Individual Tree Detection

Sam Ericksen, Oren Nardi

2022-03-15

data collected by Oren Nardi, at College of the Redwoods campus in Eureka.

Images processed in WebODM using the default "Forest" parameters

Reading a Point Cloud

First, using the LidR package, we read in the dataset in question. In this case, to make the script more universally available for all users we use the choose.file() function. This function opens a file explorer box that allows the user to choose any file from there computer as they wish. We also want to do a quick initial validity test on the data - simply ploting it.

```
las <- readLAS(file.choose())
##
## plot(las) ## initiates a 3D plot, must be run in R
##
```

We can also run a deep check on the las to see if any issues are present that may impede future analysis using las_check(). It's also nice to check and see if the file is already normalized visually, this can be done by plotting the minimum height values.

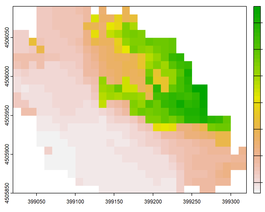
```
las_check(las)
```

```
##
##  Checking the data
##   - Checking coordinates... [0;32m <U+2713> [0m
##   - Checking coordinates type... [0;32m <U+2713> [0m
##   - Checking coordinates range... [0;32m <U+2713> [0m
##   - Checking coordinates quantization... [0;32m <U+2713> [0m
##   - Checking attributes type... [0;32m <U+2713> [0m
##   - Checking ReturnNumber validity... [0;32m <U+2713> [0m
##   - Checking NumberOfReturns validity... [0;32m <U+2713> [0m
##   - Checking ReturnNumber vs. NumberOfReturns... [0;32m <U+2713> [0m
##   - Checking RGB validity...
##  [1;33m  <U+26A0> Invalid data: RGB colors are recorded on 8 bits instead of 16 bits. [0m
##   - Checking absence of NAs... [0;32m <U+2713> [0m
##   - Checking duplicated points...
##  [1;33m  <U+26A0> 36 points are duplicated and share XYZ coordinates with other points [0m
##   - Checking degenerated ground points... [0;37m skipped [0m
##   - Checking attribute population...
##  [0;32m  <U+0001F6CB> 'gpstime' attribute is not populated [0m
##  [0;32m  <U+0001F6CB> 'PointSourceID' attribute is not populated [0m
##  [0;32m  <U+0001F6CB> 'ScanDirectionFlag' attribute is not populated [0m
##  [0;32m  <U+0001F6CB> 'EdgeOfFlightline' attribute is not populated [0m
##   - Checking gpstime incoherences
##  [0;31m  <U+2717> 1 pulses (points with the same gpstime) have points with identical ReturnNumber [0m
##   - Checking flag attributes... [0;32m <U+2713> [0m
##   - Checking user data attribute...
##  [0;32m  <U+0001F6CB> 1111572 points have a non 0 UserData attribute. This probably has a meaning [0m
## Checking the header
##   - Checking header completeness... [0;32m <U+2713> [0m
##   - Checking scale factor validity... [0;32m <U+2713> [0m
##   - Checking point data format ID validity... [0;32m <U+2713> [0m
##   - Checking extra bytes attributes validity... [0;32m <U+2713> [0m
##   - Checking the bounding box validity... [0;32m <U+2713> [0m
##   - Checking coordinate reference system... [0;32m <U+2713> [0m
## Checking header vs data adequacy
##   - Checking attributes vs. point format... [0;32m <U+2713> [0m
##   - Checking header bbox vs. actual content... [0;32m <U+2713> [0m
##   - Checking header number of points vs. actual content... [0;32m <U+2713> [0m
##   - Checking header return number vs. actual content... [0;32m <U+2713> [0m
## Checking coordinate reference system...
##   - Checking if the CRS was understood by R... [0;32m <U+2713> [0m
## Checking preprocessing already done
##   - Checking ground classification... [0;31m no [0m
##   - Checking normalization... [0;31m no [0m
##   - Checking negative outliers... [0;32m <U+2713> [0m
##   - Checking flightline classification... [0;31m no [0m
## Checking compression
##   - Checking attribute compression...
##  [0;32m  <U+0001F6CB> Compression supported only from rlas 1.6.0 [0m
```

```
epsg(las) ##get the coordinate system of the file.  Here we get 32610, which has a vertical unit of meters, so we know the Z values will be in meters
```

```
## [1] 32610
```

```
plot(pixel_metrics(las, ~min(Z), res = 10))
```




```
min(las@data$Z)
```

```
## [1] 15.97
```

Here both las_check() and a plot of minimum height values indicate a non-normalized point cloud (all values are above 0 meters). Additionally, the deep check of the file indicates a few interesting anomolies that need to be addressed:

- There are a few duplicate points that should be filtered out
- Ground points are not classified
- There is some issue with the RGB data, but this is likely only an issue for LidR, and can be ignored for now

Duplicates can be filtered very quickly:

```
las <- filter_duplicates(las)
```

Classifying Ground Points

When working with point clouds derived from photogrammetry, there is often no classifications initially associated with the points. ASPRS provides a structure for classification of point clouds to work with, but for now our main concern are ground points. Without ground points classified it is not possible to create a canopy height model, as the canopy height is derived in relation to the ground surface.

When we plotted the point cloud earlier we noticed that there aren't many points under the canopy, this makes ground classification a little tricky. Fortunately, there were ground points collected on the outskirts of the area of interest, so we can work with that.

Because of the lack of ground under the canopy we will use a Cloth Simulation Filter (Zhang et al. 2016). There are a few attributes that can be adjusted by the user, and the most important in this case is cloth rigidness, although 3L is generally for flat ground, to force some interpretation of the under-canopy ground we will use this setting. Once ground points are classified we plot just the ground points (ASPRS class 2L) to see how it performed.

```
las <- classify_ground(las, csf(rigidness = 2L))

## plot(pixel_metrics(las[las@data$Classification==2L], ~mean(Z), res = 2))
##
## plot(las[las@data$Classification==2L])
```

Plotting the average height value of the ground points shows that there are clearly some miss-classified ground points back where the canopy is very dense (Northeast corner). It looks like we can safely remove ground points above 40m during creation of a canopy height model.

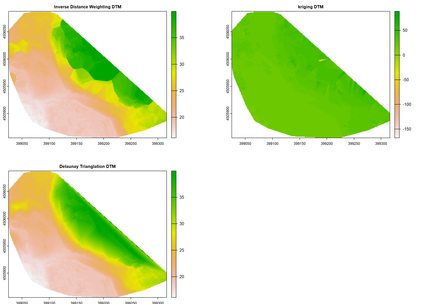
Creating a Canopy Height Model

The first step in creating a canopy height model will be the creation of a digital terrain model (DTM). There are a three methods of this included in the LidR package:

- K-Nearest Neighbor Methods
    - Inverse Distance Weighting
    - Krigging
- Delaunay Triagulation

```
grnd_points <- filter_poi(las, Classification == 2L, Z < 40) ##filter point cloud to include only non-anomalous ground points

idw_terrain <- rasterize_terrain(grnd_points, algorithm = knnidw())
krig_terrain <- rasterize_terrain(grnd_points, algorithm = kriging())
tin_terrain <- rasterize_terrain(grnd_points, algorithm = tin())
```

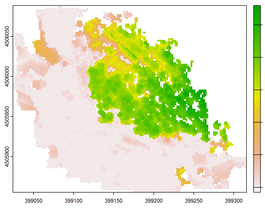


Here we can see the Kriging does not deal with extrapolation well with default parameters. IDW and TINing seem to give reasonable outputs and a closer inspection seems to indicate that TINing gave a smother slope transition when extrapolating under-canopy ground points. We will continue with the canopy height model using the DTM created using TINing, but one couldn't be blamed for considering using the IDW terrain model either.

```
## normalize the point cloud heights
las_norm <- normalize_height(las, algorithm = tin_terrain)
```

```
## Warning: 99187 points do not belong in the raster. Nearest neighbor was used to
## assign a value.
```

```
plot(pixel_metrics(las_norm, ~max(Z), res = 2))
```



Still To Come…

- *Classify noise (may reduce maximum height)*
- *Individual Tree Detection*
- *Indv. Tree points geometry*
- *plot CHM with individual trees overlaid*

References

Zhang, Wuming, Jianbo Qi, Peng Wan, Hongtao Wang, Donghui Xie, Xiaoyan Wang, and Guangjian Yan. 2016. "An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation." *Remote Sensing* 8 (6): 501. https://doi.org/10.3390/rs8060501.