Rendering a 3D city

Otto Nordgren
otto.nordgren@gmail.com

Yuri Stange
yuris@kth.se

May 21, 2012

Abstract

The purpose of this paper is to look at the possibilities to render a three dimensional replica of an area based on coordinates taken from a LIDAR map. Given x, y, z coordinates, we want to find out if the rendered world is usable as a map reference for real life scenarios and if you can include landmarks based on locations taken from OpenStreetMap. We will try to determine if this method is usable on urban areas such as the KTH Campus. We will also try to evaluate how well this method scales with larger maps and what kind of resolution that is needed on the LIDAR map. Using an open source program called Blender we were able to import the coordinates and render a three dimensional world. There was a lot of work that needed to be done with the data, such as removing duplicates and false information before the import. We concluded that this technique heavily depends on the quality of the data and it's not the optimal choice to use when rendering urban areas. However we think that this method could prove very useful when rendering less detailed areas such as open fields with soft contours. There was a problem with scalability in Blender that forced us to use a smaller version of the map. Areas such as the KTH Campus or smaller towns contains a lot of information that needs to be rendered. This can be solved by programmatically loading the data as the user moves around the map.

Statement of collaboration

The background research was shared by the writers of this report. Yuri wrote about LIDAR and LAS while Otto did the same with PLY and Blender. During the rendering phase Otto and Yuri tested different possibilities which resulted in the scripts Otto wrote. These scripts transformed the LAS-files from Lantmäteriet into workable PLY-files. Because of his experience in 3D-rendering Yuri took on the job of rendering the map and creating the faces. This led to a 3D environment which Otto could then analyze. He also wrote down the results in the report.

Contents

1 Introduction

During the beginning of the 21th century the development of online maps and reference-applications has evolved at a great speed. Much thanks to the incredible amount of available processing power and the Internet, which allows almost anyone to look up anything. An online map like Google Maps or Bing Maps gives the user the ability to see their own house from the sky and even walk the street in a two dimensional environment. The market for true three dimensional maps is however at the moment not as developed. Google Maps has three dimensional buildings on their maps but it's not possible to actually walk amongst them.

This report will look at the pros and cons in rendering a three dimensional world with objects such as buildings based on a grid of x, y, z points. The grid that we are going to use is served by Lantmäteriet and collected with a technique called LIDAR. Our approach is to render a world based on the grid with a rendering program called Blender. We will try to find out if this approach is viable method considering both usability and scalability.

2 Background

Companies like Eniro and hitta.se currently hosts services where a user can view a city in three dimensions online, an example can be seen in figure 1 and in figure 2 on page 13. They use technologies with military-maps and photos with exact positions which is then calculated in order to produce a 3D render. The problem with theese services is that they are commercial and not open for people on the ouside to edit.

2.1 LIDAR

LIDAR(Light Detection and Ranging, also called LADAR) is an optical remote sensing technology that can measure the distance to, or the properties of a target, by illuminating it with a light source, often a laser. The LIDAR system can use both ultraviolet, visible or near infrared light to illuminate an object and it's not limited to metallic targets. It handles objects well, such as rocks, rain, human-made infrastructures and even single molecules. It is the breadth of the laser which determines the resolution of the results. LIDAR systems are used in many fields, including mapping and terrain analysis.

A LIDAR system consists of four major components[8]. The source of light, the laser; which emits the radiation used to illuminate the target. The length of the laser pulse is what decides the resolution of the target. The scanner and optics, which handle the measurements made by the system. Including azimuth and elevation. Photo detectors and receiver electronics, which sensitivity plays a big role in the quality of the results. Position and navigation systems, such as

gps and gyros, which are usually on LIDARs mounted on mobile devices which need the data, such as the planes used by Lantmäteriet.

The data to be used in our project is handed to us by Lantmäteriet, which is the Swedish mapping and land registration authority. The data originates from a LIDAR system mounted on an airplane, facing the terrain. The files consist of coordinates which meshed together gives a 3d image of the terrain analyzed by the LIDAR system. The coordinates given do not only show the latitude and longitude of the points from origo, as x and y coordinates, but also their position on a third plane, as a z coordinate. That gives us the information needed to draw objects such as buildings.

2.2 PLY

PLY(Polygon File Format or Stanford Triangle Format)[4] is a file format developed at Stanford University used to represent a three dimensional object as a list of polygon. A PLY usually contains of a header, a list of vertices and a list of faces. The header contains information how the file should be read, such as if its in ASCII or binary form. In the list of vertices the first line declares the total number of vertices. Afterwards there is a complete list where each vertex is represented as a tuple of x, y and z coordinates. Each face in the list of faces are described as indices into the list of vertices.

In addition to information about vertices and faces, a PLY file can also include properties like color and texture. Each vertex and face can be individually colored or textured by adding the information next to each line in the PLY file.

```
ply
format ascii 1.0 (ascii/binary, format version number )
comment made by Greg Turk (comments keyword specified, like all lines)
comment this file is a cube
element vertex 8 ("vertex" element, 8 of them in file)
property float x (vertex contains float "x" coordinate)
property float y (y coordinate is also a vertex property)
property float z (z coordinate, too)
element face 6 (There are 6 "face" elements in the file)
property list uchar int vertex_index ("vertex_indices" is a list of ints)
end_header (delimits the end of the header)
0 0 0 (start of vertex list)
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3 (start of face list)
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

Table 1: Example of a PLY-file

2.3 LAS

The LAS[1, 3] file type is a public format which allows the exchange of 3-dimensional point cloud data between users. Although developed primarily for exchange of LIDAR point cloud data, this format supports the exchange of any 3-dimensional x,y,z tuplet. The LAS file format comes in two versions, one consisting of binary data and an ASCII version which is used by many companies. The biggest drawbacks with the ASCII version are the size of the files, the loss of LIDAR specific data and performance problems derived from reading these big files.

The format consists of a header block, variable length records, and point data. All data is in little-endian format. The header block consists of a public block followed by variable length records. The public block contains generic data such as point numbers and coordinate bounds. The variable length records contain variable types of data including projection information, meta data, and user application data. The point data consists of data such as x, y, z coordinates, gps time and others.

A library in C/C++ designed for reading and writing to/from the LAS file format, called libLAS is publicly available under the BSD license.[2]

2.4 Blender

Blender[5] is a free and open source software developed by the Blender Foundation[6] that is used for 3D modeling, creating animated films and video games. The reason we are using Blender in this project is that its free, available on many platforms and features a 3D game engine out of the box. Blender is also able to import the PLY-format and has a big community around it that provides a lot of direct help.

3 Components needed to make it work

In order to reach our goal to render a 3D map we have to use a couple of different softwarepackages and methods. We started out with some demo LIDAR-data from Lantmäteriet over the urban area of Sandviken in Sweden. The demo data is in a binary format that cannot be imported directly into Blender so we need to export it into a different one. In order to manipulate the LIDAR-data we used a library called libLAS that provides several functions to export LIDAR-data into other formats, such as txt and obj.

3.1 Computer setups

All tests and implementations were made on this two setups.

3.1.1 Yuris PC

- CPU: A quad core Intel Core i5 CPU, running at 2.67GHz
- RAM: Four gigabytes of DDR3 SDRAM clocked at 1066 MHz
- GPU: ATI Mobility Radeon HD 5650 with 1024 mb of memory
- HDD: A hard drive running with a spin velocity of 5400 rpm
- Blender version: 64 bits release based on the 2.62 build
- OS: Windows 7 Home Premium

3.1.2 Ottos Macbook Pro

- CPU: An Intel Core 2 duo CPU
- RAM: Four gigabytes of DDR3 SDRAM clocked at 1066 MHz
- GPU: NVIDIA GeForce 9400M with 256 Mb of memory
- HDD: A SSD drive
- Blender version: 64 bits release based on the 2.62 build
- OS: Mac OSX Lion

3.2 PLY files

Our approach was to export the data as ASCII text and then add ply-headers so that Blender could import it. This was done using the libLAS tool las2txt.exe which takes a LAS-file and exports each x, y and z points and inserts them into a txt-file. In order to transform the txt-file into a ply-file we added ply-headers giving the basic PLY-information as well as the amount of coordinates and saved the file as .ply. The file could then be directly imported into Blender.

3.3 Connecting the dots

One of the major problems that we have to solve is how we should connect all the points in the data that we received from Lantmäteriet. Between some of the dots we need to render a face. A face is the area between the dots, on which you can place textures etc. The x, y, z points are aligned as a 2x2 meters grid but there is nothing in the data that tells us which dots that should have a face connecting them. Since the data is in an aligned grid we figured that we could connect them as squares. The only problem with this approach is that we need to find out a clever way to find these squares, as it seems now the data is built up as rows of points. One way of solving this could be by measuring the length of each row and then grab each point in the square.

Another problem we are facing are the limitations in the blender software combined with our computer setups. Tests done in the windows setup described in 3.1.1 on the previous page showed how the immense amount of data impacts the responsiveness of the software. Simple tasks such as selecting a few vertices becomes overwhelming to the setup in use, judging by the constant reading and writing to the hard drive it is mostly due to paging to virtual memory. This points out a certain lack of scalability in our approach. A work-around would be to process the PLY-files programmatically using the python API which Blender offers[7].

After consideration we decided to alter the data provided by Lantmäteriet and work with just a subset of it. A set of ten thousand points (a hundred times a hundred meters) were selected from the PLY-file which would become our new test file. These points were selected manually due to the nature of the original LAS-file, which exhibited a big amount of duplicated data and inconsistent readings. Out of the eight million points only five millions were unique and in several occasions different readings were given for the same gps location. By analyzing the elevation of the points close to these inconsistent readings the closest ones were selected. This would give us a somewhat consistent set of points which we could work with.

The next step was creating the faces which connects these points. The easiest approach was to do it directly in the PLY-file which handles face creation by

default. Since the points form a squared shape, faces could be easily created by connecting the closest points to each other in triangles. This was done with a simple java program which created a text file following the PLY-structure for face representation, as described in 2.2.

4 Rendering a 3D world

By using Blender and PLY-files we managed to get a pretty straight forward approach on how to solve our problem. The main parts that we used in our implementation are Blender, the LIDAR-data, libLAS, a PLY-file and a self-written script to transform the data from txt to PLY. Blender, libLAS, the LIDAR-data are all things that are provided by us from outside sources and which pretty much works out of the box. Blender provides the game and rendering engine, Lantmäteriet provides the LIDAR-data and libLAS allowed us to export the LIDAR-data into a readable txt-file. The part that we had to put a lot of effort into in order to make it work is the script that transforms the txt-file into a PLY-file and connects the points with faces.

When we first tried on the approach with the PLY-file we manually exported the points from the txt-file and then added the PLY specific header to the file. We also found out that the points didn't start at the origin, and had very big coordinates which made them hard to work with in Blender. In order to make the import automatic and solve the issue with the high coordinates we wrote a script to help us out. The script is written in the programming language Ruby which is mainly used as a scripting language, and it is very simple to work with in situations like these. What the script does is that takes a txt-file as an argument and then extracts the value of the first coordinates and use those as references. It then iterates over each row and subtracts the reference value in order to make the points start at the origin.

After importing the PLY-file into blender we now had a working 3D-model which represents a 40000 m^2 area from Sandviken, Sweden. Some adjustments were needed to facilitate the handling of this model. Since the points given by Lantmäteriet weren't perfectly aligned some of these faces became really sharp and odd looking as seen in figure 3 on page 14, this would give us difficulties when appending textures to the model. The simplest way to fix this is by using a built in function in blender called beauty fill, which cleans up the model. It does that by rearranging the structure of the long, sharp faces. Creating a smoother set which is easier to work with. Blender does that without altering the positioning of the vertices, which preserves the data.

Further work was needed with the model in order to get a smooth area. That was done by using the function called smooth vertex which rearrages the positioning of vertices in the model, creating a smoother set. This does however corrupt the data imported, specially the z-positioning of the vertices.

We decided the results outweight the costs and opted for this option. The process can be seen step by step in figure 4, 5 on page 15 and figure 6 on page 15.

5 Results

What we came up with in the end was a rendered field that represents an area of Sandviken as seen in figure 7 on page 16. We had to put a lot of work into polishing the map and remove areas that were too sharp or weird looking. Since we took a pretty much random area of the data, the final version was an area representing a field with some hills. Using Blender we were then able to add some textures to the map and use the Blender game engine to simulate a character walking around. Incorporating information from OpenStreetMap into our current version is not possible. This is because we lack the information on the readings provided from Lantmäteriet required to sync a location in the LIDAR data to a location in OpenStreetMap.

6 Discussion

When starting this project our goal was to find a way to render a map from the LIDAR data provided by Lantmäteriet without the need of tampering with the data too much by hand. We thought it would be really great if we could just import the LIDAR data into Blender and sync it with OpenStreetMap. We did however find a few issues with using LIDAR and Blender and that was mainly problems with the quality of the data and scalability in Blender.

6.1 Data

In our attempts to render our 3D maps in Blender we noticed that this method needs very good data in order to be usable. The data that we used from Lantmäteriet wasn't consistent enough and included many doublets and weird information. We figured that some of the false data may be the result of birds or other flying object that interfered with the laser or misreadings from the laser itself. The inconsistency of the data made it really difficult, if not impossible, for us to find a method to go from data to final product without editing the data by hand. Before importing the data into Blender we had to apply various scripts to make the data easier to work with and sort the dots so we could create the faces. After the import we had to manually remove sharp edges and flatten misplaced dots from the map in order to make it usable.

This proved to be a tedious work which was very time consuming. In the end it became really hard for us to work with the maps that we rendered since the data was inconsistent and that we had to do our own editing. It was difficult to know if the finished map consisted of weird looking buildings or sharp hills. If you want to render a map over a urban area such as the KTH campus we

believe that the LIDAR data needs to be a lot more detailed, meaning that the resolution is to be higher and with none or little inconsistency in the data.

The solution that we could think of for this issue is to use better LIDAR data with a properly aligned grid or to put a lot of time in reworking the data before importing it into Blender. The later would be out of the scope of this project since it would take way too much time to go through that amount of data.

Another issue with LIDAR is that its data is represented as a grid of points with no consistent readings. The data we received from Lantmäteriet had different density in the amount of measurings depending on the area. The problem with this is that if you want to render an urban area with buildings you will probably run into problems when rendering the areas where the buildings end. For example if the edge of a building is located in between two x-axis points and you draw a face between them you will end up with a diagonal face from the ground to the top of the building. There is nothing in the data that tells you when the building ends so it's hard to determine how to handle situations like these.

One possible solution to this problem could be to implement an algorithm that measures the height difference between the points before drawing the face. If the difference is, say more than five meters, the algorithm will add new points in between and draw faces with a 90 degree angle instead. This will however bring up new issues since the buildings in most cases are not aligned with the grid so you will need some advanced calculations to know where to insert the new points. The best solution would be to combine the LIDAR data with ground readings made specifically on urban areas to determine the edges of buildings. That way the elevation of such points could be easily extrapolated from the data around.

6.2 Scalability

Another issue that we found is scalability with Blender. When we first imported the data over Sandviken we noticed that Blender wasn't able to handle all the points at the same time. The data over Sandviken had around 7.5 million points which at least with our computer specs was way too high. To solve this we used a smaller portion of the data. Another approach that we could think of is to implement your own game engine and load the data programmatically. If you would load the data programmatically your could decide which portion of the information to load and only load the vertices surrounding the in game character and hide the rest. As you move the character on the map you programmatically load the new surroundings.

7 Conclusion

Using LIDAR data when rendering maps have both advantages and disadvantages. What really decides the outcome of it all is how detailed the data you have is and what kind of area you are trying to render. If you want to render an urban area with buildings and a lot of details you will need data with high granularity. You will also need some pretty advanced algorithms in order to calculate the faces that should connect the points and decide if, and how, to add new ones. If you on the other hand want to render an area with soft shapes such as a field with hills we think that a technique using LIDAR data could work out very well. The possibility to sync the data with locations from OpenStreetMap also depends on the quality of the data. As long as you know the correct longitude and latitude coordinates it would be pretty easy to withdraw useful information, but since we needed to rework our data we couldn't try this out to its full extent.

We also concluded that there are some issues with scalability based on the magnitude of the area that we were trying to render. If you want to render a region with the size of a small town you need to implement some kind of game engine that programmatically loads the data into the game.

8 Appendix

Figure 1: Example of eniro's map

Figure 2: Example of Eniro's 3d-enviroment

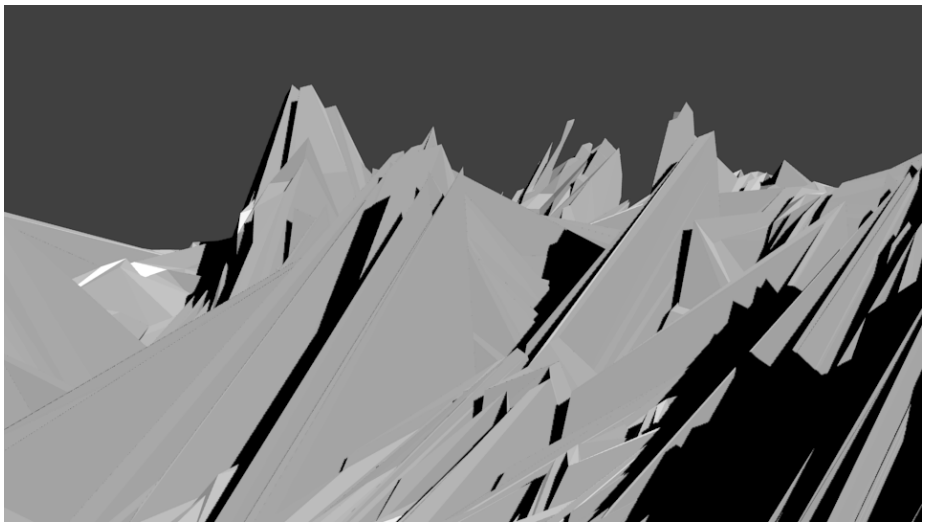

Figure 3: Rough surface

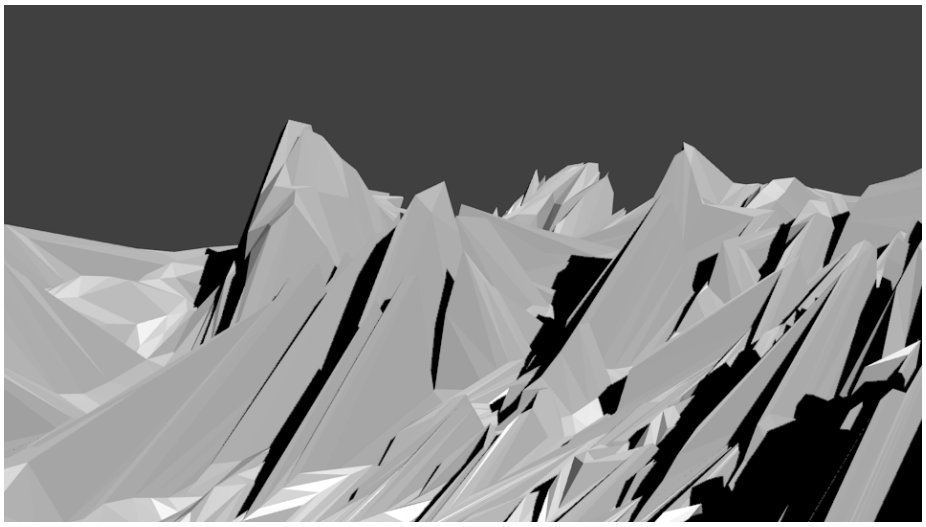

Figure 4: One step of vertex smoothing

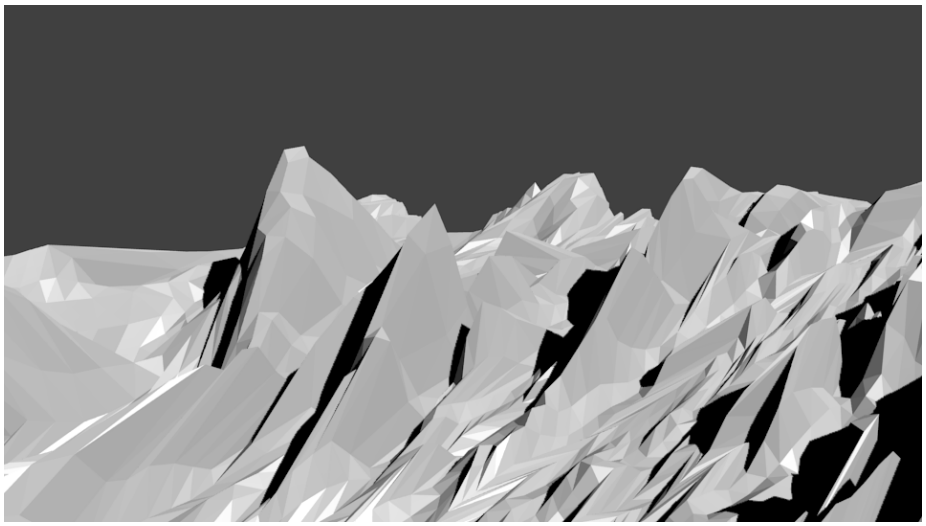

Figure 5: Two steps of vertex smoothing

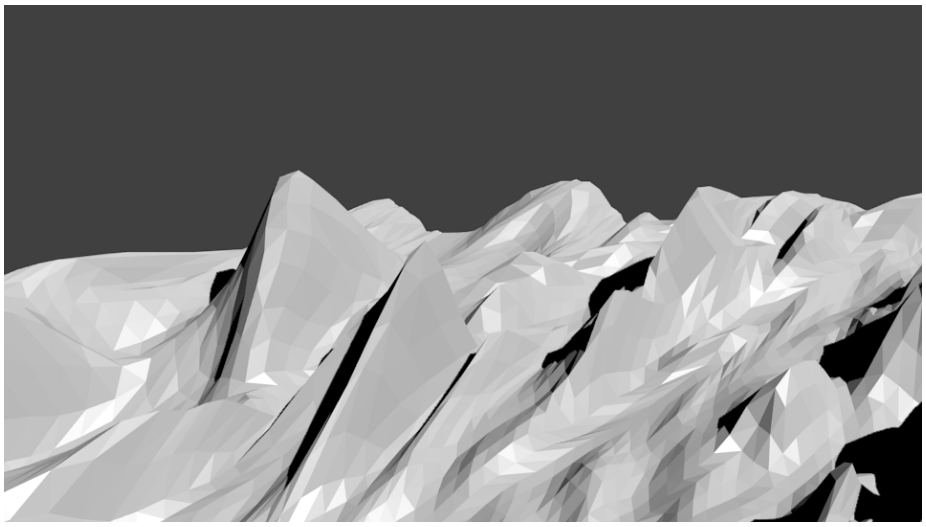

Figure 6: Three steps of vertex smoothing

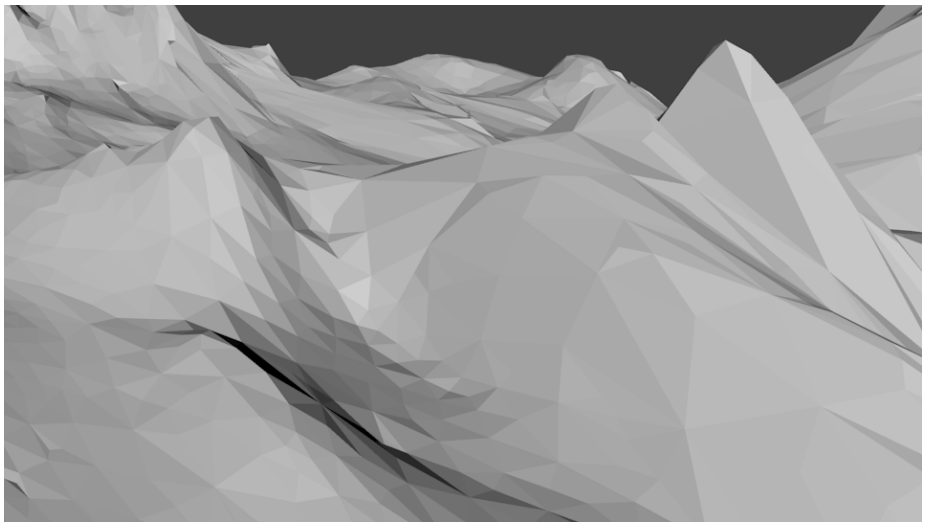

Figure 7: Rendered picture of Sandviken

References

[1] ASPRS Standards Committee
(http://www.asprs.org/a/society/committees/standards/lidar_exchange_format.html)
Retrieved 2012-03-10

[2] The libLAS LiDAR data translation toolset
(http://liblas.org/) Retrieved 2012-03-01

[3] ASPRS LIDAR Data Exchange Format Standard
(http://www.asprs.org/a/society/committees/standards/asprs_las_format_v10.pdf)
Retrieved 2012-03-11

[4] The Polygon File Format or Stanford Triangle Format
(http://local.wasp.uwa.edu.au/~pbourke/dataformats/ply/) Retrieved
2012-03-01

[5] Blender, the free open source 3D content creation suite
(http://www.blender.org/) Retrieved 2012-03-03

[6] Blender Foundation
(http://www.blender.org/blenderorg/blender-foundation/) Retrieved 2012-
03-11

[7] The Blender Python API
(http://www.blender.org/documentation/248PythonDoc/) Retrieved 2012-
03-22

[8] Cracknell, Arthur P.; Hayes, Ladson (2007) [1991]. Introduction to Remote Sensing (2 ed.). London: Taylor and Francis. ISBN 0-8493-9255-1. OCLC 70765252