WOOLPERT

Working With Lidar in Open Source and Freeware Tools

Chris Stayte

Table of Contents

Pre-Requisites

Software

This document will have several pre-requisites to complete it. The first of which are various software and tools to process, filter, translate, visualize, and query point cloud data. These tools are all either closed source free to use (freeware) or open source tools and libraries. This is in the interest of providing real world examples and procedures that can be used to produce and use point cloud data.

Materials

This document will guide you through various exercises that will require various forms of data. You should be provided the data, but in the event that you were not, most ground classified point cloud data should suffice for the examples. If you have the provided data you can place it anywhere on your computer but for the purposes of this document I will be placing it on my root drive. For me that is the **C:\** drive.

Open Source Disclaimer

This document uses examples and data that can be found on the PDAL website. For more in depth tutorials and explanation please go to https://pdal.io/.

Software

This section will break down what software we will use, why we are using it, and how to install the software. This section is important to follow so you can complete the course without fail.

Name	Version	Link
QGIS	3.8	https://qgis.org/en/site/
Visual Studio Code	1.38	https://code.visualstudio.com/
Chocolatey	N/A	https://chocolatey.org/
jq	1.5	https://stedolan.github.io/jq/
Anaconda	2019.07	http://anaconda.com
PDAL	2.0.1	https://pdal.io/
Entwine	2.1	https://entwine.io/

Version Number Noted For Possible Future Breaking Changes

Software installation will depend on other pieces of software so the order in which the tools are installed is important. For the most part, installing and using the latest version will work. If there is an issue and following the documents is not matching up with the software, re-vert back to noted versions.

This walkthrough is designed to run using 64-bit Windows 10.

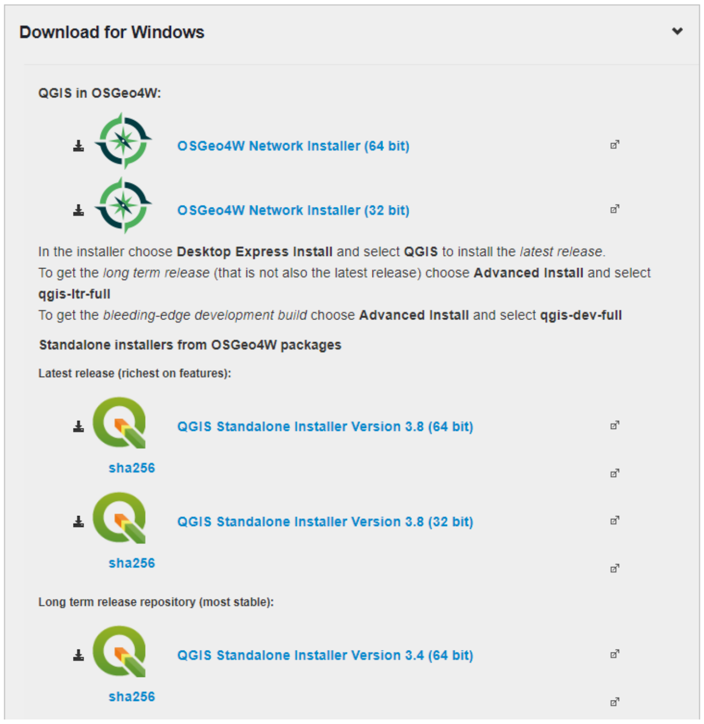
QGIS

QGIS is a user-friendly Open Source Geographic Information System (GIS) licensed under the GNU General Public License. QGIS is an official project of the Open Source Geospatial Foundation (OSGeo). It runs on Linux, Unix, macOS, Windows and Android and supports numerous vector, raster, and database formats and functionalities.

Installation

Website Link — https://qgis.org/en/site/

1. First we will need to download the software from the QGIS website.
2. Download the 64-bit. You will want the QGIS Standalone Installer. (452mb)
3. Open the installer downloaded and then click **Next ➡ I Agree ➡ Next ➡ Install** and then wait for the installer to finish. After it is complete click the **Finish** button.



Download for Windows

QGIS in OSGeo4W:

OSGeo4W Network Installer (64 bit)

OSGeo4W Network Installer (32 bit)

In the installer choose **Desktop Express Install** and select **QGIS** to install the *latest release*.
To get the *long term release* (that is not also the latest release) choose **Advanced Install** and select **qgis-ltr-full**
To get the *bleeding-edge development build* choose **Advanced Install** and select **qgis-dev-full**

Standalone installers from OSGeo4W packages

Latest release (richest on features):

QGIS Standalone Installer Version 3.8 (64 bit)
sha256

QGIS Standalone Installer Version 3.8 (32 bit)
sha256

Long term release repository (most stable):

QGIS Standalone Installer Version 3.4 (64 bit)
sha256

Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

Visual studio code will be used to edit files and work with code. Visual studio code is preferred because of the open source marketplace within the IDE as well as the perks it offers with extensions, intellisense, and customization.

Installation

Website Link— https://code.visualstudio.com/

1. The first step is to download the visual studio code installer. (51mb)
2. Once the installer has completed downloading run it and click **I Accept** ⇨ **Next** ⇨ **Next** ⇨ **Next** ⇨ **Install** and wait while the installer runs. After it completes click **Finish**.

Visual Studio should be installed and ready to use. Explore the program, check out the extension marketplace where you can find tools to assist in coding and development.

Chocolatey

The package manager for Windows - Software Management Automation

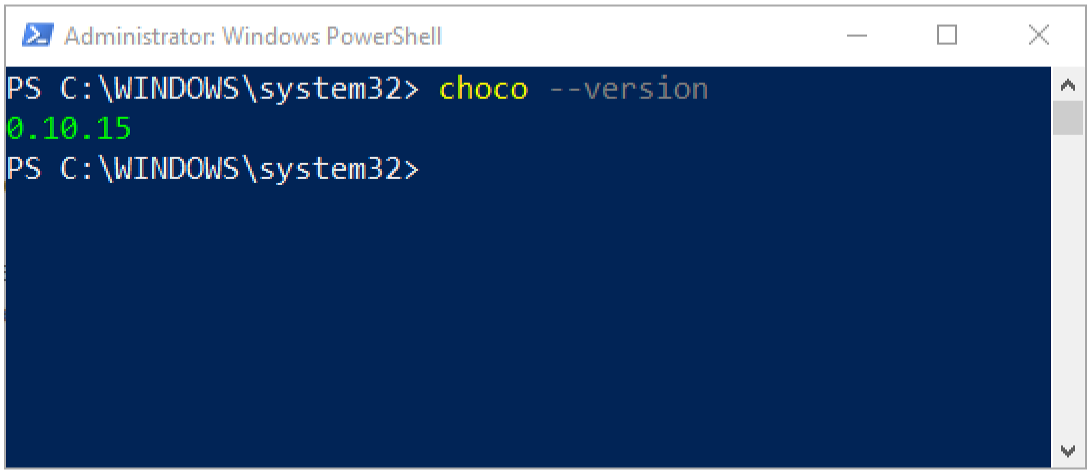
Chocolatey is a package manager for windows. Imagine packages as software applications. These applications can have a GUI (Graphical User Interface) or be run in a terminal or PowerShell. Chocolatey makes an easy unified tool to manage all applications and development environments.

The installation of Chocolatey is not a typical installation. It will be installed using the PowerShell application on your windows machine.

Installation

Website Link— https://chocolatey.org/

1. Open Windows PowerShell as an administrator.
2. Click the link above to open the Chocolatey website. After the website loads click on the 'Install Chocolatey Now' button. Scroll down on the page until you see a section labeled **Install with Powershell.exe**.
3. Paste this command into PowerShell. Click enter to run the command and wait until it finishes. After this is completed running, close PowerShell and re-open it again as administrator.
4. Type in **choco --version** into PowerShell to test if the installation ran correctly. It should look like the image below.



```
PS C:\WINDOWS\system32> choco --version
0.10.15
PS C:\WINDOWS\system32>
```

jq

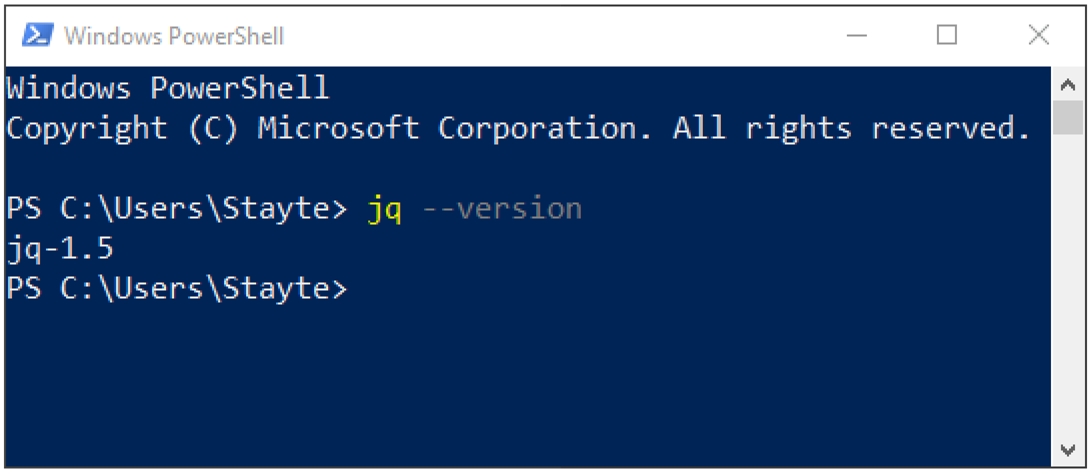
jq is a lightweight and flexible command-line JSON processor.

jq allows you to simplify the management and readability of JSON data. You can use it to slice, filter, map, and transform structured data with ease.

jq will be the first tool that we install using Chocolatey. This process will be simple and easy because of Chocolatey. Typically you must download the dll and exe files, place them on the computer, and manually path them to the computer's environment variables. This can be a convoluted, confusing and frustrating process. With one command Chocolatey makes this a very simple and streamlined process.

Installation

1. Open PowerShell as an administrator.
2. Type **choco install jq** into PowerShell and hit enter. You will encounter a part of the process where you will be asked if you want to run the script. Type **Y** and hit enter.
3. Wait until the installation is completed and then close PowerShell. We will close and re-open PowerShell to make sure the installation process is full and completed.
4. Open PowerShell (admin not required) and type **jq --version** and hit enter. If this installation was successful, you should see a version number. At the time of writing this document it will be **jq-1.5**.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Stayte> jq --version
jq-1.5
PS C:\Users\Stayte>
```

Anaconda

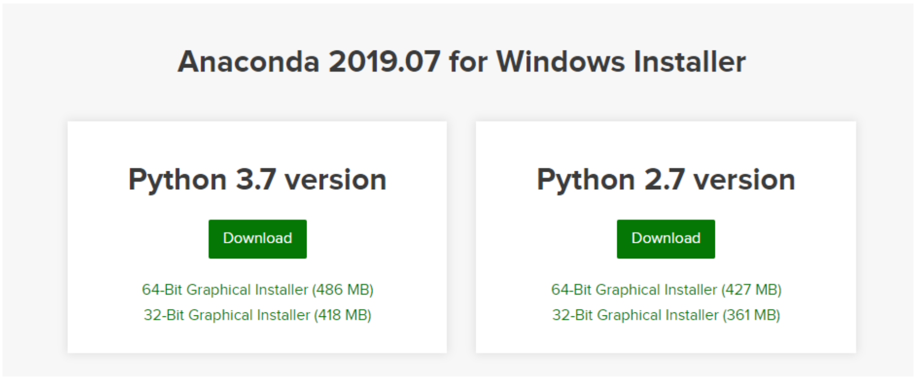
Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system Conda.

Anaconda is invaluable in setting up a development environment. Too many times as a developer or power user you are forced to use different tools and libraries that are dependent on separate environments that may clash with one another. Anaconda allows you to sandbox environments to allow easy-to-use development processes and management of tools.

Installation

Download Link - https://www.anaconda.com/distribution/

1. Download Anaconda from the Anaconda website. Choose the Windows Python 3.7 64-bit version. (486 mb)



Anaconda 2019.07 for Windows Installer

Python 3.7 version

Download

64-Bit Graphical Installer (486 MB)
32-Bit Graphical Installer (418 MB)

Python 2.7 version

Download

64-Bit Graphical Installer (427 MB)
32-Bit Graphical Installer (361 MB)

2. Once the download has completed run the installer. When the installer opens click **Next ⇒ I Agree ⇒ (All Users) Next ⇒ Next ⇒ Install ⇒ Next ⇒ Next ⇒ Finish**.
3. If the program installed correctly you should be able to open it. Go to the start menu and search for Anaconda and open up the program.

PDAL

PDAL (Point Data Abstraction Library) is a C++ BSD library for translating and manipulating point cloud data. It is very much like the GDAL library which handles raster and vector data.

PDAL is a cousin to GDAL. PDAL provides a suite of command-line applications that users can conveniently use to process, filter, translate, and query point cloud data.

We are going to use Anaconda to install PDAL. This will ensure that we get exactly what we need for PDAL to install correctly but also allow us to set up our environments that might have conflicting versions or toolsets.

Installation

Download Link—**https://pdal.io/download.html**

1. First, let's make sure that Anaconda is open. To do this search for it on Windows search bar or click on the Desktop icon.
2. Once Anaconda is open, click on the **Environments** tab.
3. Inside the environments tab, click on the **Create** button .
4. After a dialog box pops up, type in a name for the new environment. I will use **PDAL**. Make sure that you have python checked and it's set to the highest version of 3. At the time of this document the version is **3.7**. Once you have filled out everything, click on **Create**.
5. After the process is completed with creating the PDAL environment, you will need to install PDAL. Anaconda makes this process very simple. Go to the PDAL menu, click the **play** button, and click on **Open Terminal**.
6. When you read the terminal, you should see **(PDAL)** or the name you gave the environment currently in terminal. If you don't see this then you might have opened the base environment. To fix this, close out of this terminal and open up the one you created for PDAL.
7. To install PDAL you can find the command on the PDAL website for conda but you will want to type **conda install –c conda-forge pdal python-pdal gdal**. You will get prompted to accept install. Type **Y** and press enter to accept this installation.
8. Once the installation is complete, type **pdal --version** and press enter. If PDAL was installed correctly in this environment, you should get a printout of the version. As of the time of this document the version number is **pdal 2.0.1 (git-version: Release)**.

Entwine

Entwine is a data organization library for massive point clouds, designed to conquer datasets of trillions of points as well as desktop-scale point clouds. Entwine can index anything that is `PDAL`_-readable, and can read/write to a variety of sources like S3 or Dropbox. Builds are completely lossless, so no points, metadata, or precision will be discarded even for terabyte-scale datasets.

We will use Entwine to visualize las files. Entwine is not only free but it is also perfect for visualizing massive amounts of data.

Installation

Website Link — **https://entwine.io/quickstart.html**

1. First, let's make sure that Anaconda is open. To do this, search for it on windows search bar or click on the Desktop icon.
2. Once Anaconda is open, click on the **Environments** tab.
3. Inside the environments tab, click on the **Create** button .
4. After a dialog box pops up, type in a name for the new environment. I will use **Entwine**. Make sure that you have Python checked and it's set to the highest version of 3. At the time of this document the version is **3.7**. Once you have filled out everything click on **Create**.
5. After the process is completed with creating the Entwine environment you will need to install Entwine. Anaconda makes this process very simple. Go to the entwine menu, click the **play** button, and click on **Open Terminal**.
6. When you read the terminal you should see **(Entwine)** or the name you gave the environment currently in terminal. If you don't see this then you might have opened the base environment. To fix this close out of this terminal and open up the one you created for entwine.
7. To install Entwine you can find the command on the entwine website for Conda but you will want to type **conda install –c conda-forge entwine**. You will get prompted to accept install. Type **Y** and press enter to accept this installation.
8. Once the installation is complete, type **entwine --version** and press enter. If Entwine was installed correctly in this environment, you should get a printout of the version. As of the time of this document the version number is **entwine 2.1.0**.
9. Additionally you will want to run the command **conda install –c conda-forge nodejs=11.14.0 -y** and then **npm install http-server –g**. This will be used to create a server to view the Entwine databases.

Exercise 01: PDAL Info

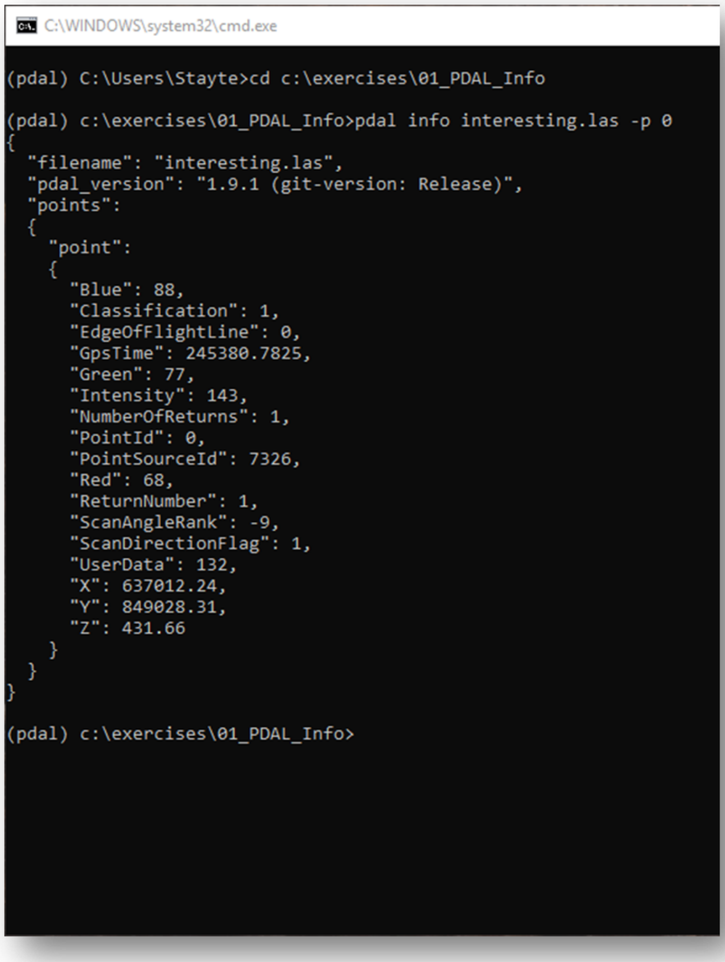
The purpose of the exercise is to use PDAL to view the information of a point cloud file. We will use PDAL to open an las header and display the information about said file.

Starting with this section, we will start to use all of the software that we have installed so far. Go ahead and open the pdal terminal.

Print the First Point

1. Using the same process as we used before, we will want to open Anaconda.
2. Go to the **Environments** tab and click on the **pdal** environment tab.
3. Click the **play** button on the pdal tab and **open in terminal**.
4. Now that the terminal is open, change the directory to exercise 01. You can use this command if you are using the provided data and place on the C:\ drive. **cd C:\exercises\01_PDAL_Info**.
5. Type in **pdal info interesting.las –p 0** and hit enter.

Congratulations! You have displayed all of the information about the first point in the las file.



```
(pdal) C:\Users\Stayte>cd c:\exercises\01_PDAL_Info

(pdal) c:\exercises\01_PDAL_Info>pdal info interesting.las -p 0
{
  "filename": "interesting.las",
  "pdal_version": "1.9.1 (git-version: Release)",
  "points":
  {
    "point":
    {
      "Blue": 88,
      "Classification": 1,
      "EdgeOfFlightLine": 0,
      "GpsTime": 245380.7825,
      "Green": 77,
      "Intensity": 143,
      "NumberOfReturns": 1,
      "PointId": 0,
      "PointSourceId": 7326,
      "Red": 68,
      "ReturnNumber": 1,
      "ScanAngleRank": -9,
      "ScanDirectionFlag": 1,
      "UserData": 132,
      "X": 637012.24,
      "Y": 849028.31,
      "Z": 431.66
    }
  }
}

(pdal) c:\exercises\01_PDAL_Info>
```

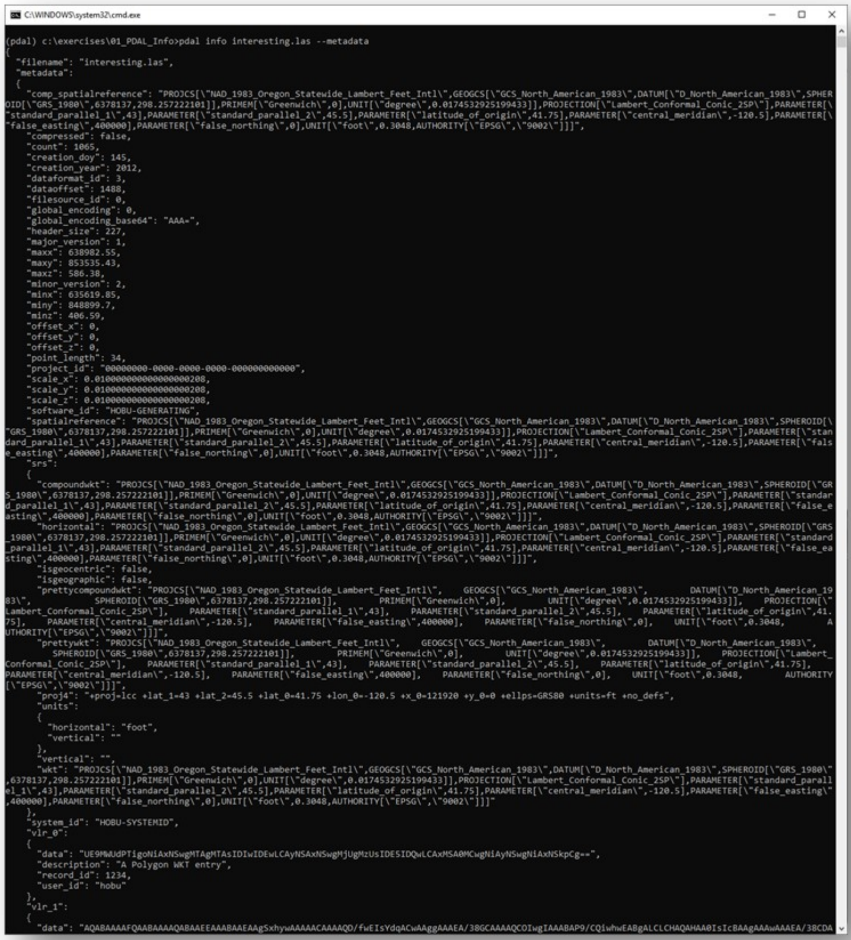
In the last step we printed the information about only one point. Now we want to view all of the information on the las file.

Printing File Metadata

1. Type **pdal info interesting.las --metadata** and press enter.

A huge blob of text will read out. You can also see this in the screenshot below. This will be all of the header information inside this las file. What you are seeing is a JSON object. This is how PDAL display's information for easier viewing, processing, and parsing. All of this JSON data looks like a jumbled mess.

You can read more about JSON data here. https://www.json.org/

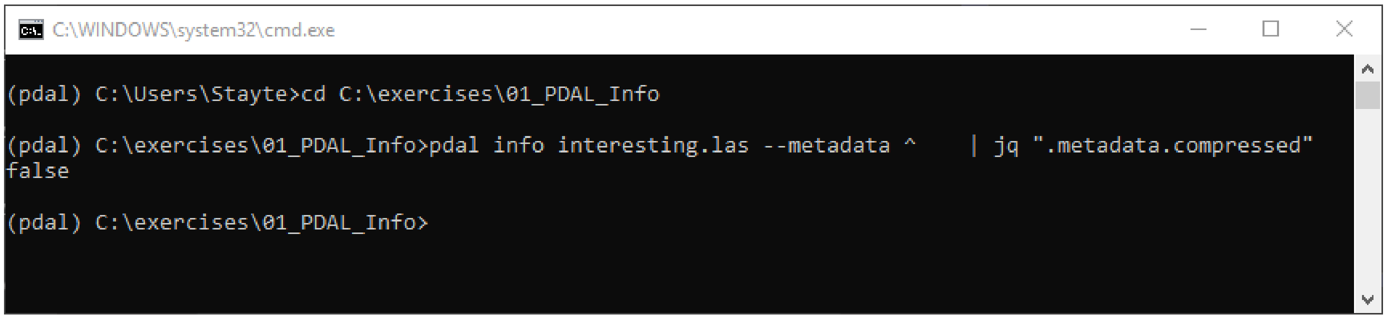

C:\WINDOWS\system32\cmd.exe

```
(pdal) c:\exercises\01_PDAL_Info>pdal info interesting.las --metadata
{
  "filename": "interesting.las",
  "metadata":
  {
    "comp_spatialreference": "PROJCS[\"NAD_1983_Oregon_Statewide_Lambert_Feet_Intl\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_American_1983\",SPHER
OID[\"GRS_1980\",6378137,298.257222101]],PRIMEM[\"Greenwich\",0],UNIT[\"degree\",0.0174532925199433]],PROJECTION[\"Lambert_Conformal_Conic_2SP\"],PARAMETER[\
"standard_parallel_1\",43],PARAMETER[\"standard_parallel_2\",45.5],PARAMETER[\"latitude_of_origin\",41.75],PARAMETER[\"central_meridian\",-120.5],PARAMETER[\
"false_easting\",400000],PARAMETER[\"false_northing\",0],UNIT[\"foot\",0.3048,AUTHORITY[\"EPSG\",\"9002\"]]]",
    "compressed": false,
    "count": 1065,
    "creation_doy": 145,
    "creation_year": 2012,
    "dataformat_id": 3,
    "dataoffset": 1488,
    "filesource_id": 0,
    "global_encoding": 0,
    "global_encoding_base64": "AAA=",
    "header_size": 227,
    "major_version": 1,
    "maxx": 638982.55,
    "maxy": 853535.43,
    "maxz": 586.38,
    "minor_version": 2,
    "minx": 635619.85,
    "miny": 848899.7,
    "minz": 406.59,
    "offset_x": 0,
    "offset_y": 0,
    "offset_z": 0,
    "point_length": 34,
    "project_id": "00000000-0000-0000-0000-000000000000",
    "scale_x": 0.01000000000000000208,
    "scale_y": 0.01000000000000000208,
    "scale_z": 0.01000000000000000208,
    "software_id": "HOBU-GENERATING",
    "spatialreference": "PROJCS[\"NAD_1983_Oregon_Statewide_Lambert_Feet_Intl\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_American_1983\",SPHEROID[\
"GRS_1980\",6378137,298.257222101]],PRIMEM[\"Greenwich\",0],UNIT[\"degree\",0.0174532925199433]],PROJECTION[\"Lambert_Conformal_Conic_2SP\"],PARAMETER[\"stan
dard_parallel_1\",43],PARAMETER[\"standard_parallel_2\",45.5],PARAMETER[\"latitude_of_origin\",41.75],PARAMETER[\"central_meridian\",-120.5],PARAMETER[\"fals
e_easting\",400000],PARAMETER[\"false_northing\",0],UNIT[\"foot\",0.3048,AUTHORITY[\"EPSG\",\"9002\"]]]",
    "srs":
    {
      "compoundwkt": "PROJCS[\"NAD_1983_Oregon_Statewide_Lambert_Feet_Intl\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_American_1983\",SPHEROID[\"GR
S_1980\",6378137,298.257222101]],PRIMEM[\"Greenwich\",0],UNIT[\"degree\",0.0174532925199433]],PROJECTION[\"Lambert_Conformal_Conic_2SP\"],PARAMETER[\"standar
d_parallel_1\",43],PARAMETER[\"standard_parallel_2\",45.5],PARAMETER[\"latitude_of_origin\",41.75],PARAMETER[\"central_meridian\",-120.5],PARAMETER[\"false_e
asting\",400000],PARAMETER[\"false_northing\",0],UNIT[\"foot\",0.3048,AUTHORITY[\"EPSG\",\"9002\"]]]",
      "horizontal": "PROJCS[\"NAD_1983_Oregon_Statewide_Lambert_Feet_Intl\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_American_1983\",SPHEROID[\"GRS
_1980\",6378137,298.257222101]],PRIMEM[\"Greenwich\",0],UNIT[\"degree\",0.0174532925199433]],PROJECTION[\"Lambert_Conformal_Conic_2SP\"],PARAMETER[\"standard
_parallel_1\",43],PARAMETER[\"standard_parallel_2\",45.5],PARAMETER[\"latitude_of_origin\",41.75],PARAMETER[\"central_meridian\",-120.5],PARAMETER[\"false_ea
sting\",400000],PARAMETER[\"false_northing\",0],UNIT[\"foot\",0.3048,AUTHORITY[\"EPSG\",\"9002\"]]]",
      "isgeocentric": false,
      "isgeographic": false,
      "prettycompoundwkt": "PROJCS[\"NAD_1983_Oregon_Statewide_Lambert_Feet_Intl\",    GEOGCS[\"GCS_North_American_1983\",        DATUM[\"D_North_American_19
83\",            SPHEROID[\"GRS_1980\",6378137,298.257222101]],        PRIMEM[\"Greenwich\",0],        UNIT[\"degree\",0.0174532925199433]],    PROJECTION[\"
Lambert_Conformal_Conic_2SP\"],    PARAMETER[\"standard_parallel_1\",43],    PARAMETER[\"standard_parallel_2\",45.5],    PARAMETER[\"latitude_of_origin\",41.
75],    PARAMETER[\"central_meridian\",-120.5],    PARAMETER[\"false_easting\",400000],    PARAMETER[\"false_northing\",0],    UNIT[\"foot\",0.3048,        A
UTHORITY[\"EPSG\",\"9002\"]]]",
      "prettywkt": "PROJCS[\"NAD_1983_Oregon_Statewide_Lambert_Feet_Intl\",    GEOGCS[\"GCS_North_American_1983\",        DATUM[\"D_North_American_1983\",
          SPHEROID[\"GRS_1980\",6378137,298.257222101]],        PRIMEM[\"Greenwich\",0],        UNIT[\"degree\",0.0174532925199433]],    PROJECTION[\"Lambert_
Conformal_Conic_2SP\"],    PARAMETER[\"standard_parallel_1\",43],    PARAMETER[\"standard_parallel_2\",45.5],    PARAMETER[\"latitude_of_origin\",41.75],
PARAMETER[\"central_meridian\",-120.5],    PARAMETER[\"false_easting\",400000],    PARAMETER[\"false_northing\",0],    UNIT[\"foot\",0.3048,        AUTHORITY
[\"EPSG\",\"9002\"]]]",
      "proj4": "+proj=lcc +lat_1=43 +lat_2=45.5 +lat_0=41.75 +lon_0=-120.5 +x_0=121920 +y_0=0 +ellps=GRS80 +units=ft +no_defs",
      "units":
      {
        "horizontal": "foot",
        "vertical": ""
      },
      "vertical": "",
      "wkt": "PROJCS[\"NAD_1983_Oregon_Statewide_Lambert_Feet_Intl\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_American_1983\",SPHEROID[\"GRS_1980\"
,6378137,298.257222101]],PRIMEM[\"Greenwich\",0],UNIT[\"degree\",0.0174532925199433]],PROJECTION[\"Lambert_Conformal_Conic_2SP\"],PARAMETER[\"standard_parall
el_1\",43],PARAMETER[\"standard_parallel_2\",45.5],PARAMETER[\"latitude_of_origin\",41.75],PARAMETER[\"central_meridian\",-120.5],PARAMETER[\"false_easting\"
,400000],PARAMETER[\"false_northing\",0],UNIT[\"foot\",0.3048,AUTHORITY[\"EPSG\",\"9002\"]]]"
    },
    "system_id": "HOBU-SYSTEMID",
    "vlr_0":
    {
      "data": "UE9MWUdPTigoNiAxNSwgMTAgMTAsIDIwIDEwLCAyNSAxNSwgMjUgMzUsIDE5IDQwLCAxMSA0MCwgNiAyNSwgNiAxNSkpCg==",
      "description": "A Polygon WKT entry",
      "record_id": 1234,
      "user_id": "hobu"
    },
    "vlr_1":
    {
      "data": "AQABAAAAFQAABAAAAQABAAEEAAABAAEAAgSxhywAAAAACAAAAQD/fwEIsYdqACwAAggAAAEA/38GCAAAAQCOIwgIAAABAP9/CQiwhwEABgALCLCHAQAHAA0IsIcBAAgAAAwAAAEA/38CDA
```

Filtering Out the Results

We have all of this metadata but to find anything useful or specific it might take some time to read through it all. For example, if you wanted to know if the file is compressed you can find this in the header file which might be time consuming to find unless we get it with filtering. Earlier we installed jq which is a JSON parser, meaning that we can filter out what we don't want to see and only get exactly what we want.

1. Type in the command **pdal info interesting.las --metadata ^ | jq ".metadata.compressed"**. Note how there are four spaces after the caret. This is important for it to work.



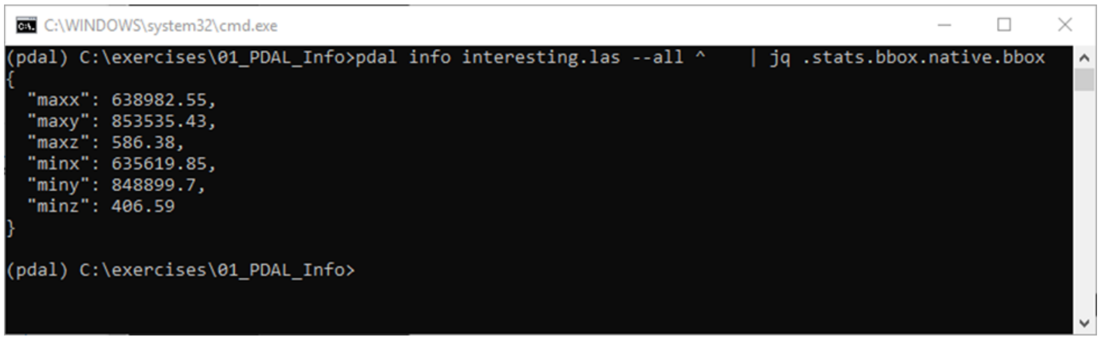
```
(pdal) C:\Users\Stayte>cd C:\exercises\01_PDAL_Info

(pdal) C:\exercises\01_PDAL_Info>pdal info interesting.las --metadata ^    | jq ".metadata.compressed"
false

(pdal) C:\exercises\01_PDAL_Info>
```

Above you can see that this particular file is not compressed.

Searching Near a Point

Here we will use PDAL to find points near a given search location. Our scenario is a simple one: we want to find the two points nearest to the midpoint of the bounding cube of our **interesting.las** data file. We need to find the midpoint of the bounding cube. To do that, we need to print the **--all** info for the file and look for the bbox output.

1. Type in the command **pdal info interesting.las --all ^ | jq .stats.bbox.native.bbox**. This will get the bbox for us. Note how there are four spaces after the caret. This is important for it to work.



```
(pdal) C:\exercises\01_PDAL_Info>pdal info interesting.las --all ^    | jq .stats.bbox.native.bbox
{
  "maxx": 638982.55,
  "maxy": 853535.43,
  "maxz": 586.38,
  "minx": 635619.85,
  "miny": 848899.7,
  "minz": 406.59
}

(pdal) C:\exercises\01_PDAL_Info>
```

Now that we have the bbox we will want to find the average of the X, Y, and Z values. This is how we will get the center point of the bbox.

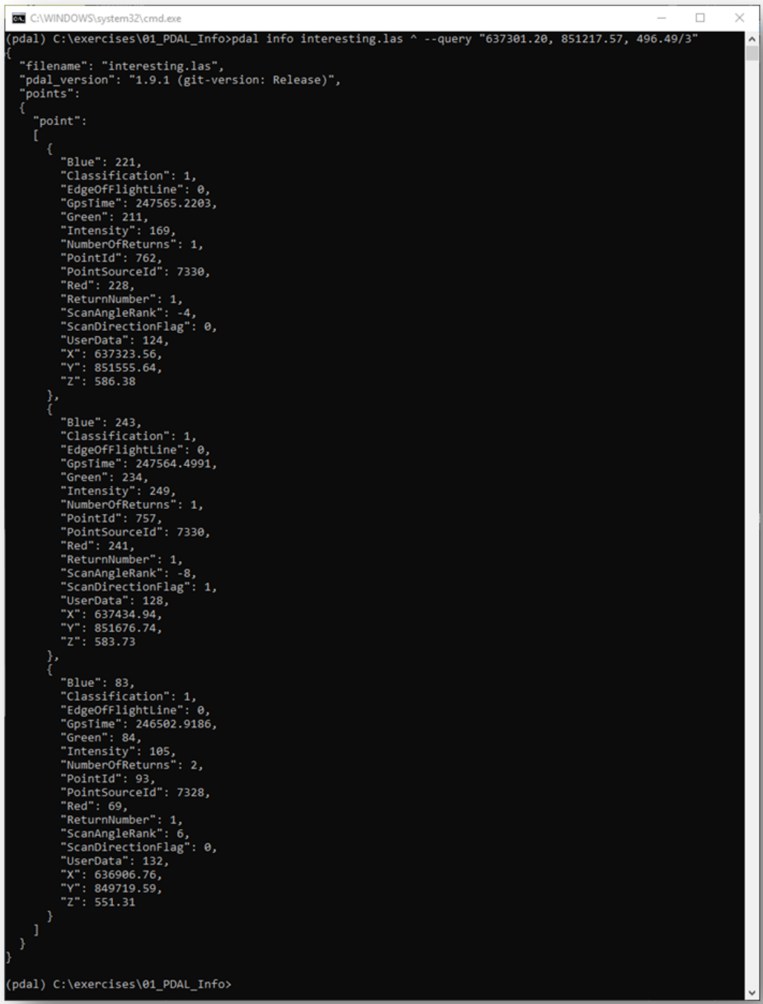
x = 635619.85 + (638982.55 - 635619.85)/2 = 637301.20

y = 848899.70 + (853535.43 - 848899.70)/2 = 851217.57

z = 406.59 + (586.38 - 406.59)/2 = 496.49

With our “center point” that we just created we will want to use the **--query** option to **pdal info** and return the three nearest points to it.

2. Use the command **pdal info interesting.las ^ query “637301.20, 851217.57, 496.49/3”** this will print out the three points and their associated information. We are not using jq here so the four spaces after the caret is not required.



```
C:\WINDOWS\system32\cmd.exe
(pdal) C:\exercises\01_PDAL_Info>pdal info interesting.las ^ --query "637301.20, 851217.57, 496.49/3"
{
  "filename": "interesting.las",
  "pdal_version": "1.9.1 (git-version: Release)",
  "points":
  {
    "point":
    [
      {
        "Blue": 221,
        "Classification": 1,
        "EdgeOfFlightLine": 0,
        "GpsTime": 247565.2203,
        "Green": 211,
        "Intensity": 169,
        "NumberOfReturns": 1,
        "PointId": 762,
        "PointSourceId": 7330,
        "Red": 228,
        "ReturnNumber": 1,
        "ScanAngleRank": -4,
        "ScanDirectionFlag": 0,
        "UserData": 124,
        "X": 637323.56,
        "Y": 851555.64,
        "Z": 586.38
      },
      {
        "Blue": 243,
        "Classification": 1,
        "EdgeOfFlightLine": 0,
        "GpsTime": 247564.4991,
        "Green": 234,
        "Intensity": 249,
        "NumberOfReturns": 1,
        "PointId": 757,
        "PointSourceId": 7330,
        "Red": 241,
        "ReturnNumber": 1,
        "ScanAngleRank": -8,
        "ScanDirectionFlag": 1,
        "UserData": 128,
        "X": 637434.94,
        "Y": 851676.74,
        "Z": 583.73
      },
      {
        "Blue": 83,
        "Classification": 1,
        "EdgeOfFlightLine": 0,
        "GpsTime": 246502.9186,
        "Green": 84,
        "Intensity": 105,
        "NumberOfReturns": 2,
        "PointId": 93,
        "PointSourceId": 7328,
        "Red": 69,
        "ReturnNumber": 1,
        "ScanAngleRank": 6,
        "ScanDirectionFlag": 0,
        "UserData": 132,
        "X": 636906.76,
        "Y": 849719.59,
        "Z": 551.31
      }
    ]
  }
}

(pdal) C:\exercises\01_PDAL_Info>
```

Exercise 02: PDAL Translation

The purpose of the exercise is to use pdal to compress ASPRS LAS data into LASzip, as well as transform the data from one projection to another.

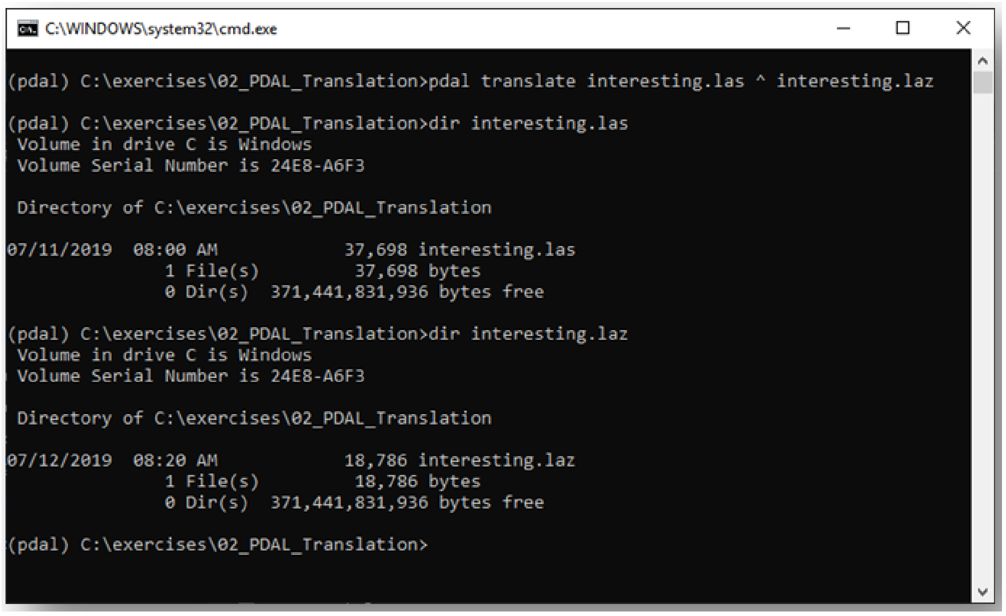
For this exercise we will need to change our directory.

Use the command **cd c:\exercises\02_PDAL_Translation**. This will put our terminal into the folder for our second exercise.

Compression

1. Use the command **pdal translate interesting.las ^ interesting.laz**. This will compress the las data to laz format.

2. Now we want to verify that the data is in fact compressed. We can do this simply by executing two commands. The first command is **dir interesting.las**, this will give us the information on the las file. The second command is **dir interesting.laz**, which will give us the information on the laz file.



```
C:\WINDOWS\system32\cmd.exe

(pdal) C:\exercises\02_PDAL_Translation>pdal translate interesting.las ^ interesting.laz

(pdal) C:\exercises\02_PDAL_Translation>dir interesting.las
 Volume in drive C is Windows
 Volume Serial Number is 24E8-A6F3

 Directory of C:\exercises\02_PDAL_Translation

07/11/2019  08:00 AM            37,698 interesting.las
               1 File(s)         37,698 bytes
               0 Dir(s)  371,441,831,936 bytes free

(pdal) C:\exercises\02_PDAL_Translation>dir interesting.laz
 Volume in drive C is Windows
 Volume Serial Number is 24E8-A6F3

 Directory of C:\exercises\02_PDAL_Translation

07/12/2019  08:20 AM            18,786 interesting.laz
               1 File(s)         18,786 bytes
               0 Dir(s)  371,441,831,936 bytes free

(pdal) C:\exercises\02_PDAL_Translation>
```

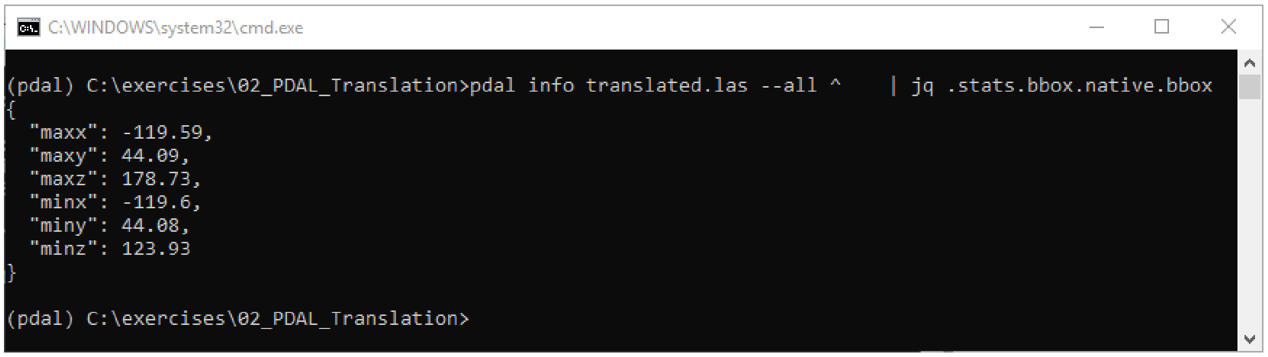
You can see here that the size of the file in the LAZ is truly compressed. Typical LASzip compression is 5:1 to 8:1, depending on the type of LiDAR. It is a compression format specifically for the ASPRS LAS model, however it will not be efficient for other types of point cloud data.

Reprojections

We will now use pdal to reproject ASPRS LAS data. The current projection is **NAD_1983_Oregon_Statewide_Lambert_Feet_Intl** but we want to project the data to **EPSG: 4326**

1. Use the command **pdal translate interesting.las ^ translated.las ^ reprojection ^ --filters.reprojection.out_srs="EPSG:4326"**

2. Let's take a look at the bbox of the file to see if we have re-projected that data. Use the command **pdal info translated.las --all ^ | jq .stats.bbox.native.bbox**. You can see the results below.



C:\WINDOWS\system32\cmd.exe

```
(pdal) C:\exercises\02_PDAL_Translation>pdal info translated.las --all ^    | jq .stats.bbox.native.bbox
{
  "maxx": -119.59,
  "maxy": 44.09,
  "maxz": 178.73,
  "minx": -119.6,
  "miny": 44.08,
  "minz": 123.93
}

(pdal) C:\exercises\02_PDAL_Translation>
```

The image above shows us the data but it's not correct. **--all** dumps all information about the file and we then use the jq command to extract out the "native" (same coordinate system as the file it-self) bounding box. As we can see, the problem is we only have two decimal places of precision on the bounding box. We are converting to a geographic coordinate system, this is not enough preci-sion.

Some formats, like **writers.las** do not automatically set scaling information. PDAL cannot really do this for you because there are a number of ways for this to be tripped up. For latitude/longitude data, you will need to set the scale to smaller values like 0.0000001. Additionally, las uses an offset value to move the origin value. Use PDAL to set that to auto so you don't have to compute it.

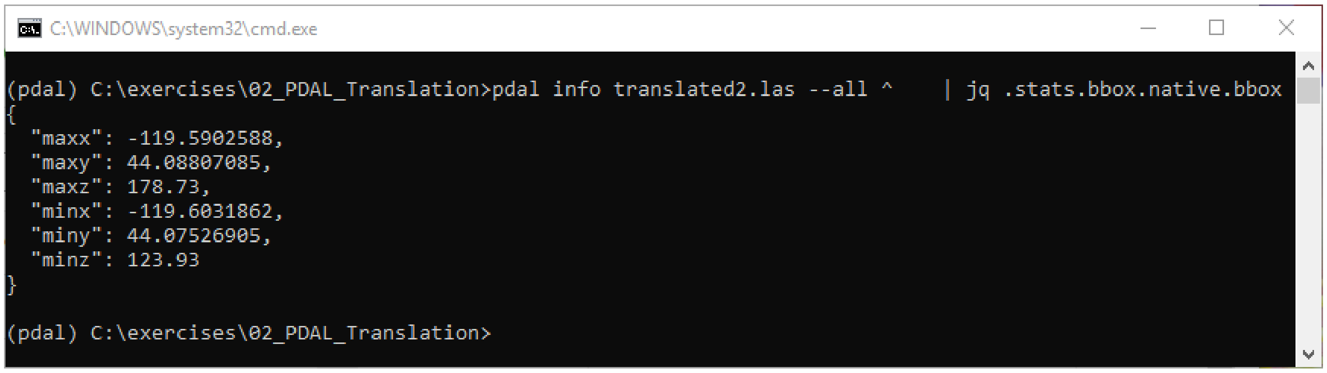
3. The command below we will use to perform the transformation correctly. Let type this into our terminal or copy/paste and see what we get.

pdal translate ^ interesting.las ^ translated2.las ^ reprojection ^ --filters.reprojection.out_srs="EPSG:4326" ^ --writers.las.scale_x=0.0000001 ^ --writers.las.scale_y=0.0000001 ^ --writers.las.offset_x="auto" ^ --writers.las.offset_y="auto"

4. To see if this transformation ran keeping accurate precision how we expected it to, we can run the info command again to verify the X, Y, and Z dimensions.

pdal info translated2.las --all ^ | jq .stats.bbox.native.bbox

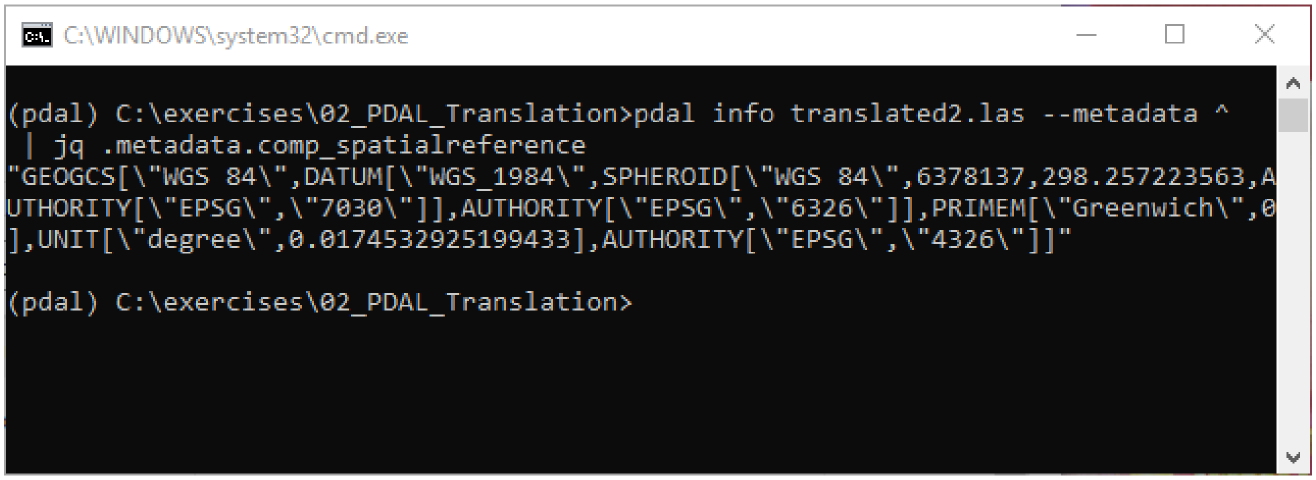
This command is the same as before except we are now looking at our new and improved processed file.



```
(pdal) C:\exercises\02_PDAL_Translation>pdal info translated2.las --all ^    | jq .stats.bbox.native.bbox
{
  "maxx": -119.5902588,
  "maxy": 44.08807085,
  "maxz": 178.73,
  "minx": -119.6031862,
  "miny": 44.07526905,
  "minz": 123.93
}

(pdal) C:\exercises\02_PDAL_Translation>
```

You can see in now that our data has the correct precision and has been transformed correctly.

5. We can type in a command to check the spatial reference. Use **pdal info translated2.las --metadata ^ | jq .metadata.comp_spatialreference**. This will print out the file's spatial reference.



```
(pdal) C:\exercises\02_PDAL_Translation>pdal info translated2.las --metadata ^
 | jq .metadata.comp_spatialreference
"GEOGCS[\"WGS 84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS 84\",6378137,298.257223563,A
UTHORITY[\"EPSG\",\"7030\"]],AUTHORITY[\"EPSG\",\"6326\"]],PRIMEM[\"Greenwich\",0
],UNIT[\"degree\",0.0174532925199433],AUTHORITY[\"EPSG\",\"4326\"]]"

(pdal) C:\exercises\02_PDAL_Translation>
```

You can see that this changed the projection to **EPSG 4326**.

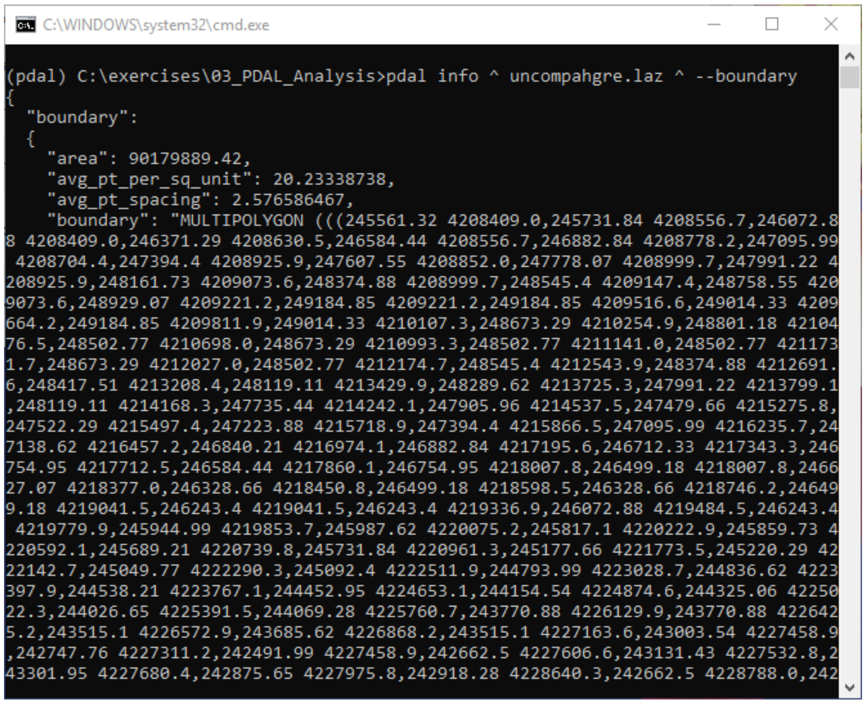
Exercise 03: PDAL Analysis

This exercise uses PDAL to find a tight-fitting boundary of an aerial scan. Printing the coordinates of the boundary for the file is quite simple using a single PDAL info call, but visualizing the boundary is more complicated.

First we need to change the directory. Use the command **cd C:\exercises\03_PDAL_Analysis**. This will put our terminal into the folder for our third exercise.

Finding the Boundary

1. We can export the boundary of an las file using PDAL. This can be used in other software to perform more analysis. The command to get a boundary is **pdal info ^ uncompahgre.laz ^ --boundary**.



C:\WINDOWS\system32\cmd.exe

```
(pdal) C:\exercises\03_PDAL_Analysis>pdal info ^ uncompahgre.laz ^ --boundary
{
  "boundary":
  {
    "area": 90179889.42,
    "avg_pt_per_sq_unit": 20.23338738,
    "avg_pt_spacing": 2.576586467,
    "boundary": "MULTIPOLYGON (((245561.32 4208409.0,245731.84 4208556.7,246072.8
8 4208409.0,246371.29 4208630.5,246584.44 4208556.7,246882.84 4208778.2,247095.99
 4208704.4,247394.4 4208925.9,247607.55 4208852.0,247778.07 4208999.7,247991.22 4
208925.9,248161.73 4209073.6,248374.88 4208999.7,248545.4 4209147.4,248758.55 420
9073.6,248929.07 4209221.2,249184.85 4209221.2,249184.85 4209516.6,249014.33 4209
664.2,249184.85 4209811.9,249014.33 4210107.3,248673.29 4210254.9,248801.18 42104
76.5,248502.77 4210698.0,248673.29 4210993.3,248502.77 4211141.0,248502.77 421173
1.7,248673.29 4212027.0,248502.77 4212174.7,248545.4 4212543.9,248374.88 4212691.
6,248417.51 4213208.4,248119.11 4213429.9,248289.62 4213725.3,247991.22 4213799.1
,248119.11 4214168.3,247735.44 4214242.1,247905.96 4214537.5,247479.66 4215275.8,
247522.29 4215497.4,247223.88 4215718.9,247394.4 4215866.5,247095.99 4216235.7,24
7138.62 4216457.2,246840.21 4216974.1,246882.84 4217195.6,246712.33 4217343.3,246
754.95 4217712.5,246584.44 4217860.1,246754.95 4218007.8,246499.18 4218007.8,2466
27.07 4218377.0,246328.66 4218450.8,246499.18 4218598.5,246328.66 4218746.2,24649
9.18 4219041.5,246243.4 4219041.5,246243.4 4219336.9,246072.88 4219484.5,246243.4
 4219779.9,245944.99 4219853.7,245987.62 4220075.2,245817.1 4220222.9,245859.73 4
220592.1,245689.21 4220739.8,245731.84 4220961.3,245177.66 4221773.5,245220.29 42
22142.7,245049.77 4222290.3,245092.4 4222511.9,244793.99 4223028.7,244836.62 4223
397.9,244538.21 4223767.1,244452.95 4224653.1,244154.54 4224874.6,244325.06 42250
22.3,244026.65 4225391.5,244069.28 4225760.7,243770.88 4226129.9,243770.88 422642
5.2,243515.1 4226572.9,243685.62 4226868.2,243515.1 4227163.6,243003.54 4227458.9
,242747.76 4227311.2,242491.99 4227458.9,242662.5 4227606.6,243131.43 4227532.8,2
43301.95 4227680.4,242875.65 4227975.8,242918.28 4228640.3,242662.5 4228788.0,242
```

You can see that this command outputs a lot of information, none of which is entirely useful. This is called a geoJSON. A geoJSON is much like a JSON file, the only difference being is that it is structured where geospatial information is stored in a format that has been set as a standard.

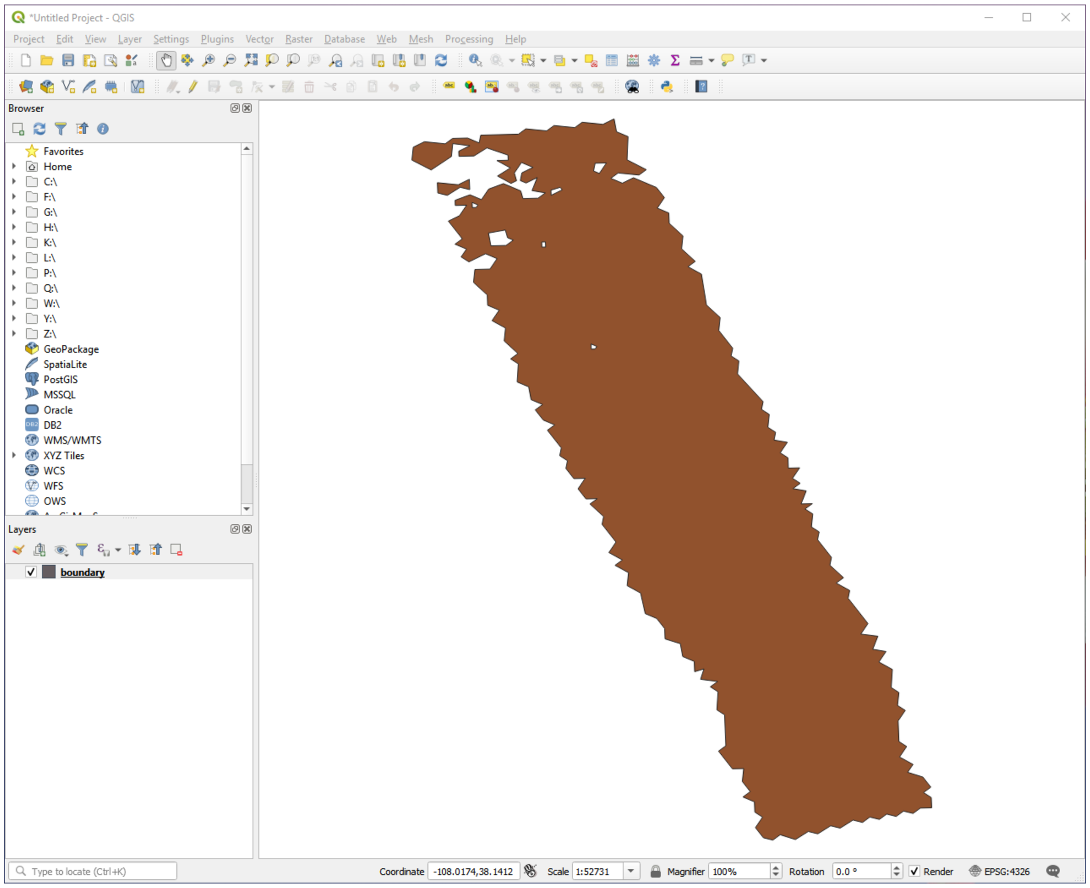
Instead of this format, we are going to export the boundary in a format that we can visualize using QGIS.

2. Use the command `pdal tindex create --tindex boundary.shp --filespec uncompahgre.laz`

This command should have exported the data into a SQLite database. We can visualize this data by using QGIS.

3. Use the windows search bar or the desktop icon to open **QGIS**.
4. Once **QGIS** is open click on **Layer** at the top.
5. Under **Layer** click on **Add Layer**.
6. Under **Add Layer** click on **Add Vector Layer**.
7. After the dialog box opens go to the source area and click on the **ellipsis (…)**.
8. Browse to the exercise 3 folder and open the **boundary.shp** file.
9. Click **add** on the dialog box then click **close**.

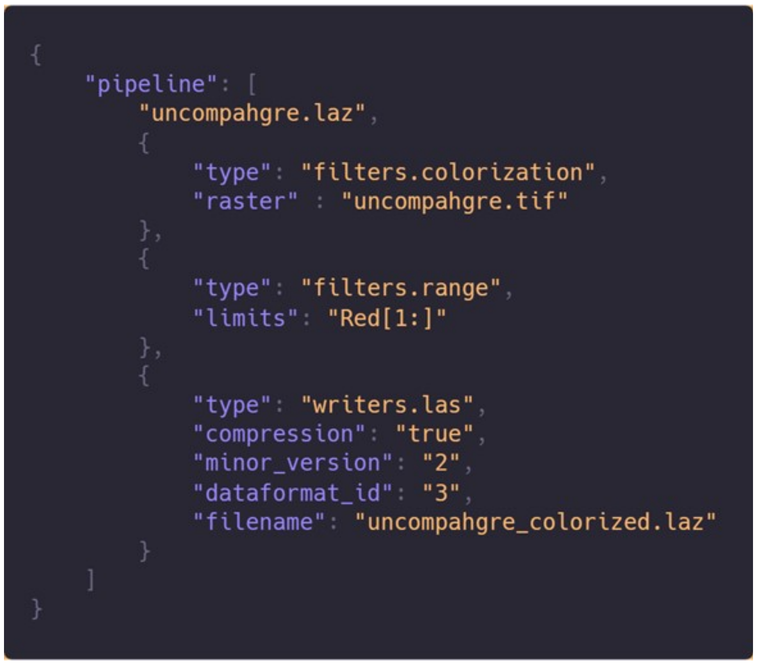
You can now see the boundary in QGIS that we created for our las file.



Colorizing Point with Imagery

The uncompahgre.laz when visualized can be viewed by elevation, return number, intensity, etc. What it does not have yet is a colorized version. We can take an un-colorized point cloud and add a red, green, blue value to each point. We will be using pdal, visual studio code and entwine for this part. We will be creating something called a pipeline. This is a syntactic sugar way for pdal to execute many commands in a sequence. This can also be chained together to batch process large workloads. Open the las file in http://plas.io to view this current las file.

1. Open **Visual Studio Code** by searching for it or clicking the icon on the desktop.
2. Create a file called **colorize.json** and save it into our folder containing exercise 03.
3. This is where we will create our pipeline. Use the image here to create the correct pipeline.



```
{
    "pipeline": [
        "uncompahgre.laz",
        {
            "type": "filters.colorization",
            "raster" : "uncompahgre.tif"
        },
        {
            "type": "filters.range",
            "limits": "Red[1:]"
        },
        {
            "type": "writers.las",
            "compression": "true",
            "minor_version": "2",
            "dataformat_id": "3",
            "filename": "uncompahgre_colorized.laz"
        }
    ]
}
```

Pipeline Breakdown

Reader

The first thing in our pipeline is telling pdal what we will work with (uncompahgre.laz).

Filters.colorization

The filters.colorization pdal filter does most of the work for this operation. We're going to use the default data scaling options. This filter wil create pdal dimension in Red, Green, and Blue.

Filters.range

A small challenge is the raster will colorize many points with NODATA values. We are going to use the filters.range to keep any points that have Red>=1.

Writers.las

We could just define the uncompahgre_colored.laz filename, but we want to add a few options to have finer control over what is written.

4. After you have created your pipeline file save it, return to the pdal terminal, and run **pdal pipeline colorize.json**.

Visualize LAS Data

Now that you have ran your pipeline you should have a new laz file that was outputted. Let’s use entwine to view this data to make sure that the colorization is complete.

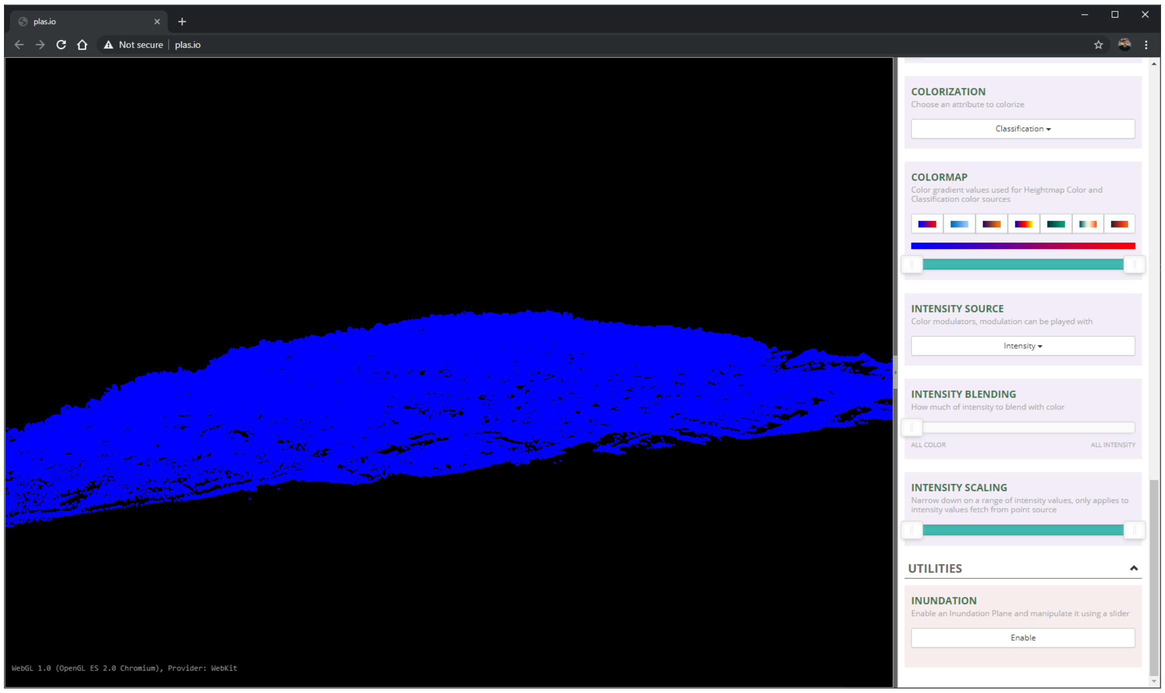
1. First we need to place our file inside of a folder. Go to your exercise 03 folder and create a folder called **colorized**.
2. Move the **uncompahgre_colorized.laz** file into this folder.
3. Open or return to the terminal that you used for **entwine**.
4. Change your directory to **cd c:\exercises\03_PDAL_Analysis**.
5. Run **entwine build –i colorized –o colorized**.
6. Now that we have created our entwine workspace or ept file we can start hosting it.
7. Run **cd ..** to go up a directory.
8. Run **http-server 03_PDAL_Analysis –p 8080** to start the server.
9. In your browser go to **http://potree.entwine.io/data/view.html?r=%22http://localhost:8080/colorized%22**

You should now be looking at a colorized point cloud that you not only colorized but hosted a server for.

Ground Classify LAS Data

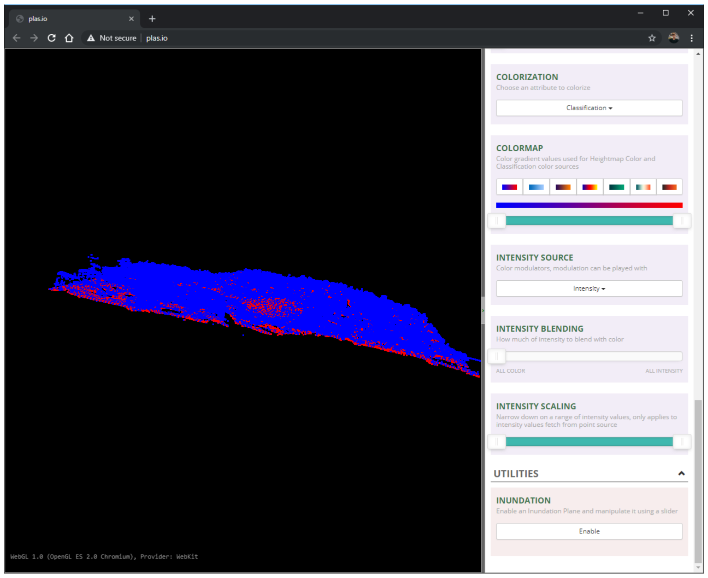
Here we will use pdal to classify ground returns using the Simple Morphological Filter (SMRF) technique.

1. Open http://plas.io and open the csite_original.laz.
2. Change the colorization to classification.


plas.io
Not secure | plas.io
COLORIZATION
Choose an attribute to colorize
Classification
COLORMAP
Color gradient values used for Heightmap Color and Classification color sources
INTENSITY SOURCE
Color modulators, modulation can be played with
Intensity
INTENSITY BLENDING
How much of intensity to blend with color
ALL COLOR
ALL INTENSITY
INTENSITY SCALING
Narrow down on a range of intensity values, only applies to intensity values fetch from point source
UTILITIES
INUNDATION
Enable an Inundation Plane and manipulate it using a slider
Enable
WebGL 1.0 (OpenGL ES 2.0 Chromium), Provider: WebKit

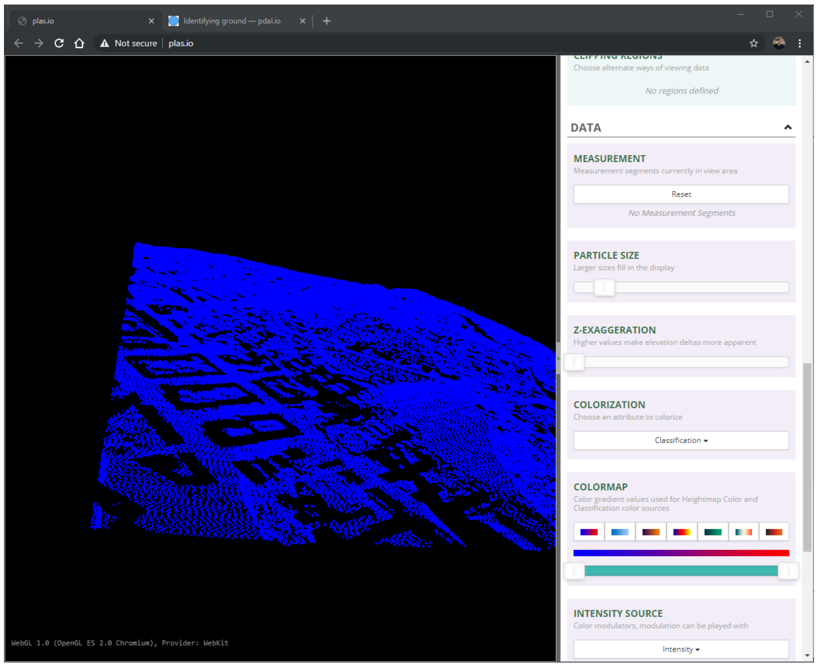
You can here that the entire las dataset has one classification. Before we can create surfaces or do any real analysis work, we need to first classify the ground. pdal can do that for us.

3. Open the **pdal** terminal in anaconda.
4. Change your directory to **cd c:\exercises\03_PDAL_Analysis**.
5. Run **pdal translate csite_original.laz –o csite_ground.laz smrf –v 4**. This command will attempt to ground classify the lidar. This will create a new file.
6. Let's open the file inside of **plas.io**. You can see that this did a good job at classifying the ground but there is still noise below the ground. This will not produce the desired results when creating surfaces. Let's clean this up a bit.


plas.io
Not secure | plas.io
COLORIZATION
Choose an attribute to colorize
Classification
COLORMAP
Color gradient values used for Heightmap Color and Classification color sources
INTENSITY SOURCE
Color modulators, modulation can be played with
Intensity
INTENSITY BLENDING
How much of intensity to blend with color
ALL COLOR
ALL INTENSITY
INTENSITY SCALING
Narrow down on a range of intensity values, only applies to intensity values fetch from point source
UTILITIES
INUNDATION
Enable an Inundation Plane and manipulate it using a slider
Enable
WebGL 1.0 (OpenGL ES 2.0 Chromium), Provider: WebKit

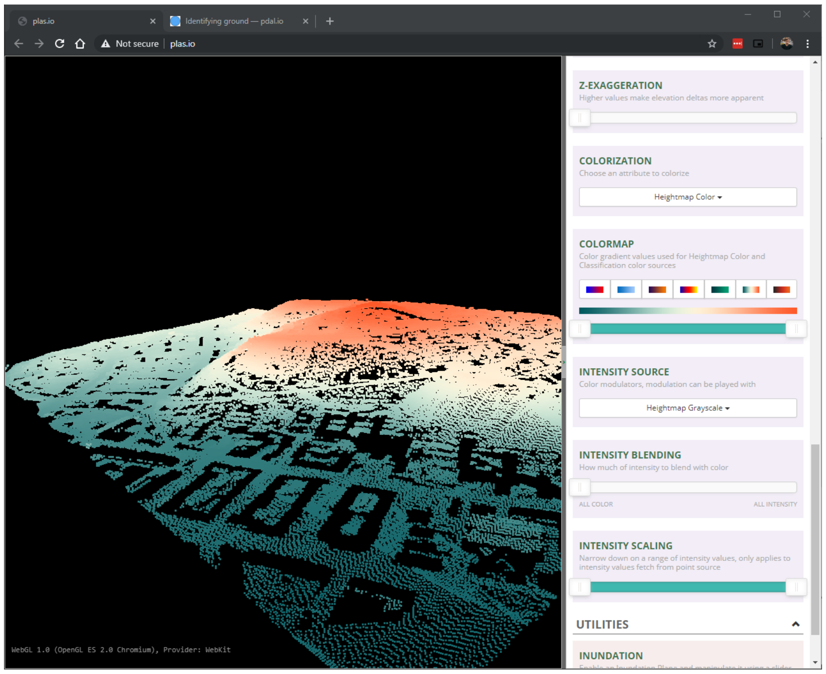
7. Run the command **pdal translate csite_original.laz –o ground.laz smrf range --filters.range.limits="Classification[2:2]" -v 4**.

If you open this up in **plas.io** then you can see we only exported the ground points. There is still noise under the ground. We will now use the translate command to stack the filters.outlier and filters.smrf stages.


plas.io
Identifying ground — pdal.io
Not secure | plas.io
Choose alternate ways of viewing data
No regions defined
DATA
MEASUREMENT
Measurement segments currently in view area
Reset
No Measurement Segments
PARTICLE SIZE
Larger sizes fill in the display
Z-EXAGGERATION
Higher values make elevation deltas more apparent
COLORIZATION
Choose an attribute to colorize
Classification
COLORMAP
Color gradient values used for Heightmap Color and Classification color sources
INTENSITY SOURCE
Color modulators, modulation can be played with
Intensity
WebGL 1.0 (OpenGL ES 2.0 Chromium), Provider: WebKit

8. Run the command **pdal translate csite_original.laz -o csite_ground_denoised.laz outlier smrf range --filters.outlier.method="statistical" --filters.outlier.mean_k=8 --filters.outlier.multiplier=3.0 --filters.smrf.ignore="Classification[7:7]" --filters.range.limits="Classification[2:2]" --writers.las.compression=true --verbose 4**.

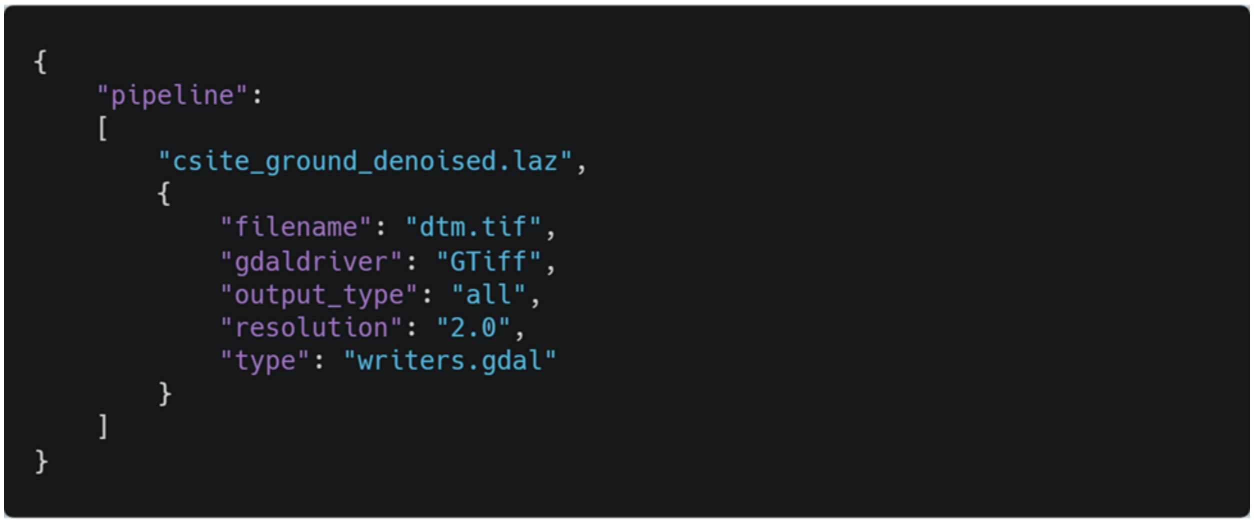
This command will give us the ground with no noise. Switch the view to the height map.


plas.io
Identifying ground — pdal.io
Not secure | plas.io
Z-EXAGGERATION
Higher values make elevation deltas more apparent
COLORIZATION
Choose an attribute to colorize
Heightmap Color
COLORMAP
Color gradient values used for Heightmap Color and Classification color sources
INTENSITY SOURCE
Color modulators, modulation can be played with
Heightmap Grayscale
INTENSITY BLENDING
How much of intensity to blend with color
ALL COLOR
ALL INTENSITY
INTENSITY SCALING
Narrow down on a range of intensity values, only applies to intensity values fetch from point source
UTILITIES
INUNDATION
WebGL 1.0 (OpenGL ES 2.0 Chromium), Provider: WebKit

Generating a DTM and Hillshade

This part will use PDAL to generate an elevation surface model using the output from the identifying ground exercise, PDAL's writers.gdal operation, and gdal to generate an elevation and hillshade surface from point cloud data. We will be using piplines again to do this process.

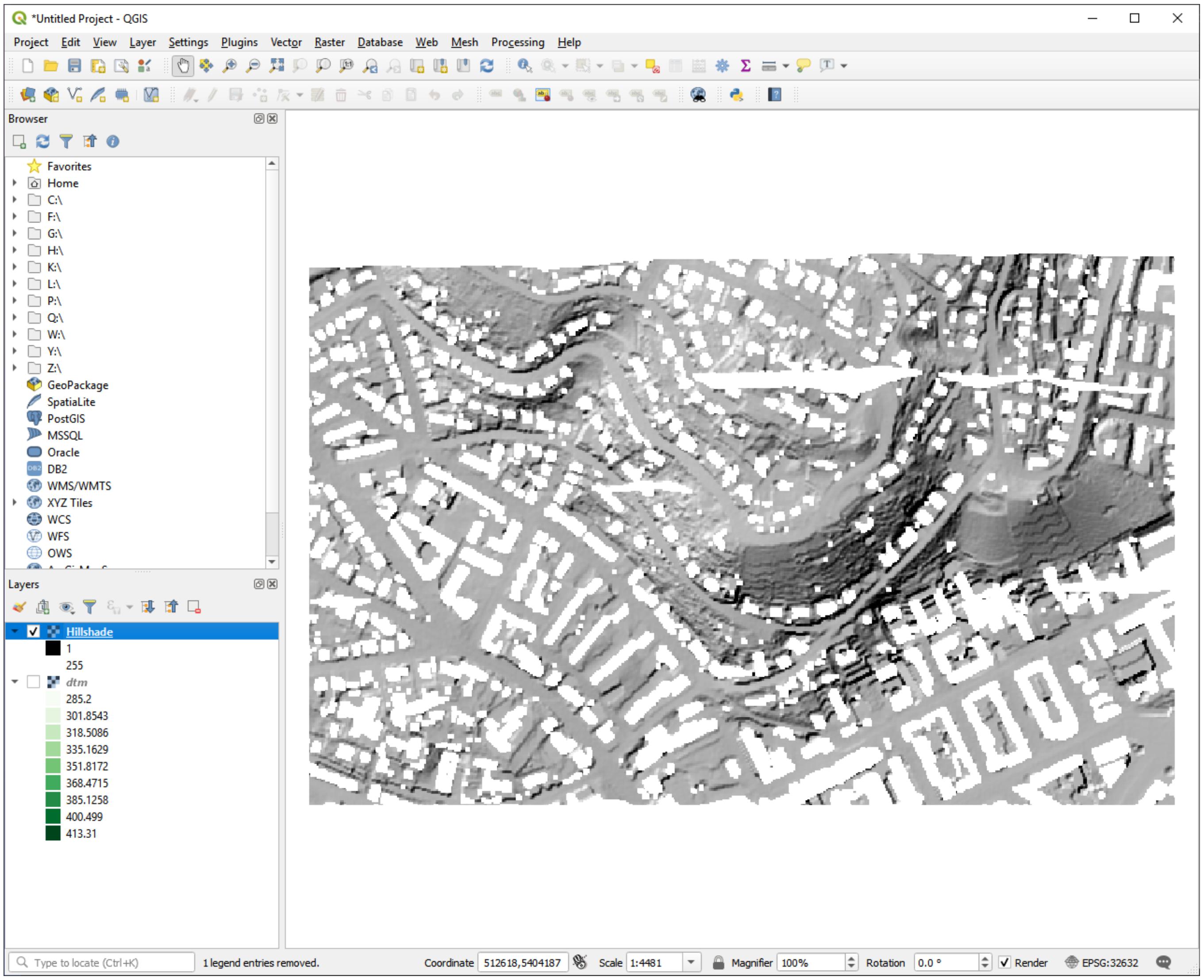
1. Inside of Visual Studio Code create a new file called **dtm.json**.



```
{
    "pipeline":
    [
        "csite_ground_denoised.laz",
        {
            "filename": "dtm.tif",
            "gdaldriver": "GTiff",
            "output_type": "all",
            "resolution": "2.0",
            "type": "writers.gdal"
        }
    ]
}
```

2. Fill in the above code into Visual Studio Code.
3. Run the pipeline **pdal pipeline dtm.json**.
4. File were written but we need to use QGIS to visualize it. Open **QGIS**.
5. Click **Layer**.
6. Click **Add Layer**.
7. Click **Add Raster Layer**.
8. Click the **ellipsis** and find the dtm.tif file in your exercise 03 folder.
9. Click **Add** and then click **Close.**
10. Right click on the **dtm** in the layers tab and click **properties**.
11. Go to the symbology table and under **Render Type** choose **Singleband pseudocolor** then click **Apply** and **OK**.
12. QGIS provides us access to GDAL processing tools, and we are going to use that to create a hillshade of our surface. Choose **Raster.**
13. Click **Analysis** and then click **Hillshade**.
14. Click the **ellipsis** under the Hillshade marker and choose **Save to File**.
15. Save the file under the exercise 03 folder as **Hillshade**.
16. Click **Run** to run the process and when the process is complete click close.

You can see that this created a **Hillshade.tif** file within your exercise 03 folder and automatically added that file to your QGIS layers tab.


*Untitled Project - QGIS
Project Edit View Layer Settings Plugins Vector Raster Database Web Mesh Processing Help
Browser
Favorites
Home
C:\
F:\
G:\
H:\
K:\
L:\
P:\
Q:\
W:\
Y:\
Z:\
GeoPackage
SpatiaLite
PostGIS
MSSQL
Oracle
DB2
WMS/WMTS
XYZ Tiles
WCS
WFS
OWS
Layers
Hillshade
1
255
dtm
285.2
301.8543
318.5086
335.1629
351.8172
368.4715
385.1258
400.499
413.31
Type to locate (Ctrl+K)
1 legend entries removed.
Coordinate 512618,5404187
Scale 1:4481
Magnifier 100%
Rotation 0.0 °
Render
EPSG:32632