Pointcloud Data Abstraction Library
About the library

- 1.1.0 November 2015
- 1.2.0 April 2016
- 1.3.0 August 2016
- 1.4.0 December 2016
- 1.5.0 April 2017
- 1.6.0 Sept/Oct 2017
- BSD licensed (supports proprietary plugins)
- C++ development
  - Git repo (https://github.com/PDAL/PDAL)
  - Pull requests welcome
1.4 / 1.5 Changes

- json everywhere - xml is gone
- gdal writer
- mbio reader (bathy data)
- smrf bare earth filter
- stream mode
- pdal metadata and pipeline VLRs options for writer.las
- filename globbing
- java/jni bindings
- greyhound reader
1.6 Changes

- filters.cpd and filters.icp
- filters.predicate and programmable merged to filter.python
- new filters.matlab
- native filters.poisson (watertight surface)
- enhancements to writers.gdal
- new openscenegraph reader/writer
- new head/tail/randomize filter
- new laz 1.4 writing
readers / writers

- readers.bpf
- readers.buffer
- readers.faux
- readers.gdal
- readers.geowave
- readers.greyhound
- readers.ilvis2
- readers.las
- readers.mbio
- readers.mrsid
- readers.nitf
- readers oci
- readers.optech
- readers.pcd
- readers.pgpointcloud
- readers.ply
- readers.pts
- readers.qfit
- readers.rxp
- readers.sbet
- readers.sqlite
- readers.text
- readers.tindex
- writers.bpf
- writers.gdal
- writers.geowave
- writers.las
- writers.matlab
- writers.nitf
- writers.null
- writers.oci
- writers.pcd
- writers.pgpointcloud
- writers.ply
- writers.sqlite
- writers.text
filters.approximatecoplanar
filters.assign
filters.chipper
filters.cluster
filters.colorinterp
filters.colorization
filters.computerange
filters.crop
filters.decimation
filters.divider
filters.eigenvalues
filters.elm
filters.estimaterank
filters.ferry
filters.greedyprojection
filters.gridprojection
filters.groupby
filters.hag
filters.hexbin
filters.iqr
filters.kdistance
filters.locate
filters.lof
filters.mad
filters.merge
filters.mongus
filters.mortonorder
filters.movingleastsquares
filters.normal
filters.outlier
filters.overlay
filters.pclblock
filters.pmf
filters.poisson
filters.predicate
filters.programmable
filters.radialdensity
filters.randomize
filters.range
filters.reprojection
filters.sample
filters.smrf
filters.sort
filters.splitter
filters.stats
filters.transformation
filters.voxelgrid
APPLICATIONS

- delta
- density
- diff
- ground
-hausdorff
- info
- merge
- pcl
- pipeline
- random
- sort
- split
- tindex
- translate
--stats filter.stats, bbox, counts, ranges, enumerations
--metadata reads header values, srs values and info
--boundary wkt and json, density, area, filters.hexbin
--dimensions limit stats to dimension(s) list
--schema dimension list and types
--point individual point values
$ pdal info --boundary source.laz
$ pdal info --boundary
source.laz
--filters.hexbin.threshold=5
--filters.hexbin.sample_size=1000000
--filters.hexbin.edge_size=1

density: 25.15629811
default density was 18.72858644
- basic format changes
- options to set readers/writers/filters
- `--json` option for reading filters from pipeline file
- `--pipeline` option to create a pipeline file

```bash
pdal translate -i myfile.las -o myfile.laz

pdal translate myfile.las myfile.laz --writers.las.system_id="Custom"
--writers.las.scale_z=0.00001

pdal translate myfile.ntf oufile.laz -f filters.reprojection
--filters.reprojection.out_srs="epsg:32641+3755"
```
- Access to the full power of pdal
- allows stacking of range of operations with one pass through data
- great for programmatically building workflows
- supports command line override of values for batch processing
$ pdal pipeline range.json
   --readers.las.filename=myfile.las
   --writers.las.filename=myrangefile.laz

$ find . -name "*.laz" | xargs -I{}
pdal pipeline range.json
   --readers.las.filename={}
   --writers.las.filename=newdir/{}

{
   "pipeline":[
      {
          "type": "readers.las",
      },
      {
          "type": "filters.range",
          "limits": "Z[0:99999]"
      },
      {
          "type": "writers.las"
      }
   ]
}

• supports globbed inputs
• tagging of stages to choose what gets applied
• complex workflows
```json
{
  "pipeline": [
    {
      "filename": "Eastman_LAZ_Final/Input/*.laz",
      "type": "readers.las"
    },
    {
      "assignment": "Classification[::]=0",
      "tag": "filtersassign",
      "type": "filters.assign"
    },
    {
      "inputs": ["filtersassign"],
      "extract": "true",
      "tag": "filtersoutlier",
      "type": "filters.outlier"
    },
    {
      "inputs": ["filtersoutlier"],
      "max_distance": "7",
      "approximate": "true",
      "tag": "filterspmf",
      "type": "filters.pmf"
    },
    {
      "inputs": ["filterspmf"],
      "length": "1000",
      "type": "filters.splitter",
      "tag": "filterssplitter"
    },
    {
      "inputs": ["filterssplitter"],
      "type": "writers.las",
      "tag": "writerslas",
      "a_srs": "EPSG:26911",
      "scale_x": "0.001",
      "scale_y": "0.001",
      "scale_z": "0.001",
      "offset_x": "auto",
      "offset_y": "auto",
      "offset_z": "auto",
      "filename": "Eastman_LAZ_Final/Output/Eastman_161115_GND_CLS_#.laz"
    },
    {
      "inputs": ["writerslas"],
      "type": "filters.merge",
      "tag": "filtersmerge"
    },
    {
      "inputs": ["filtersmerge"],
      "limits": "Classification[2:2]",
      "type": "filters.range",
      "tag": "filtersrange"
    },
    {
      "inputs": ["filtersrange"],
      "type": "writers.gdal",
      "filename": "Eastman_LAZ_Final/Output/Eastman_161115_GND_CLS_50CM_DEM.tif",
      "radius": "0.7071",
      "resolution": "0.5",
      "output_type": "idw",
      "nodata": "-9999",
      "window_size": "2",
      "gdalopts": "predictor=3, tiled=yes, compress=deflate"
    }
  ]
}
```
- assigning and reprojecting data via pipeline
- can also set via translate
- includes vertical datum reprojection (with grid shift files)

```json
{  
  "pipeline": [  
    {"type":"readers.las" },  
    { "type":"filters.reprojection",  
      "in_srs":"EPSG:26918",  
      "out_srs":"EPSG:26919"  
    },  
    { "type":"writers.las"}  
  ]
}
```

```
pdal pipeline proj.json
--readers.las.filename=src.laz
--writers.las.filename=dest.laz
```

```
ls *.laz | xargs pdal translate --filters filters.reprojection
--filters.reprojection.in_srs=epsg:32641+5773
--filters.reprojection.out_srs=epsg:32641+3775 -i {} -o newdir/{}
```
GDAL Writer

- turn a point cloud into a surface
- min/max/mean/count/stdev/idw
- output to most gdal raster types (single band)
- supports an array of gdal creation options
- stream mode for reducing memory overhead
GDAL Writer

```
{
  "pipeline": [
    {
      "type": "readers.las",
      "filename": "source.laz"
    },
    {
      "type": "filters.range",
      "limits": "Classification[2:2]"
    },
    {
      "type": "writers.gdal",
      "resolution": "1.0",
      "filename": "destination",
      "output_type": "idw",
      "gdaldriver": "GTiff",
      "gdalopts": "TILED=YES",
      "gdalopts": "COMPRESS=DEFLATE"
    }
  ]
}
```
- can use any gdal image source
- projections need to match

```json
{
    "pipeline": [
        {
            "type": "filters.colorization",
            "raster": "https://isse.cr.usgs.gov/arcgis/rest/services/Orthoimagery/USGS_EROS_Ortho_1Foot/ImageServer/exportImage?f=image&bbox=333615,4691634,330227,4689471&imageSR=26919&bboxSR=26919&size=3388,2163&format=tiff&pixelType=U8"
        }
    ]
}
```
- applies a color range (predefined or specified) to a dimension
- can set min/max or calculate, or filters.mad

```json
{
    "type": "filters.colorinterp",
    "ramp": "heat_map",
    "mad": true,
    "k": 2
}
```
Classifies a pointcloud into ground and non-ground
  ○ a filter and an app
  ○ classify and extract have been removed
  ○ use filters.range to extract
  ○ use filters.assign to clear existing classes
  ○ pdal ground uses Progressive Morphological Filter
  ○ same as a pipeline with filters.pmf
  ○ filters.smrf can often produce better/faster results
Ground - Pre
Using filters.smrf
- an app or several filters
- filters.divider - point count or count of files

```
pdal split -i source.laz -o dest_cap.las --capacity 3000000
```
- can break in tiles by size
- optional specify an origin x/y
- used with `pdal split or filters.splitter`

```bash
pdal split
-i source.laz
-o dest_length.laz
--length 400
--origin_x 476000
--origin_y 6327000
```
- an app or a filter
- applies a filters. `sort` to a file
- can be very useful to increase compression (laz)

```bash
pdal sort CO_ArkansasValley_2010_000536.laz
CO_ArkansasValley_2010_000536-time-sort.laz
    --filters.sort.dimension=GPSTime
    --writers.las.forward=all

22292036  CO_ArkansasValley_2010_000536-time-sort.laz
56629291  CO_ArkansasValley_2010_000536.laz
```
- app or filter
- creates a tile index of point clouds in any ogr vector format
- extent or hexbin boundaries, can specify output srs
- can then be used for merge/clip operations
- readers.tindex for pipeline operations

```
pdal tindex filename.db -f sqlite "*.laz" --t_srs "epsg:4326"
```
pdal tindex --merge
--tindex tileindex.shp
--filespec output.laz
--polygon "POLYGON ((476211 6327699, 476296 6327664, 476326 6327560, 476247 6327508, 476101 6327519, 476063 6327617, 476211 6327699))"
pdal density -i source.laz -o density_surface.db -f SQLite --filters.hexbin.edge_size=10
- available via pypi
- read las data to numpy with

```python
""" fetch PDAL data as a numpy array"

json = open('/data/pipeline/pipeline_read.json','rb').read
r = libpdalpython.PyPipeline(json)
r.execute()
arrays = r.arrays()
```
● http://pdal.io
  ○ rtd format
  ○ single pdf download
  ○ content reorganized
● new workshop documentation
  ○ 100+ pages
  ○ uses qgis and osgeo4w64
● new tutorials
source: http://pdal.io
  ○ dev repo at https://github.com/PDAL/PDAL

docker hub - fastest way to pdal
  ○ docker pull pdal/pdal:<release>
  ○ pdal/dependencies image for custom builds

windows: OSGeo4W64 (up to date builds)

linux centos/redhat rpms

linux debian unstable
questions?