RESEARCH ON VISUALIZATION OF GROUND LASER RADAR DATA BASED ON OSG

Haowen Huang [1,2,3], Chunmei Hu[1,2,3] *, Fang Zhang [1,2,3], Huimin Xue[1,2,3]

[1] School of Geomantic and Urban Information, Beijing University of Civil Engineering and Architecture, NO.15Yongyuan Road, Daxing District, Beijing, 102616 - (Haowen Huang, Chunmei Hu, Fang Zhang, Huimin Xue)@bucea.edu.cn
[2] Beijing Key Laboratory For Architectural Heritage Fine Reconstruction & Health Monitoring, NO.15Yongyuan Road, Daxing District, Beijing, 102616
[3] Engineering Research Center of Representative Building and Architectural Heritage Database, Ministry of Education, NO.15Yongyuan Road, Daxing District, Beijing, 102616

Commission III, WG III/5

KEY WORDS: LiDAR, Data Visualization, OSG, Point Cloud, Triangulation Network, Qt

ABSTRACT:

Three-dimensional (3D) laser scanning is a new advanced technology integrating light, machine, electricity, and computer technologies. It can conduct 3D scanning to the whole shape and form of space objects with high precision. With this technology, you can directly collect the point cloud data of a ground object and create the structure of it for rendering. People use excellent 3D rendering engine to optimize and display the 3D model in order to meet the higher requirements of real time realism rendering and the complexity of the scene. OpenSceneGraph (OSG) is an open source 3D graphics engine. Compared with the current mainstream 3D rendering engine, OSG is practical, economical, and easy to expand. Therefore, OSG is widely used in the fields of virtual simulation, virtual reality, science and engineering visualization. In this paper, a dynamic and interactive ground LiDAR data visualization platform is constructed based on the OSG and the cross-platform C++ application development framework Qt. In view of the point cloud data of .txt format and the triangulation network data file of .obj format, the functions of 3D laser point cloud and triangulation network data display are realized. It is proved by experiments that the platform is of strong practical value as it is easy to operate and provides good interaction.

1. INTRODUCTION

The three-dimensional (3D) laser scanning technology is a new means of surveying and mapping, and it has quickly become an important method of acquiring 3D spatial information because it has many advantages such as high-speed data acquisition, high precision, not contacting the target. It is widely used in ancient buildings protection, large structure deformation monitoring, and city measurement, etc. The directly acquired original point cloud data which are a set of discrete points in a 3D space represent the basis for subsequent data analysis, grid generation, and 3D visualization. The 3D visualization of data is an important segment in the application of laser scanning technology. Visualization is a relatively new research field that has developed rapidly since it was proposed by developed countries in the late 1980s. It is also a new theory, method, and technology to transform data into graphics or images on screen and allow interactive processing by means of computer graphics and the image processing technology. With the development of computer graphics technologies, Two-dimensional (2D) images are increasingly unable to meet people’s needs when expressing massive and complex information. Used as a technical method to describe and understand the phenomenon characteristics in a 3D space, 3D visualization is increasingly paid attention to by various industries (Wan, D., 2009). Therefore, it is important to find a good visualization platform for Ground LiDAR data, which is of great significance for rapidly acquiring and evaluating the quality of point cloud data and the progress planning of point clouds processing.

The 3D laser scanning technology is used to obtain high-precision and high-density 3D data, so the amount of data is especially huge. The massive point cloud data needs a powerful engine as the core of the visualization platform to ensure rapidity, efficiency, and smoothness of its operation. Traditional modes of directly using underlying graphic interfaces (e.g. OpenGL and DirectX) to develop graphics applications are posing a great many issues, such as large development complexity, long periodicity, and difficult maintenance. To resolve these issues, many excellent 3D rendering engines have been created, such as Delta3D, OGRE, OSG, Unity3d, and VTK, etc (Qiu, H., 2010). OSG is a high-performance open source 3D graphics engine, and also an open source, multi-platform graphics development package. Based on the concept of the scene graph, it provides an object-oriented framework that encapsulates the underlying details of the OpenGL. Accordingly, the developer is freed from the call to implement and optimize the underlying graphics, and many additional useful tools can be used for rapid development of graphical applications. One of the advantages of OSG is its following the open source protocol. Its user license is a modified GNU Lesser General Public License (LGPL), which is called OSGPL.

This paper combines a real-time 3D rendering engine OSG and Qt interface design with C++ programming to realize a 3D

* Corresponding author. E-mail addresses: huchunmei@bucea.edu.cn (Chunmei Hu)

visualization platform for ground LiDAR data. Experiments show that the platform can provide the user with a good 3D visualization experience of ground LiDAR data.

2. CONSTRUCTION OF DATA VISUALIZATION PLATFORM BASED ON OSG

We use Qt as GUI to integrate OSG 3D engine development visualization or model browse Windows applications with Visual Studio 2010 + Qt4.8.6 + OSG3.4.0.

2.1 OSG Graphics Engine

The OSG is an open source graphics library scene, providing scene management and rendering graphics optimization functions for the development of graphics applications. It uses a portable ANSI C++ preparation and the bottom of the OpenGL rendering API which has become industry standard and plays a key role in the level of 3D scene application. It is a middleware that provides a variety of advanced rendering and texturing functions, IO, and spatial architecture and scene organization functions for various software. So the OSG is cross platform and can run on operating systems such as Windows, GNU, Linux, IRIX, Solaris, HP, and FreeBSD. So far, the functions of OSG contain paging support for large-scale scenes, multithreading, multi display rendering, support of various file formats, and encapsulation of languages, etc. It enables programmers to create high performance, cross platform interactive graphical programs more quickly and conveniently.

2.1.1 Architecture of OSG: The OSG and its extension are located at the API level of the system, that is, the underlying graphics hardware and corresponding software drivers on the system, which encapsulates OpenGL, and supports the rest of the underlying graphics display. It can easily develop its upper application. The OSG mainly includes the OSG core library, the OSG tool library, the OSG plug-in library, and the OSG introspection library (Wang R., 2009). The OSG architecture diagram is shown in Figure 1.

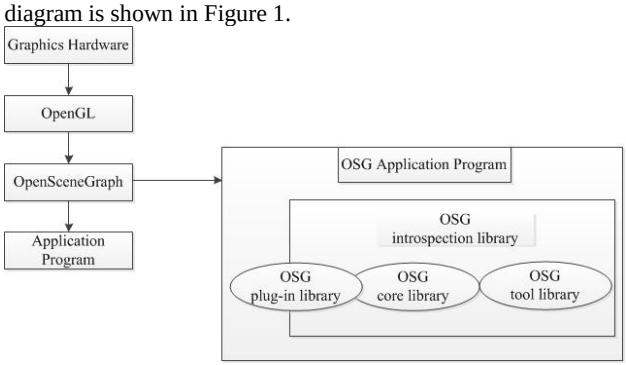

Figure 1. The OSG architecture

(1) The core library of OSG is to realize the organization and management of the core scene database, to operate the scene graph, and to provide the interface for the import of external database, which includes OSG library, osgUtil library, osgDB library and osgViewer library. The OSG library provides the basic nodes needed for creating scene graphics, as well as the management structure and methods for these nodes; The osgUtil library is a utility library that provides common public classes for operating scene graphics and content. The osgDB library is the reading and writing Library of the data, which provides the reading and writing of the data in the scene. The osgViewer library is an integrated tool for observing and managing single or multiple scenes, and it also provides multi thread, multi CPU, multi scene rendering mechanism.

(2) The OSG toolkit is an extension of the OSG Library in the OSG core library. It implements some specific functions, including osgFX library, osgParticle library, osgSim library, osgTerrain library, osgText library, and osgShadow library. The osgFX library is the scene effect library, which is used to render the special effects nodes. The osgParticle library is the particle effect library used to realize simple or complex particle effects, such as rain and snow. The osgSim library is a simulation tool library, including DOF transform nodes, texture overlay nodes and function in a variety of virtual simulation function related set. The osgTerrain library is used for rendering terrain processing library, TIF, IMAGE and DEM format elevation data; The osgText library is a text processing library, which realizes the display of dot matrix or vector text. The osgShadow library is the shadow effect library, which implements various forms of shadow rendering, including Shadow Map, Shadow Texture, shadow Volume, and the latest shadow technology based on GPU, so as to improve the authenticity of scene rendering.

(3) The OSG plug-in library, which supports a variety of third party libraries, enables the OSG to directly or indirectly import 2D images, 3D model files, and other types of files. 2D graphics file formats include .bmp，.gif，.jpeg，.pic，.png，.rgb，.tga, and .tiff. The 3D model file format includes common formats such as 3D Studio Max (.3ds), AliasWavefront (.obj), Carbon Graphics'Geo (.geo), OpenFlight (.flt) and so on. In addition, I/O operations for compressed files and file sets are also supported.

(4) The OSG introspection library allows integrating of the OSG with other development environments, such as scripting languages Python, Java, TCL, Lua. The osgIntrospection Library in the introspection library allows user software to interact with the OSG using reflective and introspective programming paradigms.

2.1.2 Installation and compilation of OSG: The source code for OSG is easy to get, but it needs to be compiled. So the installation and compilation process of OSG is as follows in Figure 2.

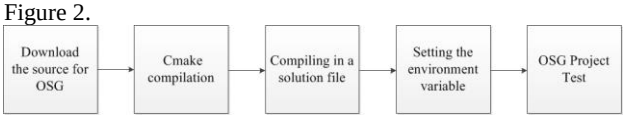

Figure 2. OSG compilation steps

(1) OSG source code download. Landing official website to download OSG source code, OSG third party library, and OSG data package.

(2) CMake compilation preparation. Building a solution by compiling OSG with the tool, CMake. At first, we extract the source file, then drag the CMakeLists.txt file into the CMake and select the development environment. After the CMake auto configuration is completed, the third party plug-in library ACTUAL_3DPARTY_DIR, the compiled OSG example, and the header files of some plug-ins are manually configured.

(3) Compiling in OpenSceneGraph.sln. After the CMake configuration is completed, VS2010 is used to open the solution again. Under the menu of batch generation, the All_BUILD is

generated, and the two versions (Debug and Release) are configured.

(4) Setting the environment variable. The following settings: OSG_FILE_PATH: C:\OSG\data, PATH: C:\OSG\bin.

(5) OSG project test. After all operations are completed, you can test in the command line (CMD) and also build a new OSG project. If the test is not wrong, it shows that the installation of OSG is completed successfully, and other operations can be done based on OSG. The test result is shown in Figure 3.

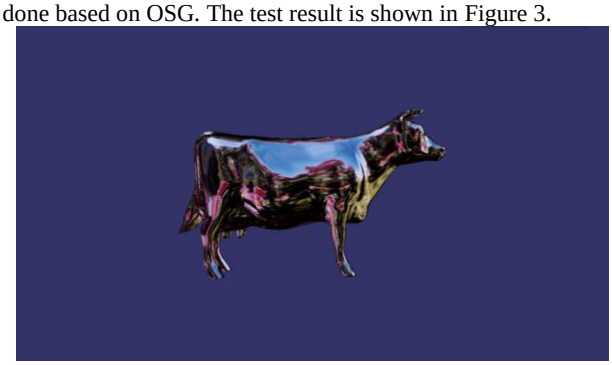

Figure 3. Test result

2.2 Qt

Qt is a multi-platform C++ graphical user interface application development framework. It can not only develop a GUI program, but also develop a non GUI program. Through the continuous development and advances, Qt, a Norway's Qt Software product, has the perfect C++ graphics library, and in recent years it gradually integrates database, OpenGL database, multimedia database (Phonon), network library, library, XML library, Web Kit library, and so on. Qt is an object oriented framework, which uses special code generation extensions, called Meta Object Compiler (MOC), and some macros. It is easy to expand and allows component programming. MOC processes Qt code into standard C++ code. Qt is widely used in the development of Opera, Google Earth and Skype. It has the following advantages: excellent cross platform characteristics, object-oriented features, rich API, support for 2D/3D graphics rendering and OpenGL, and lots of development documents and XML support.

2.3 Building scheme of Data Visualization Platform

The building of the data visualization platform requires embeding OSG in Qt. In the OSG rendering process, the external control is limited to the keyboard and mouse, and additional functions cannot be reasonably arranged. Therefore, the Qt framework is introduced to manage. The OSG is embedded in Qt so that the UI interface of Qt can be designed to compensate for the shortcomings of the 2D interface system of OSG. There are three ways to implement embedding.

In the first approach, we use multiple processes to make the two windows superimposed on a way of external control, which looks like father-and-son relationship, but is not actually a real one. There is no connection between the two windows, and mutual communication or variable exchange is difficult to achieve. Therefore, this idea is not practical.

In the second approach, the underlying OSG uses the OpenGL graphics system, and Qt also adds support for OpenGL. Through the Q type window QGL-Widget supported OpenGL window in Qt, the OSG window is derived from this class. Then, the initialization scene and timing refresh are completed, and the OSG window is embedded into Qt. QGLWidget is the multiple inheritance of QWidget and QGL, which gives this Qt type window a 3D visual effect.

In the third approach, we directly use osgViewer:: GraphicsWindowQt class to achieve the goal. With GraphicsWindowQt inheriting from GraphicsContext, we can design a new device object that encapsulates Q type window QGLWidget supported by OpenGL window in Qt, which is used for OSG rendering. Then the object based on the GraphicsWindowQt declaration initializes the view, and calls the getGLWidget () method to return a QWidget and embed it into the Qt window. The inheritance diagram of the GraphicsWindowQt is shown in Figure 4.

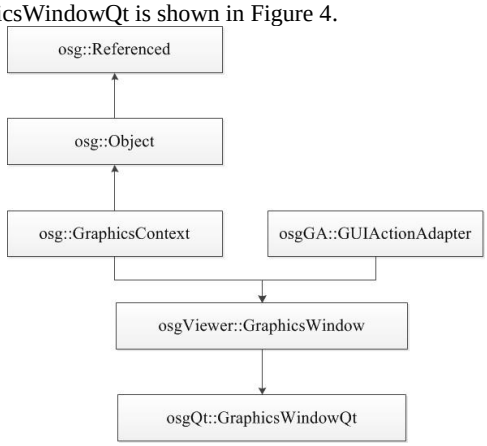

Figure 4. The inheritance of osgViewer::GraphicsWindowQt

Although the latter two are implemented on the basis of QGLWidget window class and the same principle is applied, in the third way of thinking, QGLWidget uses GraphicsWindowQt for internal encapsulation and the code is more succinct. This paper adopts the third method. First, a Qt Application project is created under VC2010, and MainWindow is selected as the main window of the platform. Then the platform interface is designed in detail in the "Qt designer". The addition of frame and Widget in the splitter control, with frame as a display of scene tree and interactive interface and widget, the viewport, as the OSG work area. Then widget will be promoted to OsgViewrWidget. In this way, the data visualization platform based on OSG and Qt is built (Figure 5).

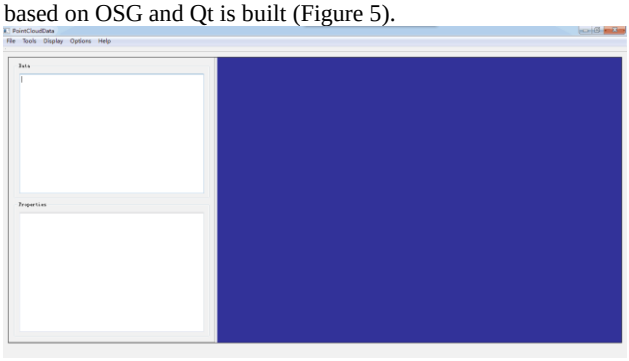

Figure 5. Main interface of data visualization platform

3. DATA VISUALIZATION BASED ON OSG AND QT

In general, there are three ways to create a geometry in the OSG. First is to use a loosely encapsulated OpenGL plotting primitives. Second is to use the basic geometry in OSG. Third is t import the scene model from the file. The first method is very flexible, but workload is huge in the face of large scenes. Consequently, we use the latter two methods to visualize the LiDAR data.

3.1 Visualization of point cloud data

We use the basic geometry of the OSG to draw point cloud, in order to realize the visualization of point cloud data. It can be roughly divided into four steps (Figure 6). In this paper, the point cloud data (.txt) of Ren Xiang gate in the Forbidden City is taken as an example.

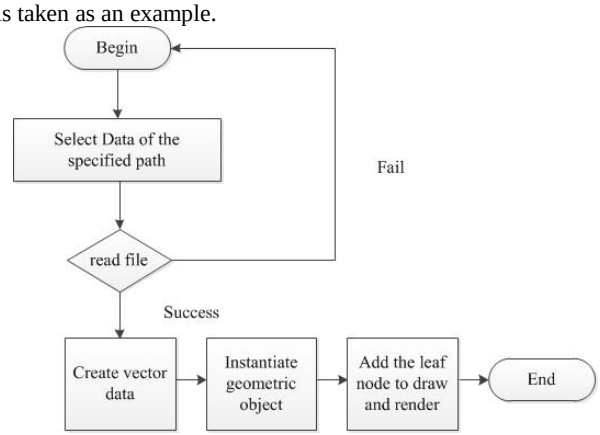

Figure 6. Step of point cloud data visualization

(1) Click on the platform to select the point cloud data of the specified path.

(2) Create various vector data, define points and color arrays. Add vertex data in reverse clockwise order.

(3) Instantiate a geometric object (osg::Geometry), and set a vertex array, a color array, a normal array, a binding method, and a data parsing. A series of three-dimensional data points are pressed into the container, and the Geometry class of OSG is used to draw the point cloud map. Among them, the specified vector array is implemented by *void setVertexArray (Array*array)* which set the vertex array, *void setColorArray (Array*array)* which set the color array and *void setNormalArray (Array*array)* which set the normal array.There are two main bindings for data setting: *void setNormalBinding (Array*array)* and *void setColorBinding (Array*array)*. Data parsing is based on the formulation of various vector data and binding modes, deciding which way to render it, implemented through *Bool addPrimitiveSet (PrimitiveSet*primitiveset)*.

(4) Add to the leaf node to draw and render it.Through sentence *osg::ref_ptr<osg::Group> root = new osg::Group;*, we create group nodes as root nodes, add leaf nodes that need to be displayed to group nodes, and use a root node to control the display of all the next sub nodes. Rendering is mainly divided into two steps. First, create viewer objects, scene browsers, and then set the read nodes to the scene by *viewer->setSceneData (root)*. In this way, the external point cloud data can be read in the built 3D scene to realize the visualization of the point cloud data (Figure 7).

Figure 7. Point cloud data visualization of Ren Xiang gate

3.2 Visualization of triangulation network data

OSG can be imported directly into most of the common model file formats. Using the interface provided to read the 3D model and the third party plug-in system management for different format files in the osgDB library, we realize the visualization of triangulation network data. It is roughly divided into the following three steps (Figure 8). This paper takes the triangulation network data (.obj) of Ren Xiang gate in the Forbidden City as an example.

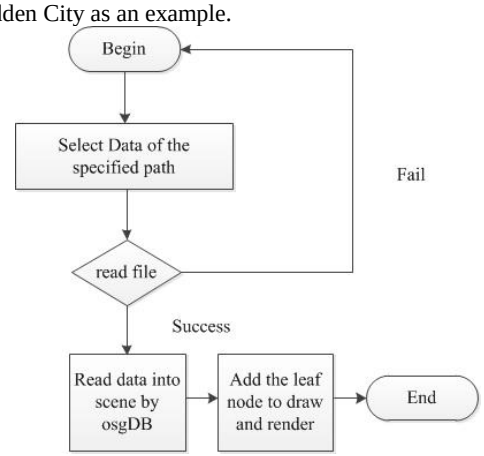

Figure 8. Step of triangulation network data visualization

Firstly, click on the platform to select the triangulated network data of the specified path. Secondly, read this data into the scene by *osg:: ref_ptr<osg:: Node> node = osgDB:: readNodeFile ("xx.obj")*. Finally, add to the leaf node to draw and render it. This part is the same as the point cloud data to realize the visualization of the triangulation network data (Figure 9).

Figure 9. Triangulation network data visualization of Ren Xiang gate

4. CONCLUSION

Based on studies of key technologies and methods of 3D visualization, the paper combines the open source 3D visualization engine OSG with the multi-platform development frame Qt to develop and realize systems, and initially constructs ground LiDAR data visualization platform with the development tool VS2010. The OSG is responsible for the rendering of the scene, and the Qt links the corresponding signals and grooves to accept the user's interaction events. The new visualization platform is of great help to provide convenient and intuitive visual services for the subsequent application of data.

In this paper, we have preliminarily discussed and tested on the visualization platform and visualization method of point cloud data and triangulated data. However, there are still some shortcomings in the research. Follow-up research will focus on the following aspects:

(1) The read and display of mass point cloud data. When the the point cloud data is large, it is very difficult to load the platform. How to improve the read and display of the mass point cloud data will be the focus of the future research.

(2) Enhancing the ability of platform spatial analysis. In this paper, we only realize the visualization of ground LiDAR data on the platform instead of integrating the pick-up analysis and editing function of point cloud data and triangulation network data. The next step is to further study how to achieve it.

(3) Due to time constraints, no comparison with other visualization platforms is made. And the platform interface design is also not beautiful enough. This is what we should work on for future research work.

REFERENCES

Dong, L. I., Yun, L. I., Wang, S. J., et al., 2014. On 3d rendering of las file based on osg. *Journal of Hengyang Normal University*, 35(6), pp. 97-100.

Lie, X. U., Wei, Q., Wang, J., 2011. Research and implementation of 3d scene management and real-time rendering technology based on osg. *Journal of the Academy of Equipment Command & Technology*, 22(3), pp. 100-104.

Liu, Y. Q., Gui-Lian, S. U., 2009. Design and implementation of graphical user interface program based on qt4. *Modern Computer*, 3, pp. 55.

Qiu, H., Chen, L. T., 2010. Design and implementation of object-oriented 3d graphics engine. *Journal of University of Electronic Science & Technology of China*, 39(1), pp. 123-127.

Wan, D.,2009. Development and application of three-dimensional visualization system for hydraulic engineering based on osg. *Computer & Digital Engineering*, 37(4), pp. 135-137.

Wang R., Qian X. L., 2009. Design and Practice of OpenSceneGraph 3d rendering engine. Tsinghua University press, pp.6-10.

Wen, Z. P., Shen, Y. C., 2009. Design and implementation of visual campus ramble system based on osg. *Computer Technology & Development*, 19(1), pp. 217-220.

Yin, Z., Wang, T., Zhou, L., Li, M., 2013. Development of 3d pipeline information system based on osg engine. *Urban Geotechnical Investigation & Surveying*, (1), pp.56-59.

Zeng, X., 2012. Webgl based lidar point clouds visualization. *Journal of Hunan University of Science & Technology*, 27(4), pp.60-64.

Zhou, K. Q., Zhao, X., Ding, Y. H., 2006. The 3d visualization's approach based on laser scanning. *Journal of Zhengzhou Institute of Surveying & Mapping*, 31(5), pp. 93-94.