for a wide range of platforms, although has predominately been tested on UNIX and Linux systems. The software is written with an object-orientated modular design throughout, allowing for new functionality to be added using the C++ application programming interface (API) with relative ease. In addition, a more user-friendly Python binding interface to SPDLib has been provided, allowing easy access to the raw pulse data and where available, the spatial index. A reader for the ITTVIS Interactive Data Language (IDL) has also been provided. Finally, a wide range of tools have been developed for common tasks (e.g., ground return classification, data management and the calculation of metrics) associated with LiDAR data processing.

2.1. SPD file format

The software library and tools presented in this paper are built on top of the SPD file format (Bunting et al., this issue), which has been designed specifically for the storage of LiDAR waveform and discrete return data acquired by TLS, ALS and space borne systems, and includes support for multiple wavelengths within a single file. The format uses a pulse-based structure as opposed to a solely point-based structure, where pulses contain all the information associated with a transmitted pulse from the sensor. This includes transmitted and received waveforms and the discrete returns, determined by Gaussian decomposition of digitised waveforms or by the sensor hardware using proprietary methods. The SPD format also supports 2D spatial indexing of the pulses, where pulses can be referenced using cartesian, spherical or polar coordinate systems and projections. These indexes can be used to significantly speed up data processing whilst allowing the data to be appropriately projected. They are particularly useful when analysing and interpreting TLS data. The format is defined within a HDF5 file, which provides a number of benefits that include broad support across a wide range of platforms and architectures and support for file compression.

2.2. Prerequisite software libraries

The SPD software library has a number of software prerequisites, which are required to built the software. External software libraries are used to speed up the development process and to build on the expertise of others. For these libraries to be included within SPDLib the software needs to be freely and openly available to the community and with a compatibility software license. When including prerequisite libraries key considerations are that they have a compatible software license, be easy to build and install by end users, provide good functionality, performance and good developer documentation.

During the development process, to date, the following libraries have been included:

- Boost (http://www.boost.org),
- HDF5 (http://www.hdfgroup.org),
- GNU Scientific Library (GSL; http://www.gnu.org/software/gsl),
- Xerces-C (http://xerces.apache.org/xerces-c),
- GDAL/OGR (http://www.gdal.org),
- LibLAS (http://www.liblas.org),
- CGAL (http://www.cgal.org),

2.2.1. Boost

The Boost libraries provide a set of software utilities, which form extensions to the C++ standard library (STL). There are alternatives to the individual components of boost, but boost provides a single installation, simplifying the installation process for the user, and the project conducts regular code review ensuring quality. The software is therefore widely used and commonly installed on UNIX systems. Within SPDLib the integer numeric types definitions, python interface, type casting and text processing components are used.

2.2.2. HDF5

The HDF5 software library, provided by the HDF group, is used to read and write HDF5 files. Using the software library provided
by the HDF5 group ensures full compatibility with the HDF5 standard and optimal input/output (I/O) performance. Many systems used for scientific computing have the HDF5 libraries installed but when used with SPDLib the optional C++ libraries need to build and the zlib library is also required.

2.2.3. GSL

The GSL library is provided under the terms of the GNU GSLL3 license and provides numerous functions for mathematical operations as well as C struct representations for mathematical vectors and matrices. Where possible the mathematical operations required within SPDLib are provided by calling functions within GSL.

2.2.4. Xerces-C

The ability to parser XML is required for the metrics command and the Xerces-C parser was used due to its general availability, good documentation and familiarity of the authors with the library.

2.2.5. GDAL/OGR

The GDAL/OGR library provides a common interface to read and write image and vector files in the formats commonly used within the remote sensing community. The GDAL library is provided under the MIT-X license and as such has been included within numerous open source and commercial software (e.g., QGIS and ArcMap), it is the only open source software of its type and has a large support community.

2.2.6. LibLAS

When SPDLib was first developed the libLAS software was the only open source library available for reading and writing the LAS file format. More recently, Martin Isenburg's LasTools LAS reader and writer has been made available and future versions of SPDLib could use this library as support is provided for the more up to date LAS 1.3 and 1.4 specifications. It is recommended that LibLAS is built with laszip support to allow compressed LAS files to be read and written.

2.2.7. CGAL

The CGAL library provides support for numerous computation geometry functions and algorithms. Within SPDLib CGAL has been used to generate triangulations and natural neighbour interpolation, used as the default interpolation technique throughout the library. CGAL was used as the project provides excellent documentation (The CGAL Project, 2012) and very good computational performance, important for operations such as interpolation.

2.2.8. CMPFIT

The CMPFIT library provides an implementation of the Levenberg–Marquardt technique for solving the least squares problem. This is used for decomposing the LiDAR waveforms to individual point returns. The CMPFIT library does not provide generic compilation scripts so the source code has therefore been included within the SPDLib source tree and build scripts so the user is not independently required to install this library.

2.3. C++ software architecture

An independent software platform was developed for SPDLib, rather than developing on an existing platform such as GRASS GIS (GRASS GIS, 2012), to provide flexibility during development. Developing within a existing platform can have a number of advantages, such as end user familiarity with the platform and acceptance within the community, but it can also put limitations on the system due to the requirements of the general platform.

Throughout SPDLib, object-oriented frameworks have been used to facilitate new functionality and software maintenance. Specifically, an abstract factory pattern (Gamma et al., 1994) is used for the I/O functionality while the abstract super class pattern (Gamma et al., 1994) has been used for the data processing frameworks. Using these patterns provides flexibility and allows rapid development of features and algorithms while masking the developer from I/O and memory management issues.

The preference for abstraction through inheritance rather than templates is due to the authors preference and knowledge but template abstraction has some advantages, such as resolving for type at compile time rather than run time, although compiler messages when using templates are often difficult to interpret.

2.3.1. I/O framework

The abstract factory design pattern has been used (Fig. 1) to provide an effective way to support a range of importers and exporters within SPDLib. To adhere to the design pattern, importers and exporters are required to implement the appropriate abstract base class and register with the factory instance. The importer includes functions to read all the input data into a single list or vector data structure while a third function (readAndProcessAllData()) uses the abstract super class pattern to allow each read pulse to be sent for processing (e.g., to be tiled or waveforms decomposed) using a class adhering to the importer processor interface. Additionally, a function is required to read any header parameters from the input file. The exporter interface provides functions for opening and closing the output file, writing a column of data (equivalent to a bin with the SPD file index) and writing a matrix of columns. To request the appropriate importer or exporter from the factory a string specifying the file type needs to be provided (e.g., 'SPD', 'LAS', etc.).

2.3.2. Processing framework

SPDLib provides a framework (Fig. 2) for processing data that builds on the grid-based index of the SPD file and therefore follows typical methodologies used within image processing. The block processing framework undertakes the processing in blocks, where blocks have an optional overlap. The block processing interface requires an input SPD file and an optional GDAL raster layer. Outputs can be either a new SPD file, an output GDAL raster or no output file is provided. During processing the spatial index resolution can be resampled to multiples of the native SPD file resolution. If an input image is provided the processing resolution is set to the pixel resolution of the input image file, which therefore must be a multiple of the SPD index resolution.

Where algorithms require the pulses within non-square regions to be selected (i.e., within a polygon) then the bounding envelope is selected and the pulses within the region are then identified. Although, this is a processing overhead the comparative performance to systems where no index is provided is significant and the majority of processing tasks do not require this type of selection. For algorithms that build on nearest neighbour selections (e.g., KNN) an assumption is commonly made to the region within which those points will be found, such that an appropriate overlap between neighbouring processing blocks is provided. The assumption can significantly reduce processing time and fits the SPD data model. For airborne LiDAR the average nearest neighbour distance can easily be estimated from the point spacing but for terrestrial laser scanning data where point spacing is more irregular this parameter needs to be estimated using the outer areas of the scan (i.e., those areas of the
lowest point spacing). Indexing with the Z-axis is not provided within the SPD file as a pulse does not have a single value for Z as multiple returns can be associated with a pulse. It is expected that points are stored in the order of the returns (i.e., from the first to last return) but it is not validated. Therefore, for selections within the Z-axis all the pulses returns need to be checked as to whether they fit any required selection criteria. This can be seen as a limitation of the pulse data model but the model is required for the storage of the waveform data.

On top of the block processing framework is a column processing framework to simplify the processing where just individual columns is required or ‘windows’ of columns (e.g., $3 \times 3$). Being built on top of the block processing framework all the inputs and outputs and options, such as the resampling of the native bin resolution, are available to this framework.

2.4. Python binding

The SPDLib software also provides a Python binding to the C++ library (using Boost Python) to allow SPD (both indexed and non-indexed), including waveform data, files to be read and written directly from Python. This allows new and more advanced
features and functions to be implemented and gives access to functionality with Python and associated libraries (e.g., Matplotlib for plotting; Fig. 3). In addition, the Python binding can be used to convert currently unsupported external formats to SPD files more easily than through the use of C++, therefore allowing other SPDLib tools and the SPD Point Viewer to be used.

2.5. Visualisation

Visualisation of the LiDAR point cloud and associated data (e.g., aerial photography) is important to many users. For example, the results of an applied algorithm (e.g., a ground return classification) or a simple overview of the distribution of returns can provide new insights into the information content of the data. The SPD Points Viewer application has been created using C++, OpenGL and QT4 to provide a cross platform visualisation tool built on top of the SPD software library. Existing libraries, such as OpenSceneGraph (2012), were not used to keep the viewer ‘light weight’ and minimise the memory footprint. The viewer was originally intended as just a relatively simple utility so the advanced functionality of these other libraries was not required, the simple OpenGL point representation and functionality of the QT OpenGL widget were therefore sufficient.

The viewer first provides a window displaying the SPD files overview (‘quicklook’) image, which maps directly onto the SPD files spatial index and allows the user to select a region of interest. A second view displays the selected region as 3D points. The 3D points can be coloured by a number of variables, including return amplitude, width, classification, RGB values (e.g., from co-registered imagery), height and elevation, while the z component of the points can be either the topographic elevation or height above ground. The colouration of points by variables provides a high degree of flexibility in visualisation that allows data to be better understood and interpreted.

To visualise the waveform data, the individual bins of the received waveform are displayed as 3D points coloured by intensity value (Fig. 4(a)) where a noise threshold (as shown in Fig. 3) is used to remove waveform bins containing only noise. Taking advantage of the pulse-based nature of the SPD file format, the viewer also provides an option to display the vectors of the pulses (Fig. 4(b)), which gives insight into the direction of pulse transmission and interaction with 3D structures and the ground surface and assists interpretation.

3. Tools and workflow

An example workflow for airborne LiDAR data processing using SPDLib for vegetation applications is shown in Fig. 5. Where multiple input files for a study are available (i.e., those associated with a number of flight lines), the first step is to decide whether to merge these into a single data file or to process the files independently as tiles indexed on to the same grid. If waveform data are provided, Gaussian decomposition is available to generate discrete returns linked to the received waveform with the SPD file. SPDLib does not currently provide functions for georeferencing pulses but this could be added in the future, if and when required.

The file(s) should be indexed to an appropriate resolution for the processing being undertaken, although a variable bin resolution of output products can be achieved by rebinning data on the fly from the existing index. If tiling is used then the index origins need to be set appropriately on the same grid. Once the SPD file has been created, the first step is to classify the ground returns, from which the height (relative to the ground) fields (i.e., on pulses and points) can be populated. The SPD file therefore contains both elevations corresponding to a vertical datum and an above-ground height attribute for each pulse origin and discrete return. After defining the above-ground height field, a range of metrics commonly applied to LiDAR data and specifically to vegetation applications can be calculated. However, rather than implementing each metric individually, a set of metric primitives and mathematical operations between metrics have been defined.
Therefore, metrics can be calculated through any combination of these primitives and operations, with these defined using an XML file. The XML interface allows the above-ground height, sensor-target range, Gaussian amplitude (or intensity) and width fields to be used and combined, facilitating calculation of a large number of existing canopy, terrain and statistical metrics and the creation of new metrics as required. Further metric primitives are also being added, primarily for processing waveform data.
3.1. File format conversion

A number of common LiDAR formats are supported for conversion to the SPD format and new formats are being progressively integrated. These include a range of ASCII formats, which differ between data providers, and the LAS binary format (through libLAS, which supports LAS version up to 1.2), which provides a standard interpretable format for discrete return data. It should be noted that some data providers and software packages do not always populate all the available data and header fields within the LAS file and this can limit the usefulness of some datasets. For example, if the points are not associated with the correct return numbers, the pulses cannot be reconstructed when the SPD file is built. Users are therefore recommended to consider the fields they require, ideally prior to acquisition of the data, to ensure they request all the data they might need. When processing large, regional, datasets consisting of a number of tiles, grid origins need to be specified to allow the output products to be integrated and mosaiced without interpolation.

3.2. Decomposition of waveform data

The SPDLib software supports Gaussian decomposition (Wagner et al., 2006) of received waveforms ($P_i$) to retrieve discrete returns. These are linked to the waveforms within the SPD file for each pulse by the time ($t_i$). The cross-overs of the waveform first derivative above a nominal noise threshold are identified and used as the starting values for the $N$ Gaussian amplitude ($P_i$) and time ($t_i$) parameters (Eq. (1)). Bounds are also placed on the pulse width parameter ($s_i$). The noise level ($\epsilon$) can be set as a constant for a particular sensor, or combination of sensor and survey properties, or solved for using the Levenberg–Marquardt method for non-linear least-squares such that

$$P_i(t) = \epsilon + \sum_{i=1}^{N} P_i e^{(t-t_i)/s_i^2}$$

(1)

Decomposing the waveform allows existing algorithms for analysing discrete return data to be used (e.g., for ground return classification) while retaining the waveform for later processing steps. This is achieved as the discrete returns and waveforms are linked through the pulse-based architecture. The assumptions of Gaussian decomposition are rarely satisfied for volumes such as canopy foliage. Therefore we do not advocate Gaussian decomposition as the processing technique that should always be applied, but it is one very commonly employed in research and commercial LiDAR methods that we have encountered (Chauve et al., 2009b; Ullrich and Pfennigbauer, 2011). Other echo detection algorithms that mimic the discrete return sensor hardware (Wagner et al., 2004) and waveform processing methods such as different deconvolution techniques (e.g., Neuenschwander, 2008) can be added in future with relative ease due to the modular framework of the software.

3.3. Ground return classification and interpolation

Two ground return classifiers are available within SPDLib. The first algorithm is an implementation of the progressive morphological filtering approach by Zhang et al. (2003), which can be applied to landscapes with variable terrain and supports the removal of a wide variety of features (e.g., vegetation and buildings). The second algorithm is the multi-scale curvature classification algorithm of Evans and Hudak (2007), which is optimised for vegetated environments. Further algorithms will be progressively added as technology, particularly for waveform LiDAR, progresses.

The above-ground height field for each return and pulse can be defined using one of two methods. The first uses a raster DTM to define the ground elevation from which the elevation difference to return is calculated. The second uses the classified ground returns to interpolate ground elevation values for each return, from which the height difference is calculated. Several interpolation algorithms are available for calculating terrain height values from the classified ground returns but the recommended (Bater and Coops, 2009) and default approach is the natural neighbour algorithm (Sibson, 1981) due its simplicity and surface quality. The same approach is used and recommended for generating DTM, DSM and CHM raster products.

3.4. Calculation of LiDAR metrics

A large body of literature (e.g., Coops et al., 2007; Hall et al., 2005; Armston et al., 2009; Evans et al., 2009) deals with the definition and formulation of metrics that can be calculated for vegetated environments from LiDAR data. These can be simple statistical moments, percentiles of the canopy height or return amplitude or count ratios (often linked to canopy cover). More advanced metrics include the parameters of probability density functions (e.g., Weibull; Coops et al., 2007) or biophysical metrics such as Foliage Projective Cover (FPC; Armston et al., 2009).

An interface allowing metrics to be easily calculated within SPDLib has been provided which includes a set of metric primitives (Table 1) defined for above-ground height, topographic elevation, return amplitude, return width and range. The returns classification (e.g., ground, vegetation, etc.), minimum and maximum values for the attribute being calculated (e.g., above-ground height) and return number (e.g., first, last, etc.) can be used to define the points and pulses used for the calculations.

Mathematical operators (Table 2) can be applied to either another mathematical operator or metric primitive to allow a range of LiDAR metrics to be derived.

To interface with the metrics command, an XML file is defined with a hierarchical list of metrics and operators. These can either be applied to the LiDAR data to produce a raster output or to a polygon shapefile where the calculated values for each polygon are added to the shapefiles attribute table.

4. Discussion

4.1. Licensing

The software is split into two parts for licensing. The main library is released under a General Public License (GPL) version 3 license (GNU, 2007) and the software to read and write SPD files is released under the MIT license (Massachusetts Institute of Technology, 1988). Whilst the software is provided completely
free of charge, it is without a warranty or promise of support. The GPL license is a so called viral license, meaning that any works derived from this software also have to be covered by the GPL license and then distributed openly. Therefore, any community improvements or changes to the algorithms have to be fed back allowing everyone to benefit from this work, as they benefitted from the original. The SPD file readers and writers are released under the MIT license to allow support for this format to be included within other software packages (e.g., commercial) without the restrictions of the viral license.

4.2. Supported systems

Currently the software is developed, built and tested on UNIX and Linux platforms; however, all the libraries and methods used support the Windows platform. Therefore, the library, tools and viewer are considered fully cross platform.

4.3. Future developments

One of the most significant future developments will be an official support for Windows alongside UNIX and Linux platforms. In terms of data processing, future developments are expected to concentrate on further algorithms for processing waveform data and specifically those that support the extraction of biophysical vegetation metrics such as Leaf Area Index (LAI) and above-ground biomass over large areas. Finally, a significant development will be support for virtual mosaics where a number of independent SPD files can referenced by a header file and then used as a single ‘super’ file for processing. This functionality is particularly important for allowing seamless products (e.g., DTM and DSM) to be created across very large areas without explicitly merging the data into a single SPD file.

5. Summary and conclusions

Versions of the SPDLib software have been in active use for over 3 years but the current form of the software has been available for about 18 months. To date a number of projects and organisations have made use of the software which is continuing to evolve and improve to support the latest technologies, such as multi-wavelength datasets. It is envisioned that the use of the software will expand as documentation improves and is made more available. The latest version of the software and documentation are available online from http://www.spdlib.org.

Acknowledgements

The authors have very much appreciated the comments and advice given by the three anonymous reviewers. Mattias Nyström is also thanked for his input and development of importers using the python bindings.

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.cageo.2013.01.010.

References

Idaho State University, 2012. BCAL LiDAR Tools for Processing and Visualizing LiDAR data (http://code.google.com/p/bcal-lidar-tools/).

Table 2

<table>
<thead>
<tr>
<th>Operator</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Two metrics</td>
</tr>
<tr>
<td>Minus</td>
<td>Two metrics</td>
</tr>
<tr>
<td>Multiply</td>
<td>Two metrics</td>
</tr>
<tr>
<td>Divide</td>
<td>Two metrics</td>
</tr>
<tr>
<td>Power</td>
<td>One metric</td>
</tr>
<tr>
<td>Absolute</td>
<td>One metric</td>
</tr>
<tr>
<td>Square root</td>
<td>One metric</td>
</tr>
<tr>
<td>Sine</td>
<td>One metric</td>
</tr>
<tr>
<td>Cosine</td>
<td>One metric</td>
</tr>
<tr>
<td>Tangent</td>
<td>One metric</td>
</tr>
<tr>
<td>Inverse sine</td>
<td>One metric</td>
</tr>
<tr>
<td>Inverse cosine</td>
<td>One metric</td>
</tr>
<tr>
<td>Inverse tangent</td>
<td>One metric</td>
</tr>
<tr>
<td>Log (base 10)</td>
<td>One metric</td>
</tr>
<tr>
<td>Natural log</td>
<td>One metric</td>
</tr>
<tr>
<td>Exponential</td>
<td>One metric</td>
</tr>
<tr>
<td>Percentage</td>
<td>Two metrics</td>
</tr>
<tr>
<td>Add constant</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>Minus constant from metric</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>Minus constant from constant</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>Multiply by constant</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>Divide metric by a constant</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>Divide constant by metric</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>Divide constant by constant</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>Metric to the power of constant</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>Constant to the power of metric</td>
<td>One metric and one const. value</td>
</tr>
<tr>
<td>No of returns above metric value</td>
<td>One metric</td>
</tr>
<tr>
<td>No of returns below metric value</td>
<td>One metric</td>
</tr>
</tbody>
</table>
to LiDAR topography. In: Second International Conference and Exhibition on Computing for Geospatial Research and Application (COM.Geo).
OpenSceneGraph, 2012. The OpenSceneGraph Website (http://www.openscenegraph.org).