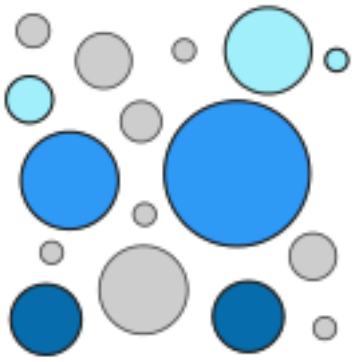


Point Cloud Processing and Analysis with PDAL

08/26/2019

Howard Butler
Pete Gadomski
Dr. Craig Glennie
Michael Smith
Dr. Adam Steer

August 26th, 2019

CONTENTS

I Introduction 3

1 Materials 7
1.1 Slides . . . 7
1.2 Workshop Materials . . . 7
1.3 USB Example Data Drive . . . 7

II Introduction to LiDAR 9

2 Types of LiDAR 13

3 Modes of LiDAR Collection 15

4 Georeferencing 17
4.1 Integrating LiDAR and GNSS/IMU data . . . 17

5 Discrete-Return vs. Full-Waveform 19

III Software Installation 23

6 Conda 25
6.1 What is Conda . . . 25
6.2 How will we use Conda? . . . 25
6.3 Installing Conda . . . 25

IV Exercises 27

7 Basic Information 29
7.1 Printing the first point . . . 29
7.2 Printing file metadata . . . 30
7.3 Searching near a point . . . 33

8 Translation **35**
8.1 Compression . . . 35
8.2 Reprojection . . . 36
8.3 Entwine . . . 39

9 Analysis **43**
9.1 Finding the boundary . . . 43
9.2 Clipping data with polygons . . . 47
9.3 Colorizing points with imagery . . . 53
9.4 Removing noise . . . 57
9.5 Visualizing acquisition density . . . 60
9.6 Thinning . . . 65
9.7 Identifying ground . . . 71
9.8 Generating a DTM . . . 77
9.9 Creating surface meshes . . . 88
9.10 Rasterizing Attributes . . . 90

10 Python **99**
10.1 Plotting a histogram . . . 99

11 Georeferencing **105**
11.1 Georeferencing . . . 105

12 Batch Processing **109**
12.1 Batch Processing . . . 109

V Final Project **115**

VI Notes **119**

13 Notes **121**

14 Notes **123**

15 Notes **125**

16 Notes **127**

17 Notes **129**

18 Notes **131**

Bibliography **133**

Index **135**

Author Howard Butler

Author Pete Gadomski

Author Dr. Craig Glennie

Author Michael Smith

Author Dr. Adam Steer

Contact howard@hobu.co

Date 08/26/2019

Part I

Introduction

1. *Introduction to LiDAR* (page 11)
2. Introduction to PDAL (https://pdal.io/about.html#about)
3. *Software Installation* (page 25)
4. *Basic Information* (page 29)
5. *Translation* (page 35)
6. *Analysis* (page 43)
7. *Georeferencing* (page 105)

CHAPTER

ONE

MATERIALS

1.1 Slides

- Slides (https://pdal-workshop-2019.s3.amazonaws.com/slides.zip)

1.2 Workshop Materials

These materials are available at https://pdal-workshop-2019.s3.amazonaws.com/ as both a PDF and an HTML website.

- PDF download (https://pdal-workshop-2019.s3.amazonaws.com/PDAL-workshop.pdf)
- HTML (https://pdal-workshop-2019.s3.amazonaws.com/PDAL-workshop-html.zip)

1.3 USB Example Data Drive

A companion USB drive containing workshop example data is required to follow along with these examples.

Note: A drive image is available for download at
https://pdal-workshop-2019.s3.amazonaws.com/PDAL-Workshop-complete.zip

Part II

Introduction to LiDAR

LiDAR is a remote sensing technique that uses visible or near-infrared laser energy to measure the distance between a sensor and an object. LiDAR sensors are versatile and (often) mobile; they help autonomous cars avoid obstacles and make detailed topographic measurements from space. Before diving into LiDAR data processing, we will spend a bit of time reviewing some LiDAR fundamentals and discussing some terms of art.

CHAPTER

TWO

TYPES OF LIDAR

LiDAR systems, generally speaking, come in one of three types:

- **Pulse-based**, or **linear-mode**, systems emit a pulse of laser energy and measure the time it takes for that energy to travel to a target, bounce off the target, and be returned to the sensor. These systems are called linear-mode because they (generally) only have a single aperture, and so can only measure distance along a single vector at any point in time. Pulse-based systems are very common, and are usually what you would think of when you think of LiDAR.
- **Phase-based** LiDAR systems measure distance via *interferometry*, that is, by using the phase of a modulated laser beam to calculate a distance as a fraction of the modulated signal's wavelength. Phase-based systems can be very precise, on the order of a few millimeters, but since they require comparatively more energy than the other two types they are usually used for short-range (e.g. indoor) scanning.
- **Geiger-mode**, or **photon-counting**, systems use extremely sensitive detectors that can be triggered by a single photon. Since only a single photon is required to trigger a measurement, these systems can operate at much much higher altitudes than linear mode systems. However, Geiger-mode systems are relatively new and suffer from very high amounts of noise and other operational restrictions, making them significantly less common than linear-mode systems.

Note: Unless otherwise noted, if we talk about a LiDAR scanner in this program, we will be referring to a pulse-based (linear) system.

CHAPTER

THREE

MODES OF LIDAR COLLECTION

LiDAR collects are generally categorized into four subjective types:

- **Terrestrial LiDAR Scanning (TLS)**: scanning with a stationary LiDAR sensor, usually mounted on a tripod.
- **Airborne LiDAR scanning (ALS)**: also called airborne laser swath mapping (ALSM), scanning with a LiDAR scanner mounted to a fixed-wing or rotor aircraft.
- **Mobile LiDAR scanning (MLS)**: scanning from a ground-based vehicle, such as a car.
- **Unmanned LiDAR scanning (ULS)**: scanning with drones or other unmanned vehicles.

With the exception of stationary TLS, LiDAR scanning generally requires the use of an integrated GNSS/IMU (Global Navigation Satellite System/Inertial Motion Unit), which provides information about the position, rotation, and motion of the scanning platform.

Note: As stated in the class description, we will focus on mobile and airborne laser scanning (MLS/ALS), though we will also use some TLS data.

CHAPTER

FOUR

GEOREFERENCING

LiDAR scanners collect information in the Scanner's Own Coordinate System (SOCS); this is a coordinate system centered at the scanner. The process of rotating, translating, and (possibly) transforming a point cloud into a real-world spatial reference system is known as **georeferencing**.

In the case of TLS, georeferencing is simply a matter of discovering the position and orientation of the static scanner. This is usually done with GNSS control points, which are used to solve for the scanner's position via least-squares.

For mobile or airborne LiDAR scanning, it is necessary to merge the scanner's points with the GNSS/IMU data. This can be done on-the-fly or as a part of a post-processing workflow. Since this is a common operation for mobile and airborne LiDAR collects, we will spend a moment discussing the methods and complications for georeferencing mobile LiDAR and GNSS/IMU data.

4.1 Integrating LiDAR and GNSS/IMU data

The LiDAR georeferencing equation is well-established; we present a version here from [Gle07]:

$$\mathbf{p}_G^l = \mathbf{p}_{GPS}^l + \mathbf{R}_b^l \left(\mathbf{R}_s^b \mathbf{r}^s - \mathbf{l}^b \right) \tag{4.1}$$

where:

- $\mathbf{p}_G^l$ are the coordinates of the target point in the global reference frame
- $\mathbf{p}_{GPS}^l$ are the coordinates of the GNSS sensor in the global reference frame
- $\mathbf{R}_b^l$ is the rotation matrix from the navigation frame to the global reference frame
- $\mathbf{R}_s^b$ is the rotation matrix from the scanner's frame to the navigation frame (boresight matrix)
- $\mathbf{r}^s$ is the coordinates of the laser point in the scanner's frame

- l^b is the lever-arm offset between the scanner's original and the navigation's origin

This equation contains fourteen unknowns, and in order to georeference a single LiDAR return we must determine all fourteen variables at the time of the pulse.

As a rule of thumb, the position, attitude, and motion of the scanning platform (aircraft, vehicle, etc) are sampled at a much lower rate than the pulse rate of the laser — rates of ~1Hz are common for GNSS/IMU sampling. In order to match the GNSS/IMU sampling rate with the sampling rate of the laser, GNSS/IMU measurements are interpolated to line up with the LiDAR measurements. Then, these positions and attitudes are combined via Equation (4.1) to create a final, georeferenced point cloud.

Note: While lever-arm offsets are usually taken from the schematic drawings of the LiDAR mounting system, the boresight matrix cannot be reliably determined from drawings alone. The boresight matrix must therefore be determined either via manual or automated boresight calibration using actual LiDAR data of planar surfaces, such as the roof and sides of buildings. The process for determining a boresight calibration from LiDAR data is beyond the scope of this class.

CHAPTER

FIVE

DISCRETE-RETURN VS. FULL-WAVEFORM

Pulse-based LiDAR systems use the round-trip travel time of a pulse of laser energy to measure distances. The outgoing pulse of a LiDAR system is roughly (but not exactly) a Gaussian:

This pulse can interact with multiple objects in a scene before it is returned to the sensor. Here is an example of a LiDAR return:

As you can see, this return pulse can be very complicated. While there is more information contained in the “full waveform” picture displayed above, many LiDAR consumers are only interested in detecting the presence or absence of an object — simplistically, the peaks in that waveform.

Full waveform data is used only in specialized circumstances. If you have or receive LiDAR data, it will usually be discrete return (point clouds). Processing full waveform data is beyond the scope of this class.

Note: PDAL is a discrete-return point cloud processing library. It does not have any functionality to analyse or process full waveform data.

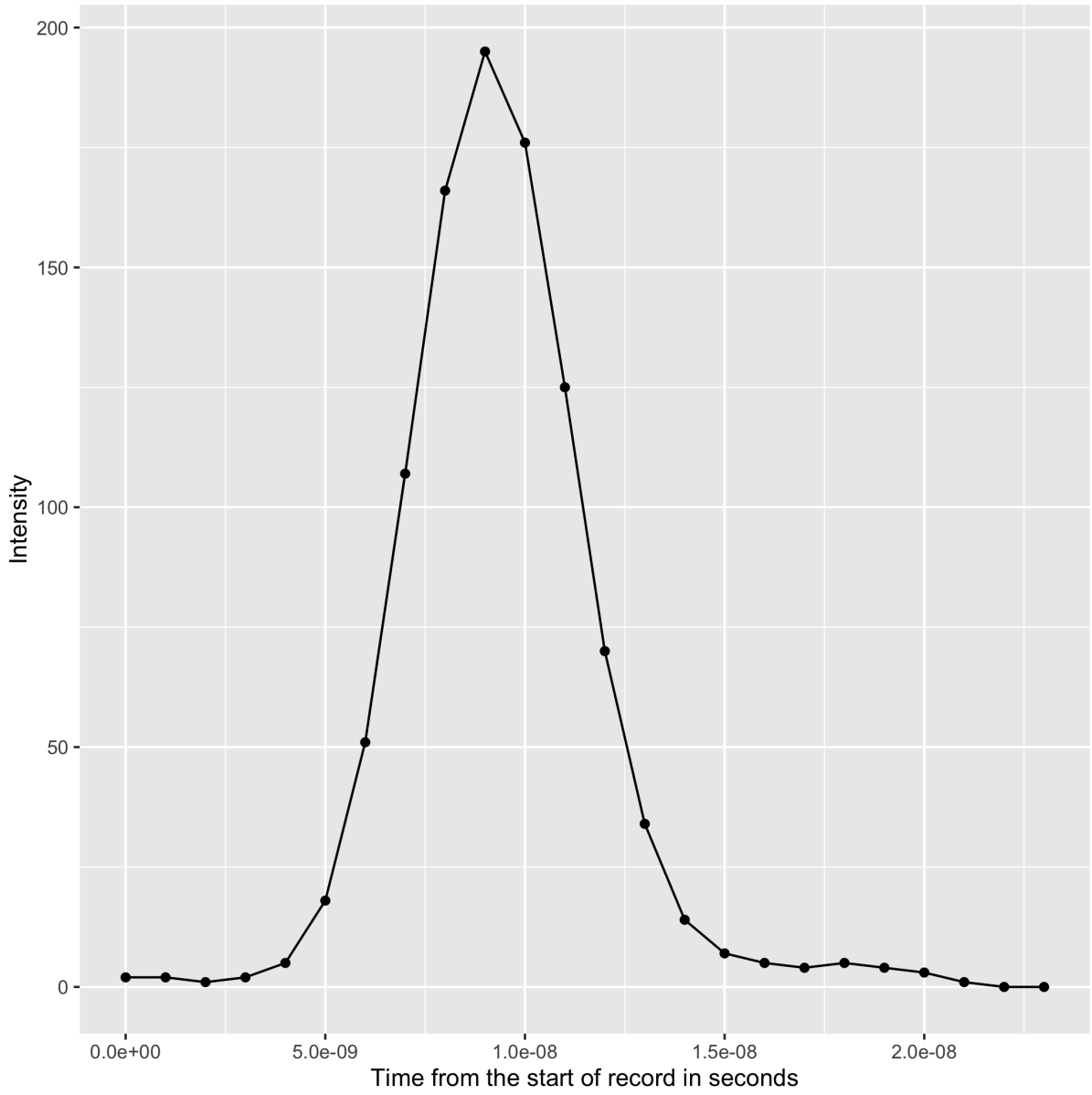


Fig. 1: A real-world outgoing LiDAR pulse.

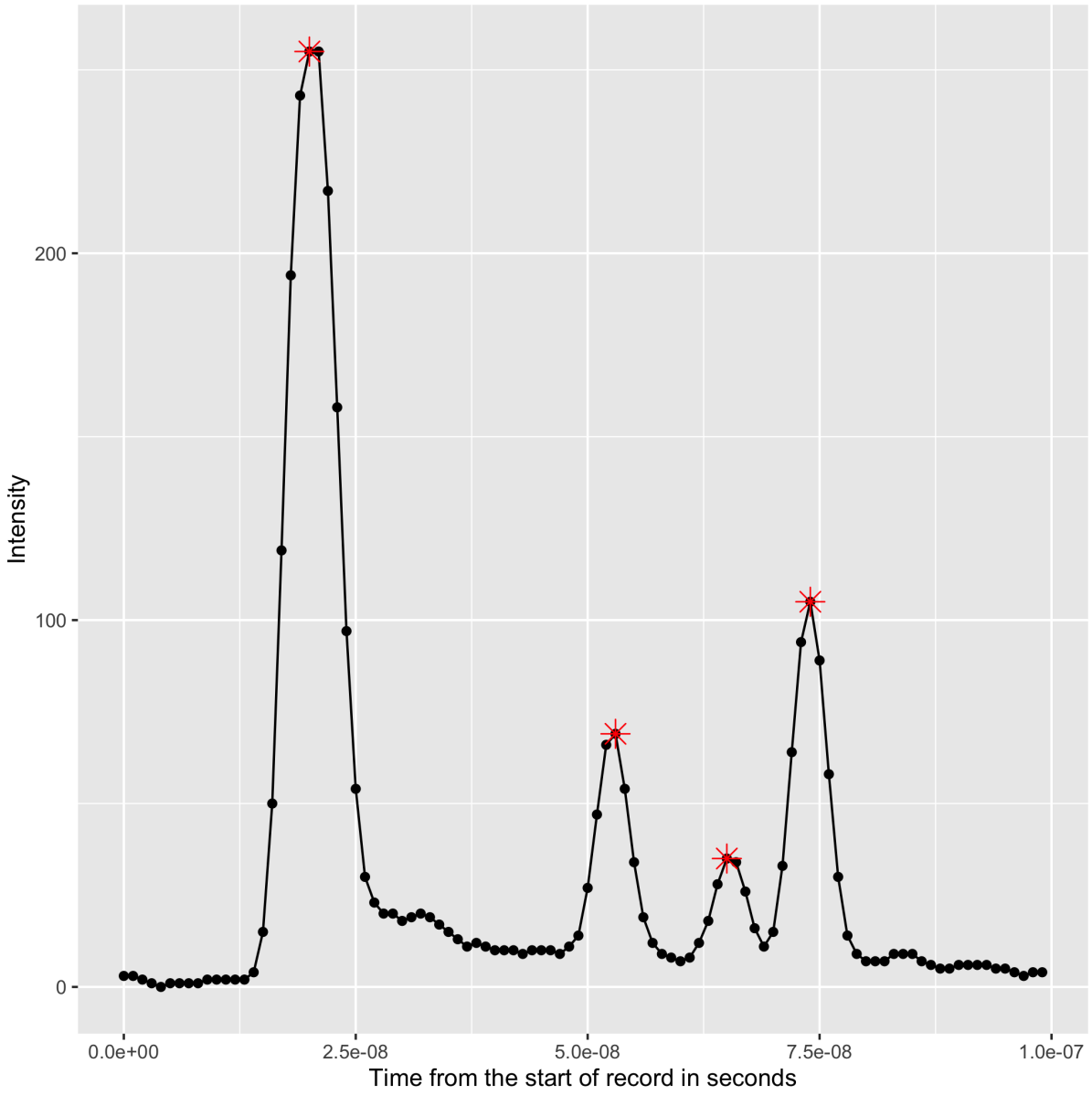


Fig. 2: A real-world incoming LiDAR return. Potential discrete-return peaks are marked in red.

Part III

Software Installation

CHAPTER

SIX

CONDA

6.1 What is Conda

Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language..

6.2 How will we use Conda?

PDAL stands on the shoulders of giants. It uses GDAL, GEOS, and many other dependencies (https://pdal.io/development/compilation/index.html#building). Because of this, it is very challenging to build it yourself. We could easily burn an entire workshop learning the esoteric build mysteries of PDAL and all of its dependencies. Fortunately, Conda provides us a fully-featured known configuration to run our examples and exercises without having to suffer so much, and provides it for Windows, Linux, and macOS.

Note: Not everyone uses Conda. Another alternative to get a known configuration is to go through the workshop using docker as your platform. A previous edition of the workshop was provided as Docker, but it was found to be a bit too difficult to follow.

6.3 Installing Conda

1. Copy the entire contents of your workshop USB key to a `PDAL` directory in your home directory (something like `C:\Users\hobu\PDAL`) or the equivalent for your OS. We will refer to this location for the rest of the workshop materials.

2. Download the Conda installer for your OS setup.
   https://docs.conda.io/en/latest/miniconda.html
3. After installing Conda, create an environment for PDAL with:

```
conda create --name pdalworkshop
```

4. Then *activate* the new environment:

```
conda activate pdalworkshop
```

5. Install PDAL, Entwine, and GDAL, and install it from **conda-forge**:

```
conda install -c conda-forge pdal gdal entwine matplotlib
```

Part IV

Exercises

CHAPTER

SEVEN

BASIC INFORMATION

7.1 Printing the first point

7.1.1 Exercise

This exercise uses PDAL to print information from the first point. Issue the following command in your *Conda Shell*.

```
pdal info ./exercises/info/interesting.las -p 0
```

Here's a summary of what's going on with that command invocation

1. `pdal`: The `pdal` application :)
2. `info`: We want to run info (https://pdal.io/apps/info.html#info-command) on the data. All commands are run by the `pdal` application.
3. `./exercises/info/interesting.las`: The file we are running the command on. PDAL will be able to identify this file is an ASPRS LAS (http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html) file from the extension, `.las`, but not every file type is easily identified. You can use a pipeline (https://pdal.io/apps/pipeline.html#pipeline-command) to override which reader (https://pdal.io/stages/readers.html#readers) PDAL will use to open the file.
4. `-p 0`: `-p` corresponds to "print a point", and `0` means to print the first one (computer people count from 0).

```
(pdalworkshop) $pdal info ./exercises/info/interesting.las -p 0
{
  "filename": "./exercises/info/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
    }
  }
}
```

7.1.2 Notes

1. PDAL uses JSON (https://en.wikipedia.org/wiki/JSON) as the exchange format when printing information from info (https://pdal.io/apps/info.html#info-command). JSON is a structured, human-readable format that is much simpler than its XML (https://en.wikipedia.org/wiki/XML) cousin.
2. You can use the writers.text (https://pdal.io/stages/writers.text.html#writers-text) writer to output point attributes to CSV (https://en.wikipedia.org/wiki/Comma-separated_values) format for other processing.
3. Output help information on the command line by issuing the `--help` option
4. A common query with `pdal info` is `--all`, which will print all header, metadata, and statistics about a file.

7.2 Printing file metadata

7.2.1 Exercise

This exercise uses PDAL to print metadata information. Issue the following command in your *Conda Shell*.

```
pdal info ./exercises/info/interesting.las --metadata
```


```
(pdalworkshop) $  pdal info ./exercises/info/interesting.las --metadata
{
  "filename": "./exercises/info/interesting.las",
  "metadata":
  {
    "comp_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[\"stand
ard_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\"]]]",
    "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": 0.010000000000000000208,
    "scale_y": 0.010000000000000000208,
    "scale_z": 0.010000000000000000208,
    "software_id": "HOBU-GENERATING",
```


Note: PDAL metadata (https://pdal.io/development/metadata.html#metadata) is returned a in a tree structure corresponding to processing pipeline that produced it.

See also:

Use the JSON (https://en.wikipedia.org/wiki/JSON) processing capabilities of your favorite processing software to selectively access and manipulate values.

- Python JSON library (https://docs.python.org/2/library/json.html)
- jsawk (https://github.com/micha/jsawk) (like `awk` but for JSON data)
- jq (https://stedolan.github.io/jq/) (command line processor for JSON)
- Ruby JSON library (http://ruby-doc.org/stdlib-2.0.0/libdoc/json/rdoc/JSON.html)

Structured Metadata Output

Many command-line utilities output their data in a human-readable custom format. The downsides to this approach are significant. PDAL was designed to be used in the context of other software tools driving it. For example, it is quite common for PDAL to be used in data validation scenarios. Other programs might need to inspect information in PDAL's output and then act based on the values. A human-readable format would mean that downstream program would need to write a parser to consume PDAL's special format.

JSON (https://en.wikipedia.org/wiki/JSON) provides a nice balance between human- and machine- readable, but even then it can be quite hard to find what you're looking for, especially if the output is long. `pdal` command output used in conjunction with a JSON parsing tool like `jq` provide a powerful inspection combination.

For example, we might only care about the `system_id` and `compressed` flag for this particular file. Our simple `pdal info --metadata` command gives us that, but it also gives us a bunch of other stuff we don't need at the moment either. Let's focus on extracting what we want using the `jq` command.

```
pdal info ./exercises/info/interesting.las --metadata \
| jq  ".metadata.compressed, .metadata.system_id"
```

```
pdal info ./exercises/info/interesting.las --metadata ^
| jq  ".metadata.compressed, .metadata.system_id"
```

```
(pdalworkshop) $ pdal info ./exercises/info/interesting.las --metadata | jq  ".metadata.compressed, .metadata.system_id"
false
"HOBU-SYSTEMID"
(pdalworkshop) $
```

Note: PDAL's JSON output is very powerfully combined with the processing capabilities of other programming languages such as JavaScript or Python. Both of these languages have excellent built-in tools for consuming JSON, along with plenty of other features to allow you to do something with the data inside the data structures. As we will see later in the workshop, this PDAL feature is one that makes construction of custom data processing workflows with PDAL very convenient.

7.2.2 Notes

1. PDAL uses JSON (https://en.wikipedia.org/wiki/JSON) as the exchange format when printing information from info (https://pdal.io/apps/info.html#info-command). JSON provides human and machine-readable text data.
2. The PDAL metadata document (https://pdal.io/development/metadata.html#metadata) contains background and information about specific metadata entries and what they mean.
3. Metadata available for a given file depends on the stage that produces the data. Readers (https://pdal.io/stages/readers.html#readers) produce same-named values where possible, but it is common that variables are different. Filters (https://pdal.io/stages/filters.html#filters) and even writers (https://pdal.io/stages/writers.html#writers) can also produce metadata entries.
4. Spatial reference system or coordinate system information is a kind of special metadata.

Spatial references are come directly from source data or are provided via options in PDAL.

7.3 Searching near a point

7.3.1 Exercise

This exercise uses PDAL to find points near a given search location. Our scenario is a simple one – we want to find the two points nearest the midpoint of the bounding cube of our `interesting.las` data file.

First 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:

```
pdal info ./exercises/info/interesting.las --all | jq .stats.bbox.
↪native.bbox
```

```
(pdalworkshop) $pdal info ./exercises/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
}
(pdalworkshop) $
```

Find the average the `X`, `Y`, and `Z` values:

```
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", issue the `--query` option to `pdal info` and return the three nearest points to it:

```
pdal info ./exercises/info/interesting.las --query "637301.20,␣
↪851217.57, 496.49/3"
```

Note: The `/3` portion of our query string tells the `query` command to give us the 3 nearest points. Adjust this value to return data in closest-distance ordering.

```
(pdalworkshop) $ pdal info ./exercises/info/interesting.las --query "637301.20, 851217.57, 496.49/3"
{
  "filename": "./exercises/info/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,
```

7.3.2 Notes

1. PDAL uses JSON (https://en.wikipedia.org/wiki/JSON) as the exchange format when printing information from info (https://pdal.io/apps/info.html#info-command). JSON is a structured, human-readable format that is much simpler than its XML (https://en.wikipedia.org/wiki/XML) cousin.
2. The `--query` option of info (https://pdal.io/apps/info.html#info-command) constructs a KD-tree (https://en.wikipedia.org/wiki/K-d_tree) of the entire set of points in memory. If you have really large data sets, this isn't going to work so well, and you will need to come up with a different solution.

CHAPTER

EIGHT

TRANSLATION

8.1 Compression

8.1.1 Exercise

This exercise uses PDAL to compress ASPRS LAS (http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html) data into LASzip (http://laszip.org).

1. Issue the following command in your *Conda Shell*.

```
pdal translate ./exercises/translation/interesting.laz \
./exercises/translation/interesting.las
```

```
pdal translate ./exercises/translation/interesting.laz ^
./exercises/translation/interesting.las
```

LAS is a very fluffy binary format. Because of the way the data are stored, there is ample redundant information, and LASzip (http://laszip.org) is an open source solution for compressing this information. Note that we are actually inflating the data here. Its laz from the workshop and we are converting it to las.

2. Verify that the laz data is compressed over the las:

```
ls -alh ./exercises/translation/interesting.laz

ls -alh  ./exercises/translation/interesting.las
```

```
dir ./exercises/translation/interesting.laz

dir ./exercises/translation/interesting.las
```

```
(pdal19) C:\>dir C:\Users\hobu\PDAL\exercises\info\interesting.laz
 Volume in drive C has no label.
 Volume Serial Number is 162A-5F2D

 Directory of C:\Users\hobu\PDAL\exercises\info

08/04/2019  05:12 PM            18,786 interesting.laz
               1 File(s)         18,786 bytes
               0 Dir(s)  11,142,668,288 bytes free

(pdal19) C:\>dir C:\Users\hobu\PDAL\exercises\info\interesting.las
 Volume in drive C has no label.
 Volume Serial Number is 162A-5F2D

 Directory of C:\Users\hobu\PDAL\exercises\info

08/04/2019  02:44 PM            37,698 interesting.las
               1 File(s)         37,698 bytes
               0 Dir(s)  11,142,668,288 bytes free

(pdal19) C:\>
```

See also:

LAS Reading and Writing with PDAL (https://pdal.io/tutorial/las.html#las-tutorial) contains many pointers about settings for ASPRS LAS (http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html) data and how to achieve specific data behaviors with PDAL.

8.1.2 Notes

1. Typical LASzip (http://laszip.org) compression is 5:1 to 8:1, depending on the type of LiDAR (https://en.wikipedia.org/wiki/Lidar). It is a compression format specifically for the ASPRS LAS (http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html) model, however, and will not be as efficient for other types of point cloud data.
2. You can open and view LAZ data in web browsers using http://plas.io

8.2 Reprojection

8.2.1 Exercise

This exercise uses PDAL to reproject ASPRS LAS (http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html) data

Issue the following command in your *Conda Shell*:

```
pdal translate ./exercises/analysis/ground/CSite1_orig-utm.laz \
  ./exercises/translation/csite-dd.laz reprojection \
  --filters.reprojection.out_srs="EPSG:4326"
```

```
pdal translate ./exercises/analysis/ground/CSite1_orig-utm.laz ^
  ./exercises/translation/csite-dd.laz reprojection ^
  --filters.reprojection.out_srs="EPSG:4326"
```

```
(pdalworkshop) C:\>pdal translate c:/Users/hobu/PDAL/exercises/analysis/ground/CSite1_orig-utm.laz ^
More?     c:/Users/hobu/PDAL/exercises/translation/csite-dd.laz ^
More?     reprojection ^
More?     --filters.reprojection.out_srs="EPSG:4326"

(pdalworkshop) C:\>
```

Unfortunately this doesn't produce the intended results for us. Issue the following `pdal info` command to see why:

```
pdal info ./exercises/translation/csite-dd.laz --all \
    | jq .stats.bbox.native.bbox
```

```
pdal info ./exercises/translation/csite-dd.laz --all ^
    | jq .stats.bbox.native.bbox
```

```
(pdalworkshop) C:\>pdal info c:/Users/hobu/PDAL/exercises/translation/csite-dd.laz --all | jq .stats.bbox.native.bbox
{
  "maxx": 9.18,
  "maxy": 48.79,
  "maxz": 426.91,
  "minx": 9.16,
  "miny": 48.78,
  "minz": 99.43
}
```

`--all` dumps all info (https://pdal.io/apps/info.html#info-command) information about the file, and we can then use the jq (https://stedolan.github.io/jq/) command to extract out the "native" (same coordinate system as the file itself) bounding box. As we can see, the problem is we only have two decimal places of precision on the bounding box. For geographic coordinate systems, this isn't enough precision.

Printing the first point confirms this problem:

```
(pdalworkshop) C:\>pdal info c:/Users/hobu/PDAL/exercises/translation/csite-dd.laz -p 0
{
  "filename": "c:/Users/hobu/PDAL/exercises/translation/csite-dd.laz",
  "pdal_version": "1.9.1 (git-version: Release)",
  "points":
  {
    "point":
    {
      "Blue": 0,
      "Classification": 0,
      "EdgeOfFlightLine": 0,
      "GpsTime": 0,
      "Green": 0,
      "Intensity": 100,
      "NumberOfReturns": 2,
      "PointId": 0,
      "PointSourceId": 0,
      "Red": 0,
      "ReturnNumber": 1,
      "ScanAngleRank": 0,
      "ScanDirectionFlag": 0,
      "UserData": 0,
      "X": 9.17,
      "Y": 48.78,
      "Z": 316.88
    }
  }
}
```

Some formats, like writers.las (https://pdal.io/stages/writers.las.html#writers-las) do not automatically set scaling information. PDAL cannot really do this for you because there are a number of ways to trip 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 of the value. Use PDAL to set that to `auto` so you don't have to compute it.

```
pdal translate \
./exercises/analysis/ground/CSite1_orig-utm.laz \
./exercises/translation/csite-dd.laz 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"
```

```
pdal translate ^
./exercises/analysis/ground/CSite1_orig-utm.laz ^
./exercises/translation/csite-dd.laz 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"
```

```
(pdalworkshop) $pdal translate \
> ./exercises/analysis/ground/CSite1_orig-utm.laz \
> ./exercises/translation/csite-dd.laz 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"
(pdal translate writers.las Warning) Auto offset for Xrequested in stream mode.  Using value of 9.16789.
(pdal translate writers.las Warning) Auto offset for Yrequested in stream mode.  Using value of 48.7835.
(pdalworkshop) $
```

Run the *pdal info* command again to verify the `X`, `Y`, and `Z` dimensions:

```
(pdalworkshop) $pdal info ./exercises/translation/csite-dd.laz --all \
>     | jq .stats.bbox.native.bbox
{
  "maxx": 9.179032939,
  "maxy": 48.78976523,
  "maxz": 426.91,
  "minx": 9.164037839,
  "miny": 48.78345443,
  "minz": 99.43
}
(pdalworkshop) $
```

8.2.2 Notes

1. filters.reprojection (https://pdal.io/stages/filters.reprojection.html#filters-reprojection) will use whatever coordinate system is defined by the point cloud file, but you can override it using the `in_srs` option. This is useful in situations where the coordinate system is not correct, not completely specified, or your system doesn't have all of the required supporting coordinate system dictionaries.
2. PDAL uses Proj.4 (http://proj4.org) library for reprojection. This library includes the capability to do both vertical and horizontal datum transformations.

8.3 Entwine

8.3.1 Exercise

This exercise uses PDAL to fetch data from an Entwine index stored in an Amazon Web Services object store (bucket). Entwine is a point cloud indexing strategy, which rearranges points into a lossless octree structure known as EPT, for Entwine Point Tiles. The structure is described here: https://entwine.io/entwine-point-tile.html.

EPT indexes can be used for visualisation as well as analysis and data manipulation at any scale.

Examples of Entwine usage can be found from very fine photogrammetric surveys to continental scale lidar management.

US Geological Survey (USGS) example data is here: https://usgs.entwine.io/

We will use a sample data set from Dublin, Ireland
http://potree.entwine.io/data/view.html?r=%22http://na-c.entwine.io/dublin/ept.json%22

1. View the `entwine.json` file in your editor. If the file does not exist, create it and paste the following JSON into it:

```
{
    "pipeline": [
        {
            "type": "readers.ept",
            "filename":"https://na-c.entwine.io/dublin/",
            "resolution": 5
        },
        {
            "type": "writers.las",
            "compression": "true",
            "minor_version": "2",
            "dataformat_id": "0",
            "filename":"dublin.laz"
        }
    ]
}
```

Note: If you use the Developer Console (https://developers.google.com/web/tools/chrome-devtools/console/) when visiting http://speck.ly or http://potree.entwine.io, you can see the browser making requests against the EPT resource at http://na-c.entwine.io/dublin/ept.json

---

2. Issue the following command in your *Conda Shell*.

```
pdal pipeline ./excercises/translation/entwine.json -v 7
```

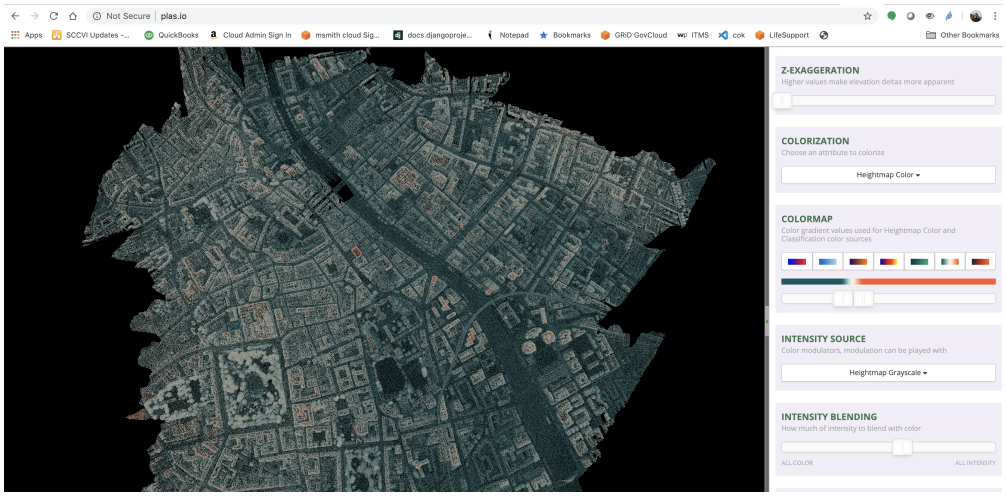
```
(pdal19) C:\Users\hobu\PDAL\exercises\translation>pdal pipeline entwine.json -v 7
(PDAL Debug) Debugging...
(pdal pipeline readers.ept Debug) GDAL debug: OGRSpatialReference::Validate: No root pointer.
(pdal pipeline readers.ept Debug) GDAL debug: OGRSpatialReference::Validate: No root pointer.
(pdal pipeline readers.ept Debug) GDAL debug: OGRSpatialReference::Validate: No root pointer.
(pdal pipeline readers.ept Debug) Endpoint: https://na-c.entwine.io/dublin/
Got EPT info
SRS: PROJCS["WGS 84 / Pseudo-Mercator",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EP
SG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHOR
ITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]],PROJECTION["Mercator_1SP"],PARAMETER["central_meridian",0],PARAMETER["scale_fac
tor",1],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["X",EAST],AX
IS["Y",NORTH],EXTENSION["PROJ4","+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nad
grids=@null +wktext +no_defs"],AUTHORITY["EPSG","3857"]]
Root resolution: 21.3828
Query resolution:  5
Actual resolution: 2.67285
Depth end: 4
Query bounds: ([-1.797693134862316e+308, 1.797693134862316e+308], [-1.797693134862316e+308, 1.797693134862316e+308], [-1.7976
93134862316e+308, 1.797693134862316e+308])
Threads: 4
(pdal pipeline readers.ept Debug) Registering dim X: double
(pdal pipeline readers.ept Debug) Registering dim Y: double
(pdal pipeline readers.ept Debug) Registering dim Z: double
(pdal pipeline readers.ept Debug) Registering dim Intensity: uint16_t
(pdal pipeline readers.ept Debug) Registering dim ReturnNumber: uint8_t
(pdal pipeline readers.ept Debug) Registering dim NumberOfReturns: uint8_t
(pdal pipeline readers.ept Debug) Registering dim ScanDirectionFlag: uint8_t
(pdal pipeline readers.ept Debug) Registering dim EdgeOfFlightLine: uint8_t
(pdal pipeline readers.ept Debug) Registering dim Classification: uint8_t
(pdal pipeline readers.ept Debug) Registering dim ScanAngleRank: float
(pdal pipeline readers.ept Debug) Registering dim UserData: uint8_t
(pdal pipeline readers.ept Debug) Registering dim PointSourceId: uint16_t
(pdal pipeline readers.ept Debug) Registering dim GpsTime: double
(pdal pipeline readers.ept Debug) Registering dim OriginId: uint32_t
(pdal pipeline Debug) Executing pipeline in standard mode.
(pdal pipeline readers.ept Debug) Overlap nodes: 78
(pdal pipeline readers.ept Debug) Overlap points: 8034506
(pdal pipeline readers.ept Debug) Data 1/78: 0-0-0-0
(pdal pipeline readers.ept Debug) Data 2/78: 1-0-0-0
(pdal pipeline readers.ept Debug) Data 3/78: 1-0-0-1
(pdal pipeline readers.ept Debug) Data 4/78: 1-0-1-0
(pdal pipeline readers.ept Debug) Data 5/78: 1-0-1-1
(pdal pipeline readers.ept Debug) Data 6/78: 1-1-0-0
```

3. Verify that the data look ok:

```
pdal info dublin.laz | jq .stats.bbox.native.bbox

pdal info dublin.laz -p 0
```

```
(pdal19) C:\Users\hobu\PDAL\exercises\translation>pdal info dublin.laz | jq .stats.bbox.native.bbox
{
  "maxx": -694128.96,
  "maxy": 7049938.84,
  "maxz": 385.37,
  "minx": -699477.88,
  "miny": 7044490.98,
  "minz": -144.24
}

(pdal19) C:\Users\hobu\PDAL\exercises\translation>pdal info dublin.laz -p 0
{
  "filename": "dublin.laz",
  "pdal_version": "1.9.1 (git-version: Release)",
  "points":
  {
    "point":
    {
      "Classification": 4,
      "EdgeOfFlightLine": 0,
      "Intensity": 7,
      "NumberOfReturns": 2,
      "PointId": 0,
      "PointSourceId": 0,
      "ReturnNumber": 2,
      "ScanAngleRank": -33,
      "ScanDirectionFlag": 1,
      "UserData": 0,
      "X": -697907.12,
      "Y": 7045474.25,
      "Z": 27.5
    }
  }
}
```

4. Visualize the data in http://plas.io



8.3.2 Notes

1. readers.ept (https://pdal.io/stages/readers.ept.html#readers-ept) contains more detailed documentation about how to use PDAL's EPT reader .

CHAPTER

NINE

ANALYSIS

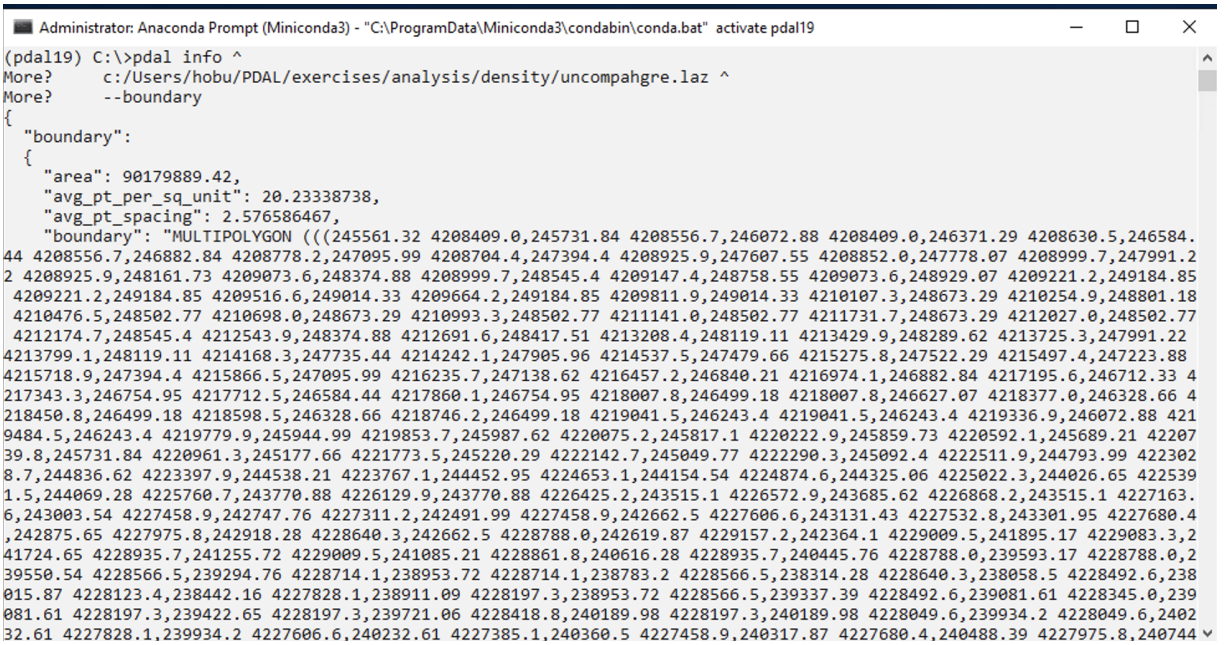
9.1 Finding the boundary

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. To complete this exercise, we are going to use qgis to view the boundary, which means we must first install it on our system.

9.1.1 Exercise

Note: We are going to run using the Uncompahgre data in the `./density` directory.

```
pdal info ./exercises/analysis/density/uncompahgre.laz --boundary
```



... a giant blizzard of coordinate output scrolls across our terminal. Not very useful.

Instead, let's generate some kind of vector output we can visualize with qgis. The `pdal tindex` is the "tile index" command, and it outputs a vector geometry file for each point cloud file it reads. It generates this boundary using the same mechanism we invoked above – filters.hexbin (https://pdal.io/stages/filters.hexbin.html#filters-hexbin). We can leverage this capability to output a contiguous boundary of the `uncompahgre.laz` file.

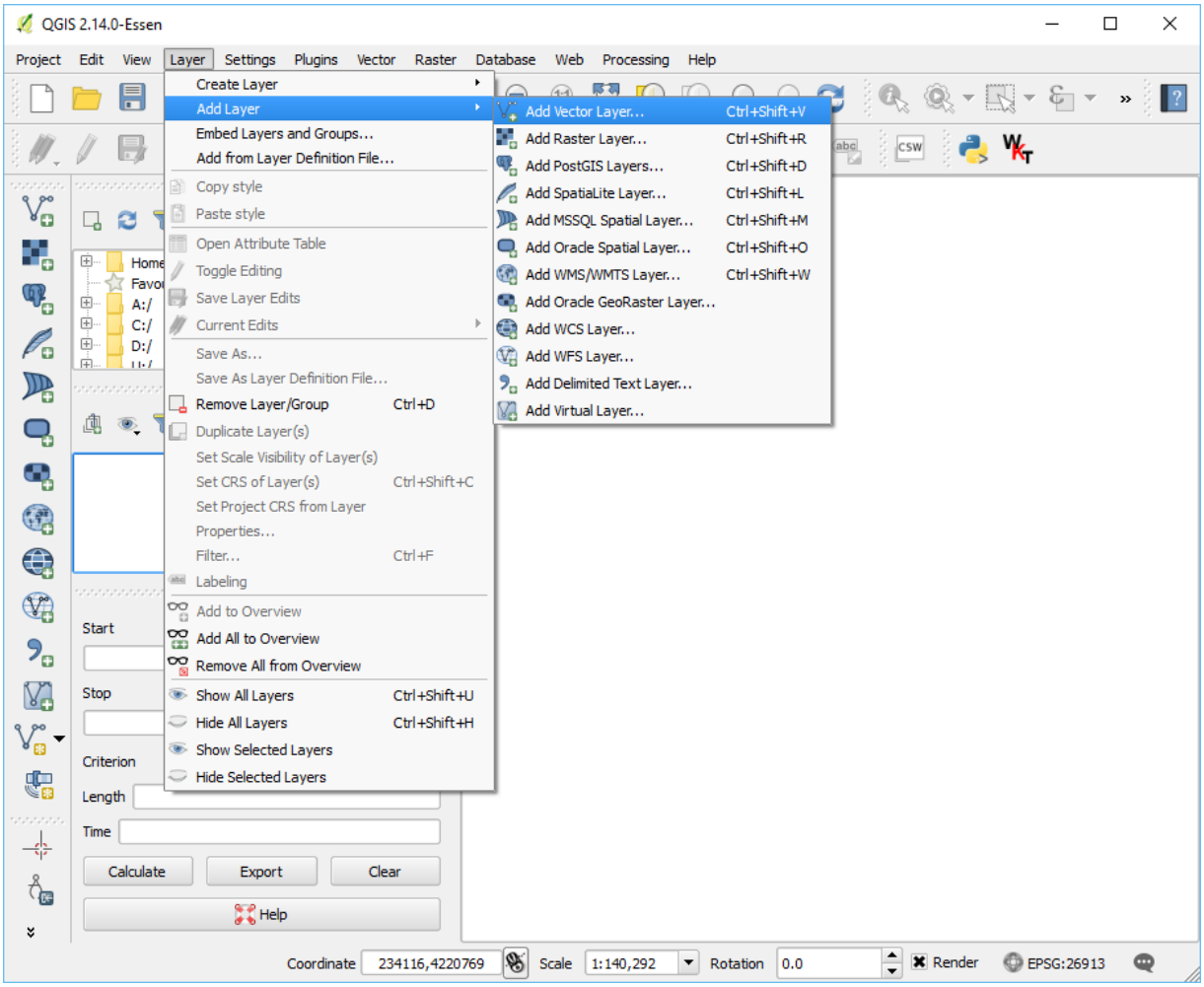
```
pdal tindex create --tindex ./exercises/analysis/boundary/boundary.
↪sqlite \
 --filespec ./exercises/analysis/density/uncompahgre.laz \
 -f SQLite
```

```
pdal tindex create --tindex ./exercises/analysis/boundary/boundary.
↪sqlite ^
 --filespec ./exercises/analysis/density/uncompahgre.laz ^
 -f SQLite
```

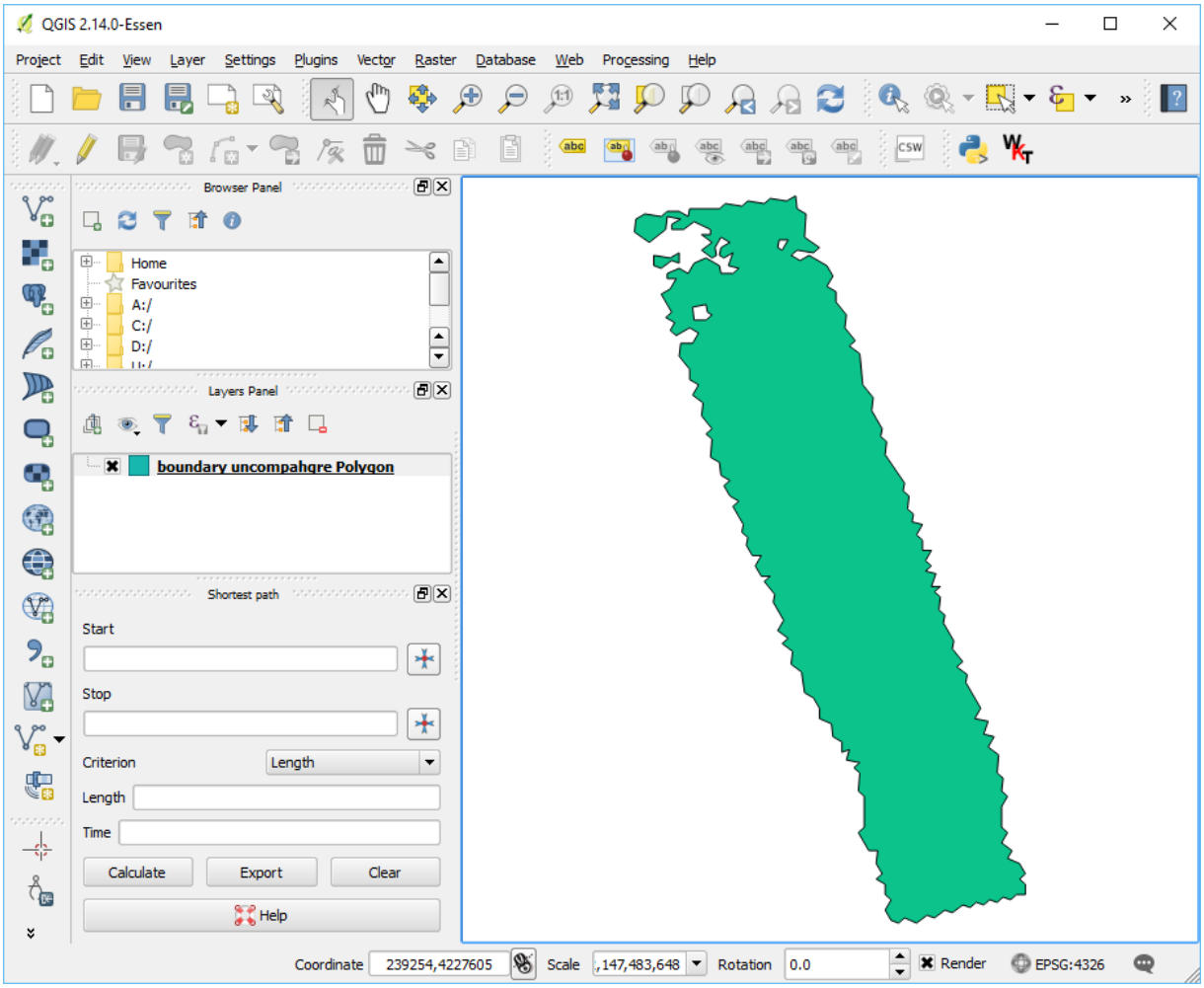
```
(pdalworkshop) $ pdal tindex create --tindex ./exercises/analysis/boundary/boundary.sqlite \
>  --filespec ./exercises/analysis/density/uncompahgre.laz \
>  -f SQLite
(pdalworkshop) $
```

Once we've run the tindex (https://pdal.io/apps/tindex.html#tindex-command), we can now visualize our output:

Open qgis and select *Add Vector Layer*:



Navigate to the `exercises/analysis/boundary` directory and then open the `boundary.sqlite` file:



9.1.2 Notes

1. The PDAL boundary computation is an approximation based on a hexagon tessellation. It uses the software at http://github.com/hobu/hexer to do this task.

2. filters.hexbin (https://pdal.io/stages/filters.hexbin.html#filters-hexbin) can also be used by the density (https://pdal.io/apps/density.html#density-command) to generate a tessellated surface. See the *Visualizing acquisition density* (page 60) example for steps to achieve this.

3. The tindex (https://pdal.io/apps/tindex.html#tindex-command) can be used to generate boundaries for large collections of data. A boundary-based indexing scheme is commonly used in LiDAR processing, and PDAL supports it through the `tindex` application. You can also use this command to merge data together (query across boundaries, for example).

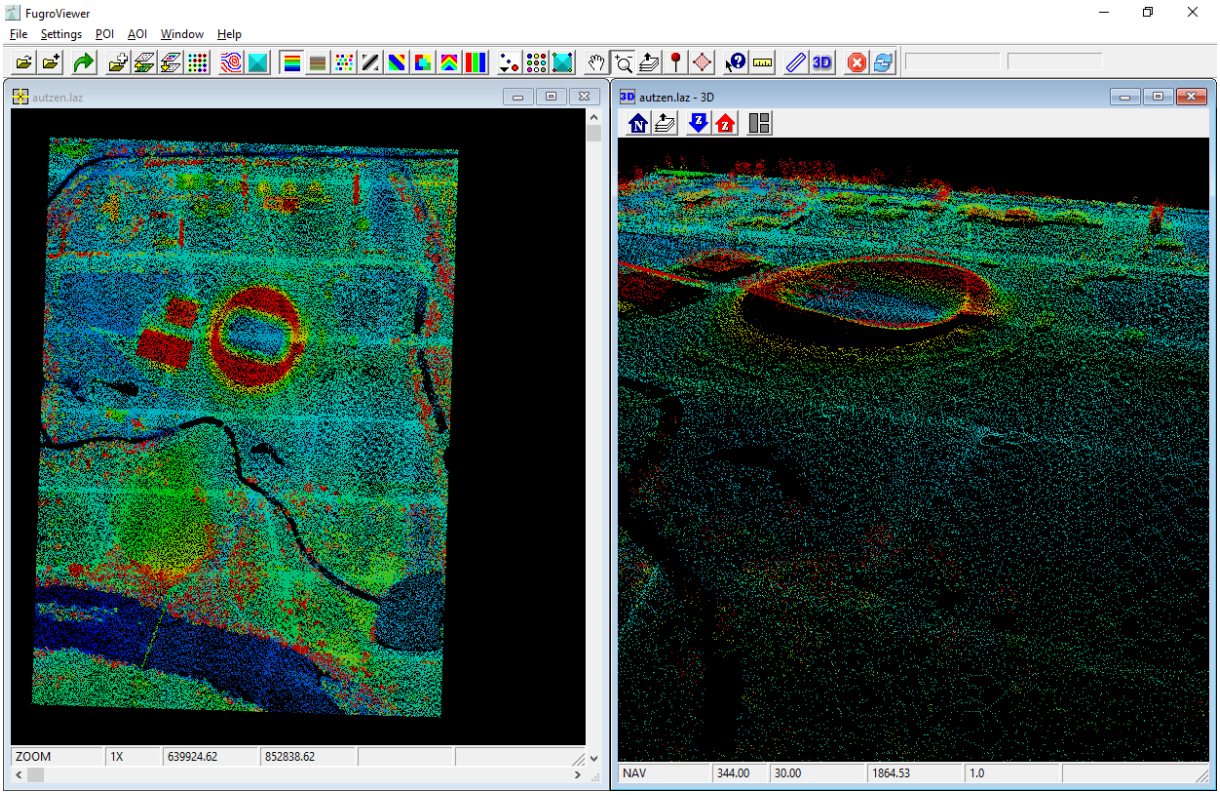
9.2 Clipping data with polygons

This exercise uses PDAL to apply to clip data with polygon geometries.

Note: This exercise is an adaption of the PDAL tutorial (https://pdal.io/tutorial/clipping/index.html#clipping).

9.2.1 Exercise

The `autzen.laz` file is a staple in PDAL and libLAS examples. We will use this file to demonstrate clipping points with a geometry. We're going to clip out the stadium into a new LAS file.



Data preparation

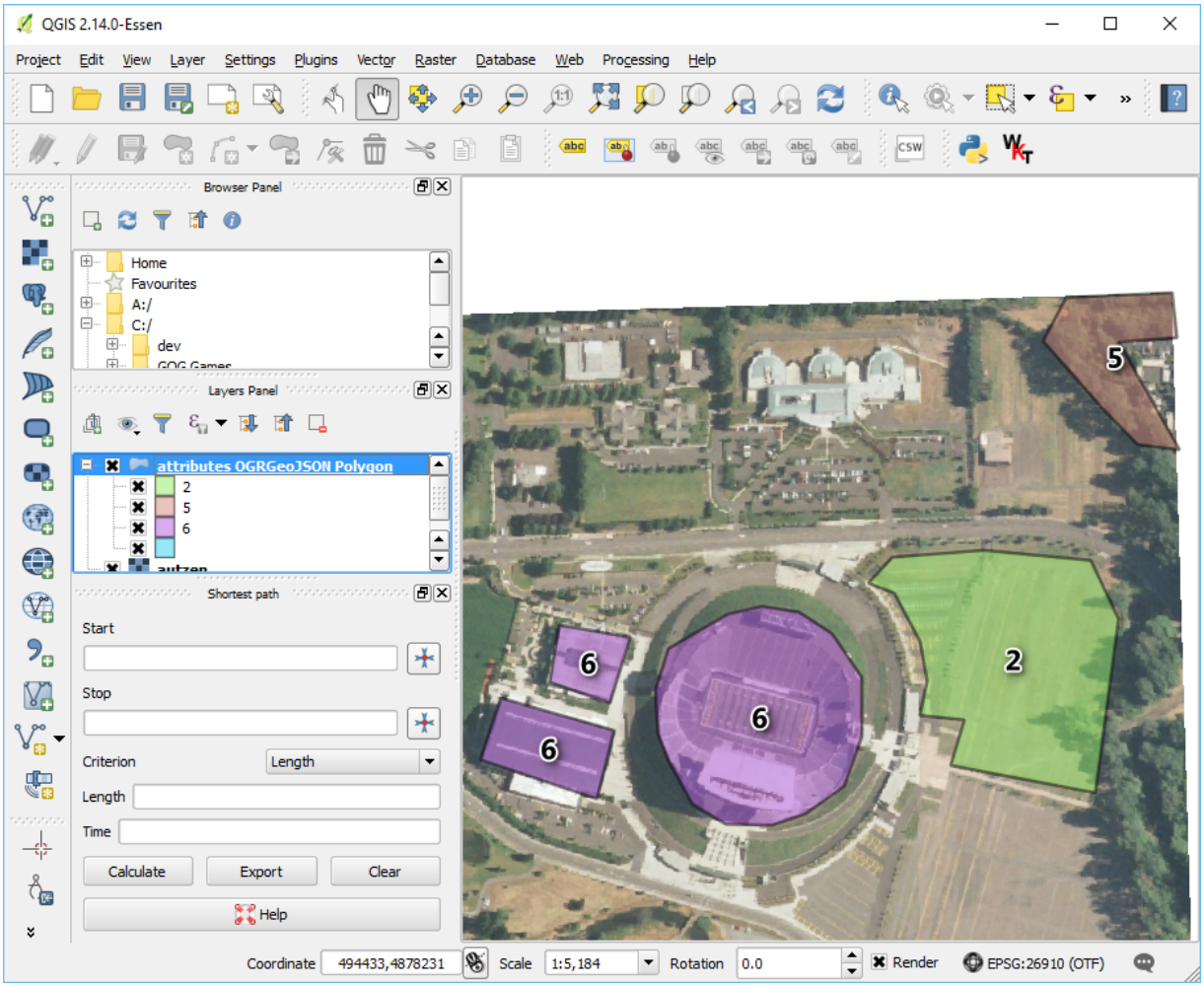
The data are mixed in two different coordinate systems. The LAZ (https://pdal.io/stages/readers.las.html#readers-las) file is in Oregon State Plane Ft. (http://www.oregon.gov/DAS/CIO/GEO/pages/coordination/projections/projections.aspx) and the GeoJSON (http://geojson.org) defining the polygons, `attributes.json`, is in EPSG:4326 (http://epsg.io/4326). We have two options – project the point cloud into the

coordinate system of the attribute polygons, or project the attribute polygons into the coordinate system of the points. The latter is preferable in this case because it will be less math and therefore less computation. To make it convenient, we can utilize OGR (http://www.gdal.org)'s VRT (http://www.gdal.org/drv_vrt.html) capability to reproject the data for us on-the-fly:

```
<OGRVRTDataSource>
    <OGRVRTWarpedLayer>
        <OGRVRTLayer name="OGRGeoJSON">
            <SrcDataSource>./exercises/analysis/clipping/attributes.
↪json</SrcDataSource>
            <SrcLayer>attributes</SrcLayer>
            <LayerSRS>EPSG:4326</LayerSRS>
        </OGRVRTLayer>
        <TargetSRS>+proj=lcc +lat_1=43 +lat_2=45.5 +lat_0=41.75 +lon_
↪0=-120.5 +x_0=399999.9999999999 +y_0=0 +ellps=GRS80 +units=ft +no_
↪defs</TargetSRS>
    </OGRVRTWarpedLayer>
</OGRVRTDataSource>
```

Note: This VRT file is available in your workshop materials in the `./exercises/analysis/clipping/attributes.vrt` file. You will need to open this file, go to line 4 and replace `./` with the correct path for your machine.

A GDAL or OGR VRT is a kind of "virtual" data source definition type that combines a definition of data and a processing operation into a single, readable data stream.



Note: The GeoJSON file does not have an externally-defined coordinate system, so we are explictly setting one with the LayerSRS parameter. If your data does have coordinate system information, you don't need to do that. See the OGR VRT documentation (http://www.gdal.org/drv_vrt.html) for more details.

Pipeline breakdown

```
{
    "pipeline": [
        "./exercises/analysis/clipping/autzen.laz",
        {
            "column": "CLS",
            "datasource": "./exercises/analysis/clipping/attributes.
↪vrt",
            "dimension": "Classification",
```

(continues on next page)

(continued from previous page)

```
            "layer": "OGRGeoJSON",
            "type": "filters.overlay"
        },
        {
            "limits": "Classification[6:6]",
            "type": "filters.range"
        },
        "./exercises/analysis/clipping/stadium.las"
    ]
}
```

Note: This pipeline is available in your workshop materials in the `./exercises/analysis/clipping/clipping.json` file. Remember to replace each of the three occurrences of `./` in this file with the correct location for your machine.

1. Reader

`autzen.laz` is the LASzip (http://laszip.org) file we will clip.

2. filters.overlay (https://pdal.io/stages/filters.overlay.html#filters-overlay)

The filters.overlay (https://pdal.io/stages/filters.overlay.html#filters-overlay) filter allows you to assign values for coincident polygons. Using the VRT we defined in *Data preparation* (page 47), filters.overlay (https://pdal.io/stages/filters.overlay.html#filters-overlay) will assign the values from the `CLS` column to the `Classification` field.

3. filters.range (https://pdal.io/stages/filters.range.html#filters-range)

The attributes in the `attributes.json` file include polygons with values `2`, `5`, and `6`. We will use filters.range (https://pdal.io/stages/filters.range.html#filters-range) to keep points with `Classification` values in the range of `6:6`.

4. Writer

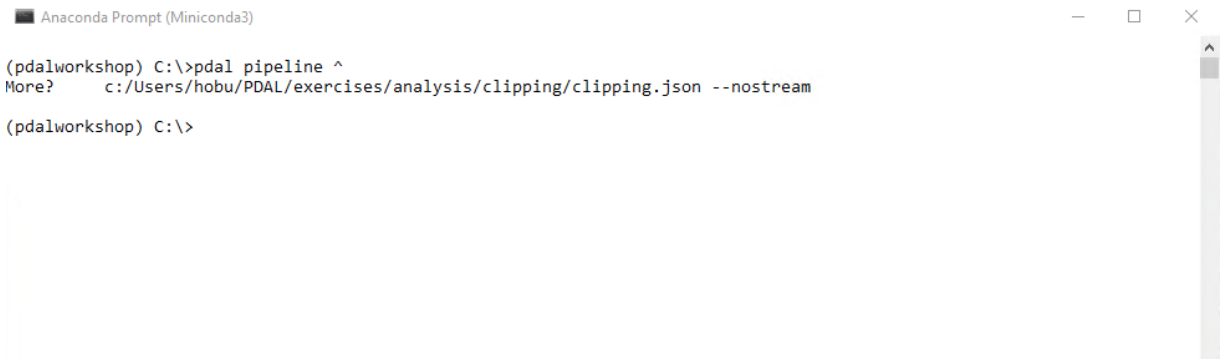
We will write our content back out using a writers.las (https://pdal.io/stages/writers.las.html#writers-las).

Execution

Invoke the following command, substituting accordingly, in your *Conda Shell*:

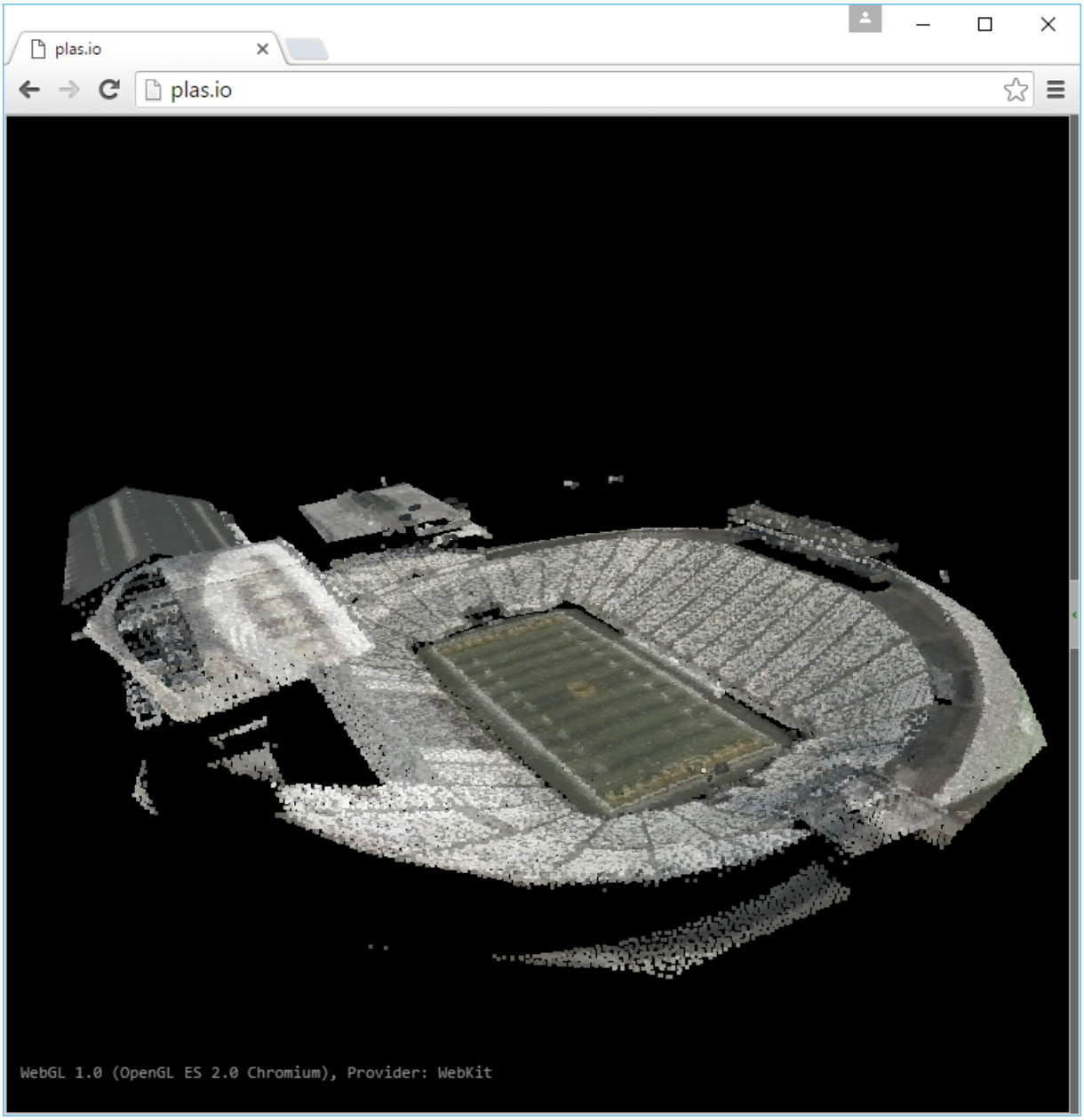
The *–nostream* option disables stream mode. The point-in-polygon check (see notes) performs poorly in stream mode currently.

```
pdal pipeline ./exercises/analysis/clipping/clipping.json --nostream
```



Visualization

Use one of the point cloud visualization tools you installed to take a look at your `./exercises/analysis/clipping/stadium.las` output. In the example below, we opened the file to view it using the http://plas.io website.



9.2.2 Notes

1. filters.overlay (https://pdal.io/stages/filters.overlay.html#filters-overlay) does point-in-polygon checks against every point that is read.
2. Points that are *on* the boundary are included.

9.3 Colorizing points with imagery

This exercise uses PDAL to apply color information from a raster onto point data. Point cloud data, especially LiDAR (https://en.wikipedia.org/wiki/Lidar), do not often have coincident color information. It is possible to project color information onto the points from an imagery source. This makes it convenient to see data in a larger context.

9.3.1 Exercise

PDAL provides a filter (https://pdal.io/stages/filters.html#filters) to apply color information from raster files onto point cloud data. Think of this operation as a top-down projection of RGB color values on the points.

Because this operation is somewhat complex, we are going to use a pipeline to define it.

```
{
    "pipeline": [
        "./exercises/analysis/colorization/uncompahgre.laz",
        {
            "type": "filters.colorization",
            "raster": "./exercises/analysis/colorization/casi-2015-
↪04-29-weekly-mosaic.tif"
        },
        {
            "type": "filters.range",
            "limits": "Red[1:]"
        },
        {
            "type": "writers.las",
            "compression": "true",
            "minor_version": "2",
            "dataformat_id": "3",
            "filename":"./exercises/analysis/colorization/
↪uncompahgre-colored.laz"
        }
    ]
}
```

Note: This JSON file is available in your workshop materials in the `./exercises/analysis/colorization/colorize.json` file. Remember to open this file and replace each occurrence of `./` with the correct path for your machine.

Pipeline breakdown

1. Reader

After our pipeline errata, the first item we define in the pipeline is the point cloud file we're going to read.

```
"./exercises/analysis/colorization/uncompahgre.laz",
```

2. filters.colorization (https://pdal.io/stages/filters.colorization.html#filters-colorization)

The filters.colorization (https://pdal.io/stages/filters.colorization.html#filters-colorization) PDAL filter does most of the work for this operation. We're going to use the default data scaling options. This filter will create PDAL dimensions `Red`, `Green`, and `Blue`.

```
{
    "type": "filters.colorization",
    "raster": "./exercises/analysis/colorization/casi-2015-04-29-
↪weekly-mosaic.tif"
},
```

3. filters.range (https://pdal.io/stages/filters.range.html#filters-range)

A small challenge is the raster will colorize many points with NODATA values. We are going to use the filters.range (https://pdal.io/stages/filters.range.html#filters-range) to filter keep any points that have `Red >= 1`.

```
{
    "type": "filters.range",
    "limits": "Red[1:]"
},
```

4. writers.las (https://pdal.io/stages/writers.las.html#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. These include:

```
{
    "type": "writers.las",
    "compression": "true",
```

(continues on next page)

(continued from previous page)

```
    "minor_version": "2",
    "dataformat_id": "3",
    "filename":"./exercises/colorization/analysis/uncompahgre-
↪colored.laz"
}
```

1. `compression`: LASzip (http://laszip.org) data is ~6x smaller than ASPRS LAS.
2. `minor_version`: We want to make sure to output LAS 1.2, which will provide the widest compatibility with other softwares that can consume LAS.
3. `dataformat_id`: Format 3 supports both time and color information

Note: writers.las (https://pdal.io/stages/writers.las.html#writers-las) provides a number of possible options to control how your LAS files are written.

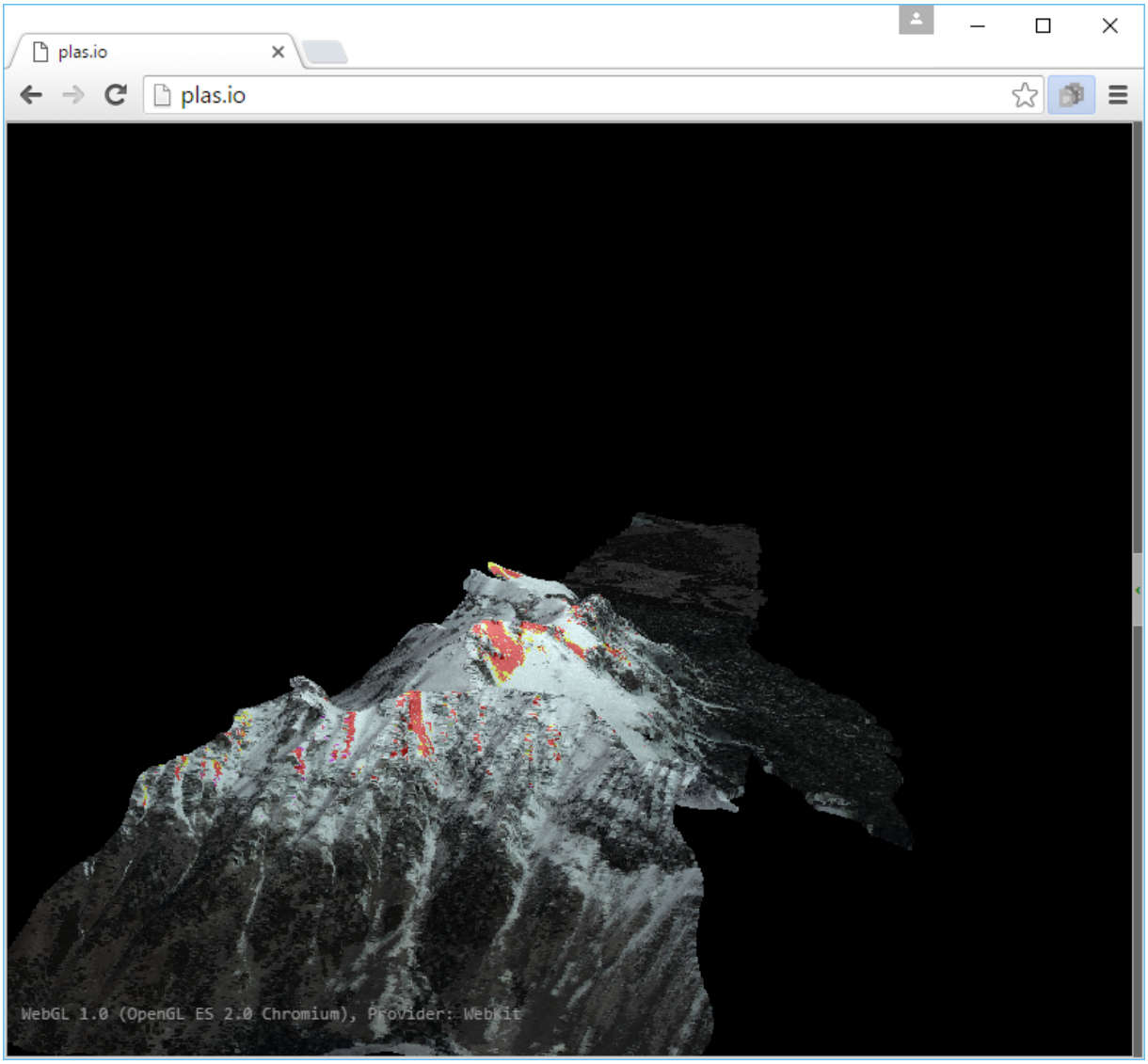
Execution

Invoke the following command, substituting accordingly, in your *Conda Shell*:

```
pdal pipeline ./exercises/analysis/colorization/colorize.json
```

Visualization

Use one of the point cloud visualization tools you installed to take a look at your `uncompahgre-colored.laz` output. In the example below, we simply opened the file using the http://plas.io website.



9.3.2 Notes

1. Applying color information that is not time-coincident with the point cloud data will mean you will see discontinuities.
2. GDAL is used to read the image source. Any GDAL-readable data format can be used.
3. There are performance considerations to be aware of depending on the raster format and type being used. See filters.colorization (https://pdal.io/stages/filters.colorization.html#filters-colorization) for more information.
4. These data are of Uncompahgre Basin (https://en.wikipedia.org/wiki/Uncompahgre_River) courtesy of the NASA Airborne Snow Observatory (http://aso.jpl.nasa.gov/).

9.4 Removing noise

This exercise uses PDAL to remove unwanted noise in an airborne LiDAR collection.

9.4.1 Exercise

PDAL provides the outlier filter (https://pdal.io/stages/filters.outlier.html#filters-outlier) to apply a statistical filter to data.

Because this operation is somewhat complex, we are going to use a pipeline to define it.

```
{
    "pipeline": [
        "./exercises/analysis/denoising/18TWK820985.laz",
        {
            "type": "filters.outlier",
            "method": "statistical",
            "multiplier": 3,
            "mean_k": 8
        },
        {
            "type": "filters.range",
            "limits": "Classification![7:7],Z[-100:3000]"
        },
        {
            "type": "writers.las",
            "compression": "true",
            "minor_version": "2",
            "dataformat_id": "0",
            "filename":"./exercises/analysis/denoising/clean.laz"
        }
    ]
}
```

Note: This pipeline is available in your workshop materials in the `./exercises/analysis/denoising/denoise.json` file.

Pipeline breakdown

1. Reader

After our pipeline errata, the first item we define in the pipeline is the point cloud file we're going to read.

```
"./exercises/analysis/denoising/18TWK820985.laz",
```

2. filters.outlier (https://pdal.io/stages/filters.outlier.html#filters-outlier)

The PDAL outlier filter (https://pdal.io/stages/filters.outlier.html#filters-outlier) does most of the work for this operation.

```
{
    "type": "filters.outlier",
    "method": "statistical",
    "multiplier": 3,
    "mean_k": 8
},
```

3. filters.range (https://pdal.io/stages/filters.range.html#filters-range)

At this point, the outliers have been classified per the LAS specification as low/noise points with a classification value of 7. The range filter (https://pdal.io/stages/filters.range.html#filters-range) can remove these noise points by constructing a range (https://pdal.io/stages/filters.range.html#ranges) with the value `Classification![7:7]`, which passes every point with a `Classification` value **not** equal to 7.

Even with the filters.outlier (https://pdal.io/stages/filters.outlier.html#filters-outlier) operation, there is still a cluster of points with extremely negative `Z` values. These are some artifact or miscomputation of processing, and we don't want these points. We can construct another range (https://pdal.io/stages/filters.range.html#ranges) to keep only points that are within the range $-100 <= Z <= 3000$.

Both ranges (https://pdal.io/stages/filters.range.html#ranges) are passed as a comma-separated list to the range filter (https://pdal.io/stages/filters.range.html#filters-range) via the `limits` option.

```
{
    "type": "filters.range",
    "limits": "Classification![7:7],Z[-100:3000]"
},
```

4. writers.las (https://pdal.io/stages/writers.las.html#writers-las)

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

```
{
    "type": "writers.las",
    "compression": "true",
    "minor_version": "2",
    "dataformat_id": "0",
    "filename":"./exercises/analysis/denoising/clean.laz"
}
```

1. `compression`: LASzip (http://laszip.org) data is ~6x smaller than ASPRS LAS.
2. `minor_version`: We want to make sure to output LAS 1.2, which will provide the widest compatibility with other softwares that can consume LAS.
3. `dataformat_id`: Format 3 supports both time and color information

Note: writers.las (https://pdal.io/stages/writers.las.html#writers-las) provides a number of possible options to control how your LAS files are written.

Execution

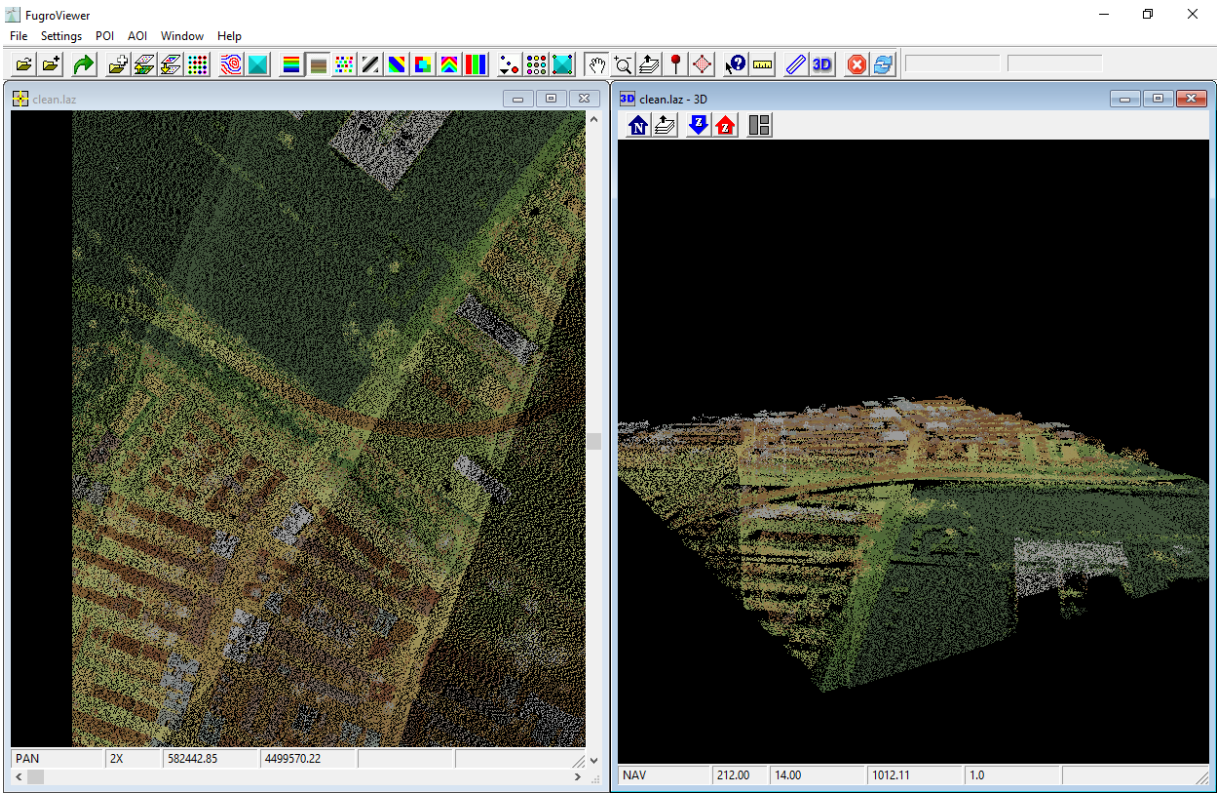
Invoke the following command, substituting accordingly, in your ‘ Shell‘:

```
pdal pipeline ./exercises/analysis/denoising/denoise.json
```

```
(pdalworkshop) $ pdal density ./exercises/analysis/density/uncompahgre.laz  \
> -o ./exercises/analysis/density/density.sqlite \
> -f SQLite
(pdalworkshop) $
```

Visualization

Use one of the point cloud visualization tools you installed to take a look at your `clean.laz` output. In the example below, we simply opened the file using the Fugro Viewer (http://www.fugroviewer.com/)



9.4.2 Notes

1. Control the aggressiveness of the algorithm with the `mean_k` parameter.
2. filters.outlier (https://pdal.io/stages/filters.outlier.html#filters-outlier) requires the entire set in memory to process. If you have really large files, you are going to need to split (https://pdal.io/stages/filters.splitter.html#filters-splitter) them in some way.

9.5 Visualizing acquisition density

This exercise uses PDAL to generate a density surface. You can use this surface to summarize acquisition quality.

9.5.1 Exercise

PDAL provides an application (https://pdal.io/apps/density.html#density-command) to compute a vector field of hexagons computed with filters.hexbin (https://pdal.io/stages/filters.hexbin.html#filters-hexbin). It is a kind of simple interpolation, which we will use for visualization in QGIS (http://qgis.org).

Command

Invoke the following command, substituting accordingly, in your ‘ Shell‘:

```
pdal density ./exercises/analysis/density/uncompahgre.laz  \
-o ./exercises/analysis/density/density.sqlite \
-f SQLite
```

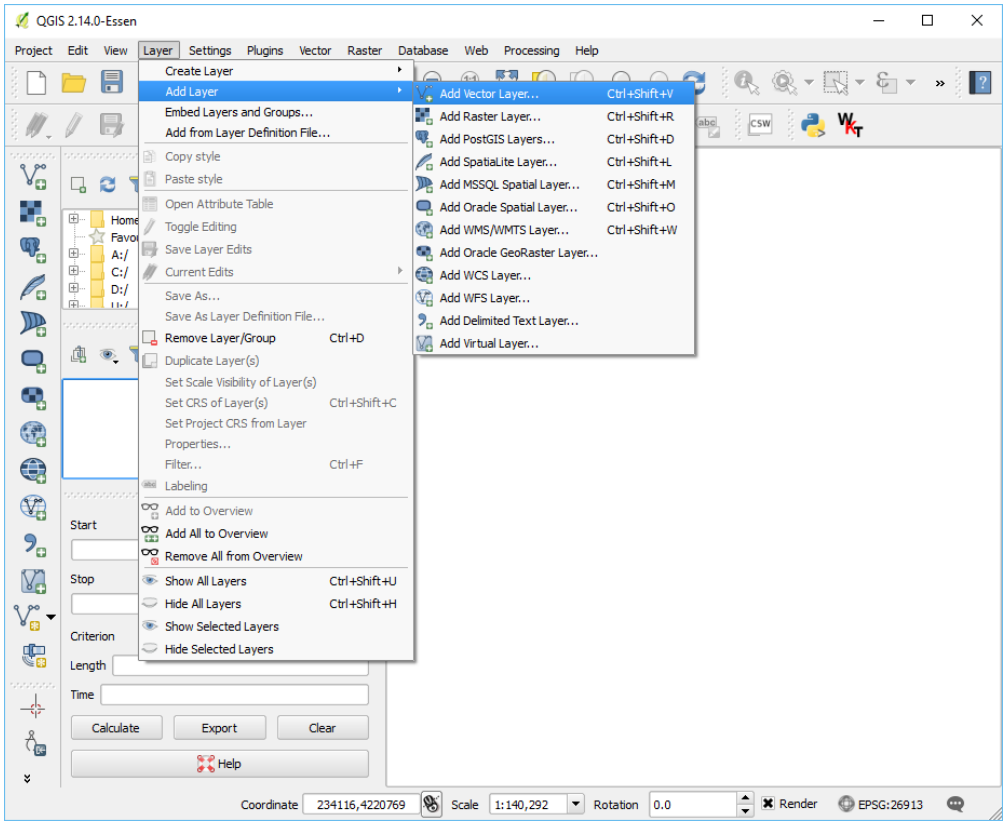
```
pdal density ./exercises/analysis/density/uncompahgre.laz  ^
-o ./exercises/analysis/density/density.sqlite ^
-f SQLite
```

```
(pdalworkshop) C:\>pdal density ^
More?    c:/Users/hobu/PDAL/exercises/analysis/density/uncompahgre.laz ^
More?    -o c:/Users/hobu/PDAL/exercises/analysis/density/density.sqlite ^
More?    -f SQLite

(pdalworkshop) C:\>
```

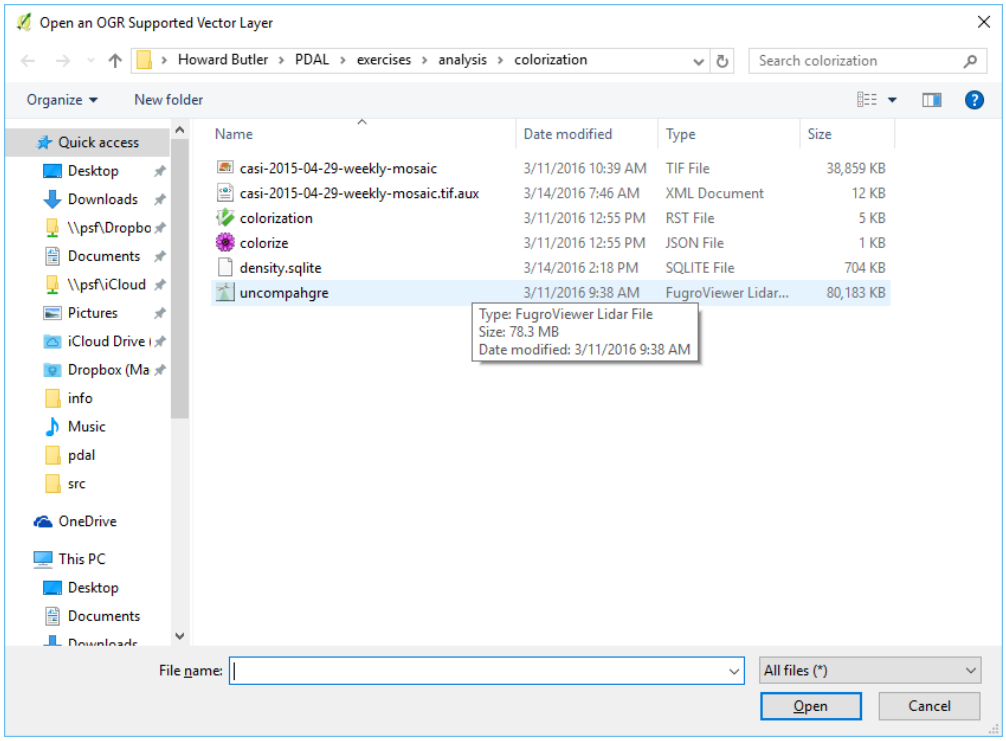
Visualization

The command uses GDAL to output a SQLite (http://sqlite.org) file containing hexagon polygons. We will now use QGIS (http://qgis.org) to visualize them.

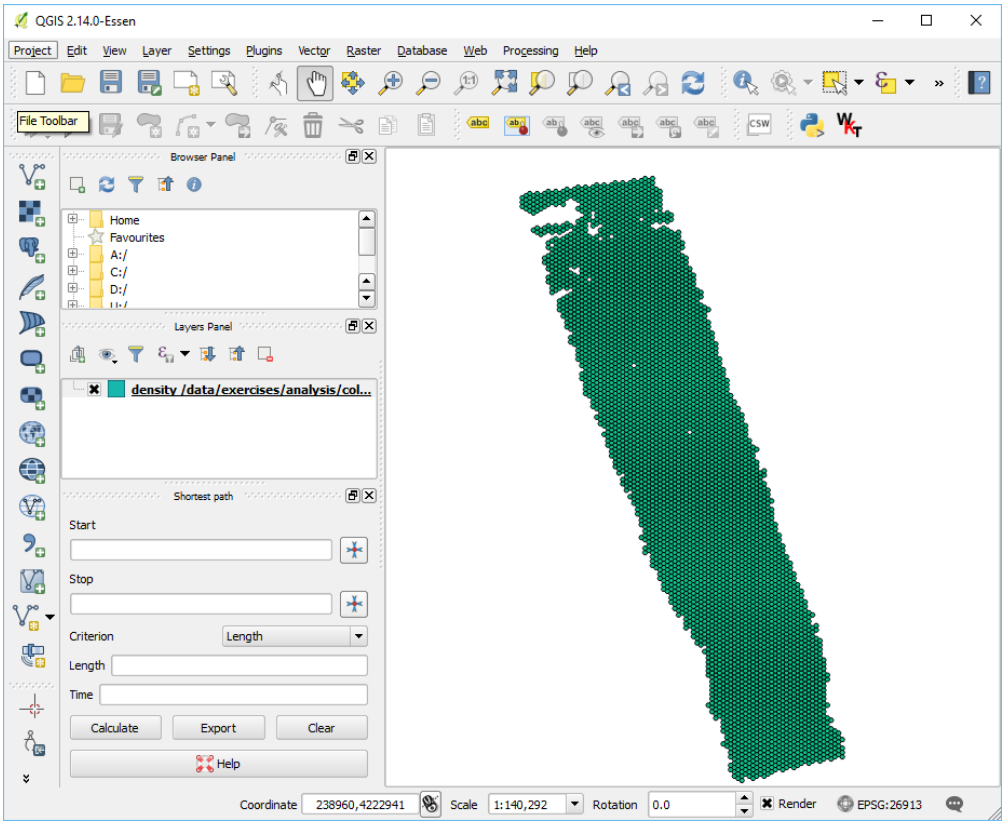
1. Add a vector layer



2. Navigate to the output directory

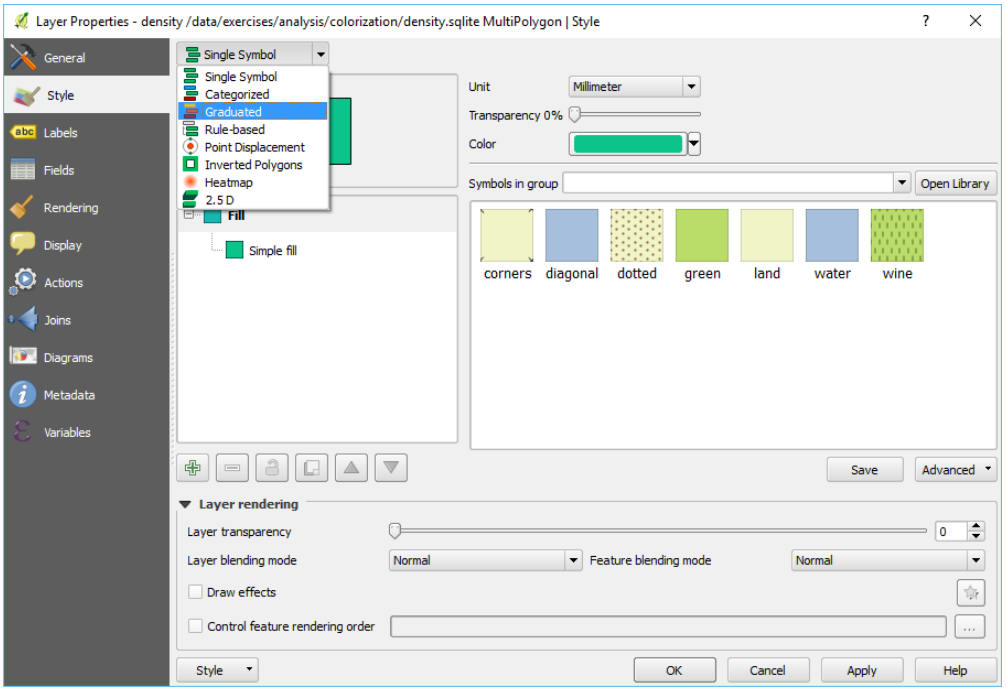


3. Add the `density.sqlite` file to the view

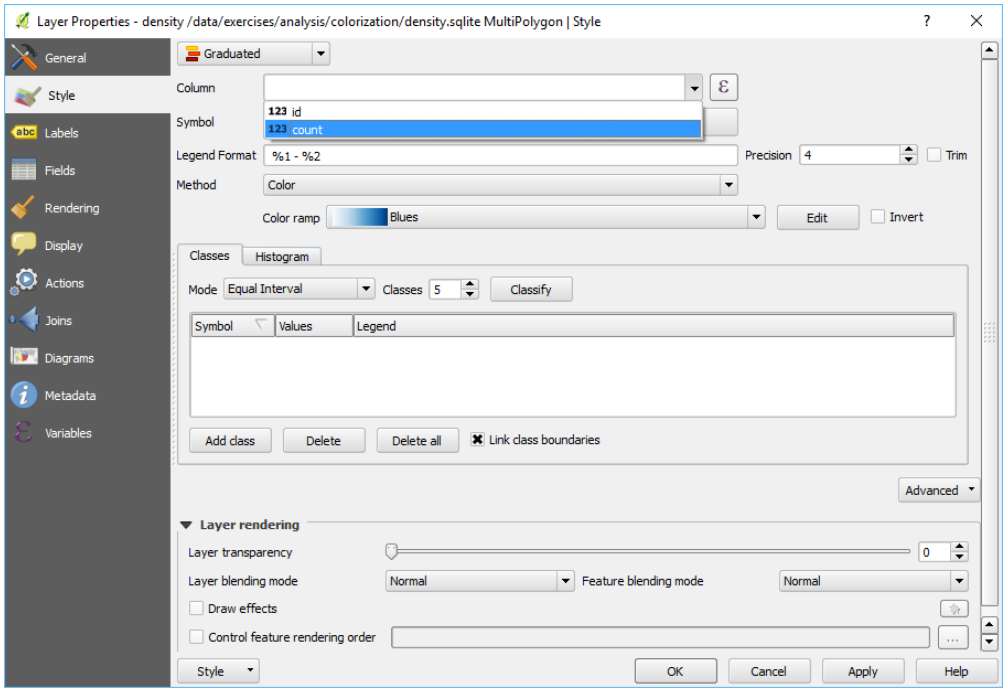


4. Right click on the `density.sqlite` layer in the *Layers* panel and then choose `Properties`.

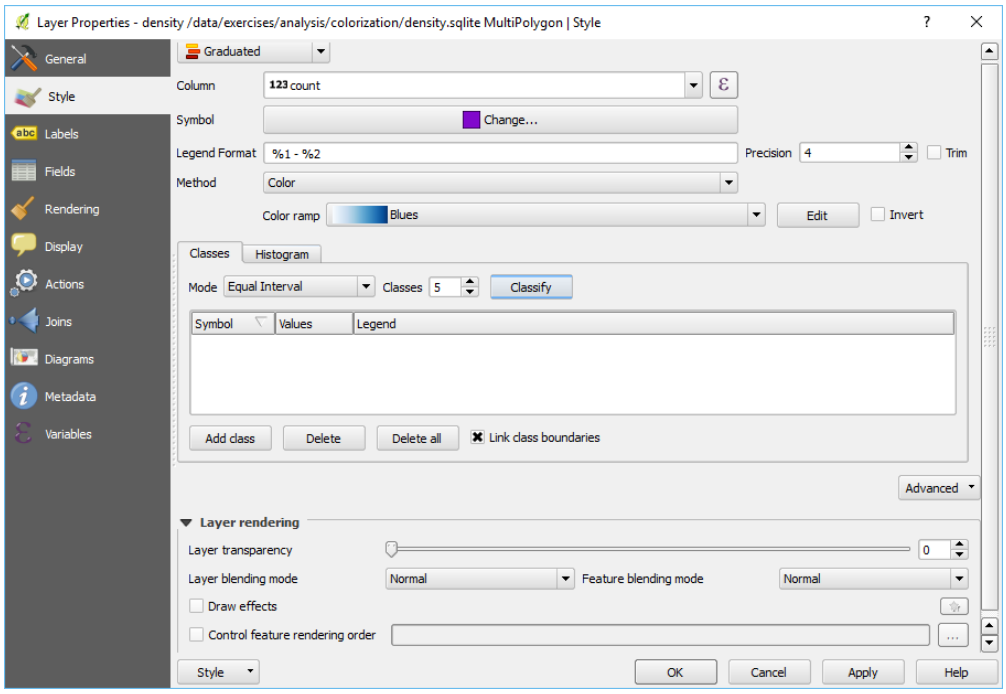
5. Pick the `Graduated` drop down



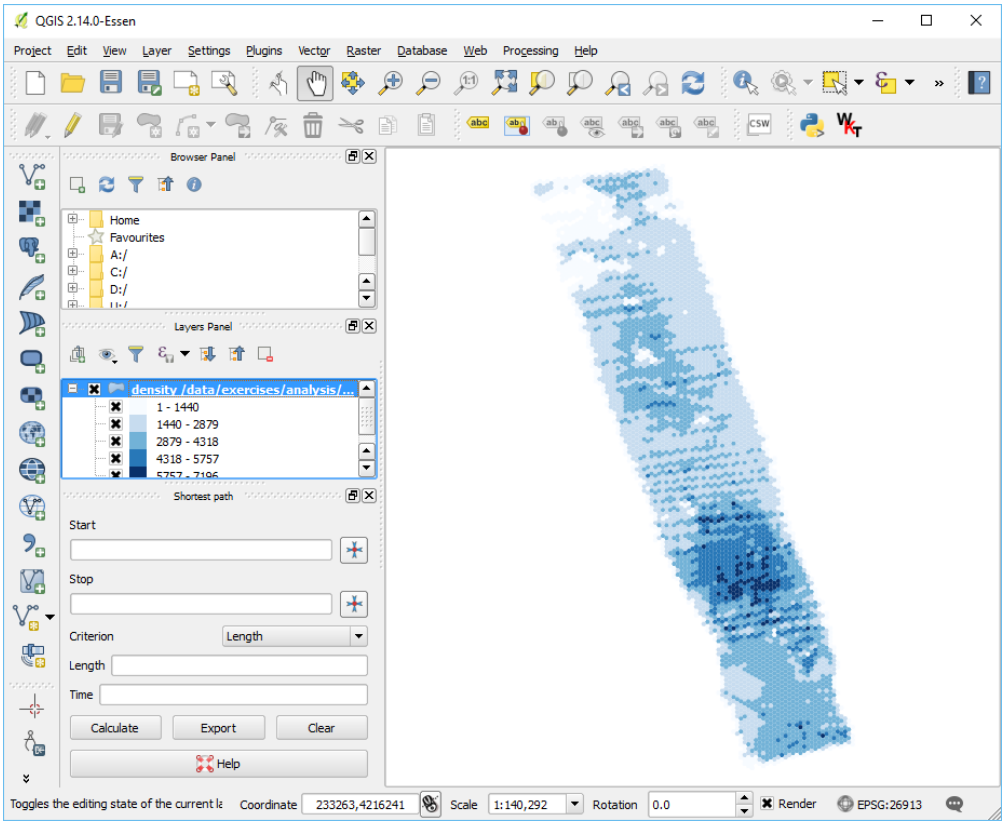
6. Choose the `Count` column to visualize



7. Choose the `Classify` button to add intervals



8. Adjust the visualization as desired



9.5.2 Notes

1. You can control how the density hexagon surface is created by using the options in filters.hexbin (https://pdal.io/stages/filters.hexbin.html#filters-hexbin).

   The following settings will use a hexagon edge size of 24 units.

   ```
   --filters.hexbin.edge_size=24
   ```

2. You can generate a contiguous boundary using PDAL (https://pdal.io/)'s tindex (https://pdal.io/apps/tindex.html#tindex-command).

9.6 Thinning

This exercise uses PDAL to subsample or thin point cloud data. This might be done to accelerate processing (less data), normalize point density, or ease visualization.

9.6.1 Exercise

As we showed in the *Visualizing acquisition density* (page 60) exercise, the points in the *uncompahgre.laz* file are not evenly distributed across the entire collection. While we will not get into reasons why that particular property is good or bad, we note there are three different sampling strategies we could choose. We can attempt to preserve shape, we can try to randomly sample, and we can attempt to normalize posting density. PDAL provides capability for all three:

- Poisson using the filters.sample (https://pdal.io/stages/filters.sample.html#filters-sample)
- Random using a combination of filters.decimation (https://pdal.io/stages/filters.decimation.html#filters-decimation) and filters.randomize (https://pdal.io/stages/filters.randomize.html#filters-randomize)
- Voxel using filters.voxelgrid (https://pdal.io/stages/filters.voxelgrid.html#filters-voxelgrid)

In this exercise, we are going to thin with the Poisson method, but the concept should operate similarly for the filters.voxelgrid (https://pdal.io/stages/filters.voxelgrid.html#filters-voxelgrid) approach too.

Command

Invoke the following command, substituting accordingly, in your *Conda Shell*:

```
pdal translate ./exercises/analysis/density/uncompahgre.laz \
./exercises/analysis/thinning/uncompahgre-thin.laz \
sample --filters.sample.radius=20
```

```
pdal translate ./exercises/analysis/density/uncompahgre.laz ^
./exercises/analysis/thinning/uncompahgre-thin.laz ^
sample --filters.sample.radius=20
```

```
(pdalworkshop) $pdal translate ./exercises/analysis/density/uncompahgre.laz \
> ./exercises/analysis/thinning/uncompahgre-thin.laz \
> sample --filters.sample.radius=20
(pdalworkshop) $
```

Visualization

http://plas.io has the ability to switch on/off multiple data sets, and we are going to use that ability to view both the `uncompahgre.laz` and the `uncompahgre-thin.laz` file.

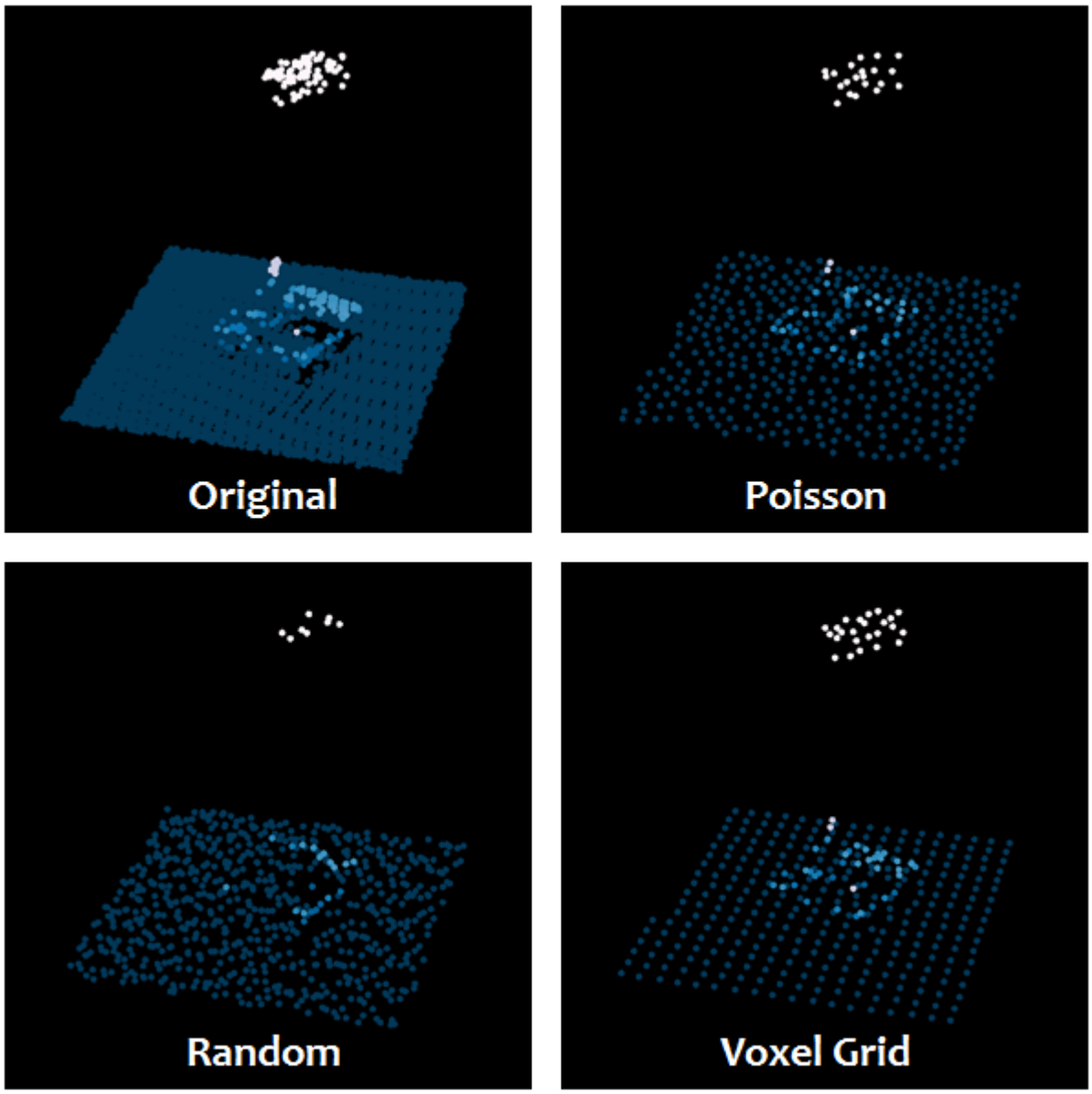


Fig. 1: Thinning strategies available in PDAL

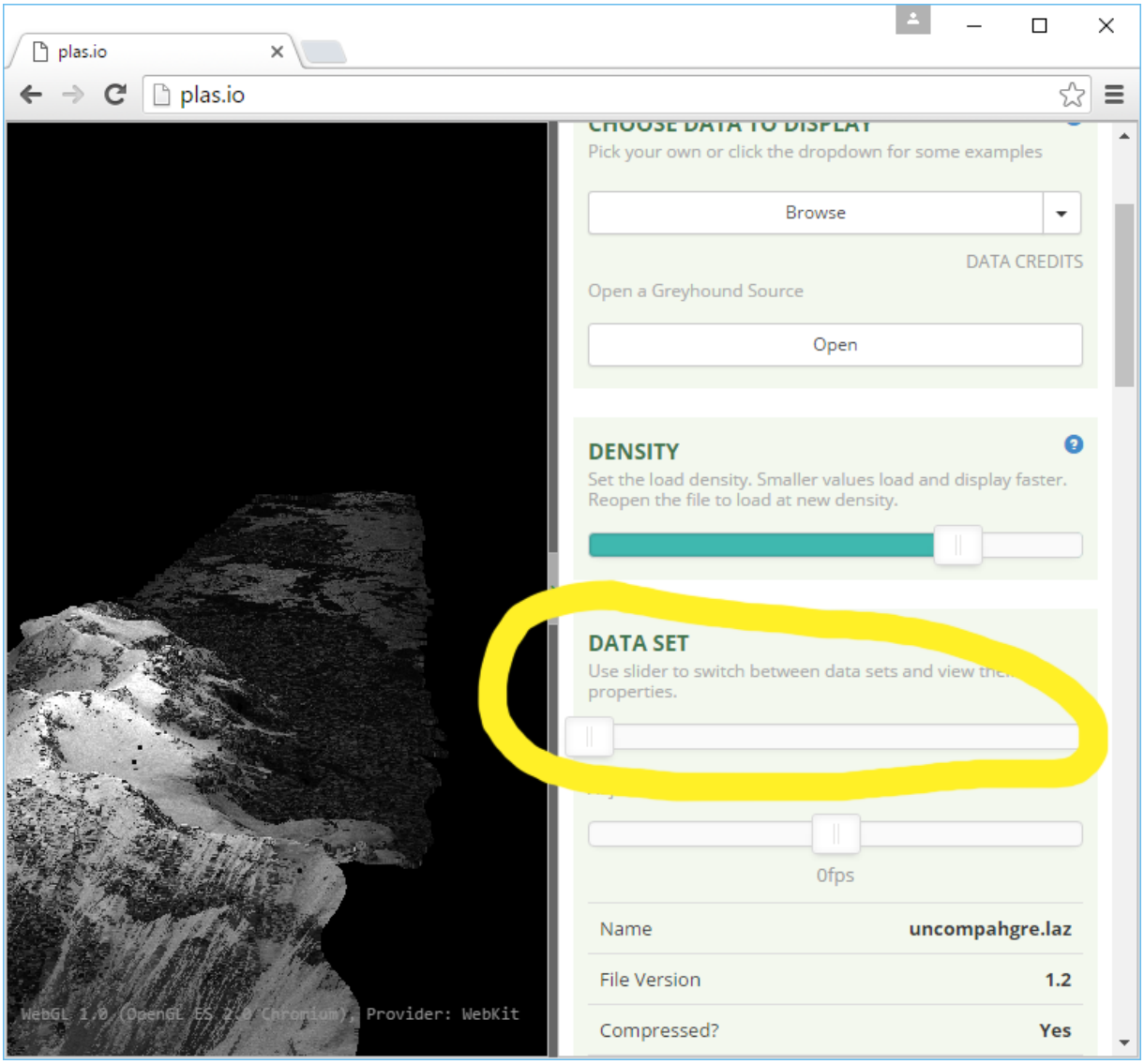


Fig. 2: Selecting multiple data sets in http://plas.io

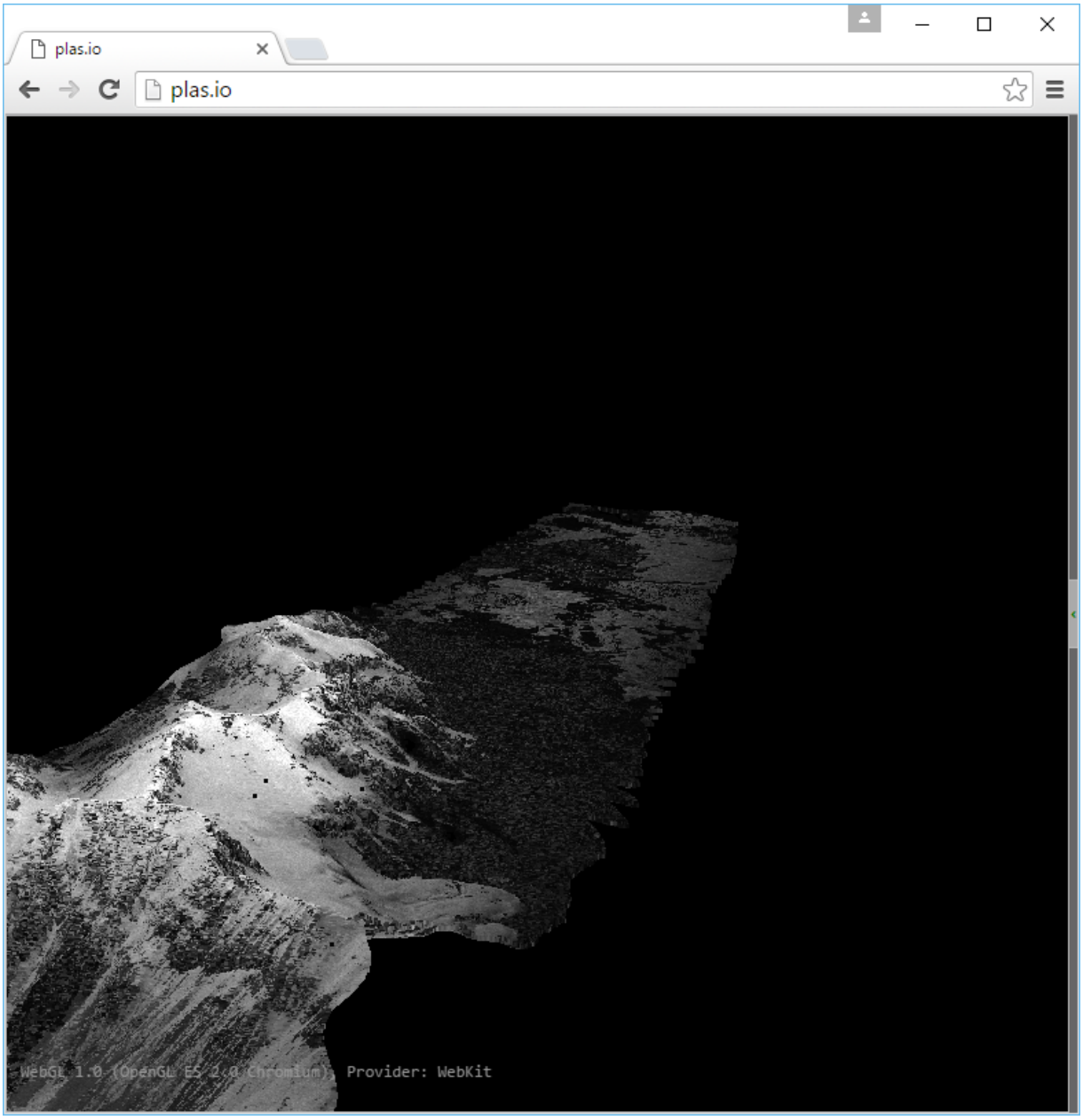


Fig. 3: Full resolution Uncompahgre data set

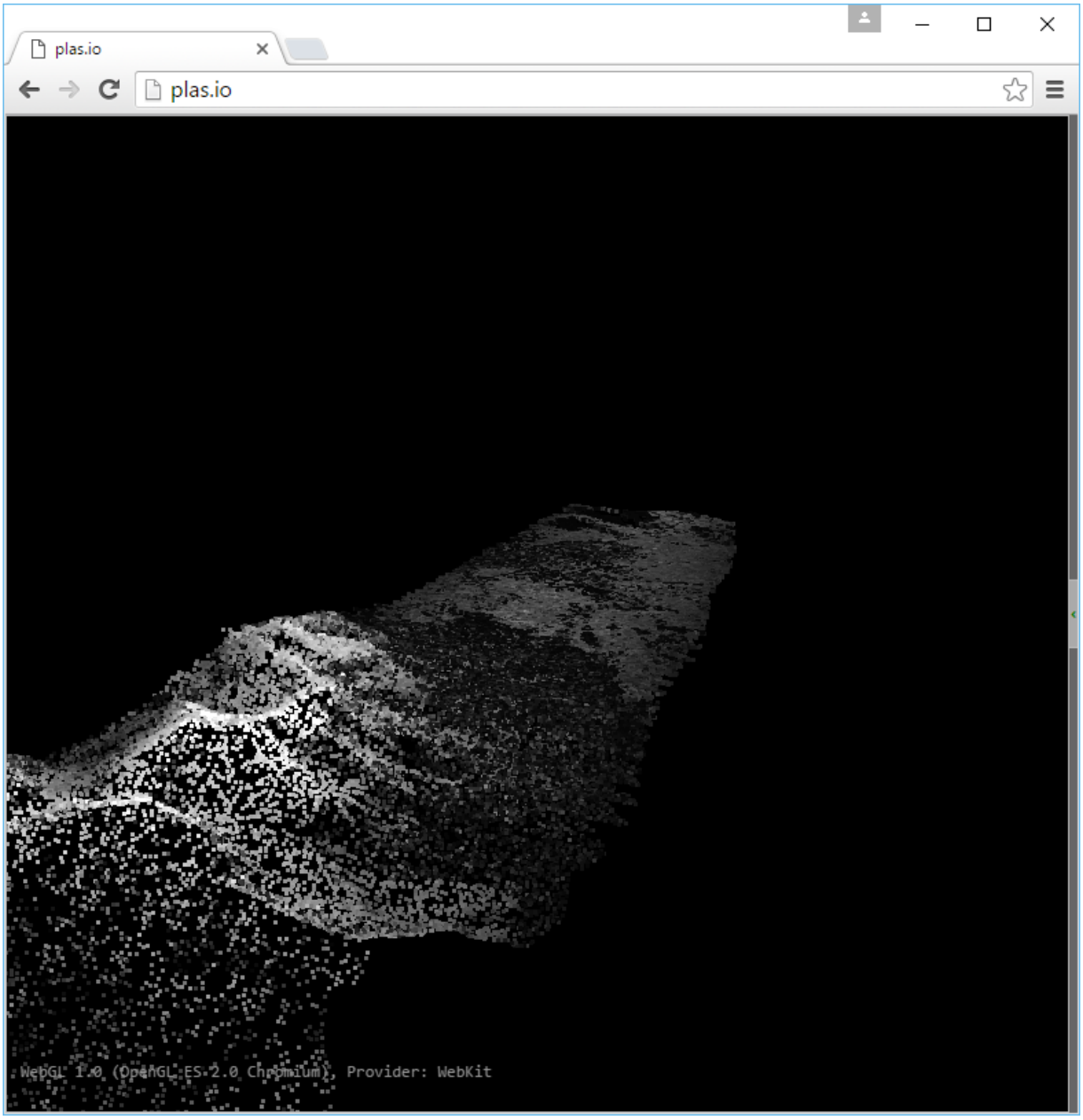


Fig. 4: Uncompahgre thinned at a radius of 20m

9.6.2 Notes

1. Poisson sampling is non-destructive. Points that are filtered with filters.sample (https://pdal.io/stages/filters.sample.html#filters-sample) will retain all attribute information.

9.7 Identifying ground

This exercise uses PDAL to classify ground returns using the *Simple Morphological Filter (SMRF)* technique.

Note: This excerise is an adaptation of the Identifying ground returns using ProgressiveMorphologicalFilter segmentation (https://pdal.io/tutorial/pcl_ground/index.html#pcl-ground) tutorial on the PDAL website by Brad Chambers. You can find more detail and example invocations there.

9.7.1 Exercise

The primary input for Digital Terrain Model (https://en.wikipedia.org/wiki/Digital_elevation_model) generation is a point cloud with ground vs. not-ground classifications. In this example, we will use an algorithm provided by PDAL, the *Simple Morphological Filter* technique to generate a ground surface.

See also:

You can read more about the specifics of the SMRF algorithm from *[Pingle2013]_*

Command

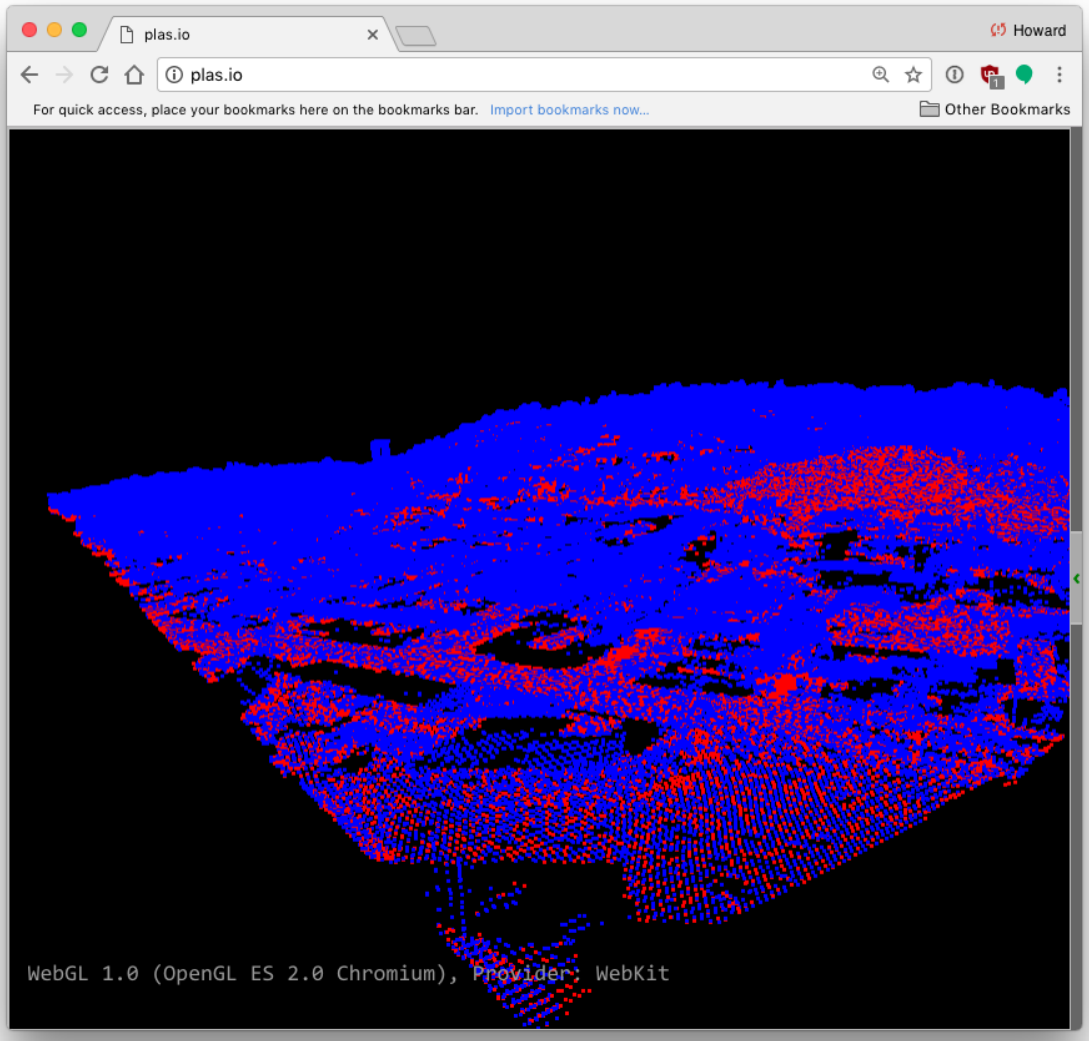
Invoke the following command, substituting accordingly, in your *Conda Shell*:

```
pdal translate ./exercises/analysis/ground/CSite1_orig-utm.laz \
-o ./exercises/analysis/ground/ground.laz \
smrf \
-v 4
```

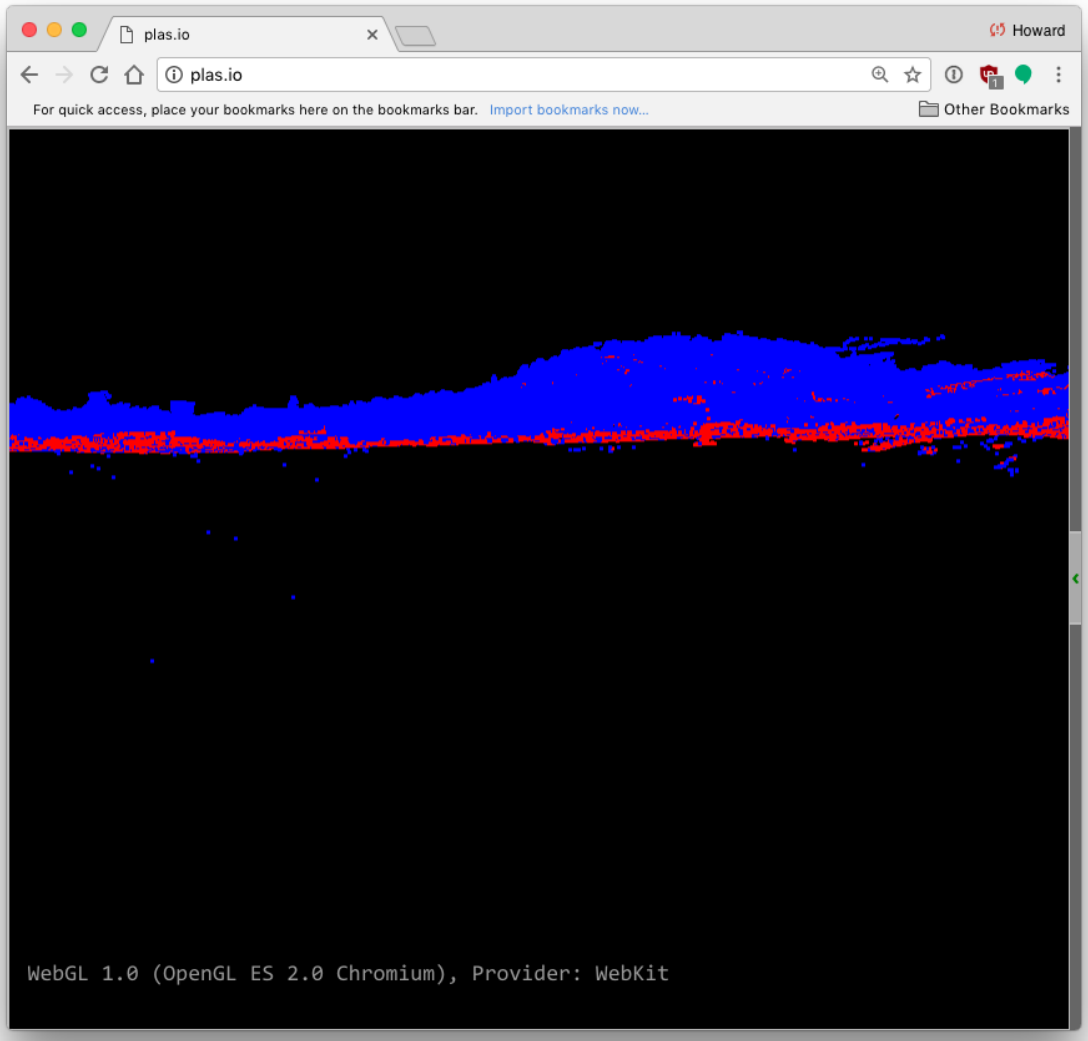
```
pdal translate ./exercises/analysis/ground/CSite1_orig-utm.laz ^
-o ./exercises/analysis/ground/ground.laz ^
smrf ^
-v 4
```

```
(pdalworkshop) $pdal translate ./exercises/analysis/ground/CSite1_orig-utm.laz \
> -o ./exercises/analysis/ground/ground.laz \
> smrf \
> -v 4
(PDAL Debug) Debugging...
(pdal translate readers.las Debug) GDAL debug: OGRSpatialReference::Validate: No root pointer.
(pdal translate readers.las Debug) GDAL debug: OGRSpatialReference::Validate: No root pointer.
(pdal translate readers.las Debug) GDAL debug: OGRSpatialReference::Validate: No root pointer.
(pdal translate Debug) Executing pipeline in standard mode.
(pdal translate filters.smrf Debug) progressiveFilter: radius = 1       767170 ground   4631 non-ground (0.60%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 1       576488 ground   195313 non-ground       (25.31%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 2       519149 ground   252652 non-ground       (32.74%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 3       492155 ground   279646 non-ground       (36.23%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 4       473156 ground   298645 non-ground       (38.69%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 5       454156 ground   317645 non-ground       (41.16%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 6       433943 ground   337858 non-ground       (43.78%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 7       414492 ground   357309 non-ground       (46.30%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 8       399457 ground   372344 non-ground       (48.24%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 9       388449 ground   383352 non-ground       (49.67%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 10      382942 ground   388859 non-ground       (50.38%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 11      379683 ground   392118 non-ground       (50.81%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 12      377098 ground   394703 non-ground       (51.14%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 13      374993 ground   396808 non-ground       (51.41%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 14      373622 ground   398179 non-ground       (51.59%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 15      372487 ground   399314 non-ground       (51.74%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 16      372248 ground   399553 non-ground       (51.77%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 17      371884 ground   399917 non-ground       (51.82%)
(pdal translate filters.smrf Debug) progressiveFilter: radius = 18      371674 ground   400127 non-ground       (51.84%)
(pdal translate writers.las Debug) Wrote 1366408 points to the LAS file
(pdalworkshop) $
```

As we can see, the algorithm does a great job of discriminating the points, but there's a few issues.



There's noise underneath the main surface that will cause us trouble when we generate a terrain surface.



Filtering

We do not yet have a satisfactory surface for generating a DTM. When we visualize the output of this ground operation, we notice there's still some noise. We can stack the call to SMRF with a call to a the *filters.outlier* technique we learned about in denoising.

1. Let us start by removing the non-ground data to just view the ground data:

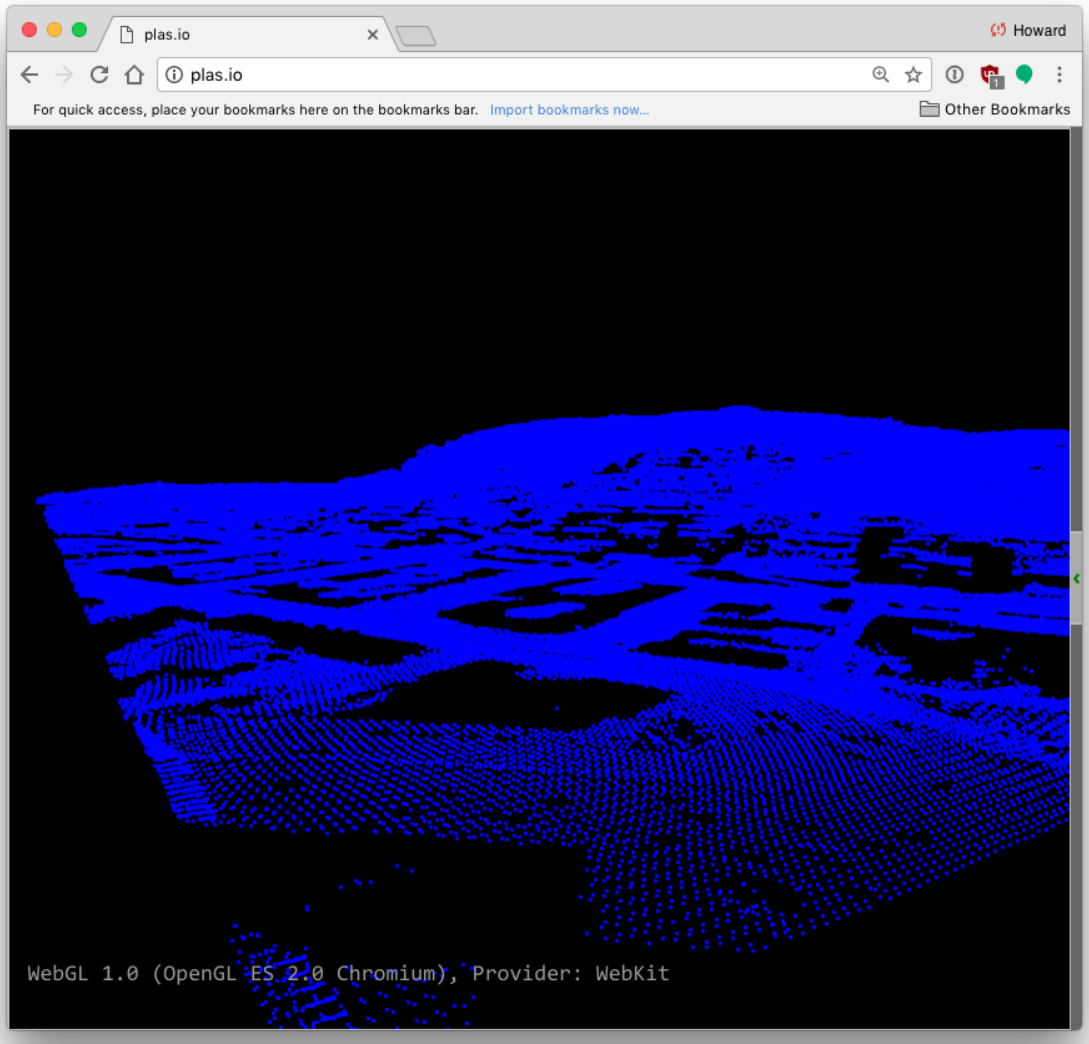
```
pdal translate \
./exercises/analysis/ground/CSite1_orig-utm.laz \
-o ./exercises/analysis/ground/ground.laz \
smrf range \
--filters.range.limits="Classification[2:2]" \
```

(continues on next page)

(continued from previous page)

```
-v 4
```

```
pdal translate ^
./exercises/analysis/ground/CSite1_orig-utm.laz ^
-o ./exercises/analysis/ground/ground.laz ^
smrf range ^
--filters.range.limits="Classification[2:2]" ^
-v 4
```



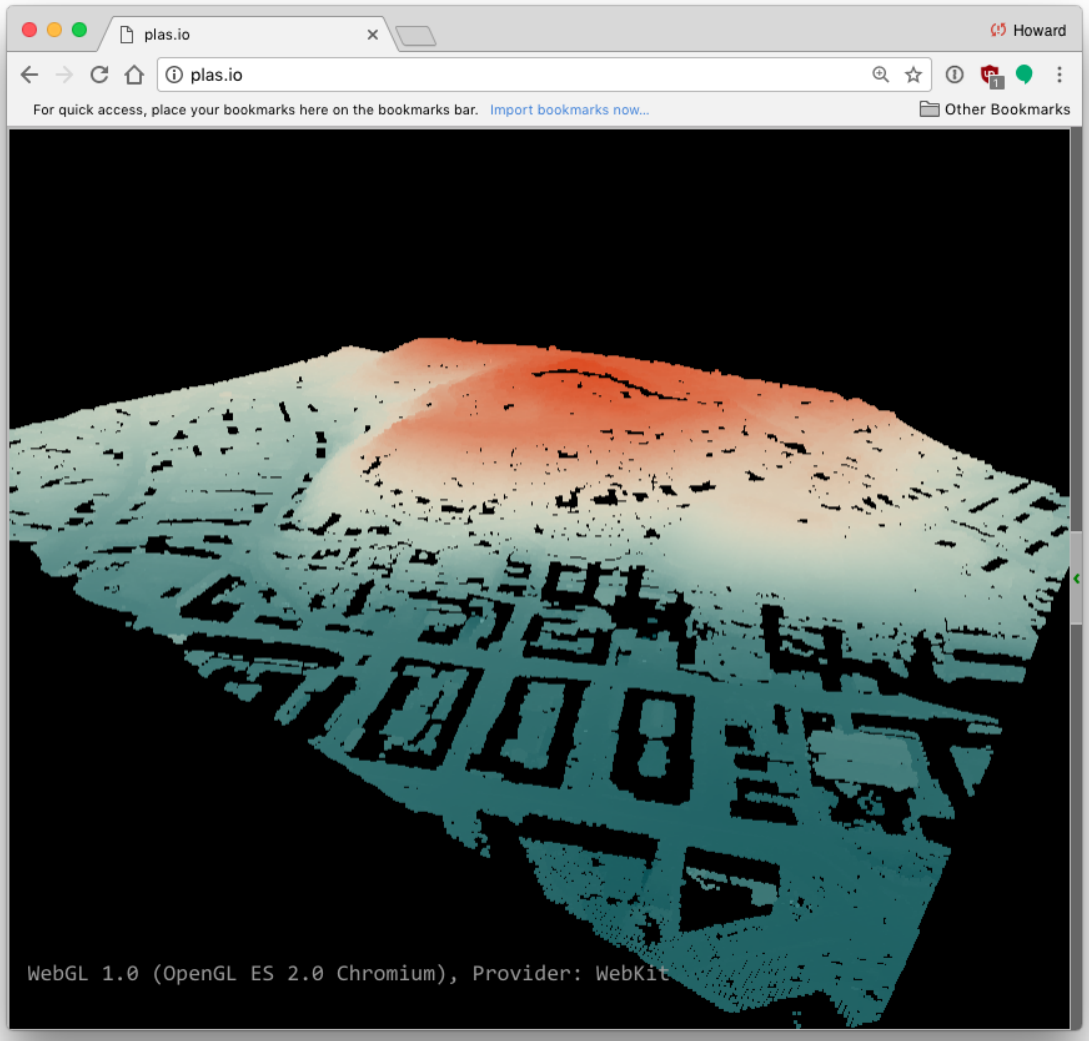
2. Now we will instead use the translate (https://pdal.io/apps/translate.html#translate-command) command to stack the filters.outlier (https://pdal.io/stages/filters.outlier.html#filters-outlier) and filters.smrf (https://pdal.io/stages/filters.smrf.html#filters-smrf) stages:

```
pdal translate ./exercises/analysis/ground/CSite1_orig-utm.laz \
-o ./exercises/analysis/ground/denoised-ground-only.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
```

```
pdal translate ./exercises/analysis/ground/CSite1_orig-utm.laz ^
-o ./exercises/analysis/ground/denoised-ground-only.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
```

In this invocation, we have more control over the process. First the outlier filter merely classifies outliers with a `Classification` value of 7. These outliers are then ignored during SMRF processing with the `ignore` option. Finally, we add a range filter to extract only the ground returns (i.e., `Classification` value of 2).

The result is a more accurate representation of the ground returns.



9.8 Generating a DTM

This exercise uses PDAL to generate an elevation model surface using the output from the *Identifying ground* (page 71) exercise, PDAL's writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) operation, and GDAL (http://gdal.org/) to generate an elevation and hillshade surface from point cloud data.

9.8.1 Exercise

Note: The primary input for Digital Terrain Model (https://en.wikipedia.org/wiki/Digital_elevation_model) generation is a point cloud with ground classifications. We created this file, called `denoised-ground-only.laz`, in the *Identifying ground* (page 71) exercise. Please produce that file by following that exercise before starting this one.

Command

Invoke the following command, substituting accordingly, in your *Conda Shell*:

PDAL capability to generate rasterized output is provided by the writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) stage. There is no application (https://pdal.io/apps/index.html#apps) to drive this stage, and we must use a pipeline.

Pipeline breakdown

```
{
    "pipeline": [
        "./exercises/analysis/ground/denoised-ground-only.laz",
        {
            "filename":"./exercises/analysis/dtm/dtm.tif",
            "gdaldriver":"GTiff",
            "output_type":"all",
            "resolution":"2.0",
            "type": "writers.gdal"
        }
    ]
}
```

Note: This pipeline is available in your workshop materials in the `./exercises/analysis/dtm/dtm.json` file. Make sure to edit the filenames to match your paths.

1. Reader

`denoised-ground-only` is the LASzip (http://laszip.org) file we will clip. You should have created this output as part of the *Identifying ground* (page 71) exercise.

2. writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal)

The writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) writer that bins the point cloud data into an elevation surface.

Execution

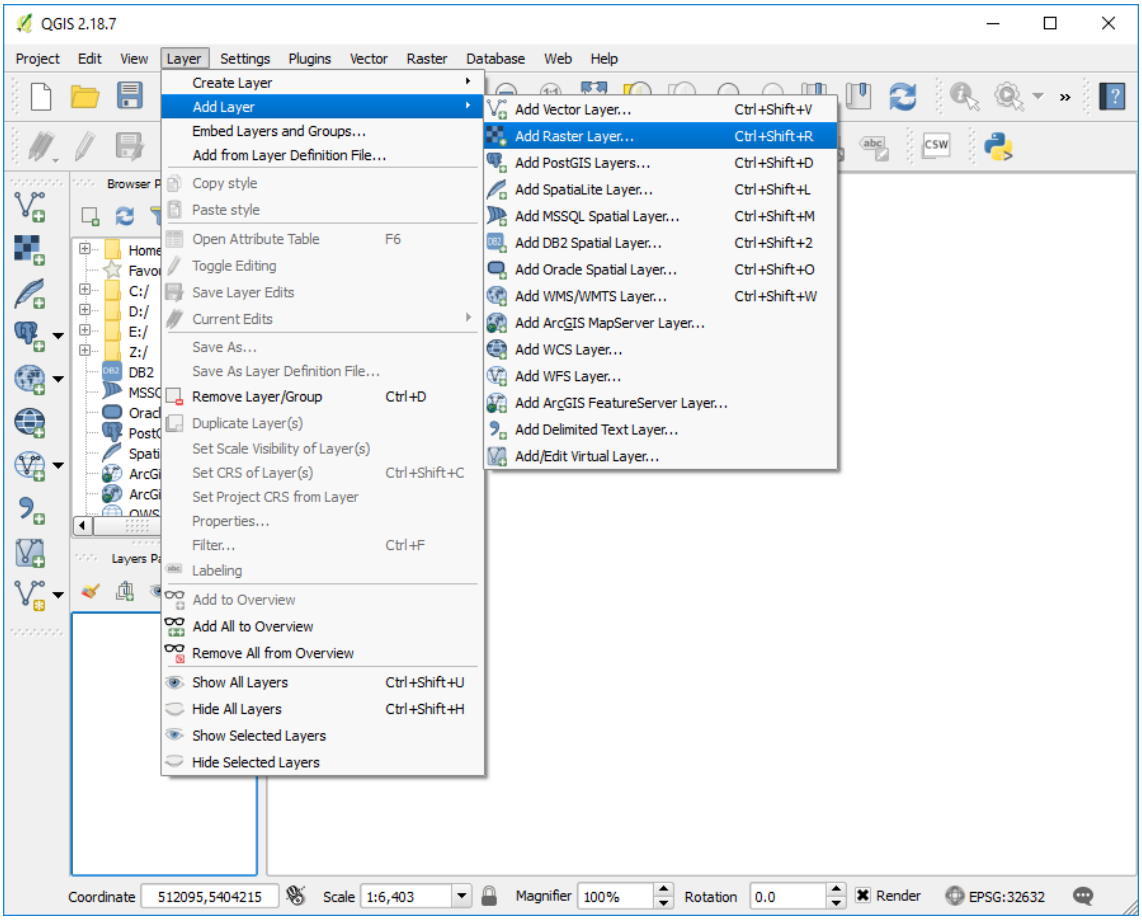
```
1 pdal pipeline ./exercises/analysis/dtm/gdal.json
```

```
(pdalworkshop) $ pdal pipeline ./exercises/analysis/dtm/gdal.json
(pdalworkshop) $
```

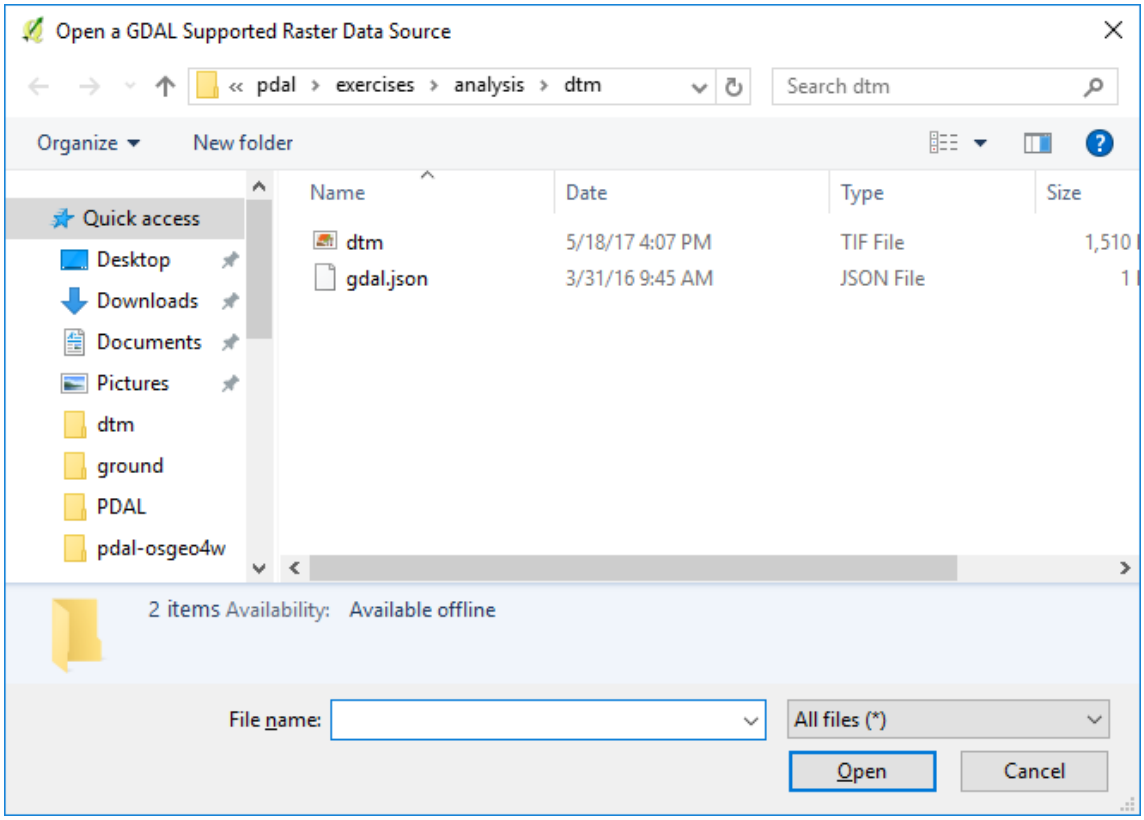
Visualization

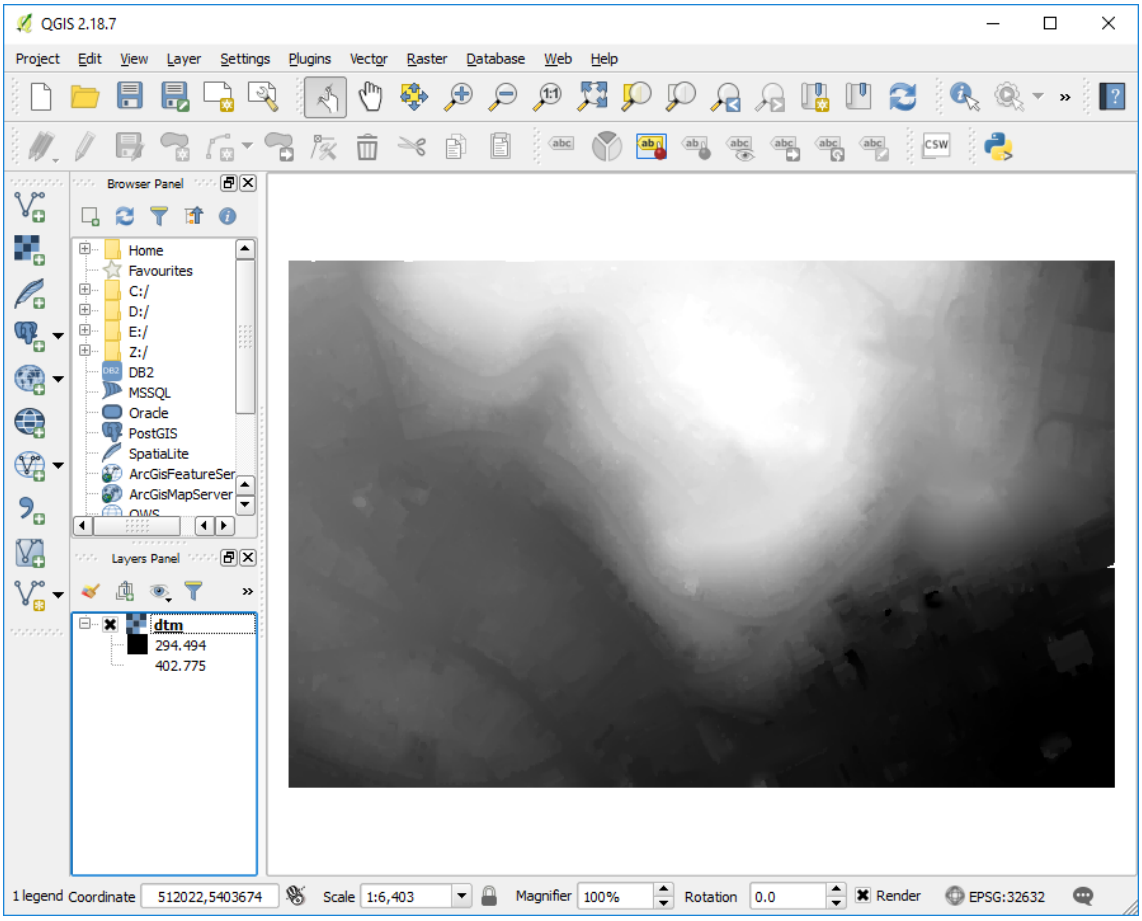
Something happened, and some files were written, but we cannot really see what was produced. Let us use qgis to visualize the output.

1. Open qgis and *Add Raster Layer*:

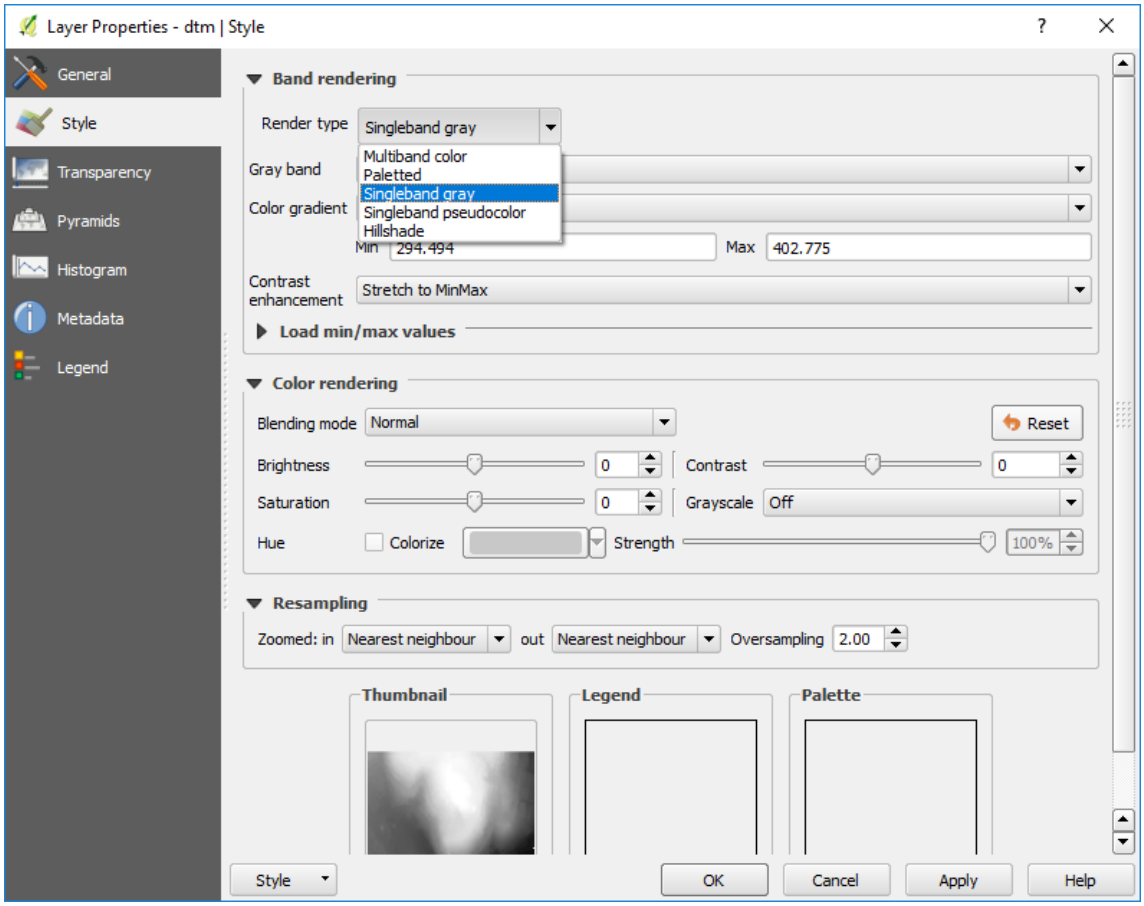


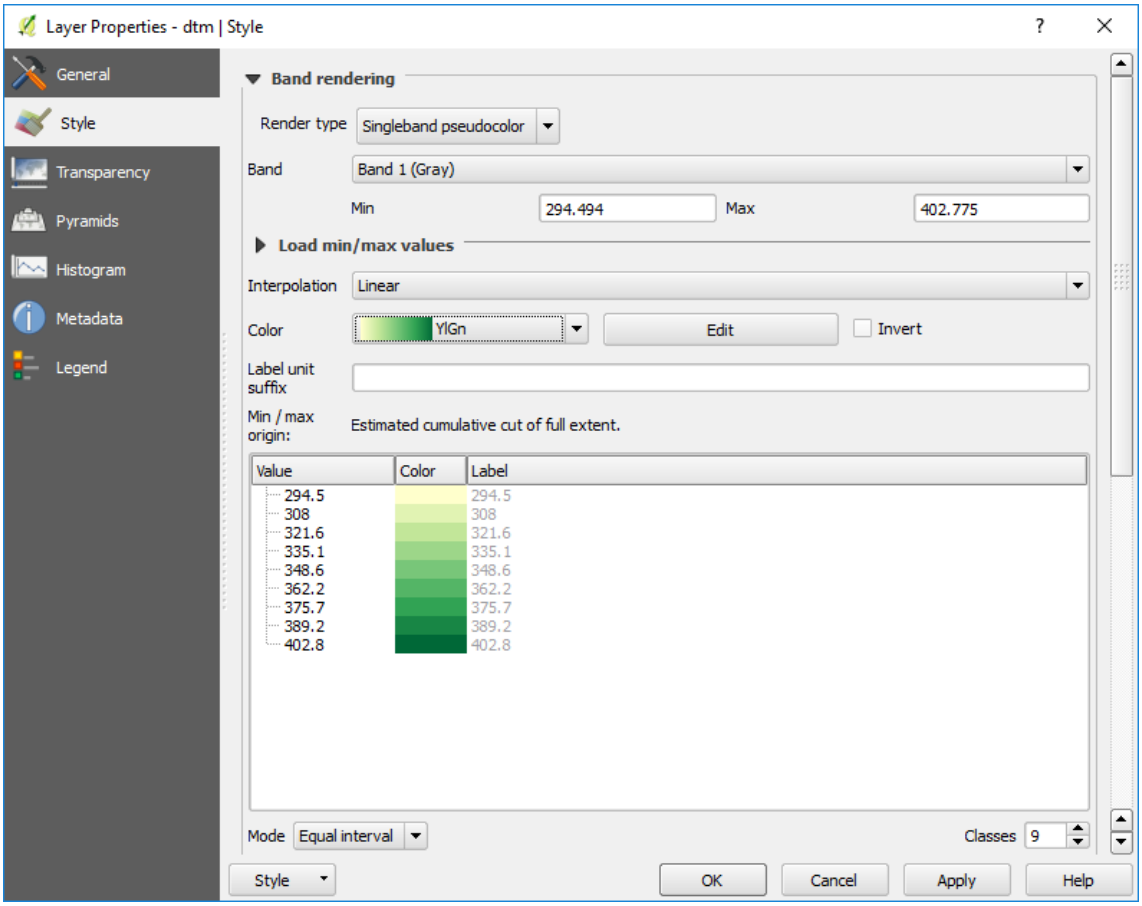
2. Add the *dtm.tif* file from your `./exercises/analysis/dtm` directory.


Open a GDAL Supported Raster Data Source
« pdal › exercises › analysis › dtm
Search dtm
Organize
New folder
Quick access
Desktop
Downloads
Documents
Pictures
dtm
ground
PDAL
pdal-osgeo4w
Name
Date
Type
Size
dtm
5/18/17 4:07 PM
TIF File
gdal.json
3/31/16 9:45 AM
JSON File
2 items Availability: Available offline
File name:
All files (*)
Open
Cancel

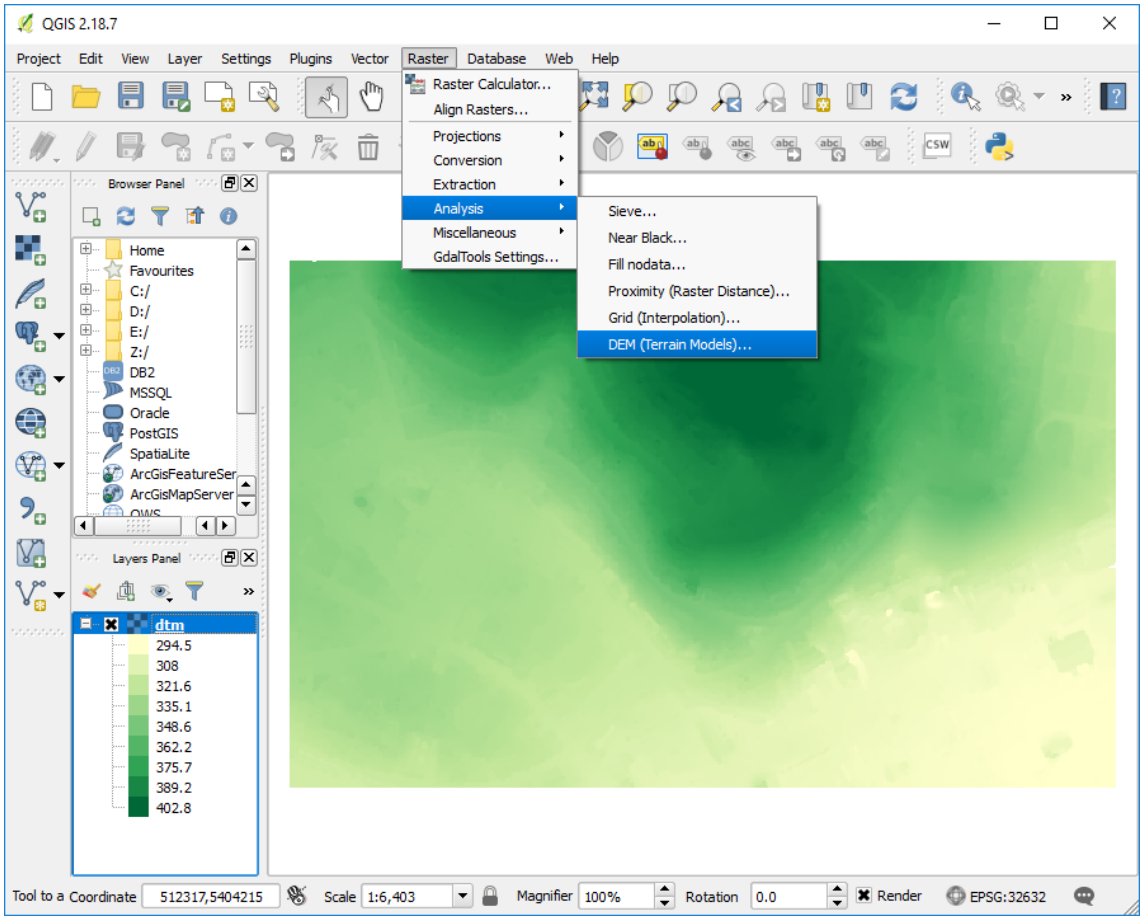


3. Classify the DTM by right-clicking on the *dtm.tif* and choosing *Properties*. Pick the pseudocolor rendering type, and then choose a color ramp and click *Classify*.

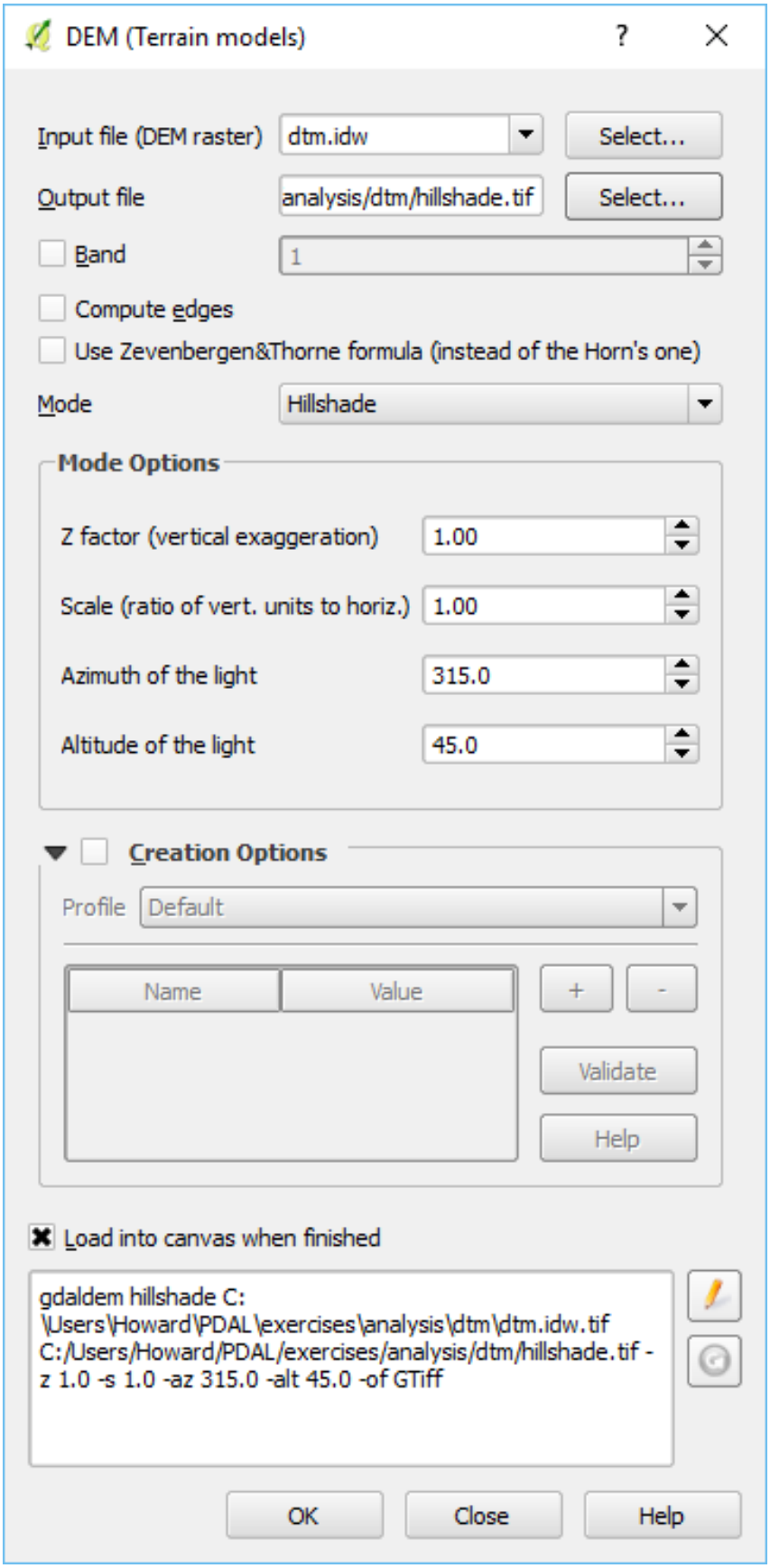

Layer Properties - dtm | Style
General
Style
Transparency
Pyramids
Histogram
Metadata
Legend
Band rendering
Render type
Singleband gray
Multiband color
Paletted
Singleband gray
Singleband pseudocolor
Hillshade
Gray band
Color gradient
Min
294.494
Max
402.775
Contrast enhancement
Stretch to MinMax
Load min/max values
Color rendering
Blending mode
Normal
Reset
Brightness
Contrast
Saturation
Grayscale
Off
Hue
Colorize
Strength
100%
Resampling
Zoomed: in
Nearest neighbour
out
Nearest neighbour
Oversampling
2.00
Thumbnail
Legend
Palette
Style
OK
Cancel
Apply
Help



4. qgis provides access to GDAL (http://gdal.org/) processing tools, and we are going to use that to create a hillshade of our surface. Choose *Raster–>Analysis–>Dem*:



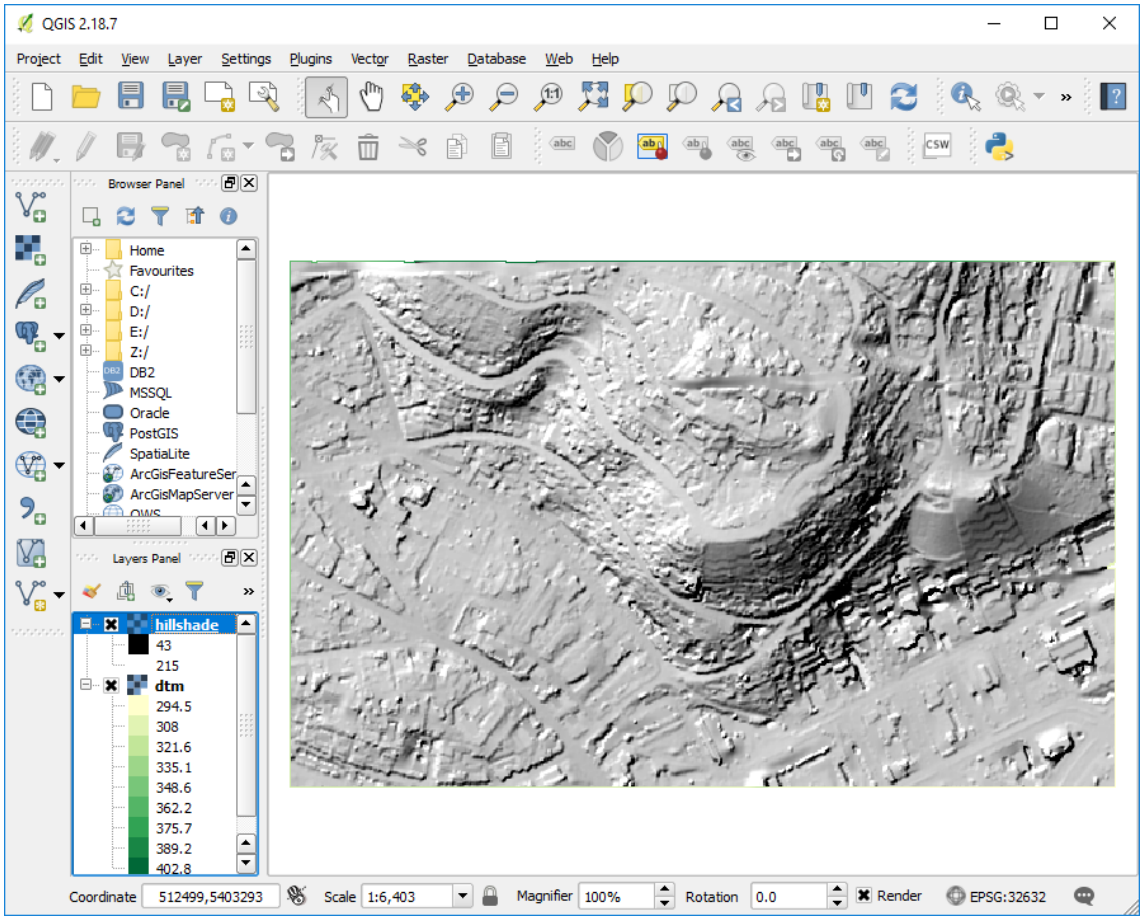
5. Click the window for the *Output file* and select a location to save the `hillshade.tif` file.


DEM (Terrain models)
Input file (DEM raster)
dtm.idw
Select...
Output file
analysis/dtm/hillshade.tif
Select...
Band
1
Compute edges
Use Zevenbergen&Thorne formula (instead of the Horn's one)
Mode
Hillshade
Mode Options
Z factor (vertical exaggeration)
1.00
Scale (ratio of vert. units to horiz.)
1.00
Azimuth of the light
315.0
Altitude of the light
45.0
Creation Options
Profile
Default
Name
Value
+
-
Validate
Help
Load into canvas when finished
gdaldem hillshade C:\Users\Howard\PDAL\exercises\analysis\dtm\dtm.idw.tif C:/Users/Howard/PDAL/exercises/analysis/dtm/hillshade.tif -z 1.0 -s 1.0 -az 315.0 -alt 45.0 -of GTiff
OK
Close
Help

```
gdaldem hillshade ./exercises/analysis/dtm/dtm.tif \
./exercises/analysis/dtm/hillshade.tif \
-z 1.0 -s 1.0 -az 315.0 -alt 45.0 \
-of GTiff
```

```
gdaldem hillshade ./exercises/analysis/dtm/dtm.tif ^
./exercises/analysis/dtm/hillshade.tif ^
-z 1.0 -s 1.0 -az 315.0 -alt 45.0 ^
-of GTiff
```

6. Click *OK* and the hillshade of your DTM is now available



9.8.2 Notes

1. gdaldem (http://www.gdal.org/gdaldem.html), which powers the qgis DEM tools, is a very powerful command line utility you can use for processing data.
2. writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) can be used for large data, but it does not interpolate a typical TIN (https://en.wikipedia.org/wiki/Triangulated_irregular_network) surface model.

9.9 Creating surface meshes

This exercise uses PDAL to create surface meshes. PDAL is able to use a number of meshing filters: https://pdal.io/stages/filters.html#mesh. Three of these are 'in the box', without needing plugins compiled. These are 2D Delaunay triangulation, Greedy projection meshing and Poisson surface meshing.

In this exercise we'll create a Poisson surface mesh - a watertight isosurface - from our input point cloud.

9.9.1 Exercise

We will create mesh models of a building and its surrounds using an entwine data input source.

After running each command, the output *.ply* file can be viewed in Meshlab or CloudCompare.

See also:

PDAL implements Mischa Kazhdan's Poisson surface reconstruction algorithm. For details see *[Kazhdan2006]_*

Note: *writers.ply* will write out mesh vertices by default. In this exercise we set the attribute *faces="true"*. Try using the ply writer without it. Also, if you're using a machine with a lot of processing power, try increasing the *depth* parameter for a more detailed mesh.

Command

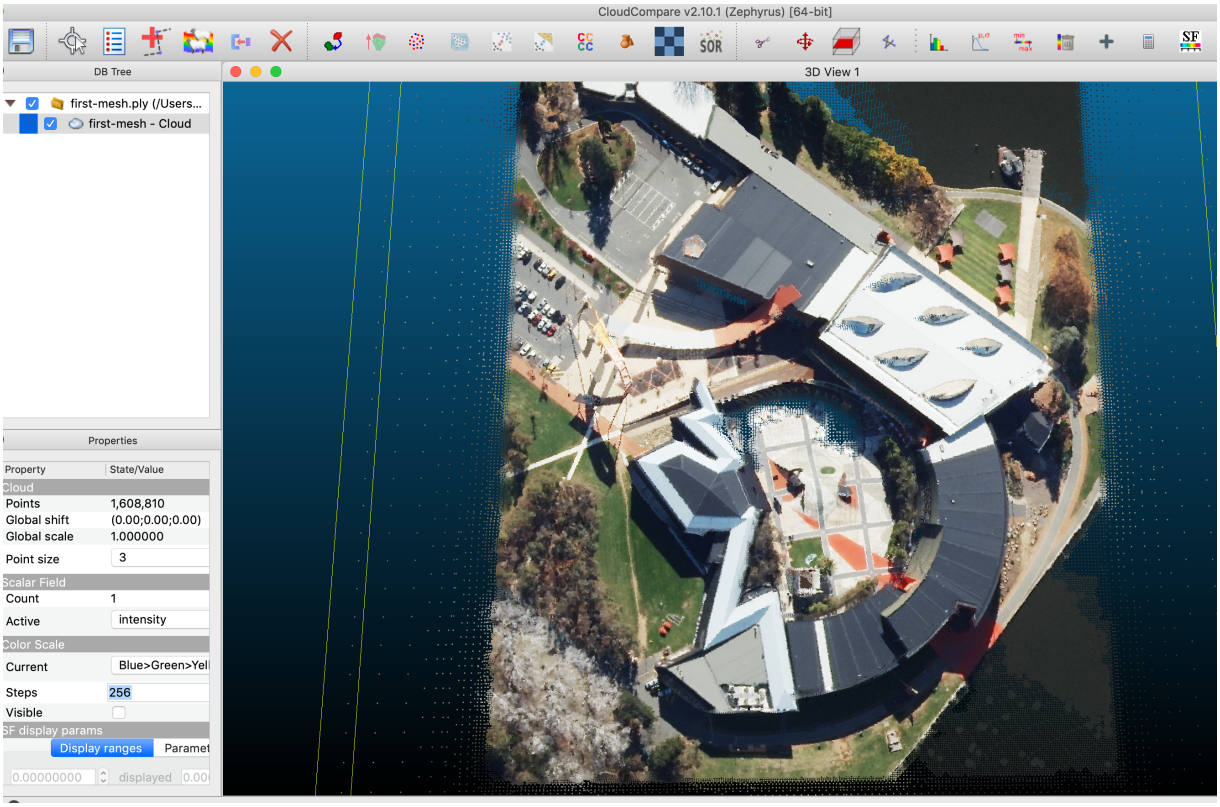
Invoke the following command, substituting accordingly, in your *Conda Shell*:

```
pdal translate -i ept://http://act-2015-rgb.s3.amazonaws.com \
 -o ./exercises/analysis/meshing/first-mesh.ply \
 poisson --filters.poisson.depth=16 \
 --readers.ept.bounds="([692738, 692967], [6092255,6092562])" \
 --verbose 4
```

```
pdal translate -i ept://http://act-2015-rgb.s3.amazonaws.com ^
 -o ./exercises/analysis/meshing/first-mesh.ply ^
 poisson --filters.poisson.depth=16 ^
 --readers.ept.bounds="([692738, 692967], [6092255,6092562])" ^
 --verbose 4
```

```
(pdalworkshop) $ pdal translate -i ept://http://act-2015-rgb.s3.amazonaws.com \
>  -o ./exercises/analysis/meshing/first-mesh.ply \
>  poisson --filters.poisson.depth=16 \
>  --readers.ept.bounds="([692738, 692967], [6092255,6092562])" \
>  --verbose 2
# Read input into tree:
#   Got kernel density:
#     Got normal field:
#       Finalized tree:
#  Set FEM constraints:
#Set point constraints:
Got average:
(pdalworkshop) $
```

You can view the mesh in Cloud Compare, you should see something similar to



Filtering

If we want to just mesh a building, or just terrain, or both we can apply a *range* filter based on point classification. These data have ground labelled as class 2, and buildings as 6.

In this exercise we will create a poisson mesh surface of a building and the ground surrounding it, using the same data subset as above and adding a filters.range

(https://pdal.io/stages/filters.range.html#filters-range) stage to limit the set of points used in mesh creation.

Command

Invoke the following command, substituting accordingly, in your *Conda Shell*:

```
pdal translate -i ept://http://act-2015-rgb.s3.amazonaws.com \
 -o ./exercises/analysis/meshing/building-exercise.ply \
 range poisson \
 --filters.range.limits="Classification[2:2],Classification[6:6]" \
 --filters.poisson.depth=16 \
 --readers.ept.bounds="([692738, 692967], [6092255,6092562])" \
 --verbose 4
```

```
pdal translate -i ept://http://act-2015-rgb.s3.amazonaws.com ^
 -o ./exercises/analysis/meshing/building-exercise.ply ^
 range poisson ^
 --filters.range.limits="Classification[2:2],Classification[6:6]" ^
 --filters.poisson.depth=16 ^
 --readers.ept.bounds="([692738, 692967], [6092255,6092562])" ^
 --verbose 4
```

```
(pdalworkshop) $ pdal translate -i ept://http://act-2015-rgb.s3.amazonaws.com \
>  -o ./exercises/analysis/meshing/building-exercise.ply \
>  range poisson \
>  --filters.range.limits="Classification[2:2],Classification[6:6]" \
>  --filters.poisson.depth=16 \
>  --readers.ept.bounds="([692738, 692967], [6092255,6092562])" \
>  --verbose 2
# Read input into tree:
#   Got kernel density:
#     Got normal field:
#       Finalized tree:
#  Set FEM constraints:
#Set point constraints:
Got average:
(pdalworkshop) $
```

9.10 Rasterizing Attributes

This exercise uses PDAL to generate a raster surface using a fully classified point cloud with PDAL's writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal).

9.10.1 Exercise

Note: The exercise fetches its data from a Entwine (https://entwine.io) service that organizes the point cloud collection for the entire country of Denmark. You can view the data online at http://potree.entwine.io/data/denmark.html

Command

PDAL capability to generate rasterized output is provided by the writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) stage. There is no application (https://pdal.io/apps/index.html#apps) to drive this stage, and we must use a pipeline.

Pipeline breakdown

```
{
  "pipeline":[
    {
        "type":"readers.ept",
        "filename":"http://na-c.entwine.io/dk",
        "bounds":"([1401016, 1410670], [7476527, 7484590])",
        "resolution": 5
    },
    {
      "type":"writers.gdal",
      "filename":"denmark-classification.tif",
      "dimension":"Classification",
      "data_type":"uint16_t",
      "output_type":"mean",
      "resolution": 5
    }
  ]
}
```

Note: This pipeline is available in your workshop materials in the `./exercises/analysis/dtm/dtm.json` file. Make sure to edit the filenames to match your paths.

1. Reader

```
    {
        "type":"readers.ept",
        "filename":"http://na-c.entwine.io/dk",
        "bounds":"([1401016, 1410670], [7476527, 7484590])",
        "resolution": 5
    },
```

The data is read from a EPT resource that contains the Denmark data. We're going to download a small patch of data by the Copenhagen airport area that is the limited to a spatial resolution of 5m.

2. writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal)

The writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) writer that bins the point cloud data with classification values.

```
    {
      "type":"writers.gdal",
      "filename":"denmark-classification.tif",
      "dimension":"Classification",
      "data_type":"uint16_t",
      "output_type":"mean",
      "resolution": 5
    }
```

Execution

Issue the pipeline (https://pdal.io/pipeline.html#pipeline) operation to execute the interpolation:

```
pdal pipeline ./exercises/analysis/rasterize/classification.json -v 3
```

```
{
  "pipeline":[
    {
        "type":"readers.ept",
        "filename":"http://na-c.entwine.io/dk",
        "bounds":"([1401016, 1410670], [7476527, 7484590])",
        "resolution": 5
    },
    {
      "type":"writers.gdal",
```

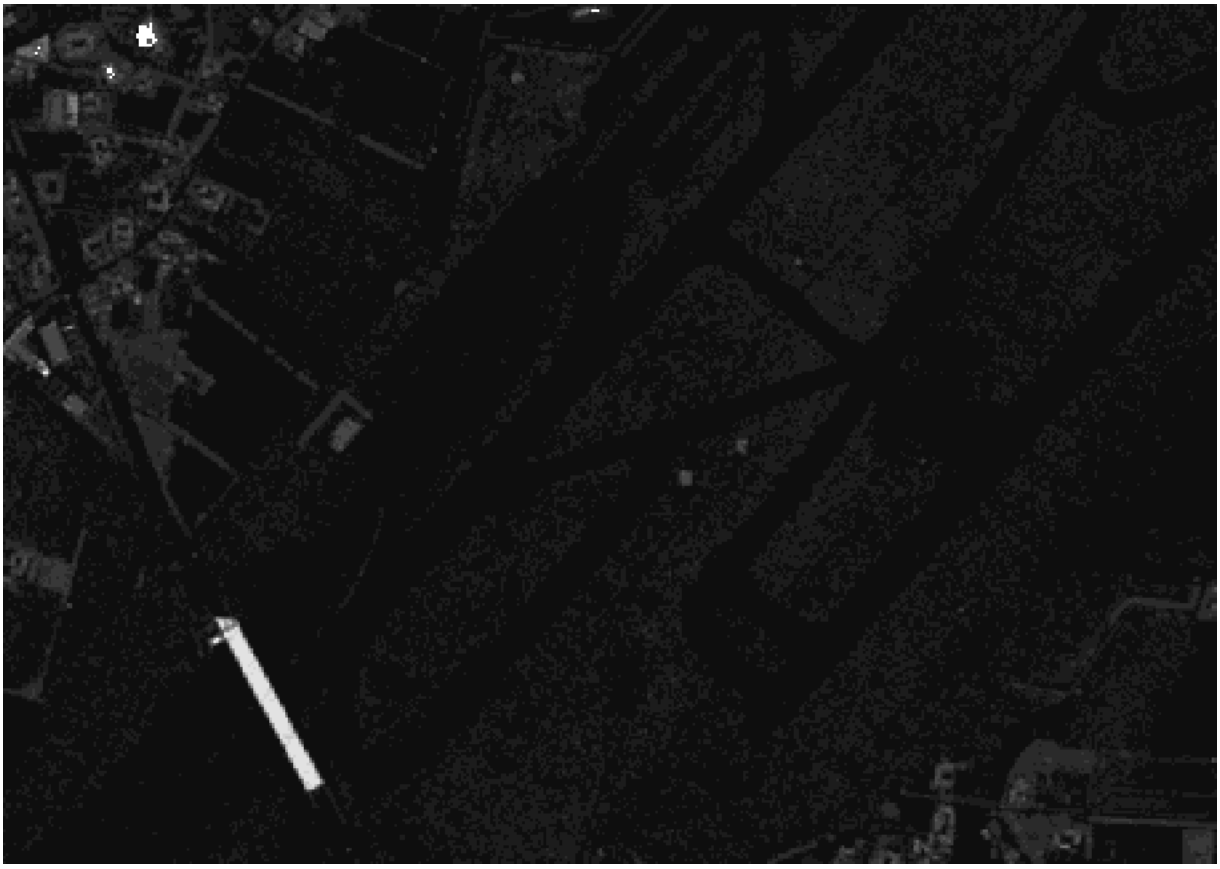
(continues on next page)

(continued from previous page)

```
        "filename":"denmark-classification.tif",
        "dimension":"Classification",
        "data_type":"uint16_t",
        "output_type":"mean",
        "resolution": 5
      }
    ]
}
```

```
(pdalworkshop) $ pdal pipeline ./exercises/analysis/rasterize/classification.json -v 3
(PDAL Debug) Debugging...
(pdal pipeline readers.ept Debug) Endpoint: http://na-c.entwine.io/dk/
Got EPT info
SRS: PROJCS["WGS 84 / Pseudo-Mercator",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326
"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]],PROJECTION["Mercator_1SP
"],PARAMETER["central_meridian",0],PARAMETER["scale_factor",1],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001"
]],AXIS["X",EAST],AXIS["Y",NORTH],EXTENSION["PROJ4","+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +wk
text +no_defs"],AUTHORITY["EPSG","3857"]]
Root resolution: 3108.53
Query resolution:  10
Actual resolution: 6.07135
Depth end: 10
Query bounds: ([1102422, 1107468], [7762273, 7777901], [-1.797693134862316e+308, 1.797693134862316e+308])
Threads: 4
(pdal pipeline readers.ept Debug) Registering dim X: double
(pdal pipeline readers.ept Debug) Registering dim Y: double
(pdal pipeline readers.ept Debug) Registering dim Z: double
(pdal pipeline readers.ept Debug) Registering dim Intensity: uint16_t
(pdal pipeline readers.ept Debug) Registering dim ReturnNumber: uint8_t
(pdal pipeline readers.ept Debug) Registering dim NumberOfReturns: uint8_t
(pdal pipeline readers.ept Debug) Registering dim ScanDirectionFlag: uint8_t
(pdal pipeline readers.ept Debug) Registering dim EdgeOfFlightLine: uint8_t
(pdal pipeline readers.ept Debug) Registering dim Classification: uint8_t
(pdal pipeline readers.ept Debug) Registering dim ScanAngleRank: float
(pdal pipeline readers.ept Debug) Registering dim UserData: uint8_t
(pdal pipeline readers.ept Debug) Registering dim PointSourceId: uint16_t
(pdal pipeline readers.ept Debug) Registering dim GpsTime: double
(pdal pipeline readers.ept Debug) Registering dim Red: uint16_t
(pdal pipeline readers.ept Debug) Registering dim Green: uint16_t
(pdal pipeline readers.ept Debug) Registering dim Blue: uint16_t
(pdal pipeline Debug) Executing pipeline in standard mode.
(pdal pipeline readers.ept Debug) Overlap nodes: 79
(pdal pipeline readers.ept Debug) Overlap points: 7242657
(pdal pipeline readers.ept Debug) Data 1/79: 0-0-0-0
(pdal pipeline readers.ept Debug) Data 2/79: 1-0-1-1
(pdal pipeline readers.ept Debug) Data 3/79: 2-1-2-2
(pdal pipeline readers.ept Debug) Data 4/79: 3-2-5-4
(pdal pipeline readers.ept Debug) Data 5/79: 4-4-11-8
(pdal pipeline readers.ept Debug) Data 6/79: 5-8-22-16
(pdal pipeline readers.ept Debug) Data 7/79: 5-8-23-16
(pdal pipeline readers.ept Debug) Data 8/79: 6-16-45-32
(pdal pipeline readers.ept Debug) Data 9/79: 6-16-46-32
```

Visualization



Basic interpolation of data with writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) will output raw classification values into the resulting raster file. We will need to add a color ramp to the data for a satisfactory preview.

Unfortunately, this doesn't give us a very satisfactory image to view. The reason is there is no color ramp associated with the file, and we're looking at pixel values with values from 0-31 according to the ASPRS LAS specification.

We want colors that correspond to the classification values a bit more directly. We can use a color ramp to assign explicit values. qgis allows us to create a text file color ramp that gdaldem can consume to apply colors to the data.

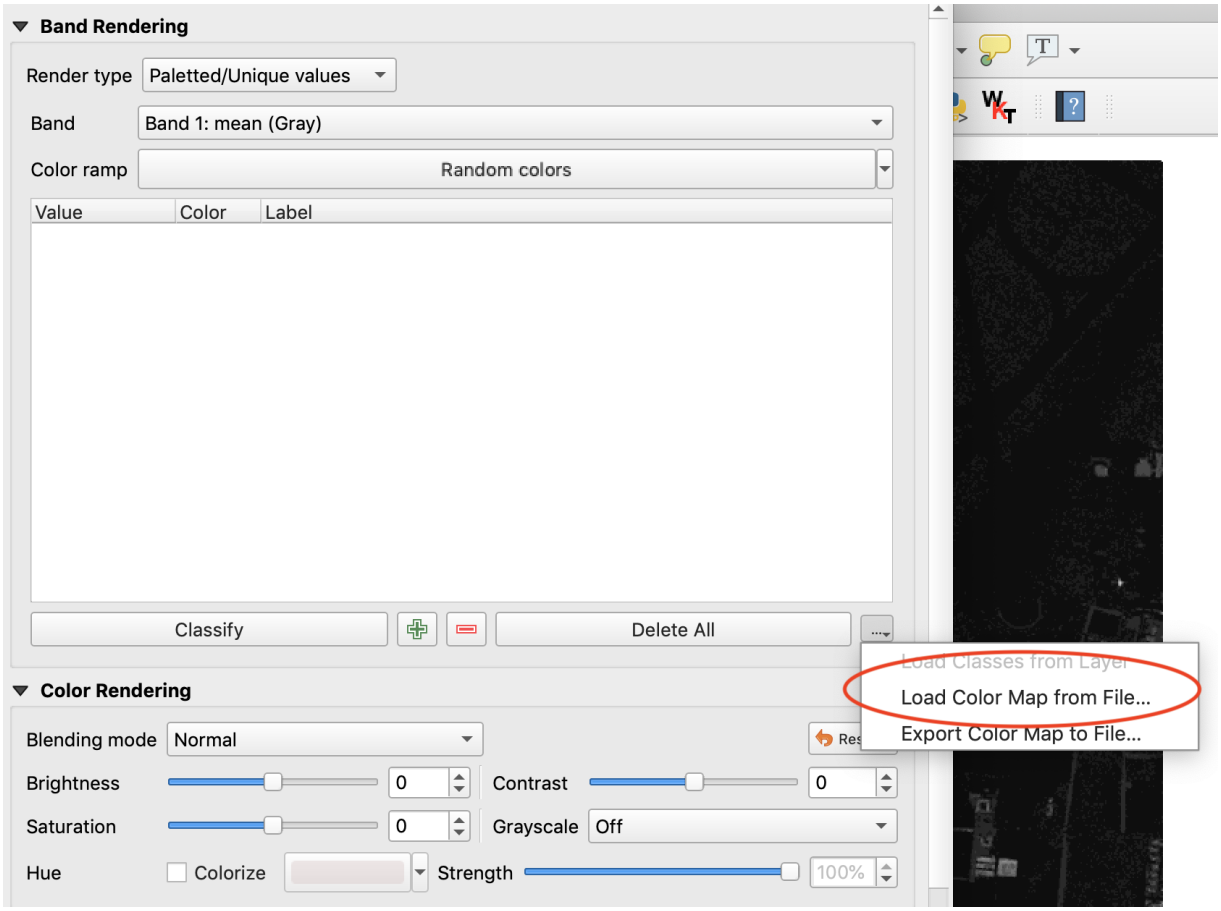
```
# QGIS Generated Color Map Export File
2 139 51 38 255 Ground
3 143 201 157 255 Low Veg
4 5 159 43 255 Med Veg
5 47 250 11 255 High Veg
6 209 151 25 255 Building
7 232 41 7 255 Low Point
8 197 0 204 255 reserved
```

(continues on next page)

(continued from previous page)

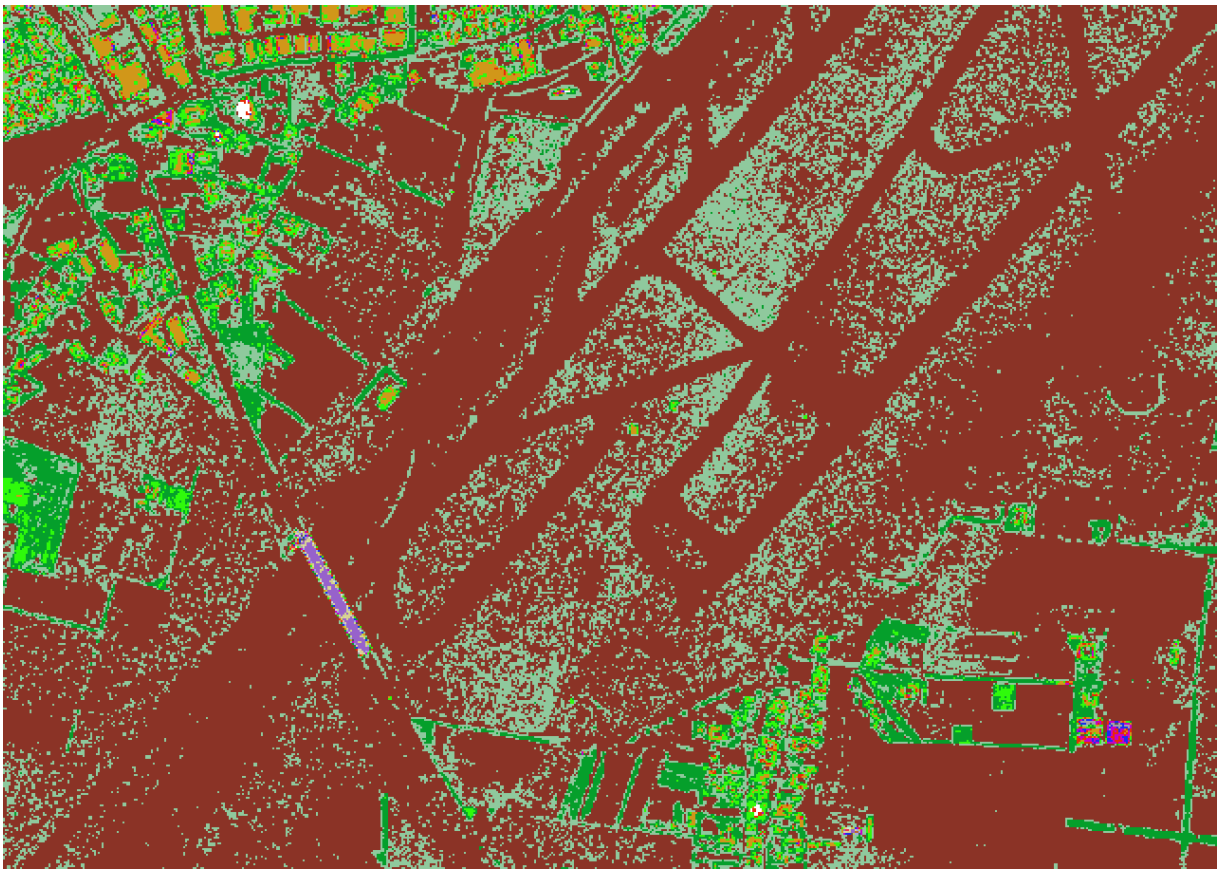
```
9 26 44 240 255 Water
10 165 160 173 255 Rail
11 81 87 81 255 Road
12 203 210 73 255 Reserved
13 209 228 214 255 Wire - Guard (Shield)
14 160 168 231 255 Wire - Conductor (Phase)
15 220 213 164 255 Transmission Tower
16 214 211 143 255 Wire-Structure Connector (Insulator)
17 151 98 203 255 Bridge Deck
18 236 49 74 255 High Noise
19 185 103 45 255 Reserved
21 58 55 9 255 255 Reserved
22 76 46 58 255 255 Reserved
23 20 76 38 255 255 Reserved
26 78 92 32 255 255 Reserved
```

With this ramp, you can load the color values into QGIS as a color ramp if you change the layer to Palatted/Unique Values, and then load the color ramp file:



With the ramp, we can also use gdaldem (http://www.gdal.org/gdaldem.html) to apply it to a new image:

```
gdaldem color-relief denmark-classification.tif ramp.txt classified-
↪color.png -of PNG
```



Intensity

With PDAL's ability to override pipeline via commands, we can generate a relative intensity image:

```
pdal pipeline ./exercises/analysis/rasterize/classification.json \
--writers.gdal.dimension="Intensity" \
--writers.gdal.data_type="float" \
--writers.gdal.filename="intensity.tif" \
-v 3

gdal_translate intensity.tif intensity.png -of PNG
```

```
pdal pipeline ./exercises/analysis/rasterize/classification.json ^
--writers.gdal.dimension="Intensity" ^
--writers.gdal.data_type="float" ^
--writers.gdal.filename="intensity.tif" ^
```

(continues on next page)

(continued from previous page)

```
-v 3

gdal_translate intensity.tif intensity.png -of PNG
```

The same pipeline can be used to generate a preview image of the Intensity channel of the data by overriding pipeline arguments at the command line.

9.10.2 Notes

1. writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) can output any dimension PDAL can provide, but it is is up to the user to interpolate the values. For categorical data, neighborhood smoothing might produce undesirable results, for example.
2. Pipeline (https://pdal.io/pipeline.html#pipeline) contains more information about overrides and organizing complex pipelines.

CHAPTER

TEN

PYTHON

10.1 Plotting a histogram

10.1.1 Exercise

PDAL doesn't provide every possible analysis option, but it strives to make it convenient to link PDAL to other places with substantial functionality. One of those is the Python/Numpy universe, which is accessed through PDAL's Python (https://pdal.io/python.html#python) bindings and the filters.python (https://pdal.io/stages/filters.python.html#filters-python) filter. These tools allow you to manipulate point cloud data with convenient Python tools rather than constructing substantial C/C++ software to achieve simple tasks, compute simple statistics, or investigate data quality issues.

This exercise uses PDAL to create a histogram plot of all of the dimensions of a file. matplotlib (https://matplotlib.org/) is a Python package for plotting graphs and figures, and we can use it in combination with the Python (https://pdal.io/python.html#python) bindings for PDAL to create a nice histogram. These histograms can be useful diagnostics in an analysis pipeline. We will combine a Python script to make a histogram plot with a pipeline (https://pdal.io/apps/pipeline.html#pipeline-command).

Note: Python allows you to enhance and build functionality that you can use in the context of other Pipeline (https://pdal.io/pipeline.html#pipeline) operations.

PDAL Pipeline

We're going to create a PDAL Pipeline (https://pdal.io/pipeline.html#pipeline) to tell PDAL to run our Python script in a filters.python (https://pdal.io/stages/filters.python.html#filters-python) stage.

```
{
    "pipeline": [
        {
            "filename": "./exercises/python/athletic-fields.laz"
        },
        {
            "type": "filters.python",
            "function": "make_plot",
            "module": "anything",
            "pdalargs": "{\"filename\":\"./exercises/python/
↪histogram.png\"}",
            "script": "./exercises/python/histogram.py"
        },
        {
            "type": "writers.null"
        }
    ]
}
```

Note: This pipeline is available in your workshop materials in the `./exercises/python/histogram.json` file.

Python script

The following Python script will do the actual work of creating the histogram plot with matplotlib (https://matplotlib.org/). Store it as `histogram.py` next to the `histogram.json` Pipeline (https://pdal.io/pipeline.html#pipeline) file above. The script is mostly regular Python except for the `ins` and `outs` arguments to the function – those are special arguments that PDAL expects to be a dictionary of Numpy dictionaries.

Note: This Python file is available in your workshop materials in the `./exercises/python/histogram.py` file.

```
# import numpy
import numpy as np

# import matplotlib stuff and make sure to use the
# AGG renderer.
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import matplotlib.mlab as mlab

# This only works for Python 3. Use
# StringIO for Python 2.
from io import BytesIO

# The make_plot function will do all of our work. The
# filters.programmable filter expects a function name in the
# module that has at least two arguments -- "ins" which
# are numpy arrays for each dimension, and the "outs" which
# the script can alter/set/adjust to have them updated for
# further processing.
def make_plot(ins, outs):

    # figure position and row will increment
    figure_position = 1
    row = 1

    fig = plt.figure(figure_position, figsize=(6, 8.5), dpi=300)

    for key in ins:
        dimension = ins[key]
        ax = fig.add_subplot(len(ins.keys()), 1, row)

        # histogram the current dimension with 30 bins
        n, bins, patches = ax.hist( dimension, 30,
                                    normed=0,
                                    facecolor='grey',
                                    alpha=0.75,
                                    align='mid',
                                    histtype='stepfilled',
                                    linewidth=None)

        # Set plot particulars
        ax.set_ylabel(key, size=10, rotation='horizontal')
        ax.get_xaxis().set_visible(False)
        ax.set_yticklabels('')
        ax.set_yticks((),)
        ax.set_xlim(min(dimension), max(dimension))
        ax.set_ylim(min(n), max(n))

        # increment plot position
        row = row + 1
        figure_position = figure_position + 1

```

(continues on next page)

(continued from previous page)

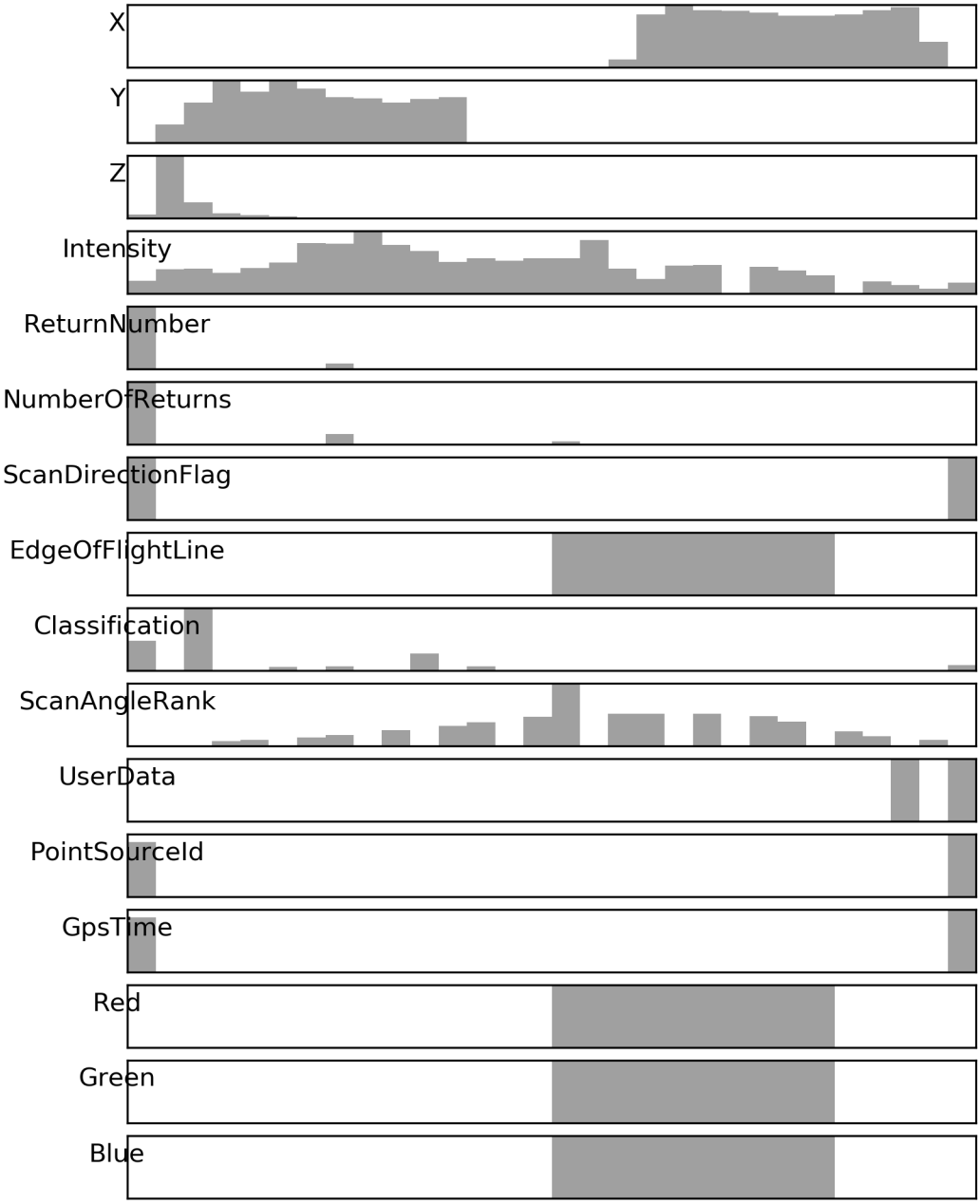
```
    # We will save the PNG bytes to a BytesIO instance
    # and the nwrite that to a file.
    output = BytesIO()
    plt.savefig(output,format="PNG")

    # a module global variable, called 'pdalargs' is available
    # to filters.programmable and filters.predicate modules that contains
    # a dictionary of arguments that can be explicitly passed into
    # the module by the user. We passed in a filename arg in our `pdal pipeline` call
    if 'filename' in pdalargs:
        filename = pdalargs['filename']
    else:
        filename = 'histogram.png'

    # open up the filename and write out the
    # bytes of the PNG stored in the BytesIO instance
    o = open(filename, 'wb')
    o.write(output.getvalue())
    o.close()


    # filters.programmable scripts need to
    # return True to tell the filter it was successful.
    return True
```

Run `pdal pipeline`

```
pdal pipeline ./exercises/python/histogram.json
```

```
(pdalworkshop) $ pdal pipeline ./exercises/python/histogram.json
anything:40: MatplotlibDeprecationWarning:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'density' instead.
anything:47: UserWarning: Attempting to set identical left == right == 0 results in singular transformations; automatically expanding.
(pdalworkshop) $
```

Output


X
Y
Z
Intensity
ReturnNumber
NumberOfReturns
ScanDirectionFlag
EdgeOfFlightLine
Classification
ScanAngleRank
UserData
PointSourceId
GpsTime
Red
Green
Blue

10.1.2 Notes

1. writers.null (https://pdal.io/stages/writers.null.html#writers-null) simply swallows the output of the pipeline. We don't need to write any data.
2. The `pdalargs` JSON needs to be escaped because a valid Python dictionary entry isn't always valid JSON.

CHAPTER

ELEVEN

GEOREFERENCING

11.1 Georeferencing

As discussed *in the introduction* (page 15), laser returns from a mobile LiDAR (https://en.wikipedia.org/wiki/Lidar) system must be georeferenced, i.e. placed into a local or global coordinate system by combining data from the laser and from a GNSS/IMU. As of this writing, PDAL does **not** include generic georeferencing tools — this is considered future work. However, the Optech (http://www.teledyneoptech.com/) csd file format includes both laser return and GNSS/IMU data in the same file, and the PDAL csd reader includes built in georeferencing support.

In this section, we will demonstrate how to georeference an Optech (http://www.teledyneoptech.com/) csd file and reproject that file into a UTM projection.

Note: Optech's (http://www.teledyneoptech.com/) csd format is just one of several vendor-specific data formats PDAL supports; we also support data files directly from RIEGL (http://riegl.com/) sensors and from several project-specific government platforms.

11.1.1 Exercise

The file *S1C1_csd_004.csd* contains airborne data from an Optech (http://www.teledyneoptech.com/) sensor. Without georeferencing these points, they would be impossible to interpret — once they are georeferenced, we will be able to inspect and analyze these points like any other point cloud.

In addition to georeferencing, we are going to make two other tweaks to our point cloud:

- The point cloud is, by default, in WGS84 (https://en.wikipedia.org/wiki/Geodetic_datum), but we will reproject these points to a UTM (https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system) coordinate system for visualization purposes.

- Because these are raw data coming from the sensor, these data are noisy. In particular, there are a few points *very* close to the sensor which were probably caused by air returns or laser light reflecting off of part of the airplane or sensor. These points have very high intensity values, which will screw up our visualization. We will use the filters.range (https://pdal.io/stages/filters.range.html#filters-range) PDAL filter to drop all points with very high intensity values.

Note: These data were provided by Dr. Craig Glennie and were collected by NCALM (http://ncalm.cive.uh.edu/), the National Center for Airborne Laser Mapping. The collect area is southwest of Austin, TX.

Command

Invoke the following command, substituting accordingly, into your ‘ Conda Shell‘:

```
pdal translate \
./exercises/georeferencing/S1C1_csd_004.csd \
./exercises/georeferencing/S1C1_csd_004.laz \
reprojection range \
--filters.reprojection.out_srs="EPSG:32614" \
--filters.range.limits="Intensity[0:500]"
```

```
pdal translate ^
./exercises/georeferencing/S1C1_csd_004.csd ^
./exercises/georeferencing/S1C1_csd_004.laz ^
reprojection range ^
--filters.reprojection.out_srs="EPSG:32614" ^
--filters.range.limits="Intensity[0:500]"
```

```
(pdalworkshop) $ pdal translate \
> ./exercises/georeferencing/S1C1_csd_004.csd \
> ./exercises/georeferencing/S1C1_csd_004.laz \
> reprojection range \
> --filters.reprojection.out_srs="EPSG:32614" \
> --filters.range.limits="Intensity[0:500]"
(pdalworkshop) $
```

Visualization

View your georeferenced point cloud in http://plas.io.

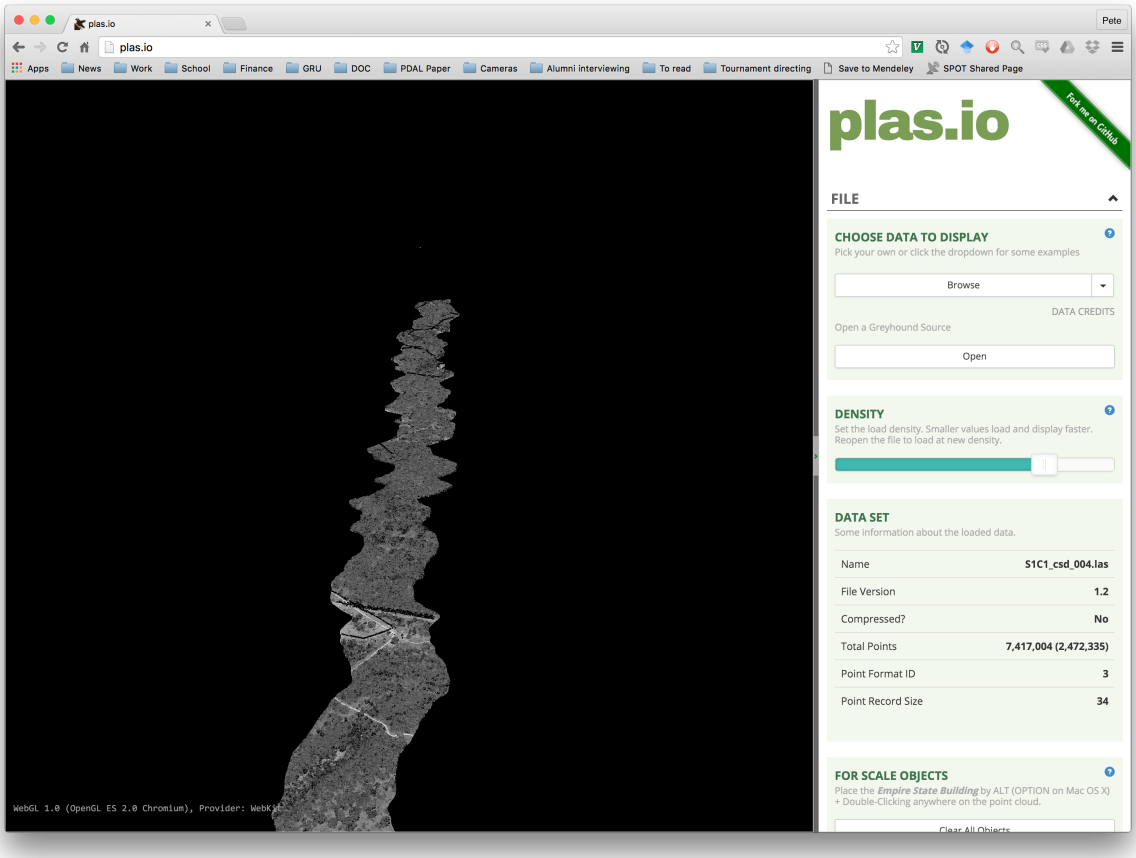


Fig. 1: Our airborne laser point cloud after georeferencing, reprojection, and intensity filtering.

CHAPTER

TWELVE

BATCH PROCESSING

12.1 Batch Processing

PDAL doesn't handle matching multiple file inputs except for glob handling for merge operations, but does allow for command line substitution parameters to make batch processing simpler, substitutions. Substitions work with both Pipeline (https://pdal.io/pipeline.html#pipeline) operations as well as with other applications such as translate (https://pdal.io/apps/translate.html#translate-command).

12.1.1 Operating system variations

How substitutions are passed generally depends on the operating system and tools available. In the unix/linux environments, this is primarily using the *find* and *ls* programs to get lists of files (either with directories or just filenames) and the *xargs* or *parallel* program to pass those files to the `pdal` application (although *-exec* with *find* can also be used). These tools are available in the *docker* environment if you are running *PDAL* under docker. They are also available under Windows one installs *Cygwin* or *MinGW*. They are also available if Git for Windows is installed. They are also available as win32 command line programs installed from the GNU Findutils (https://www.gnu.org/software/findutils/findutils.html). They are available for MacOS and Linux.

12.1.2 Windows native tools

Subtitions can be handled directly in windows using *PowerShell* syntax.

While there are a number of ways to generate lists of files, the *Get-ChildItem* is used here, along with the *foreach* option to pass each separate filepath to the pdal application.

12.1.3 Example - Batch compression of LAS files to LAZ - PowerShell:

To compress a series of LAS files in one directory into compressed LAZ files in another directory, the *PowerShell* syntax would be:

```
Get-ChildItem .\DIR1\*.las | foreach {pdal translate -i .\DIR1\$($_.
↪BaseName).las ^
-o .\DIR2\$($_.BaseName).laz}
```

Note the use of the *$($_.BaseName)* syntax for the files passed. This option on the *$($_)* shortcut for the full filename, removes the directory and the extension on the file and allows the user to set the path and extension manually.

12.1.4 Example - Parallel Batch compression of LAS files to LAZ - PowerShell:

This use of the *PowerShell* syntax doesn't allow a user to execute more than one process at a time. There is a free download of the *xargs* program that provides parallel execution available at http://www.pirosa.co.uk/demo/wxargs/ppx2.exe. For this tool, the file names are passed with using the *{}* syntax.

```
Get-ChildItem .\dir1\ | Select-Object -ExpandProperty BaseName ^
| .\ppx2.exe -P 3 pdal translate -i ".\dir1\{}.las" -o ".\dir2\{}.laz
↪"
```

12.1.5 Example - Batch compression of LAS files to LAZ - Bash:

To compress a series of LAS files in one directory into compressed LAZ files in another directory, the *Bash* syntax would be:

```
ls ./dir1/*.las | parallel -I{} \
pdal translate -i ./dir1/{/.}.las -o ./dir2/{/.}.laz
```

In *Parallel*, then *{/.}* syntax means strip the directory and the extension and just use the basename of the file. This allows you to easily change the output format and the location.

12.1.6 Example - Parallel Batch compression of LAS files to LAZ - Bash:

Parallel, as its name implies, allows paralell operations. Adding the *-j* syntax indicates the number simultaneous jobs to run

```
ls ./dir1/*.las | parallel -I{} -j 4 \
pdal translate -i ./dir1/{/.}.las -o ./dir2/{/.}.laz
```

12.1.7 Exercise - Pipeline Substitions:

For the most flexibility, pipelines are used to apply a series of opertations to a file (or group of files). In this excersise, we build on the *Generating a DTM* (page 77) pipeline example, but run this pipline over 4 files and reproject, calculate a bare earth using the filters.smrf (https://pdal.io/stages/filters.smrf.html#filters-smrf) filter, remove those points that aren't bare earth with filters.range (https://pdal.io/stages/filters.range.html#filters-range) and then write the output using the writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal).

The pipeline we are using is:

```
{
    "pipeline": [
        {
            "type":"readers.las"
        },
        {
            "type": "filters.reprojection"
        },
        {
            "type": "filters.smrf"
        },
        {
            "type":"filters.range",
            "limits":"Classification[2:2]"
        },
        {
            "gdaldriver":"GTiff",
            "output_type":"idw",
            "resolution" :"2.0",
            "type": "writers.gdal"
        }
    ]
}
```

You might have spotted that this pipeline doesn't have any input or output file references, or a value for the output spatial reference. We will be adding those at the command line, not within the actual pipeline and using the substitutions syntax to do this.

```
PS ./exercises/batch> Get-ChildItem  ./exercises/batch/
 ↪source/*.laz | ^
foreach {pdal pipeline ./exercises/batch/batch_srs_gdal.
 ↪json ^
```

(continues on next page)

(continued from previous page)

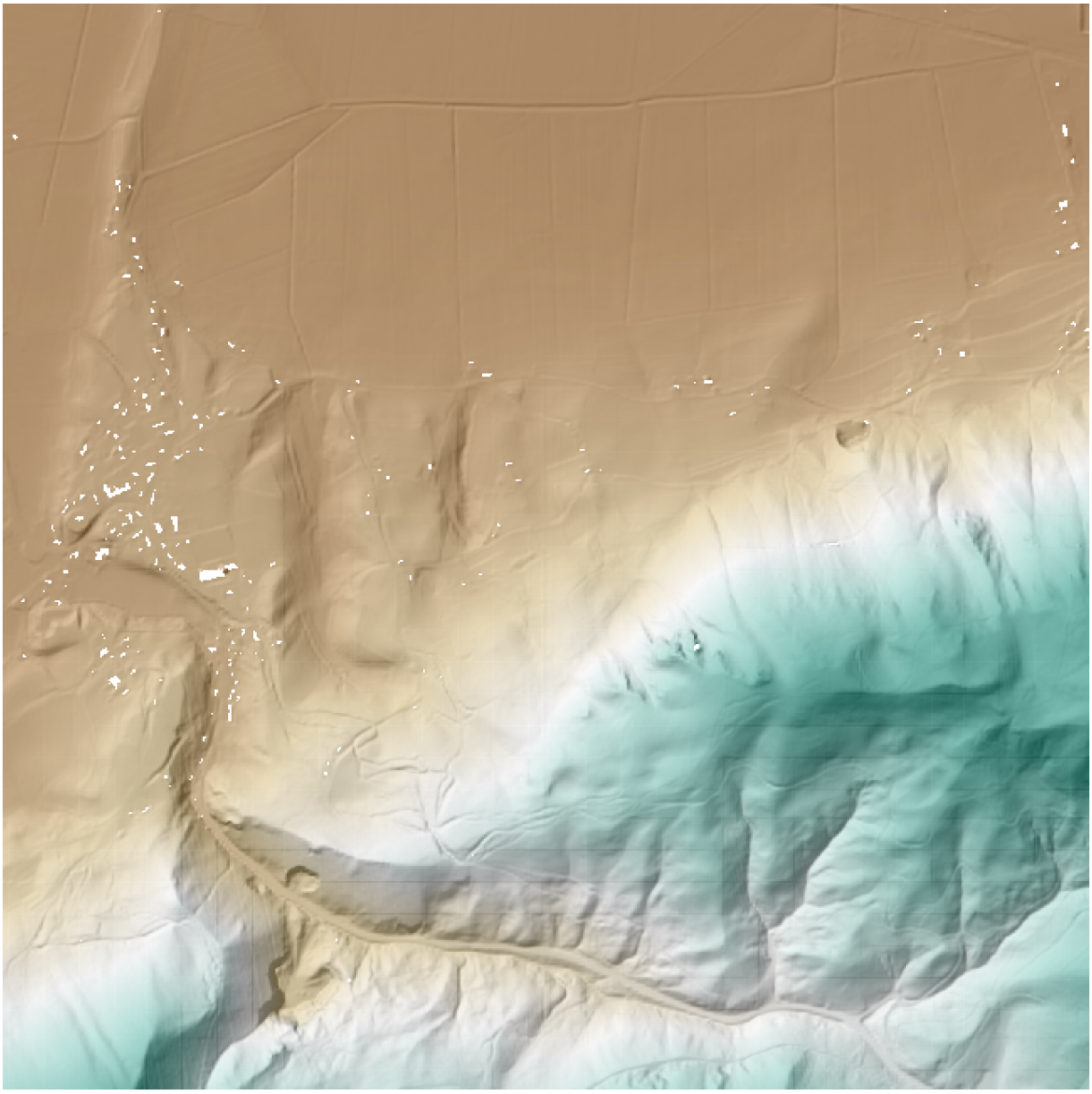
```
--readers.las.filename=./source/$($_.BaseName).laz ^
--writers.gdal.filename=./dtm/$($_.BaseName).tif ^
--filters.reprojection.in_srs=epsg:3794 ^
--filters.reprojection.in_srs=epsg:32733}
```

```
ls ./exercises/batch_processing/source/*.laz | \
parallel -I{} pdal pipeline ./exercises/batch_processing/
 ↪batch_srs_gdal.json \
 --readers.las.filename={} \
 --writers.gdal.filename=./exercises/batch_processing/dtm{/.
 ↪}.tif \
 --filters.reprojection.in_srs=epsg:3794 \
 --filters.reprojection.out_srs=epsg:32733
```

Once you have your dtms created with pdal, combine them to a single file with:

```
gdalbuildvrt  ./exercises/batch_processing/dtm.vrt ./exercises/batch_
 ↪processing/dtm*.tif
```

You can then visualize the vrt with *qgis*. Add the vrt twice, and set the properties of the lower layer to hillshade. Set the upper layer to Singleband PseudoColor and choose a pleasing color ramp. Then set the transparency of the upper layer to 50% and you'll get a nice display of the terrain.



Part V

Final Project

The final project brings together a number of PDAL processing workflow operations into a single effort It builds upon the exercises to enable you to use the capabilities of PDAL in a coherent processing strategy, and it will give you ideas about how to orchestrate PDAL in the context of larger data processing scenarios.

Given the following pipeline for fetching the data, complete the rest of the tasks:

```
{
    "pipeline": [
        {
            "type": "readers.ept",
            "filename":"http://na-c.entwine.io/dublin/",
            "bounds":"([-697041.0, -696241.0], [7045398.0, 7046086.
↪0],[-40, 400])"

        },
        {
            "type": "writers.las",
            "compression": "true",
            "minor_version": "2",
            "dataformat_id": "0",
            "filename":"st-stephens.laz"
        }
    ]
}
```

- Read data from an EPT resource using readers.ept (https://pdal.io/stages/readers.ept.html#readers-ept) (See *Entwine* (page 39))
- Thin it by 1.0 meter spacing using filters.sample (https://pdal.io/stages/filters.sample.html#filters-sample) (See *Thinning* (page 65))
- Filter out noise using filters.outlier (https://pdal.io/stages/filters.outlier.html#filters-outlier) (See *Removing noise* (page 57))
- Classify ground points using filters.smrf (https://pdal.io/stages/filters.smrf.html#filters-smrf) (See *Identifying ground* (page 71))
- Compute height above ground using filters.hag (https://pdal.io/stages/filters.hag.html#filters-hag)
- Generate a digital terrain model (DTM) using writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal) (See *Generating a DTM* (page 77))
- Generate a average vegetative height model using writers.gdal (https://pdal.io/stages/writers.gdal.html#writers-gdal)

Note: You should review specific *Exercises* (page 29) for specifics how to achieve each task.

Part VI

Notes

CHAPTER

THIRTEEN

NOTES

CHAPTER

FOURTEEN

NOTES

CHAPTER

FIFTEEN

NOTES

CHAPTER

SIXTEEN

NOTES

CHAPTER

SEVENTEEN

NOTES

CHAPTER

EIGHTEEN

NOTES

BIBLIOGRAPHY

[Gle07] Craig L. Glennie. Rigorous 3D error analysis of kinematic scanning LIDAR systems. *Journal of Applied Geodesy*, jan 2007.

INDEX

B

boundary, 43

C

capstone, 117
classification, 71, 88, 90
classifications, 58
Clipping, 47
Colorization, 53
Conda, 25
coordinate system, 30
csd, 105
CSV, 30

D

Denoising, 57
Density, 60
density, 65
DSM, 77
DTM, 77

E

elevation model, 77
EPT, 40

F

filtering, 71, 88

G

GDAL, 53
georeferencing, 15, 105
GNSS/IMU, 15, 105
ground, 71, 88

H

hexagon tessellation, 60
histogram, 104

I

info command, 29
installation, 29
intensity, 90

J

JSON, 30

M

matplotlib, 104
metadata, 30

N

nearby, 33
nearest, 33
Numpy, 104

O

OGR, 43, 47, 60
Optech, 105
outliers, 57

P

poisson, 65
Potree, 40
project, 117
Python, 104

Q

QGIS, 43
query, 33

R

range filter, 58

Raster, 53
rasterization, 90
Reprojection, 36
RGB, 53
RIEGL, 105

S

sample, 65
search, 33
SOCS, 15
software installation, 25
spatial reference system, 30
Start Here, 29

T

thinning, 65

U

UTM, 36, 105

V

Vector, 47
voxel sampling, 65

W

web services, 40
WGS84, 36, 105