http://dx.doi.org/10.15576/GLL/2013.3.105

GLL

Geomatics, Landmanagement and Landscape No. 3 • 2013, 105–115

THE TESTING OF PCL: AN OPEN-SOURCE LIBRARY FOR POINT CLOUD PROCESSING

Maria Zygmunt

Summary

In the era of the rapid development of the terrestrial laser scanning technology (TLS), that facilitates the acquisition of large data volumes in the form of three-dimensional point clouds, a need arises to create modern solutions to enable effective and efficient data processing. This paper presents the PCL (Point Cloud Library). This is an Open Source project that contains many of the key algorithms for TLS data, designed for filtering, registration, estimation of function, surface reconstruction, modeling and segmentation, as well as structures for identifying and matching objects. The article discusses the structure of the project as well as its most useful modules from the point of view of the TLS data processing. It describes the format used by the PCL data storage. Also, a practical example is enclosed, as a basis for the discussion of consecutive steps of working with the library, from recording input data in the correct format to visualization of the effects of the used algorithm.

Keywords

terrestrial laser scanning • post-processing • Point Cloud Library • algorithm

1. Introduction

In the era of the rapid development of the terrestrial laser scanning technology (TLS), that facilitates the acquisition of large data volumes in the form of three-dimensional point clouds, the need arises to create modern solutions to enable their effective and efficient processing. Working with observations acquired in a continuous manner, requires applications adapted to their size and structure [Bohler et al. 2002]. The nature of TLS data and above all, their volume, virtually imposes the need to use procedures which minimize operator intervention. Therefore, a special role in the efficient processing of point clouds is played by specialized algorithms.

The issue of TLS data post processing has been providing an ample scope for research for years. Initially, algorithms were developed to help simplify and automate the preprocessing of point clouds. A function for automatic measurement noise and unnecessary data filtration would be created [Sithole and Vosselmann 2003], the registration of scans [Wendt 2004, Bae and Lichti 2004, Dold and Brenner 2006], or coloring of point clouds

[Kadobayashi et al. 2004]. Over time, there have been more and more sophisticated algorithms, i.e., segmentation procedures [Bauer et al. 2005, Belton and Lichti 2006.] and edge detection [Pu and Vosselman 2006, Weber et al. 2010a] which still constitute an enormous part of the ongoing research. Intensive work is also being done on automatic discovery and modeling of objects from point clouds [Vosselman 2009, Weber et al. 2010b]. Today, specialized algorithms dedicated to a specific purpose are created with a rising frequency. Features automatically extracting industrial installations [Rabanni 2006], elements of road infrastructure [Pu et al. 2011] and individual valuation characteristics of forest stands [Bienert et al. 2006, Wezyk et al. 2007] from point clouds have already been created.

The development of terrestrial laser scanning and broadening the possibilities of its application in many sectors of the economy has resulted, not only in the development of scientific research but also the rapid progress, in the field of software to process point clouds [Staiger 2003, Fröhlich and Mettenleiter 2004]. In recent years, many programs at different levels of sophistication have been available on the market. Some of them are simple visualization browsers; others are powerful applications enabling advanced post processing of TLS data. Unfortunately, most existing point cloud processing software does not allow for the modification of existing procedures, sometimes even concealing the latter. Laser input data are subject to unspecified operations, bringing results that are difficult to verify which considerably restricts the freedom of scientific research. This makes it difficult to work on new algorithms. No access to source code, makes it impossible to improve the efficiency of algorithms and adapt them to specific applications.

The appearance of the PCL programming library which is completely free and open for any modifications (Open Source) and at the same time containing a number of algorithms for processing and visualization of point clouds, raises high hopes. This paper demonstrates the potential of the PCL project, in the development of research on algorithms for post-processing terrestrial laser scanning data. The library has been presented from a theoretical point of view, as well as on a practical example of using the steps in working with PCL from the recording of input data, in the correct format to the visualization of the effects of the algorithm used.

2. What is PCL?

PCL or Point Cloud Library is a cross platform, large scale Open Source project dedicated to the processing of point clouds, operating under a Berkeley Software Distribution (BSD) license. It has been designed mainly for commercial and scientific research. It allows free and fully legal copying of the source code of the algorithms contained, in the library and any modification thereof.

The first algorithms, which are the foundation of the PCL, were developed as a part of the PhD work by Radu Bogdan Rusu at the Technische Universitaet Muenchen in Munich [www.willowgarage.com]. Further more, intensive work on the project was carried out in the Willow Garage research lab, involved in the development of hardware

and free software for robotics. An alpha version of the library was made available in 2010, as a part of the Robot Operating System (ROS), serving primarily to manipulate robots in complex 3D environments. In March 2011, PCL began to function as a separate project with its own domain [www.pointclouds.org]. Two months later, the first mature version 1.0 of the library was published. Currently, it is available in version 1.6 and is still being further developed.

Just two months after the publication of version 1.0 of the PCL, it was supported by more than 120 scientists from six continents and 30 different universities and research institutes, which contributed to the rapid development of the library. The project is being developed by leading institutions and researchers from around the world and funded by a number of international organizations and large corporations, such as Trimble and Leica Geosystems. PCL's quality was recognized in 2011 at the Open Source World Challenge in Seoul, where the library received the highest Grand Prix award [www.pointclouds.org].

The PCL can work with the top popular operating systems: Windows, Linux and MacOS. The algorithms of the library are written in C++. All codes are created according to clearly presented, uniform rules, ensuring consistency and clarity of the language.

The library, in addition to the source codes of the implemented algorithms, also provides extensive Application Programming Interface (API) documentation. API shows in detail all the functions and structure of the interface, presents fragments of the code that can be used for quick testing of specific algorithms and provides links to publications describing the principles of operation of some of them.

3. The PCL file format – PCD

PCL has its own data format – PCD (Point Cloud Data), which facilitates the best adaptation of the input data to the algorithms contained in the library and guarantees the highest performance. A PCD file consists of a header, specifying the properties of the point cloud and data encoded by means of one of the two ASCII main methods or binary. Header, in consecutive lines it defines [www.pointclouds.org]:

1. VERSION – the PCD file version (usually .7).
2. FIELDS – the name of each dimension/field that a point can have (e.g. FIELDS x y z).
3. SIZE – the size of each dimension in bytes (e.g. a float is 4).
4. TYPE – the type of each dimension as a character (I = signed, U = unsigned, F = float).
5. COUNT – the number of elements in each dimension (e.g. x, y, or z would only have 1).
6. WIDTH – the width of the point cloud dataset in the number of points (the total number of points in the cloud for unorganized datasets, total number of points in a row of an organized).
7. HEIGHT – the height of the point cloud dataset, in the number of points (set to 1 for unorganized point clouds, total number of rows of an organized point cloud).

8. VIEWPOINT – an acquisition viewpoint for the points: translation (tx ty tz) + quaternion (qw qx qy qz).
9. POINTS – the total number of points in the cloud.
10. DATA – the data type that the point cloud data is stored in (ASCII or binary).

4. PCL structure

The Point Cloud Library contains many of the key algorithms designed for filtering, registration, estimating of function, surface reconstruction, modeling and segmentation, as well as structures for identifying and matching objects. Algorithms are divided into smaller, modular libraries, containing the functions for specific tasks assigned to them [Rusu and Cousins 2011].

Particularly useful when working with laser scan data are the following modules:

1. IO module – used to write and read clouds in PCD format. It includes algorithms for creating and reading PCD files and changing the data from ASCII to binary or vice versa.
2. Filter module – designed to eliminate unnecessary and erroneous point clouds. It contains a number of point cloud downsampling algorithms based on the normal ones and established on the maximum number of points or averaging the position of the neighboring points. In addition, it provides many features to reduce measurement noise, i.e., on the basis of the distribution of neighboring points, the intensity of reflection, or statistics and algorithms that remove points inside or outside the designated area such as a box or a convex hull.
3. Registration module – it can accommodate multiple point clouds into one global system. It includes algorithms for connecting clouds on the basis of keypoints or normal of planes. It provides a number of functions iteratively performing transformations based on various criteria, such as the maximum number of repetitions or exceeding the set threshold. It also provides structures viewing the quality and stability of established connections and rejecting the invalid ones that exceed the specified tolerance limits, as well as optimizing connection errors.
4. Keypoints module – designed to detect specific, clearly identifiable point in the cloud, the so-called keypoints. It contains a number of algorithms for the extraction of keypoints, e.g. through the use of normal surfaces and the intensity of reflection.
5. Segmentation module – used to divide the point cloud into smaller clusters. It contains a number of advanced algorithms to segment data based on multiple criteria, built e.g. on the basis of Euclidean distance. It also provides functions for extracting points within e.g. established geometric figures.
6. Surface module – it allows for the construction of three-dimensional surfaces. Contains algorithms for generating surfaces either on the basis of a 3D grid or normal surfaces. Also, it includes a number of functions to smooth the surface and

transitions between them, as well as to improve the structure of the grid, fill holes in it, and to add texture.

7. Features module – is used to estimate the function of a point cloud. It contains a number of advanced features to detect the curvature of surfaces, mainly determined based on the proximity of points, geometric functions, the angle criterion, the technique of spatial data distribution, normal, intensity, or histograms.
8. Visualization module – used for visualization of algorithm operation results. It uses the VTK (Visualization Toolkit) library, which provides a 3D graphical environment used for visualization. It contains a number of algorithms for simple displaying of point clouds, as well as more complex functions for rendering and setting visual properties such as colors, sizes, points, drawing basic 3D shapes on the screen such as cylinders, spheres, lines, polygons, or creating them with parametric equations.

5. Example usage: downsampling a point cloud

In terrestrial laser scanning, unlike in classical land surveying, observations are replaced with single point data sets with very large data volumes. In most studies, high density of point clouds provided by scanners is not needed and what is more, excess data impedes effective processing. Therefore, in order to save computer memory, size reduction is done through the downsampling of the point cloud data. The proper execution of this operation facilitates further processing of the data, while retaining the essential information [Bohler et al. 2002]. The PCL library, among the algorithms, also provides a number of features for downsampling point clouds. One of them was used in this example.

A statue of Pope John XXIII (Figure 1a), located in Wrocław Street St Martin was used as test object. The monument's data, in the form of point cloud, was acquired by a TLS device FARO Focus3D and was stored in ASCII format. In the example, the PCL library in the latest available version (1.6) was used, along with the free development environment Visual Studio C++ 2010 Express and Open Source tools to manage the process of compiling: CMake 2.8.6. The work began with a test store format point cloud library – PCD, through appropriate wording of the header file, declaring the properties of the point cloud (Figure 1b).

The test point cloud was downsampled using the VoxelGrid algorithm, operating on the basis of a three-dimensional grid of so-called voxels. The grid is created based on the input point cloud, and the size of blocks is a filter parameter. Cloud points (based on their location) are sorted into the appropriate voxels. Then within each block, the centroid of the points in it, is calculated. The algorithm disperses the input point cloud, replacing it with one composed of designated centers of gravity [www.pointclouds.org].

a)

b)

```
 # .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z
SIZE 4 4 4
TYPE F F F
COUNT 1 1 1
WIDTH 99962
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 99962
DATA ascii
43.4985 58.0794 3.6565
43.4892 58.0916 3.6542
43.4812 58.1067 3.6508
43.4522 58.1075 3.6541
43.4379 58.1080 3.6558
43.4227 58.1088 3.6574
43.4024 58.0928 3.6645
43.5554 58.1116 3.6256
43.5455 58.1237 3.6233
```

Source: a) silesian.eu, b) author's study

Fig. 1. Test object: a) object photography and b) test point cloud in PCD format

The source code for the algorithm (Figure 2), according to its tasks, can be divided into four main parts:

A. Adding header files with descriptions of the required interfaces.

B. Reading the input point cloud *monument.pcd* contained in the catalog *cloud,* placed in the source files of the library.

C. Downsampling the data, based on a grid composed of voxels measuring 0.01 m × 0.01 m × 0.01 m and transferring the filtered clouds to the output directory *cloud_filtered*.

D. Saving the downsampled data, in the file *monument_downsample.pcd* contained in the directory *cloud_filtered* placed in the source files of the library.

In order to process the algorithm source code into machine language, in a CMakeLists.txt file (Figure 3), object files that should arise as a result of compiling were defined. Then, it was determined how to combine them with the library and install.

Once the executable file was available, the created program was run by means of $ ./voxel_grid, resulting in a message containing information on the number of input and output point clouds (Figure 4).

```
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/filters/voxel_grid.h>

int
main (int argc, char** argv)
{
  sensor_msgs::PointCloud2::Ptr cloud (new sensor_msgs::PointCloud2 ());
  sensor_msgs::PointCloud2::Ptr cloud_filtered (new sensor_msgs::PointCloud2 ());


  pcl::PCDReader reader;
  reader.read ("monument.pcd", *cloud);

  std::cerr << "PointCloud before filtering: " << cloud->width * cloud->height
       << " data points (" << pcl::getFieldsList (*cloud) << ").";


  pcl::VoxelGrid<sensor_msgs::PointCloud2> sor;
  sor.setInputCloud (*cloud);
  sor.setLeafSize (0.01f, 0.01f, 0.01f);
  sor.filter (*cloud_filtered);

  std::cerr << "PointCloud after filtering: " << cloud_filtered->width * cloud_filtered-
>height
       << " data points (" << pcl::getFieldsList (*cloud_filtered) << ").";

  pcl::PCDWriter writer;
  writer.write ("monument_downsampled.pcd", *cloud_filtered,
         Eigen::Vector4f::Zero (), Eigen::Quaternionf::Identity (), false);

  return (0);
}
```

A

B

C

D

Source: author's study

Fig. 2. VoxelGrid – source code

```
cmake_minimum_required ( VERSION 2.8.6)

 project ( voxel_grid )

 find_package ( PCL 1.6 )

 include_directories ( ${ PCL_INCLUDE_DIRS } )
 link_directories ( ${ PCL_LIBRARY_DIRS } )
 add_definitions ( ${ PCL_DEFINITIONS } )

 add_executable ( voxel_grid voxel_grid.cpp )
 target_link_libraries ( voxel_grid ${ PCL_LIBRARIES } )
```

Source: author's study

Fig. 3. The CMakeLists.txt file for the VoxelGrid algorithm

```
PointCloud before filtering: 99962 data points <xyz>.P
PointCloud after filtering: 3942 data points <xyz>.
```

Source: author's study

Fig. 4. The effect of the VoxelGrid algorithm (0.01 m × 0.01m × 0.01m)

The VoxelGrid algorithm was used again, on the same input data by increasing the dimensions of a single voxel to 0.04 m × 0.04 m × 0.04 m. Having acquired (Figure 5):

```
PointCloud before filtering: 99962 data points <xyz>.P
PointCloud after filtering: 1421 data points <xyz>.
```

Source: author's study

Fig. 5. The effect of the VoxelGrid algorithm (0.04 m × 0.04m × 0.04m)

Thanks to the visualization module, the downsampled could points were presented immediately after the operation of the algorithm. In order to view the input and output data, one of the simplest functions of the module – CloudViewer – was used (Figure 6).

```
#include <pcl/visualization/cloud_viewer.h>
#include <iostream>
#include <pcl/io/io.h>
#include <pcl/io/pcd_io.h>

int
main ()
{
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud;
    pcl::io::loadPCDFile ("monument.pcd", *cloud);

    pcl::visualization::CloudViewer viewer("Cloud Viewer");

    viewer.showCloud(cloud);
    while (!viewer.wasStopped ())
    {
    }
}
```

Source: author's study

Fig. 6. CloudViewer – source code

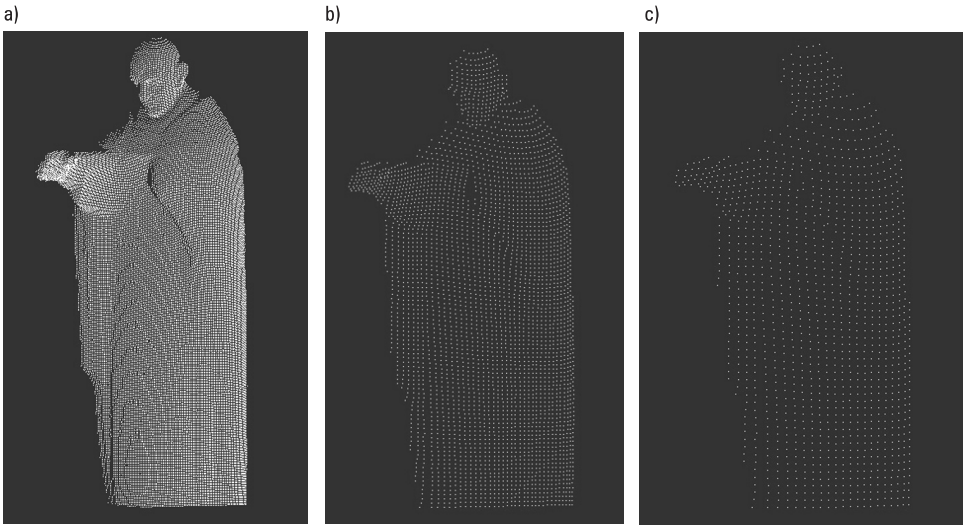
Again, to manage the process of compiling in the program, the file CMakeLists.txt (Figure 7) was used.

Then, using the $ ./ cloud_viewer, the executable file was run, visualizing the cloud (Figure 8).

```
cmake_minimum_required ( VERSION 2.8.6 )

project (cloud_viewer)

find_package (PCL 1.6)

include_directories(${PCL_INCLUDE_DIRS})
link_directories(${PCL_LIBRARY_DIRS})
add_definitions(${PCL_DEFINITIONS})

add_executable (cloud_viewer cloud_viewer.cpp)
target_link_libraries (cloud_viewer ${PCL_LIBRARIES})
```

Source: author's study

Fig. 7. The CMakeLists.txt file for the CloudViewer algorithm



Source: author's study

Fig. 8. Point cloud visualization: a) input point cloud, b) downsampled point cloud – 0.01m and c) downsampled point cloud – 0.04 m

6. Conclusion

The PCL library provides advanced algorithms for point cloud processing and their visualization. It runs on an Open Source license, allowing for free and fully legal copying of the source code, encouraging research into the performance and efficiency of the algorithms. In addition, any of them can be modified which allows for customization of the existing algorithms for the future implementation of independent projects and the creation of new and increasingly effective solutions.

A practical example of PCL's application, showing the downsampling a TLS point cloud, has demonstrated the effectiveness of the library and the ease of implementation of the source code. Using simple steps, the input data was saved in the correct format, operations to the point cloud were performed and the achieved results visualized.

Terrestrial laser scanning is developing towards the creation of procedures and algorithms that minimize the time consumption and labor-intensiveness of operator studies. The initial results presented, in the article have shown that a generally available programming library, providing key algorithms for point cloud processing is the hope for the development of a post-processing automation technology for data acquired by terrestrial laser scanning. Work on the application of PCL in this respect shall be continued.

Acknowledgments

Many thanks to Mr. Marek Jędrzejczak of the WROGEO company from Wrocław, for sharing the point cloud used for this paper.

References

Bae K.H., Lichti D.D. 2004. Automated registration of unorganized point clouds from terrestrial laser scanners. IAPRS, Istanbul, Turkey, 12–23 July, 222–227.

Bauer J., Karner K., Schindler K., Klaus A., Zach C. 2005. Segmentation of building from dense 3D point-clouds. Proceedings of the ISPRS. Workshop Laser scanning, Enschede, the Netherlands, 12–14 September.

Belton D., Lichti D.D. 2006. Classification and segmentation of terrestrial laser scanner point clouds, using local variance information. IAPRS, 36, 5, Dresden, 25–27 September.

Bienert A., Scheller S., Keane E., Mullooly G., Mohan F. 2006. Application of terrestrial laser scanners for the determination of forest inventory parameters. Image Engineering and Vision Metrology. ISPRS Commission V Symposium, WG 3, Dresden, 25–27.

Bohler W., Heinz G., Marbs A., Siebold M. 2002. 3D scanning software – an introduction. International Workshop on Scanning for Cultural Heritage Recording (CIPA 2002), Corfu, Greece, 9–13.

Dold C., Brenner C. 2006. Registration of terrestrial laser scanning data using planar patches and image data. IAPRS, 36, 5, Dresden, 25–27 September.

Fröhlich C., Mettenleiter M. 2004. Terrestrial laser scanning: New perspectives in 3D surveying. IAPRS&SIS, Freiburg, Germany, 36, 8/W2, 7–13.

Kadobayashi R., Kochi N., Otani H., Furukawa R. 2004. Comparison and evaluation of laser scanning and photogrammetry and their combined use for digital recording of cultural heritage. IAPRS&SIS, Istanbul, Turkey, 401–406.

Pu S., Rutzinger M., Vosselman G., Elberink S. 2011. Recognizing basic structures from mobile laser scanning data for road inventory studies. ISPRS J. Photogram. Remote Sens., 66, 28–39.

Pu S., Vosselman G. 2006. Automatic extraction of building features from terrestrial laser scanning. IAPRS, 36, 5, Dresden, Germany, 25–27 September.

Rabanni T. 2006. Automatic reconstruction of Industrial Installations using point clouds and images. PhD thesis. NCG, Delft, Netherlands, Publications on Geodesy, 62.

Rusu R.B., Cousins S. 2011. 3D is here: Point cloud library (PCL). In IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China.

Sithole G., Vosselmann G. 2003. Report: ISPRS Comparison of filters. Department of Geodesy, Faculty of Civil Engineering and GeoSciences, Delft University of Technology.

Staiger R. 2003. Terrestrial laser scanning – technology, systems and applications. 2nd Regional Conference FIG, Marrakech, Morocco.

Weber C., Hahmann S., Hagen H. 2010a. Methods for feature detection in Point Clouds. Shape Modeling International Conference (SMI), Aix-en-Provence, France.

Weber C., Hahmann S., Hagen H. 2010b. Sharp feature detection in point clouds. Shape Modeling International Conference (SMI), Aix-en-Provence, France.

Wendt A. 2004. On the automation of the registration of point clouds using the metropolis algorithm. IAPRS. Istanbul, Turkey, 12–23 July, 106–111.

Wezyk P., Koziol K., Glista M., Pierzchalski M. 2007. Terrestrial laser scanning versus traditional forest inventory, first results from the polish forests. Proceedings of ISPRS Workshop on laser scanning 2007 and SilviLaser. Espoo, Finland, 12–14 September, 424–430.

www.pointclouds.org – PCL official website.

www.willowgarage.com – Willow Garage official website.

Mgr inż. Maria Zygmunt
Uniwersytet Rolniczy w Krakowie
Katedra Geodezji
30–149 Kraków, ul. Balicka 253 a
e-mail: mary.zygmunt@gmail.com