LIDAR DATA PROCESSING WITH PDAL

PREREQUISITES

- Basic knowledge of LiDAR technology and LiDAR data
- Basic understanding of JSON structures
- Windows or Linux command line interface

BEFORE YOU BEGIN

1. Follow PDAL quick-start guide to set up your computer
2. Download the following datasets to your computer and keep the file paths handy:
   - `20140326_18TWL850045.laz`: tile 18TWL850045 directly downloaded from the 2013-2014 USGS Post Sandy NYC LiDAR dataset

EXERCISE 1: READ LAS/LAZ FILES

This exercise aims to get you up and started with PDAL. You will use `info` – one of the PDAL applications to print out information about a point cloud file.

1. Open a terminal with PDAL activated. Replace the `<intput_path>` in the command below by the path to the LAS/LAZ file you downloaded, and run the command:

```
pdal info --metadata <input_path>
```

   The command should print out the content stored in the LAS/LAZ file's header and the variable length records in the JSON format.

   - Look for the JSON key `count` in the JSON string printed out. The value of that key is the number of points in the file (e.g. 19,549,079).
   - Look for the key `spatialreference` or `srs`. The value of that key is the string defining the coordinate system of the data in the file, which is encoded in the Well-Known Text (WKT) format.
   - The token `AUTHORITY["EPSG","4759"]` under the key `GEOGCS` indicates that the horizontal datum of the dataset is the North American Datum 1983 (NAD83) with EPSG code of 4759. The horizontal coordinates are in degrees.
   - The attribute `NAVD88 height` under the key `VERT_CS` indicates that the vertical coordinates are in the orthometric vertical datum NAVD88 (geoid12b). The vertical coordinates are in meters.

2. Now remove the `--metadata` tag from the command and run again.

```
pdal info <input_path>
```

This time, instead of printing the metadata, PDAL prints some statistics computed from the data file: the spatial bounding box (key `bbox`) in several different formats; and average, count, maximum, minimum, standard deviation, and variance for each of the dimensions (including coordinates and attributes).

3. Modify the command to print out the first 10 points in the file. The output is a JSON string containing the points with all of their attributes.

```
pdal info <input_path> -p 0-10
```

EXERCISE 2: PERFORM COORDINATE TRANSFORMATION

As you see in Exercise 1, the horizontal coordinates in LAZ file are in degrees while the vertical coordinates are in meters. The mismatch prevents the file to be correctly parsed by CloudCompare and other software which are unaware of the unit mismatch or the coordinate system definitions embedded in the file. In this exercise, we will transform the data into the local map projection for New York and Long Island (EPSG:2263), which represents all of the coordinates in US survey feet.

You can use either the `translate` or the `pipeline` applications of PDAL to perform the coordinate transformation. `pipeline` has higher capability while `translate` is good for quick and simple transformation / translation. You will use `pipeline` for this exercise.

To use `pipeline`, you need to define the data processing workflow (i.e. the pipeline) in JSON format and provide that to PDAL's `pipeline` application. A JSON pipeline file contains an ordered sequence of stages. The first stage is always a reader, which is responsible for getting data into the pipeline from a data source (i.e. a file or a database). The subsequent stages are filters (or transformers), each of which performs some kind of transformation to the data received from the immediate preceding stage, and passes on the result to the next stage. The last stage in a pipeline is a writer, which writes the data back to a data source.

The pipeline in this exercise has 3 stages: (1) a reader to read the LAZ file, (2) a filter to perform the coordinate transformation, and (3) a writer to write the data back to a new LAZ file. You can see them numbered in the JSON block below.

1. Put the text below (which actually encodes a JSON object) to any text editor (notepad, sublime text, etc.). Replace `input.laz` and `output.laz` by the paths to your input and output files. Save the text as a JSON file (e.g. epsg4759to2263.json).

```
{
  "pipeline":[
    //// (1) read LAS/LAZ ////
    {
      "type":"readers.las",//reader type: LAS/LAZ
      "filename":"input.laz"//input path
    },
    //// (2) transform coordinates ////
    {
      "type":"filters.reprojection",//filter type: reproject (tranform CRS)
      "in_srs":"epsg:4759",//source (in) CRS
      "out_srs":"epsg:2263"//target (out) CRS
    },
    //// (3) write LAS/LAZ ////
    {
      "type":"writers.las",// writer type: LAS/LAZ
      "scale_x":"0.01",//define the scale parameters based on the data's
expected accuracy
```

```
      "scale_y":"0.01",
      "scale_z":"0.01",
      "offset_x":"auto",
      "offset_y":"auto",
      "offset_z":"auto",
      "filename":"output.laz"
    }
  ]
}
```

The JSON file together with the comments are self-explained. There are only a few things to notice here:

- PDAL's LAS readers can handle both LAS and LAZ.
- You only define the horizontal CRS transformation here and PDAL is smart enough to scale the vertical coordinates accordingly. In some cases, you will need to explicitly instruct PDAL by using a proj4 or a WKT string. You can find more information about that here.
- You should define the scale parameters when writing LAS/LAZ files. The scale parameters allow you to prevent numeric precision loss when storing your data. Internally, LAS/LAZ keeps coordinates as 8-byte long integers. You can check the LAS format specification for more information about the transformation between the internal integer representation and the real coordinates. For now, you can think of the scale parameters as an equivalence to the number of decimal digits when writing down real numbers. As the data's accuracy is lower than 1/10 ft, keeping 2 decimal digits is a fair choice to have some redundancy. And that translates to scale parameters of 0.01 for all of the dimensions.

2. Open your PDAL terminal as in Exercise 1 and run the following command. Remember to replace `<pipeline_file.json>` by the path to the JSON file you created in step 1. The command parses the pipeline (i.e. data processing workflow) from the JSON file with all of the parameters and executes the data processing.

```
pdal pipeline <pipeline_file.json>
```

After the execution completed, you can open the output file in CloudCompare and correctly view, measure, and process the data. The process described in this exercise should enable you to access and correctly prepare any georeferenced LiDAR dataset. The data made available by the USGS (currently more than 10 trillion points covering approximately 53% of the country) are available for download and view online:

- FTP download: ftp://rockyftp.cr.usgs.gov/vdelivery/Datasets/Staged/NED/LPC/projects/
- Online 3D viewers: https://usgs.entwine.io/

PDAL can do much more than coordinate transformation. Examples of things PDAL can do include: data subsampling, cleaning (remove noise and outliers), splitting a file into tiles, clipping a portion, filtering the data by a dimension, colorizing a point cloud using a raster image. PDAL also has python binding and can be used in a Jupyter notebook. This short hands-on is just enough for you to get started. But if you have enough interest, the extensive resource on PDAL website will let you go much further.