Lab 6: 3D Point clouds from photos vs. LiDAR

TA: Anthony Stewart

2023-05-02

Objectives:

- Rumple index and canopy height models
- Digital Aerial Photogrammetry (DAP)
- Linear regression analysis with rasters
- Creating forest characterization maps

Data and Software:

- LAB6Data.zip folder, downloadable from Canvas
- Data from lab 5
- Downloads from Washington DNR lidar Portal
- RStudio
- CloudCompare

What you will turn in:

Lab 6 Submission Quiz (https://canvas.uw.edu/courses/1633883/quizzes/1856519)

Welcome to Lab 6 for ESRM433/SEFS533!

In this lab we will be comparing Digital Aerial Photogrammetry point clouds with ALS. We will be creating a forest ecology metric called the rumple index with both sets of data. Then, we'll look for relationships between field data and our rumple index and extrapolate our metrics out across an area.

Lastly, you will be given a task to identify forests with specific characteristics in Pack Forest.

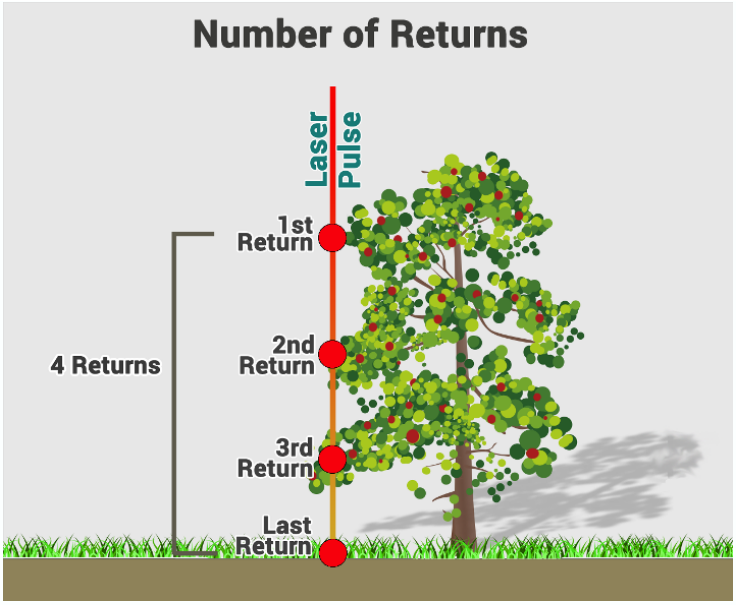
INTRO: Ecological meaning of lidar metrics

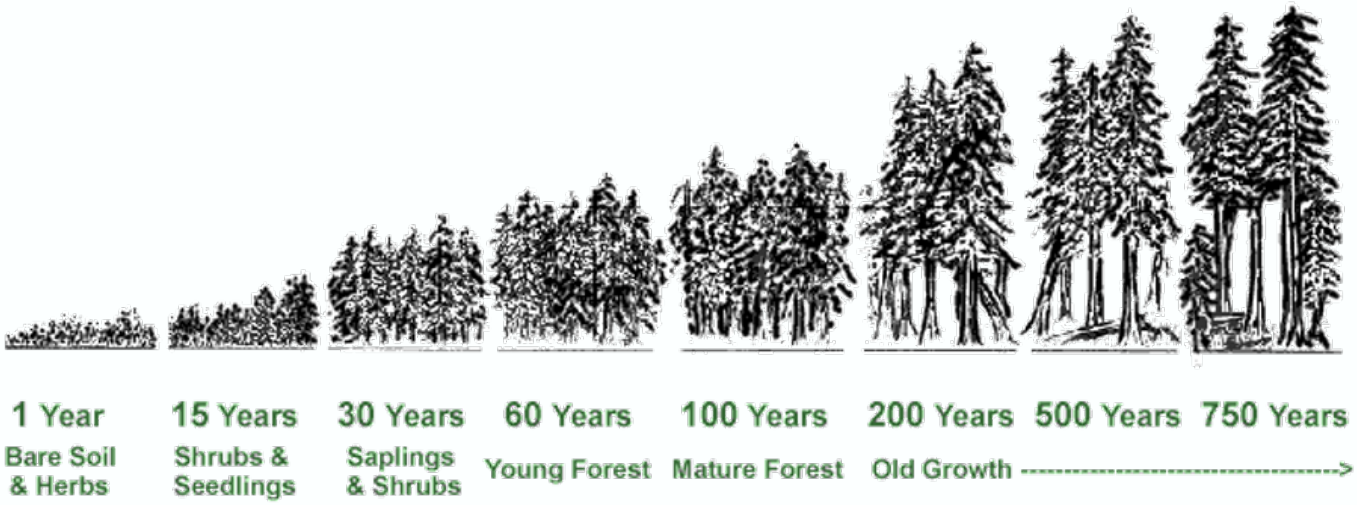
The number and locations of returns depends on the height and density of forest canopy. Some pulses may have more or less than 4 returns. If there are many pulses that hit this tree with similar locations, the mean values would be between the 2nd and 3rd returns, the max would be the 1st returns, the 95th percentile would be just below the 1st returns, and the 25th percentile would be around the location of the 3rd return.

`Mean` , `Max` , `95th percentile` , & `25th percentile` are often used lidar metrics in ecology. We can make a few general statements about these metrics and how they relate to forest structure. There are many reasons

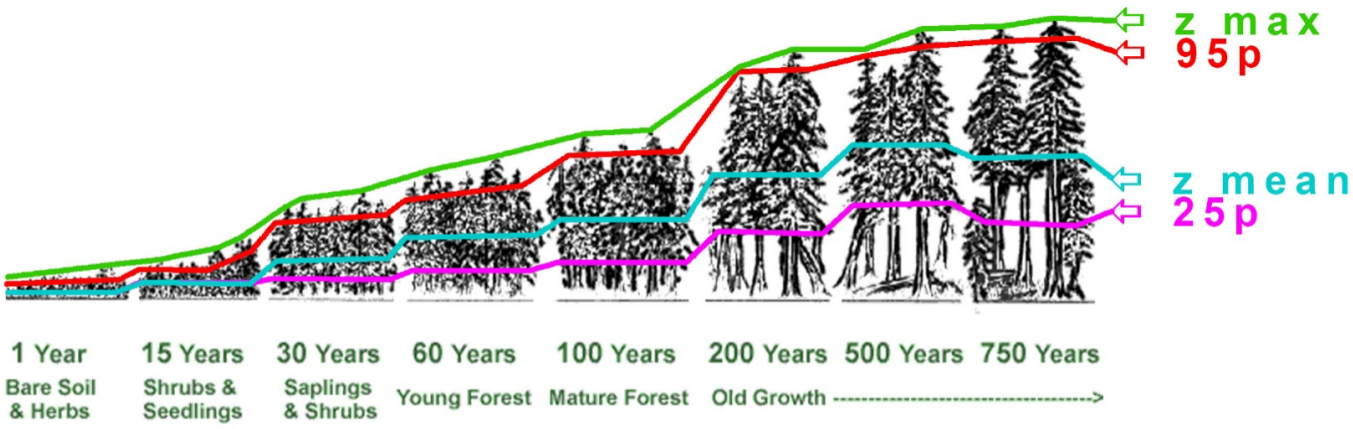
why these generalizations won't always be true, but hopefully this will aid in visualizing why we use these metrics.

To generalize about forest age and structure and percentiles, let's look at forest structure at different successional stages:





A conceptual diagram of where our lidar metrics would be located at the different successional stages:

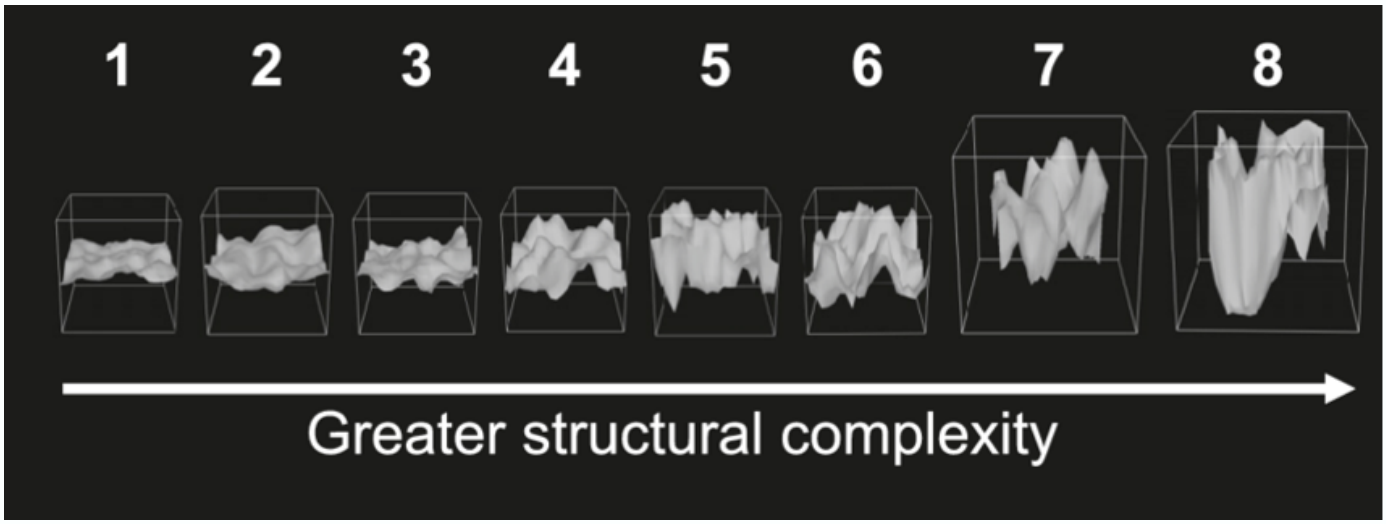


Direct measurements from lidar like `Z max` indicate a specific, tangible property of a forest or tree. In this case `Z max` is potentially the tallest tree in an area, however, noise in the data might make `Z max` unreliable so `95th percentile` / `95p` may be reasonable to take as a direct measurement of tallest trees.

Other metrics like `Z mean` and `25th percentile` /' `25p` aren't necessarily a direct measurement of forest

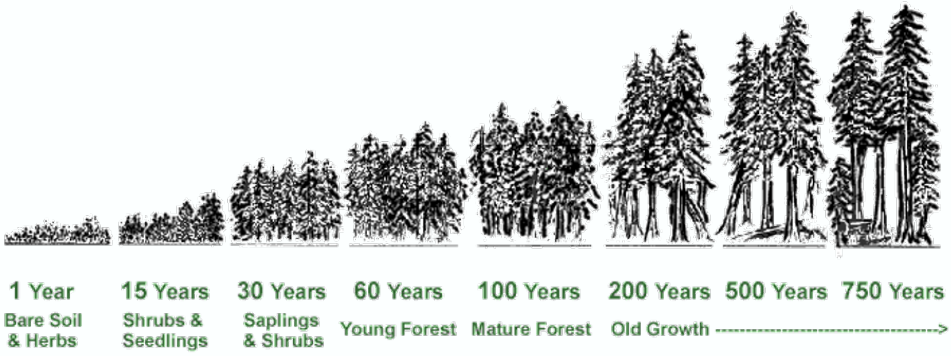
structure but they can provide a lot of information about the general structure of a forest. 25p often correlates with the average height to crown base in a forest.

Indirect measurements such as `Rumple` are calculated from the lidar returns in order to better characterize forest structure. In this case `Rumple` is the ratio of the surface area of a canopy height model to the area of that model projected to the ground. **A higher rumple, the more complex the canopy surface**. Rumple has been used to model basal area, QMD, and other forest metrics.



Rumple Index - Like a blanket over the forest canopy!

Conceptually, think about an idyllic old growth forest. There will be some giant tall trees with gaps in the forest canopy. The gaps and tall trees will result in a **higher rumple value**. An even aged stand in a plantation will likely have a **relatively "smooth" or low rumple** as the canopy is more homogeneous. Keep in mind that disturbance events like fire, disease, logging also change forest structure.



QUESTION 1: Based on the illustration above, which age class would you expect to have the highest rumple, and the lowest rumple? Why? Note that this is a generalization and isn't necessarily true in all cases.

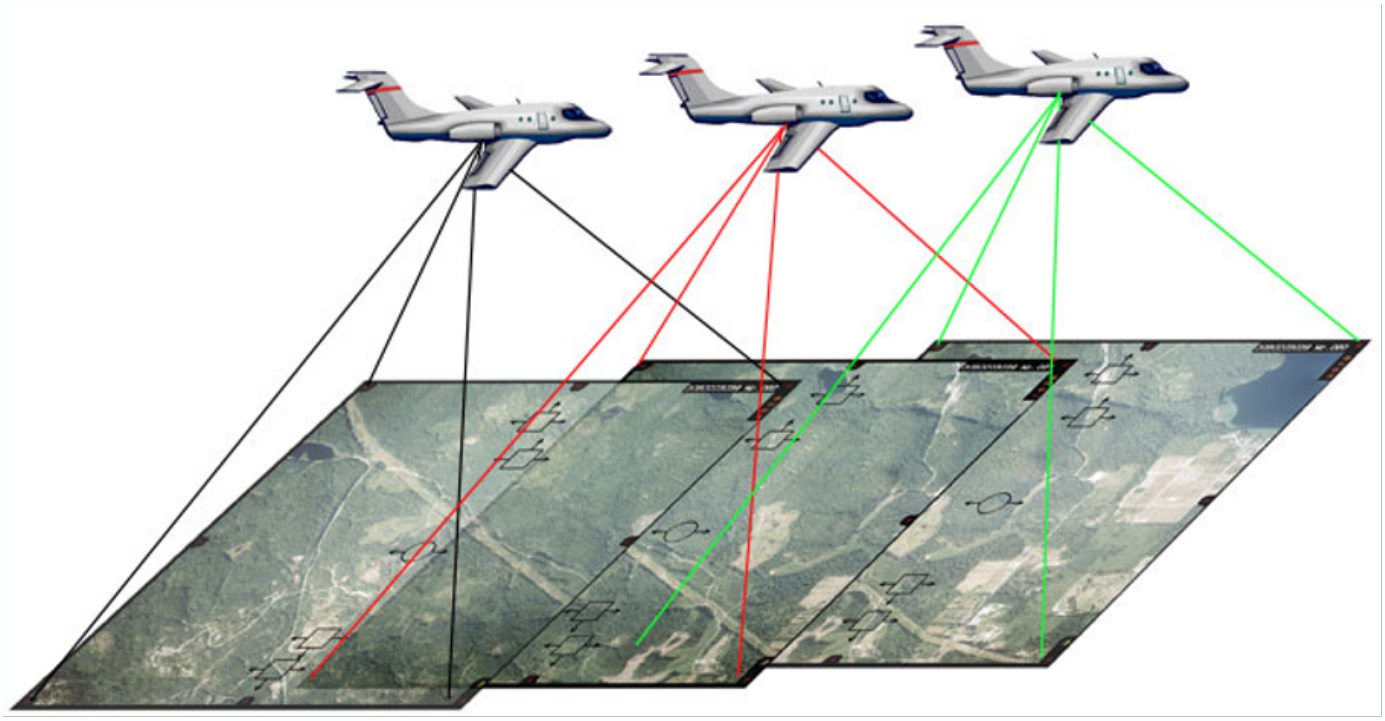
Other forest metrics like `Trees Per Acre (TPA)` are not directly measured by lidar but relationships can be found between observed `TPA` and lidar metrics. In our example above, there would be many more `TPA` in the younger forests. Younger forests tend to have a lower `25p` and `zmean`, therefore, `25p` and `zmean` in a multiple linear regression may be able to model `TPA` while not being a direct measurement.

A relationship in one forest type likely won't be the same in a different forest type. Some generalized metrics may be possible, but new models should be developed for different forests.

Digital Aerial Photogrammetry (DAP)

Photogrammetry is the science of creating structural information and measurements from photographs and imagery. Outputs include orthorectified photographs to 3D models. In order to create 3D models, photogrammetry relies on stereo imagery or the use of overlapping images to emulate a 3D model of a surface. One of the best examples of this is Google's 3D imagery in Google Earth: https://youtu.be/suo_aUTUpps (https://youtu.be/suo_aUTUpps)

The video focuses on Googles creation of 3D buildings and topography but the basics apply to the data that we are working with today.



Unfortunately, DAP isn't readily available off the internet like lidar data is. Fortunately, you have friends in SEFS that can provide you with some data.

The DAP point cloud that we are working with was derived from National Agriculture Imagery Program (NAIP).

https://www.usgs.gov/centers/eros/science/usgs-eros-archive-aerial-photography-national-agriculture-imagery-program-naip (https://www.usgs.gov/centers/eros/science/usgs-eros-archive-aerial-photography-national-agriculture-imagery-program-naip)

NAIP imagery is readily available for download, but the imagery that is made public is seamless orthographically corrected imagery. The images are basically merged into one orthomosaic where distortion from the oblique angle at the edges of images, and differences in topography are removed.

https://en.wikipedia.org/wiki/Orthophoto (https://en.wikipedia.org/wiki/Orthophoto)

To create a DAP point cloud, you need those raw images as those oblique angles and differences due to topography are what allow for structure to be modeled. DAP is often called Structure from Motion. As you can imagine, getting a point cloud from images can be very helpful in forest monitoring. DAP can be done with images from airplanes or from drones. Drones will generate much higher resolution images (<3cm/px) while NAIP imagery has significantly lower spatial resolution (1m to 0.5m). However, the nation-wide coverage with NAIP imagery every year or two makes it a fantastic resource for resampling forested areas.

For more information about DAP and forests:

Goodbody, T.R.H., Coops, N.C. & White, J.C. Digital Aerial Photogrammetry for Updating Area-Based Forest

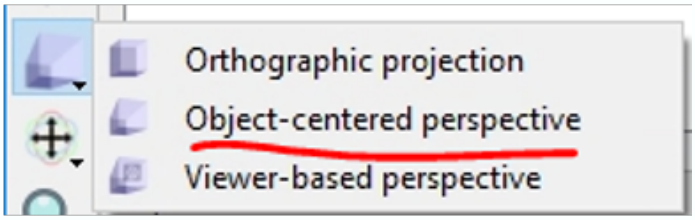
Inventories: A Review of Opportunities, Challenges, and Future Directions. Curr Forestry Rep 5, 55–75 (2019). https://doi.org/10.1007/s40725-019-00087-2 (https://doi.org/10.1007/s40725-019-00087-2)

PART 1: Comparing DAP to ALS, Rumple

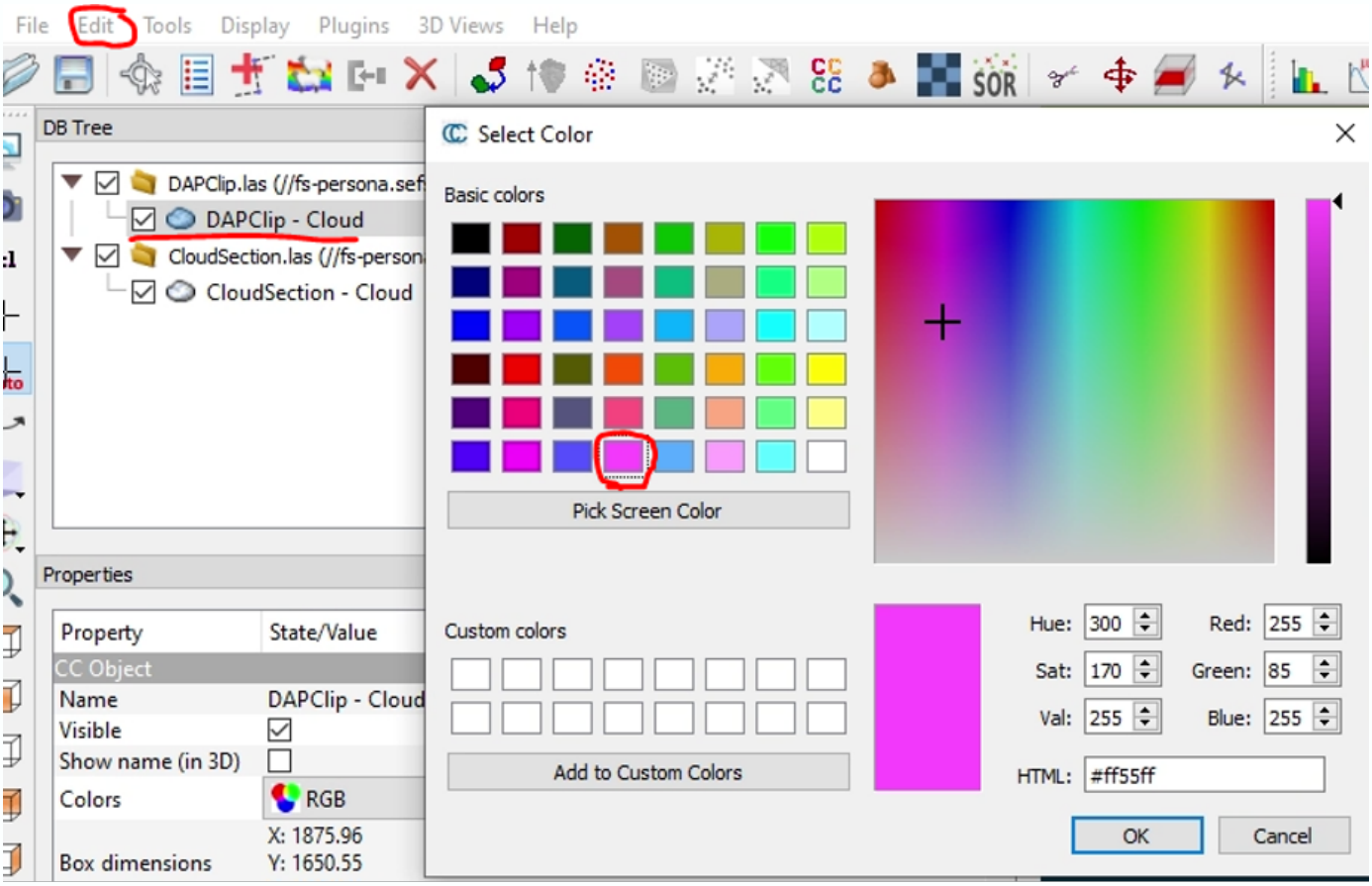
Ok so we have 2 different ways of creating 3D models for forest structure. But why would we choose one or the other?

Let's compare the DAP point cloud to the ALS point cloud. The ALS point cloud is `LAB5/CloudSection.las` The DAP point cloud is `LAB6/DAPClip.las`

In CloudCompare, I suggest using Object-centered perspective, and having the EDL shader turned on. Also, adjust point size to larger when zoomed into a cloud.

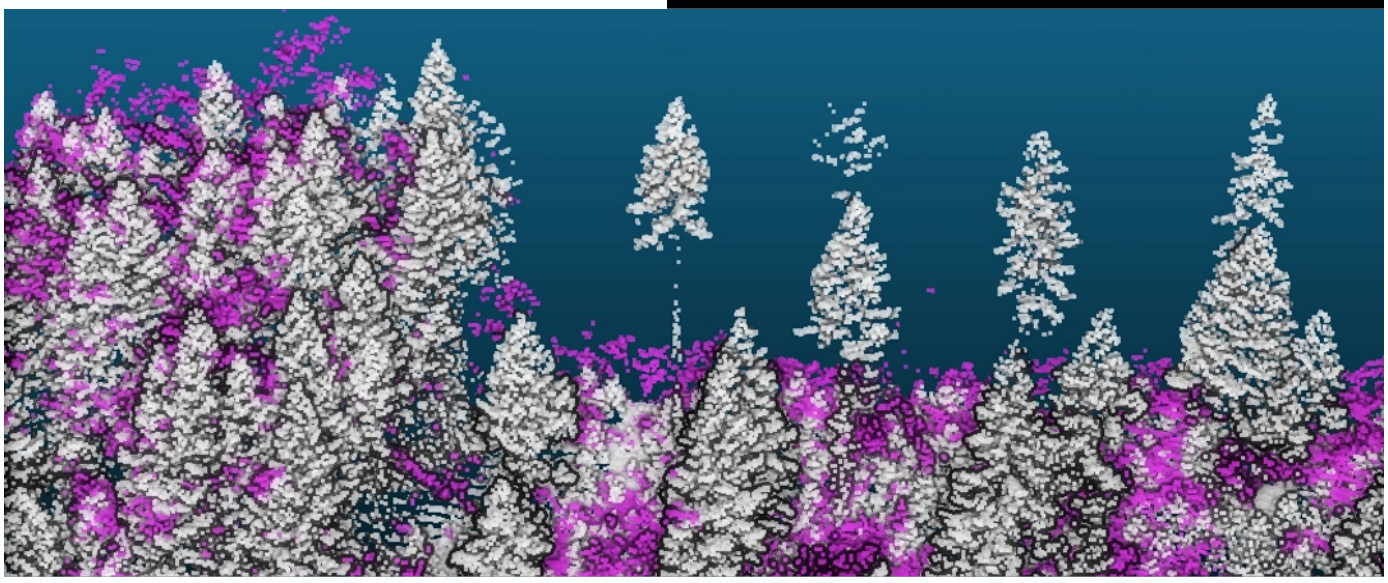


Drag both those files into the same instance of CloudCompare. To visually compare two point clouds, it is helpful to color them differently. Going to `Edit > Colors > Set Unique`



I am going to use pink for the DAP, and white for the ALS, but you can use whatever colors you want. The clip extent doesn't match exactly so points on the edge won't match, but internally the points should be very similar. Take a moment and look around the cloud. Where do points align well, and where are points missing from either layer missing?

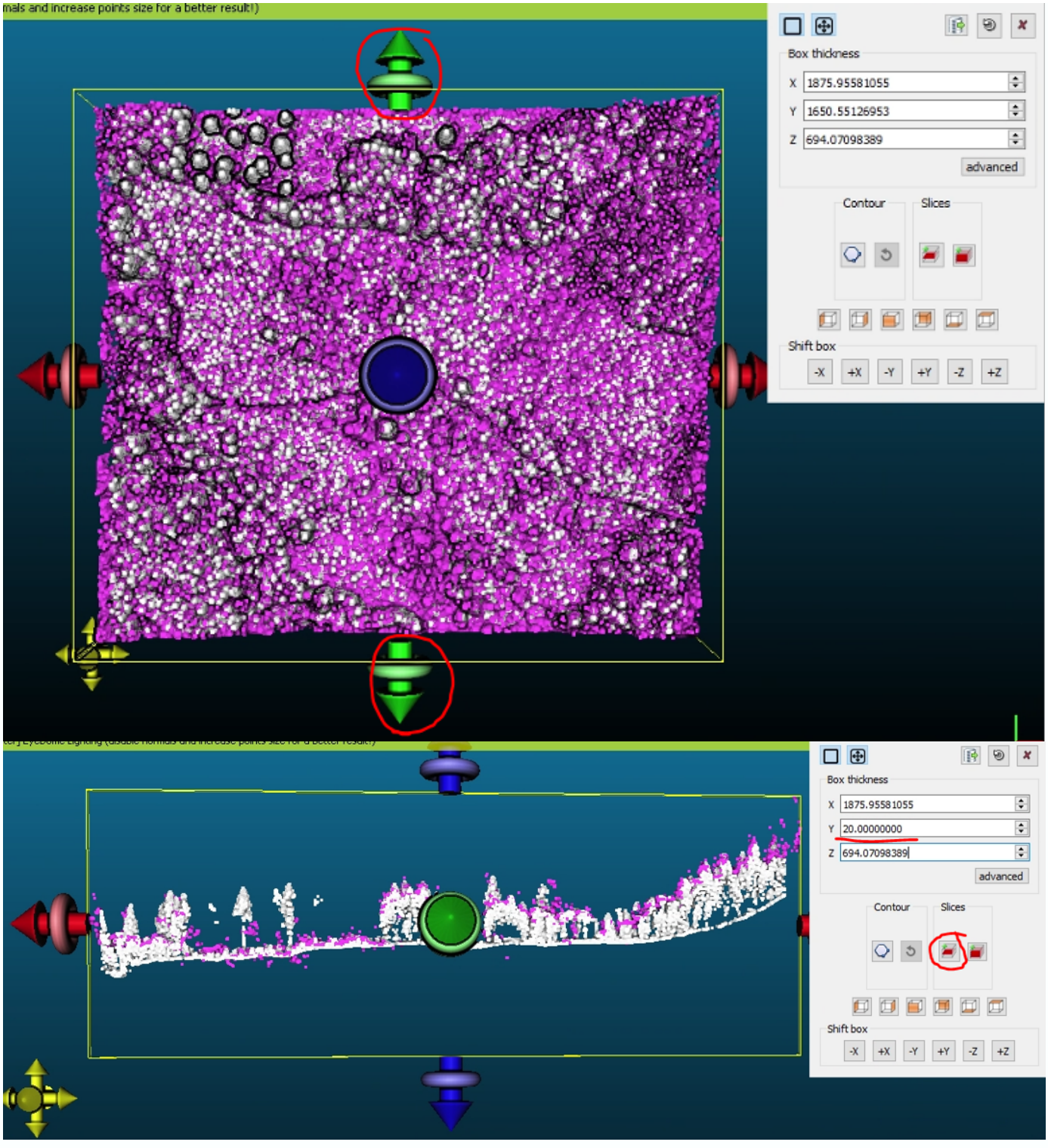
Look at the trees in the NW corner of the clip.



QUESTION 2: The white is lidar and the pink is DAP. The clustered trees on the left of the image above have some errant points above them, but generally are captured fairly well with both the lidar and DAP. The single trees that are standing on the right of the image seem to be missed entirely with the DAP point cloud. Why do you think this would occur?

Let's take a slice of the point clouds to get a profile. Center your view to overhead using the top view button.

Highlight both the ALS and DAP point clouds, and select the Cross Section tool.



QUESTION 3: Include a screen shot of your slice. Describe the differences between the ALS and DAP point clouds. Why does the DAP point cloud only model the tops of the trees and not penetrate all the way to the ground?

Back to RStudio and `lidR`!

Let's get into quantitatively analyzing the differences between Lidar and DAP

This block of code below installs a bunch of libraries we need for this lab. Some are optional

```
install.packages("lidR") #downloads and installs lidR and dependent packages
install.packages("rgl") #downloads and installs graphis package
install.packages("gstat") #optional, only needed for lasground
install.packages("sf") #needed for readLAScatalog. DO NOT Sellect "yes" for binary
install.packages("mapview") #needed to view tiles with world imagry
install.packages("BiocManager")#needed to install EBImage
BiocManager::install("EBImage") #Installs EBImage
install.packages("rgdal") #required for mapping, should already be installed.
install.packages("concaveman")#optional, needed to make detailed, non-overlaping hull
s
```

Now let's load our libraries for this lab. Again, some are optional

```
library(lidR) #loads the lidR package into RStuio
library(rgl) #loads package into RStudio
library(gstat) #optional
library(sf) #needed for readLAScatalog
library(mapview) #needed for ploting files on basemaps
library(BiocManager) #loading BiocManager
library(EBImage)#loading EBImage
library(rgdal) #run if you get an error "rgdal required"
library(concaveman) #optional
library(leaflet)
```

Make sure you set your working directory using `setwd`

```
setwd("/Users/Anthony/OneDrive - UW/University of Washington/Teaching/SEFS433_Lidar/L
abs/")
```

Assuming that you have your `CloudSection.las` in a LAB5 folder and `DAPClip.las` in LAB6 along with `VendorDTM_Clip.tif` . For this exercise, you won't be creating a tin from the point cloud to normalize the cloud, but rather you will be using the vendor created DTM to help save some time. Load the ALS, DAP, and DTM into R, and check them out if you want.

```
ALS <- readLAS("Lab 5/LAB5Data/CloudSection.las")
DAP <- readLAS("Lab 6/LAB6Data/DAPClip.las")
```

```
## Warning: Invalid data: 100729 points with a 'return number' greater than the
## 'number of returns'.
```

```
DTM <- rast("Lab 6/LAB6Data/VendorDTM_Clip.tif")

#plot(ALS)
```

```
#plot(DAP)
```

Next we are going to normalize the points clouds using a familar command:

```
?normalize_height
```

`ALSN` is going to be our normalized aerial lidar point cloud and `DAPN` is going to be our normalized photogrammetry point cloud

```
ALSN <- normalize_height(ALS,DTM) #using the vendor clip DTM instead of creating a ti
n()
DAPN <- normalize_height(DAP,DTM) #normalize the DAP point cloud using vendor DTM


writeLAS(ALSN, file="LAB 6/LAB6Data/ALSN.las") #writes out the normalized cloud
writeLAS(DAPN, file="LAB 6/LAB6Data/DAPN.las") #take them into CloudCompare if you wa
nt
```

Not required for the lab, but I would encourage you to take your normalized clouds into CloudCompare to literally compare the clouds. Because we normalized the ALS cloud, when we clip the data, we won't have to normalize each clip as we did in the previous lab.

Create the clips for the plots, stick them all into one file and then look at it:

This should look familiar!

```
#create 5 circle clips with the normalized ALS cloud
P1 <- clip_circle(ALSN,1636,-2636,60)
P2 <- clip_circle(ALSN,1430,-2230,60)
P3 <- clip_circle(ALSN,1216,-2425,60)
P4 <- clip_circle(ALSN,1279,-2725,60)
P5 <- clip_circle(ALSN,1139,-2174,60)
ALSplots <- rbind(P1,P2,P3,P4,P5) #combine the las clips
#plot(ALSplots)
```

Let's go ahead and create a rumple index values for our plots, the first step for each plot is to create a canopy height model:

```
chm1 <- rasterize_canopy(P1, 1.64, p2r(3.28)) #creates the canopy surface model using
p2R
chm1R <- rumple_index(chm1)             #creates the rumple index value for the clip
chm2 <- rasterize_canopy(P2, 1.64, p2r(3.28))
chm2R <- rumple_index(chm2)
chm3 <- rasterize_canopy(P3, 1.64, p2r(3.28))
chm3R <- rumple_index(chm3)
chm4 <- rasterize_canopy(P4, 1.64, p2r(3.28))
chm4R <- rumple_index(chm4)
chm4 <- rasterize_canopy(P4, 1.64, p2r(3.28))
chm4R <- rumple_index(chm4)
chm5 <- rasterize_canopy(P5, 1.64, p2r(3.28))
chm5R <- rumple_index(chm5)
ALSRumple <- rbind(chm1R,chm2R,chm3R,chm4R,chm5R) #combines rumple values
#plot (ALSRumple, xlab = "CHM ALS Point Clouds", ylab = "Rumple Index")
```

Remember: - rasterize canopy is creating a raster of our point cloud from the previously defined plots - `p2r()` is a point to raster algorithm in the lidR package that creates a digital surface model based on a setting the resolution of the cells and attributing those cells with Z values of the points. check out `?p2r` for more info.

Looking at the output of the `rumple_index` we can see that it is a single value as the function has calculated a rumple index for the entire plot

QUESTION 4: Using the ALS data, What plot had the highest rumple value? What plot had the lowest rumple value? Submit a screen shot of the las cloud for both of the plots. These are the P1, P2, P3 etc… and you can use plot()

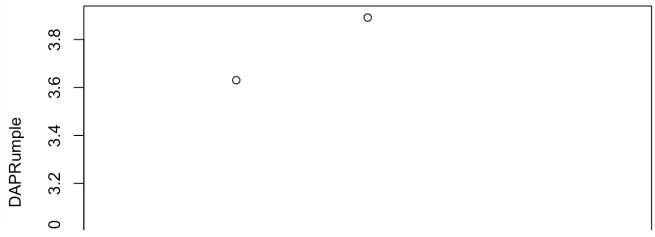
Now let's do the same thing but for the DAP data:

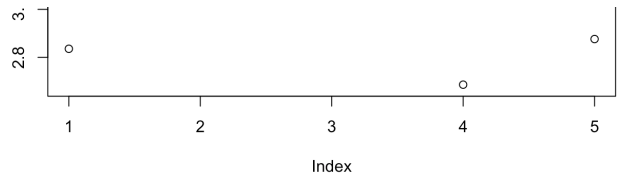
```
#create 5 circle clips with the normalized DAP cloud
D1 <- clip_circle(DAPN,1636,-2636,60)
D2 <- clip_circle(DAPN,1430,-2230,60)
D3<- clip_circle(DAPN,1216,-2425,60)
D4 <- clip_circle(DAPN,1279,-2725,60)
D5 <- clip_circle(DAPN,1139,-2174,60)
DAP_PLOTS <- rbind(D1,D2,D3,D4,D5)
```

```
## Warning: Invalid data: 1826 points with a 'return number' greater than the
## 'number of returns'.
```

```
plot(DAP_PLOTS)
```

```
# Creating rumple indexs for each of the plots
# ?rumple_index #check this out again if needed
chmD1 <- rasterize_canopy(D1, 1.64, p2r(3.28))
chmD1R <- rumple_index(chmD1)
chmD2 <- rasterize_canopy(D2, 1.64, p2r(3.28))
chmD2R <- rumple_index(chmD2)
chmD3 <- rasterize_canopy(D3, 1.64, p2r(3.28))
chmD3R <- rumple_index(chmD3)
chmD4 <- rasterize_canopy(D4, 1.64, p2r(3.28))
chmD4R <- rumple_index(chmD4)
chmD5 <- rasterize_canopy(D5, 1.64, p2r(3.28))
chmD5R <- rumple_index(chmD5)
DAPRumple <- rbind(chmD1R,chmD2R,chmD3R,chmD4R,chmD5R)
plot(DAPRumple)
```





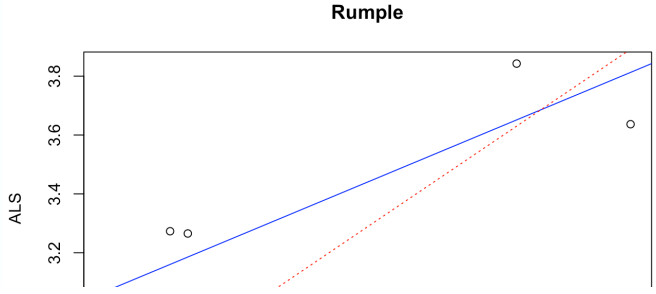
QUESTION 5: Using the DAP data, What plot had the highest rumple value? What plot had the lowest rumple value? Are they the same as using the ALS data? If not, why do you think they were different? Submit a screen shot of the las cloud for both the high and low rumple value plots similar to QUESTION 4 except using the D1, D2, D3 etc… with plot().

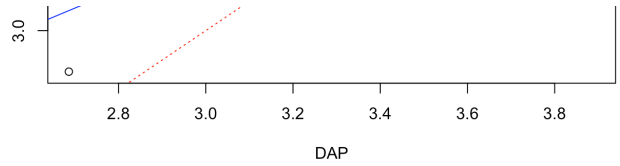
One quick analysis we can do is to check the correlation between ALS and DAP Rumple's and see if there is a relationship or if they don't match. We would expect that they should since they are attempting to measure the same 3D structure.

```
reg <- lm (ALSRumple ~ DAPRumple) # We are using the ALSRumple as the Y/Dependent Var
iable and DAPRumple as the X/Independent Variable
summary(reg)
```

```
##
## Call:
## lm(formula = ALSRumple ~ DAPRumple)
##
## Residuals:
##    chm1R    chm2R    chm3R    chm4R    chm5R
##   0.1126   0.1911 -0.1763 -0.2074   0.0799
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.4078     0.6207   2.268   0.1081
## DAPRumple     0.6181     0.1927   3.207   0.0491 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2079 on 3 degrees of freedom
## Multiple R-squared:  0.7742, Adjusted R-squared:  0.6989
## F-statistic: 10.28 on 1 and 3 DF,  p-value: 0.04908
```

```
plot(DAPRumple, ALSRumple, xlab="DAP", ylab="ALS", main="Rumple")
abline(reg, col="blue") #regression line
abline(0,1, col="red", lty = 3) #1 to 1 line
```





The blue line is the regression line , and the red is a 1 to 1 line. If a perfect match, the red and blue would be the same.

QUESTION 6: What is your p-value and R-squared for ALSRumple compared to DAPRumple? Is there a significant relationship?

PART 2: Comparing DAP to ALS, Grid Metrics

So far we have only created cloud metrics for plots. *A single value for an entire plot*. It is possible to create a grid across a lidar tile and generate metrics **for every grid cell**. We are going to do that, with the intention of further comparing DAP to ALS.

To make the comparison easier, we want to make sure our two tiles are the exact same extent:

```
print(DAPN) #check out the extent and basic las information
```

```
## class        : LAS (v1.2 format 2)
## memory       : 73.2 Mb
## extent       : 646.695, 2522.651, -3317.954, -1667.403 (xmin, xmax, ymin, ymax)
## coord. ref.  : NA
## area         : 3.1 kunits²
## points       : 1.37 million points
## density      : 0.44 points/units²
## density      : 0.41 pulses/units²
```

```
print(ALSN) #Extent must match
```

```
## class        : LAS (v1.2 format 1)
## memory       : 477.2 Mb
## extent       : 681.32, 2481.66, -3300.66, -1685.71 (xmin, xmax, ymin, ymax)
## coord. ref.  : NA
## area         : 2.92 kunits²
## points       : 7.36 million points
## density      : 2.52 points/units²
## density      : 1.13 pulses/units²
```

The print command gets you the basic information about the las file. You should be familiar with this command. Unfortunately, the extents don't match exactly. An easy way to get the extents to match that will result in output rasters with aligned cells is to simply clip the las files, just like you did for the plots, but this time use a rectangle.

```
#?clip_rectangle
#Clipping the ALS and DAP data
ALSNc <- clip_rectangle(ALSN, 700, -3300, 2400, -1700)
#for a direction comparison, the extent must match
DAPNc <- clip_rectangle(DAPN, 700, -3300, 2400, -1700)

print(ALSNc)
```

```
## class        : LAS (v1.2 format 1)
## memory       : 453.5 Mb
## extent       : 700, 2399.99, -3300, -1700.01 (xmin, xmax, ymin, ymax)
## coord. ref.  : NA
## area         : 2.72 kunits²
## points       : 6.99 million points
## density      : 2.57 points/units²
## density      : 1.15 pulses/units²
```

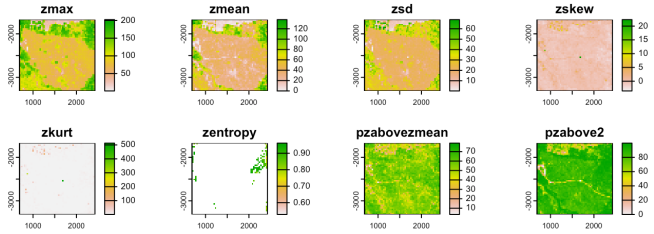
```
print(DAPNc)
```

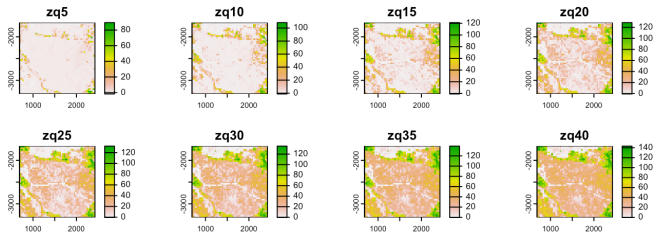
```
## class        : LAS (v1.2 format 2)
## memory       : 60.9 Mb
## extent       : 700.003, 2399.999, -3300, -1700.001 (xmin, xmax, ymin, ymax)
## coord. ref.  : NA
## area         : 2.72 kunits²
## points       : 1.23 million points
## density      : 0.45 points/units²
## density      : 0.42 pulses/units²
```

Once our extents match, we can use grid_metrics to create our `.stdmetrics` for our Z values. If we ran `pixel_metrics` on `DAPNc` and `ALSNc`, we would get one value for all metrics for the entire extent. Exactly what we got when running `pixel_metrics` on our plots.

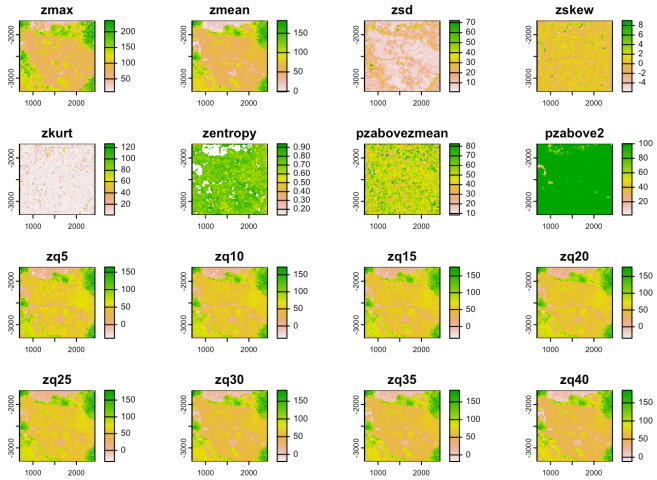
`pixel_metrics` lets us input a grid size (res) and the output will be a raster with raster cells of our defined resolution. One raster for each metric and they get all pilled together in a raster brick in R.

```
?cloud_metrics
#creates standard elevation metrics for all Z values
ALSrasters <- pixel_metrics(ALSNc, ~stdmetrics_z(Z), res = 32.8)
#output is a raster brick, output won't include intensity metrics
DAPrasters <- pixel_metrics(DAPNc, ~stdmetrics_z(Z), res = 32.8)
plot(ALSrasters) #checking out all the rasters in the brick
```






```
plot(DAPrasters) #not all rasters will be shown
```




```
names(ALSrasters) #names of all rasters in brick
```

```
##  [1] "zmax"         "zmean"        "zsd"          "zskew"        "zkurt"
##  [6] "zentropy"     "pzabovezmean" "pzabove2"     "zq5"          "zq10"
## [11] "zq15"         "zq20"         "zq25"         "zq30"         "zq35"
## [16] "zq40"         "zq45"         "zq50"         "zq55"         "zq60"
## [21] "zq65"         "zq70"         "zq75"         "zq80"         "zq85"
## [26] "zq90"         "zq95"         "zpcum1"       "zpcum2"       "zpcum3"
## [31] "zpcum4"       "zpcum5"       "zpcum6"       "zpcum7"       "zpcum8"
## [36] "zpcum9"
```

```
names(DAPrasters) #rasters are the cloud_metrics!
```

```
##  [1] "zmax"         "zmean"        "zsd"          "zskew"        "zkurt"
##  [6] "zentropy"     "pzabovezmean" "pzabove2"     "zq5"          "zq10"
## [11] "zq15"         "zq20"         "zq25"         "zq30"         "zq35"
## [16] "zq40"         "zq45"         "zq50"         "zq55"         "zq60"
## [21] "zq65"         "zq70"         "zq75"         "zq80"         "zq85"
## [26] "zq90"         "zq95"         "zpcum1"       "zpcum2"       "zpcum3"
## [31] "zpcum4"       "zpcum5"       "zpcum6"       "zpcum7"       "zpcum8"
## [36] "zpcum9"
```

The lines should all look familiar to you. The additional `res = 32.8` tells R that we want output rasters with a resolution of 32.8 units. Our units in these clouds is feet, so the output will be 10m squares.

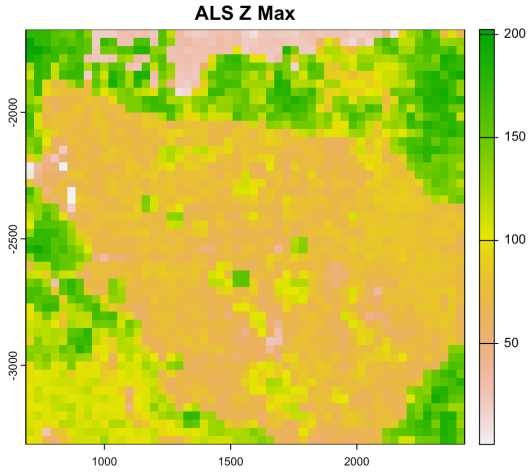
The `plot` function will show you a partial list of the rasters in the raster stack. Note that not all are showing.

You can use `names` to get the names of all the rasters in the raster stack. The names should look familiar to you. They are the `cloud_metrics` ! If you were processing a larger amount of lidar data, you can be more selective about what metrics you generate.

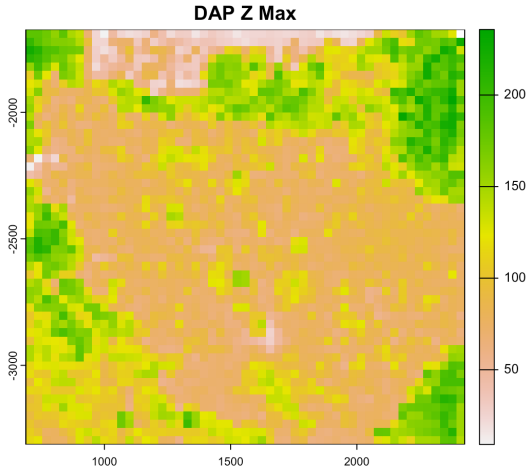
You can select which raster you want from the raster stack with `$` such as `ALSrasters$zmax` . pressing tab after `$` can show you the available layers as well

Let's look at our output for `zmax` , and compare our two rasters with a scatter plot:

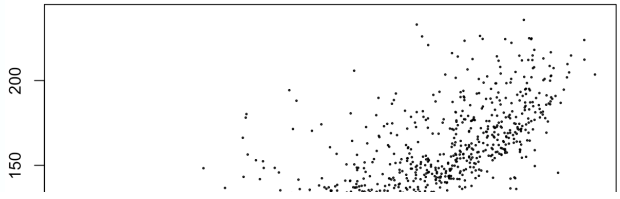
```
par(mar = c(4, 4, .1, .1))
plot(ALSrasters$zmax, main="ALS Z Max") #cell size is 10m
```

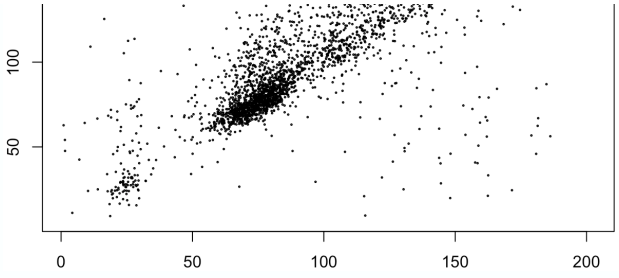


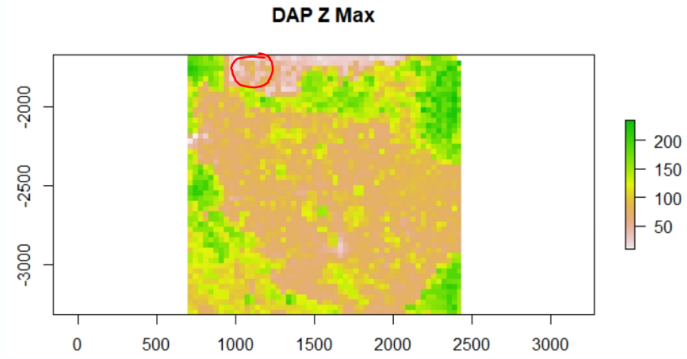

```
plot(DAPrasters$zmax, main="DAP Z Max") #the maxium Z value within each cell
```




```
#scatter plot of matched cell values. Won't work if extents differ
plot(ALSrasters$zmax, DAPrasters$zmax, xlab="DAP Zmax",ylab="ALS Zmax")
```







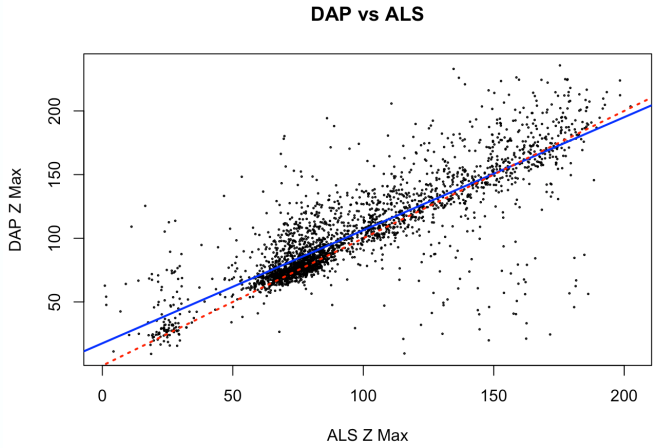
You can really see the omission of the tall, solo trees that we looked at in the beginning of the lab…

The maps between DAP and ALS look close, with a few differences, but how close are they actually? To quantify the answer to that question, we can preform a linear regression on the matched cells of the rasters:

```
#we can do a basic linear regression to see if our cell values "match"
L1 <- ALSrasters$zmax #identifies L1 as being the ALS zmax raster
L2 <- DAPrasters$zmax
s <- c(L1, L2) #creates a raster stack for the regression
v <- data.frame(na.omit(values(s))) #turns rasters into a dataframe
names(v) <- c('L1', 'L2') #arranges the data for the regression
m <- lm(L2 ~ L1, data=v) #regression comparing matched raster cells
summary (m)
```

```
##
## Call:
## lm(formula = L2 ~ L1, data = v)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -145.429   -9.359   -3.361    7.976  100.740
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.53321    1.18094   14.85   <2e-16 ***
## L1           0.88783    0.01142   77.76   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.03 on 2647 degrees of freedom
## Multiple R-squared:  0.6955, Adjusted R-squared:  0.6954
## F-statistic:  6047 on 1 and 2647 DF,  p-value: < 2.2e-16
```

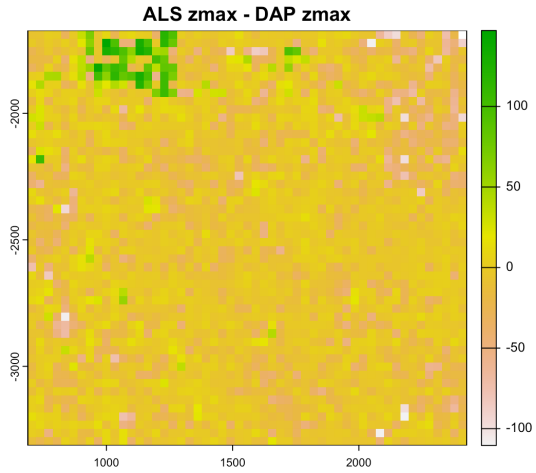
```
plot(L1, L2, xlab="ALS Z Max", ylab="DAP Z Max", main="DAP vs ALS")
abline(m, col="blue", lwd = 2)
abline(0,1, col="red", lty = 3, lwd = 2) #1 to 1 line
```



QUESTION 7: Pick a metric other than zmax from your raster outputs. All the options can be seen using names(ALSrasters). Include screenshots of the DAP and ALS versions of your metric rasters (like the ones above). Preform a linear regression on the rasters and report your findings. Include p-value, r-squared value, and a short discussion. Make sure to include a properly labeled figure of the scatter plot with the regression line and the 1 to 1 line added.

We can lastly create a raster of the difference between our ALS and DAP metric rasters. Check out the biggest difference where those solo standing tall trees were…

```
ZDiff <- (ALSrasters$zmax - DAPrasters$zmax)
plot(ZDiff, main="ALS zmax - DAP zmax")
```



QUESTION 8: Create a difference raster of a metric other than zmax. It can be the same metric you used for the regression, or a different one. Label and caption your figure.

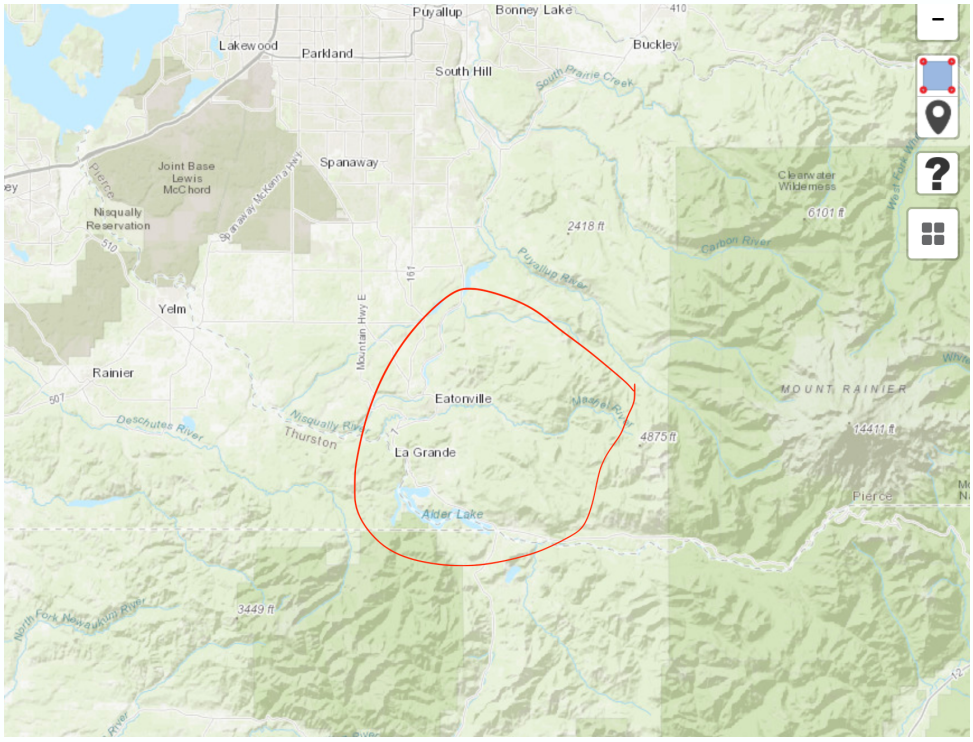
PART 3: Metric Rasters from `lascatalog`

We are only going to process two small lidar point cloud tiles from Pack forest.

- We are going to produce metric rasters from the tiles using our own defined functions
- We will use plot data with both field and lidar observations to build a linear regression that models the **Basal Area** in ft^2/ha from lidar predictor variables.
- We will use the linear regression model to predict Basal Area across the entire study area of the two tiles
- Finally we will determine which areas within the two tiles have both a **canopy closure > 70% and a Basal Area > 160**, these are believed to be habitat preferences for the **Spotted Owl**. As a note, we aren't doing full habitat modeling, just identifying forest that likely meets those two elements.

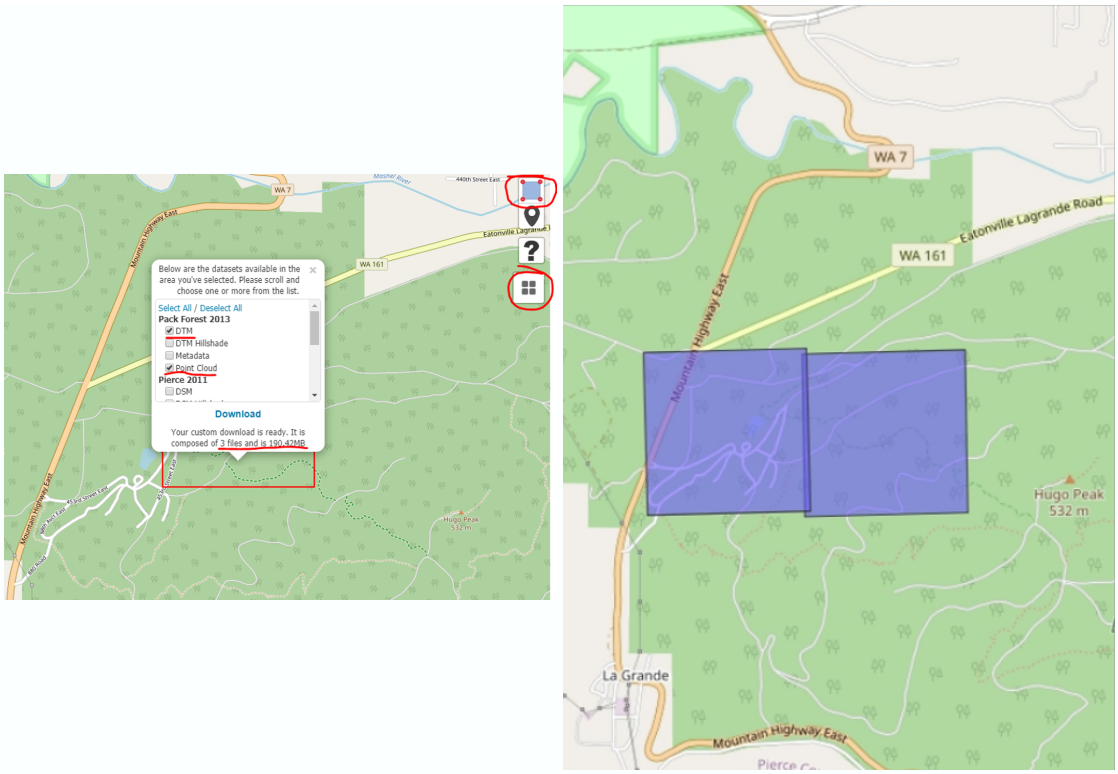
> Thome, Darrin M., Cynthia J. Zabel, and Lowell V. Diller. "Forest Stand Characteristics and Reproduction of Northern Spotted Owls in Managed North-Coastal California Forests." The Journal of Wildlife Management 63, no. 1 (1999): 44–59. https://doi.org/10.2307/3802486 (https://doi.org/10.2307/3802486).

Go to Washington DNR lidar portal and find Pack forest



Around here somewhere

I would suggest using the open street maps layer as Pack forest is clearly labeled. Use the area select tool to draw a rectangle around the approximate same area. You want the DTM and the Point cloud from Pack Forest 2013. "3 files and is 190.42MB" It should look like the two squares below



Once you download the data from the lidar portal, unzip the folder and move the dtm and laz folders directly into your LAB6 folder for easy access when we move to R.

Back to R

We are going to be using `readLAScatalog` . We've used this before in Lab 3 & 4 but now we are actually going to use some of the power in the catalog tool.

`readLAScatalog` reads the header (remember `las@header` command?) of all the LAS files of a given folder. The header of a LAS file contains, among others, the bounding box of the point cloud, meaning that it's possible to know where the file is situated spatially without reading a potentially massive amount of data. The variable class in R is called a `LAScatalog` class.

A `LAScatalog` class is a representation in R of a las file or a collection of las files not loaded in memory. A regular computer cannot load the entire point cloud in R if it covers a broad area. In `lidR` , we use a `LAScatalog` to process datasets that cannot fit in memory.

Import your data by pointing the `readLAScatalog` function to the file folder (NOT the file itself) holding your laz file then fix the `crs` issues in `lidR` (because it doesn't like ft):

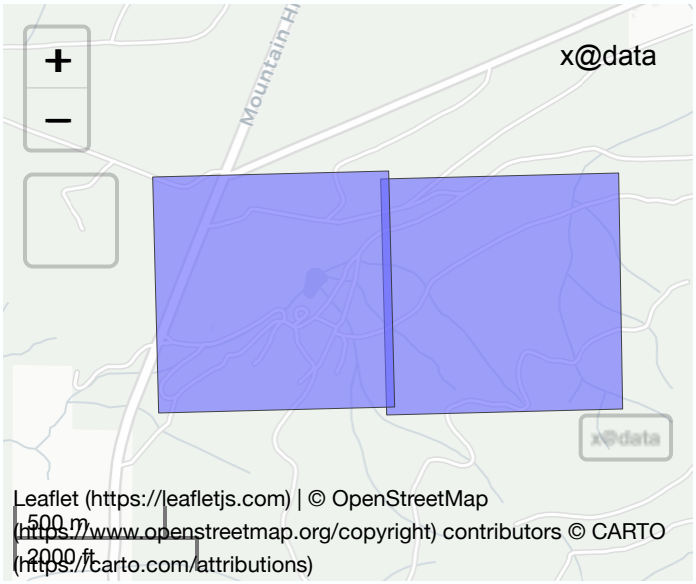
```
?readLAScatalog
#folder you put the 2 Pack forest laz files from DNR
ALS <- readLAScatalog("Lab 6/LAB6Data/pack_forest_2013/laz/")
```

```
## Be careful, some tiles seem to overlap each other. lidR may return incorrect outpu
ts with edge artifacts when processing this catalog.
```

```
#DTM you downloaded from DNR
DTM <- rast("Lab 6/LAB6Data/pack_forest_2013/dtm/pack_forest_2013_dtm_1.tif")
#we know lidR doesn't like feet...
NEWCRS <- "EPSG:2927"
crs(ALS) #checking crs
crs(ALS) <- NEWCRS #changing crs
crs(DTM)
crs(DTM)<- NEWCRS
```

Lets take a look at our DTM and laz files to make sure they are in the correct location now. Remember a `LAScatalog` variable will just show the squares of the tiles, not the point clouds.

```
plot(ALS, mapview = T)
```



Now lets look at the DTM. One confusing thing with `mapview` is that it doesn't like `SpatRasters` which is the new file format for rasters using the `terra` package. So you'll notice here that I've used the `raster` package to convert the DTM back to the old raster format using `raster::raster()`. Someday we'll have to use just `terra` and everything will be updated and work together, but today is not that day…

```
mapview(raster::raster(DTM), map.types = "Esri.WorldImagery", alpha = 0.7, layer.name
= "DTM (ft)")
```

```
## Warning in rasterCheckSize(x, maxpixels = maxpixels): maximum number of pixels for
Raster* viewing is 5e+05 ;
## the supplied Raster* has 42955200
##  ... decreasing Raster* resolution to 5e+05 pixels
##  to view full resolution set 'maxpixels =  42955200 '
```

We are now going to *normalize* both tiles with one set of commands, using the vendor supplied DTM. It is a little more complicated than what we did before.

Make sure to check out the documentation `?LAScatalog`.

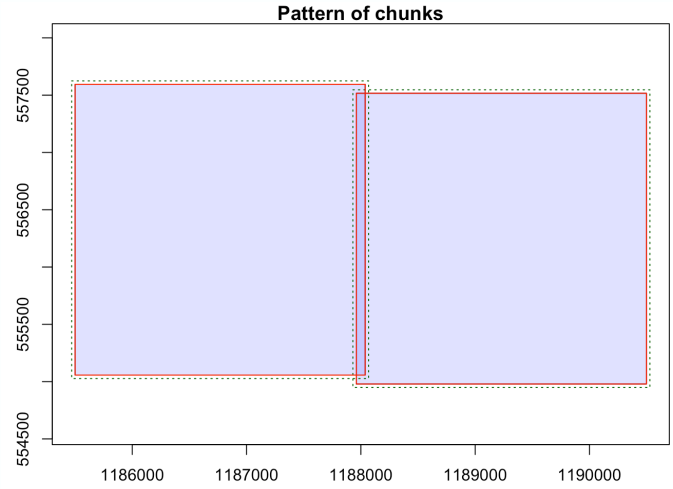
In lidR, each function that supports a LAScatalog as input will respect these processing options. Internally, processing a catalog is almost always the same and relies on a few steps:

1. Define chunks. A chunk is an arbitrarily-defined region of interest (ROI) of the collection. Altogether, the chunks are a wall-to-wall set of ROIs that encompass the whole dataset.
2. Loop over each chunk (in parallel or not).
3. For each chunk, load the points inside the ROI into R, run some R functions, return the expected output.
4. Merge the outputs of the different chunks once they are all processed to build a continuous (wall-to-wall) output.

So basically, a LAScatalog is an object that allows for batch processing. For us that means:

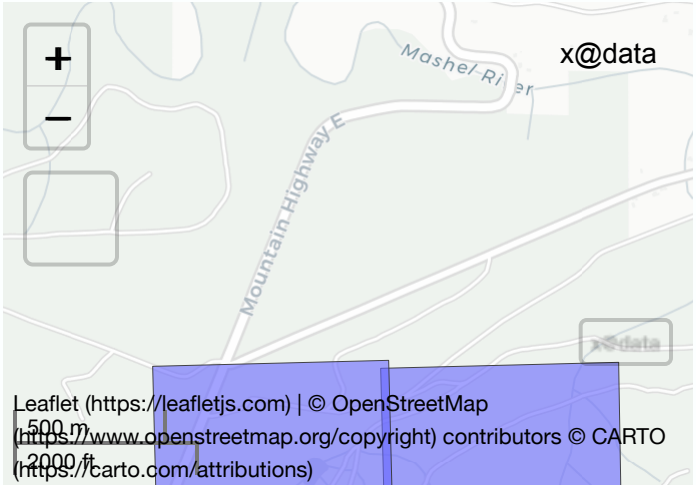
- We have to define `chunk size` and a `location` to write the normalized laz files as it processes them before we start using the functions. We do this by setting the `opt_chunk_size()` and `opt_output_files()`.
- Additionally, we set the `opt_laz_compression()` to make sure we output as a .laz compressed file
- Before we were only dealing with data that was small enough that it could be written out to R, and with only processing 2 tiles, R could handle it, but if you were to process all the tiles at once for Pack Forest, this is the same process you would use. **This is probably the longest running command we have done so far:**

```
?`LAScatalog-class`
?normalize_height #extra steps are needed for working with a catalog

#the chunk size. 0 means the entire tile. Since the tiles are small
opt_chunk_size(ALS) <- 0

#location for output files. Files need to be written out of R due to size
opt_output_files(ALS) <- "Lab 6/LAB6Data/{ORIGINALFILENAME}_Norm"
#There are differnet naming conventions {ORIGINALFILENAME} keeps original file name f
rom the tile

opt_laz_compression(ALS) <- TRUE #we want output as laz vs las

#This will take a long time... Imagine if you did this for more than 2 tiles...
#in your "plots" window, it updates you on what tiles are being worked on.
ALSN <- normalize_height(ALS, DTM) #Run normalize on the tiles using DTM you download
ed
```




```
## Chunk 1 of 2 (50%): state ✓
## Chunk 2 of 2 (100%): state ✓
```

```
plot(ALSN, mapview=TRUE)
```



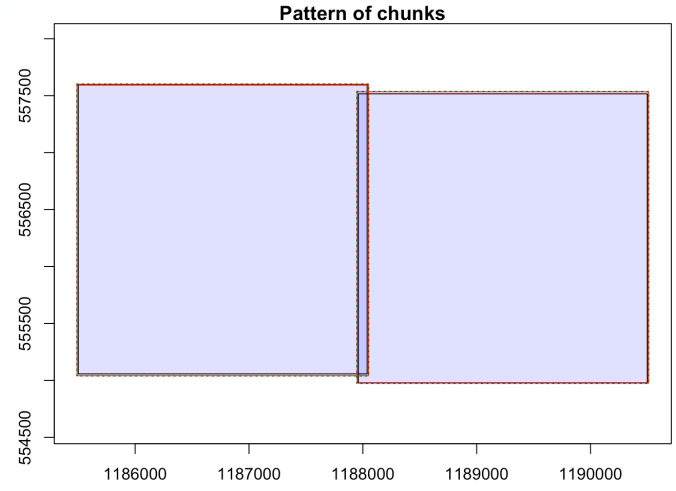
There are many more " `opt_` " parameters that can be set using las catalog. All the documentation is in the `lascatalog` help pages.

If you were dealing with really large tiles or a very weak computer, you can define chunk size to 500 or some other value. lidR will bite off chunks of that size to process.

A chunk of 500 would be a 500 ft x 500 ft section and there would be created an output file for each chunk. You only need to write files to a directory if the data is really big (e.g. point clouds). Rasters don't need to be written out to a external folder as R can handle the rasters (at least our two rasters). We can also filter out points we don't want to include in our processing.

Next, calculate the canopy cover for our two tiles. The expression `sum(Z>6.56)/sum(Z>=0)` should be familiar to you. `>=` is "greater than or equal to". We are filtering out points that have a z value below 0.

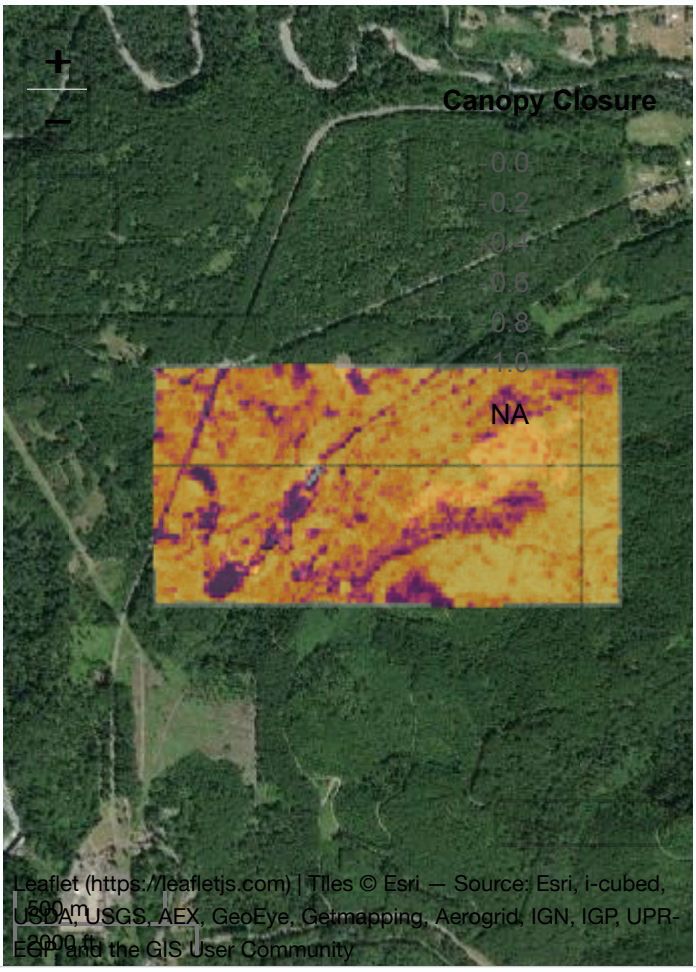
```
opt_chunk_size(ALSN) <- 0 #the chunk size. 0 means the entire tile.
#If output_files = "" outputs are returned in R
opt_output_files(ALSN) <- ""

#run rlas:::lasfilterusage() for list of filters
opt_filter(ALSN) <- " -drop_z_below 0"
PackCC <- pixel_metrics(ALSN, ~sum(Z>6.56)/sum(Z>=0), 32.8)
```




```
## Chunk 1 of 2 (50%): state ✓
## Chunk 2 of 2 (100%): state ✓
```

One thing to notice with our `mapview` plot is that some of the canopy closure doesn't match the satellite imagery. This is probably due to the difference in dates since we are using 2013 lidar and the `Esri.WorldImagery` is more recent

```
mapview(raster::raster(PackCC), alpha.regions=0.7, map.types = "Esri.WorldImagery", l
ayer.name = "Canopy Closure")
```



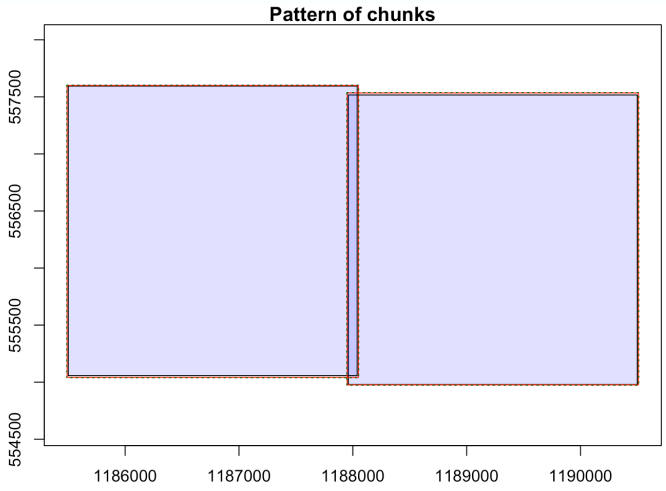

```
writeRaster(PackCC,filename = ("Lab 6/LAB6Data/PackCC.tif"), overwrite=TRUE)
```

QUESTION 9: use mapview() to create a map over ESRI.Worldimagry of your canopy closure metric raster using the raster::raster() conversion. You can include alpha.regions=0.5 to set the transparency of your raster. As always, use ?mapview for more information. Fully caption the figure with a very brief discussion as to how it was made. Think of it as a figure in a paper you are submitting. Don't want just snippets of R code, but rather a conceptual description of how it was made. Feel free to take the raster into ArcGIS to make a map there if you like.

We now can make metric rasters for all `stdmetrics`, or just a few that we are interested in. In the last lab, you estimated Basal Area `BA` from multiple linear regression. There was a significant relationship between `BA` and the percentage of all points above 6.56ft + average point height. **We can produce rasters for these metrics plus others.**

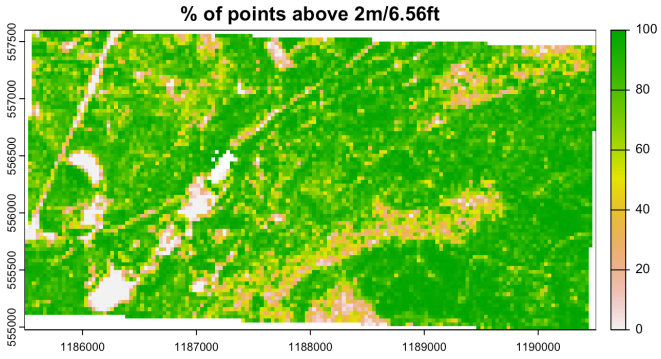
We can also make a custom function to get the percentage of points above 6.56ft and the mean z value. If you want, you can just run all the standard metrics with `~stdmetrics` instead of `~f(z)`:

```
#If output_files = "" outputs are returned in R
opt_output_files(ALSN) <- ""
#run rlas:::lasfilterusage() for list of filters
opt_filter(ALSN) <- " -drop_z_below 0"
#making our own function
f = function(x) {list(above2percent = sum(x>6.56)/length(x>=0)*100, zmean = mean(x))}

#Applying function
PackZ <- pixel_metrics(ALSN, ~f(Z), 32.8)
```

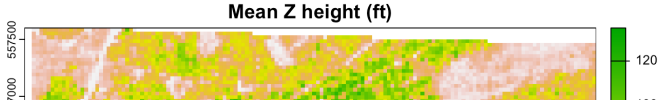


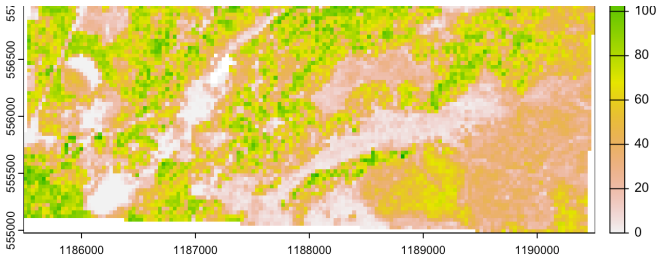

```
## Chunk 1 of 2 (50%): state ✓
## Chunk 2 of 2 (100%): state ✓
```

```
plot(PackZ$above2percent, main = "% of points above 2m/6.56ft")
```




```
plot(PackZ$zmean, main = "Mean Z height (ft)")
```





Part 4: Regression with Rasters and “Habitat”

In Part 3 I referred to the lab 5 regression analysis. We can use the coefficients from a linear regression to take cell values from input rasters, to create a new raster. Specifically, we can use the coefficients from the multiple linear regression to model `BA` from our `above2percent` and `zmean`.

We will use the more extensive plot data and cloud metrics from FUSION that you used in the first part of lab 5:

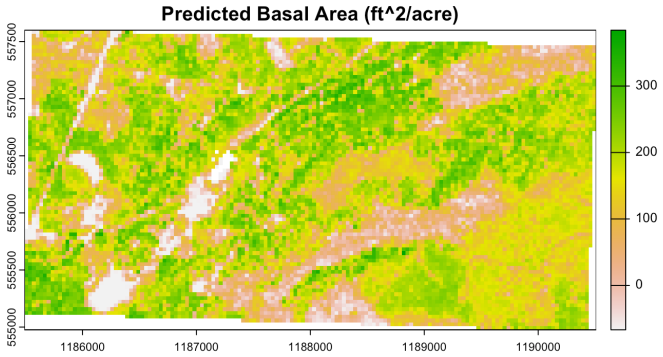
```
fieldplots <- read.csv("Lab 5/LAB5Data/plotdata.csv")
lidarplots <- read.csv("Lab 5/LAB5Data/cloudmetrics.csv")

BA_field <- fieldplots$BA
zmean_lidar <- (lidarplots$Elev.mean)*3.28 # I suspect these lidar plot heights are i
n meters so I'm converting them to feet to match the field data
above2percent_lidar <- lidarplots$Percentage.all.returns.above.6.56

#build a linear model that predicts Basal Area
BA_mod <- lm(BA_field ~ above2percent_lidar + zmean_lidar)
summary(BA_mod)
```

```
##
## Call:
## lm(formula = BA_field ~ above2percent_lidar + zmean_lidar)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -88.172 -32.718  -8.936  20.975 228.016
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -66.7765    17.1199  -3.901 0.000206 ***
## above2percent_lidar   1.5584     0.3363   4.634 1.46e-05 ***
## zmean_lidar           2.1887     0.3258   6.717 2.98e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 53.89 on 76 degrees of freedom
## Multiple R-squared:  0.6945, Adjusted R-squared:  0.6865
## F-statistic:  86.4 on 2 and 76 DF,  p-value: < 2.2e-16
```

We can use the `predict` function to generate a Basal Area `BA` map from our linear regression. This basically takes the `above2percent` raster and the `zmean` raster and combines them into a new raster using the linear modeling coefficients. This is the basis for generating prediction maps!

First we need to stack our rasters together and make sure they have the same names as the variables in the model. Conveniently our `PackZ` is already a raster stack, so we just need to change the names

```
#
# #?terra::predict
#
names(PackZ) <- c("above2percent_lidar", "zmean_lidar")


BA_predict <- predict(PackZ, BA_mod)
plot(BA_predict, main = "Predicted Basal Area (ft^2/acre)")
```



Again, we don't know how accurate our BA model is without going out and ground truthing sections of the area.

Also, the regression we used was based on plot data from a different area so the relationships may be wrong. I suspect we are over estimating BA with our model. Who wants to head out to Pack forest to measure some trees?

The main takeaway here is that we can generate lidar metrics, model their relationships to forest structure variables, then map those variables across our entire lidar point cloud area

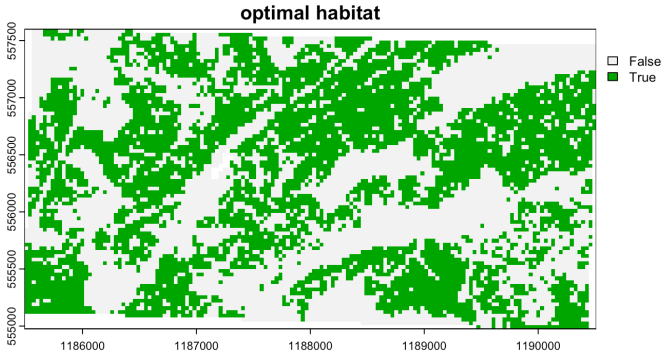
QUESTION 10: use mapview() to create a map over Esri.WorldImagry of your BA metric raster using the raster::raster() conversion. You can include alpha.regions=0.5 to set the transparency of your raster. As always, use ?mapview for more information. Fully caption the figure with a very brief discussion as to how it was made without referring to the R code, but rather a conceptual description of how it was made. Also does the map match the imagery? Why or why not? Feel free to take the raster into ArcGIS to make a map there if you like.

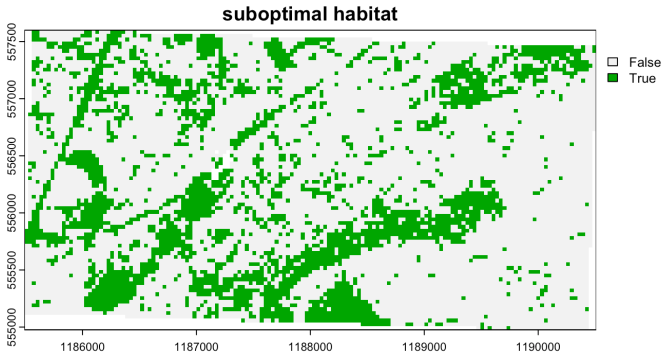
LAST STEP!

OK, you were told that Spotted Owls like canopy closure > 70% and BA > 160 ft^2/acre. You are now able to identify areas within our lidar data that meet those two criteria. We are going to use a very simple "yes/no" determination.

We can take our rasters `BA_predict` and `PackCC` and create a new raster with values of either 0 or 1 to denote if that location has a BA > 160 ft^2/acre & Canopy Closure > 70% Run:

```
BA_predict_optimal <- (BA_predict > 160 & PackCC > 0.7)
BA_predict_suboptimal <- (BA_predict < 160 & PackCC < 0.7)


plot(BA_predict_optimal, main = "optimal habitat")
plot(BA_predict_suboptimal, main = "suboptimal habitat")
```





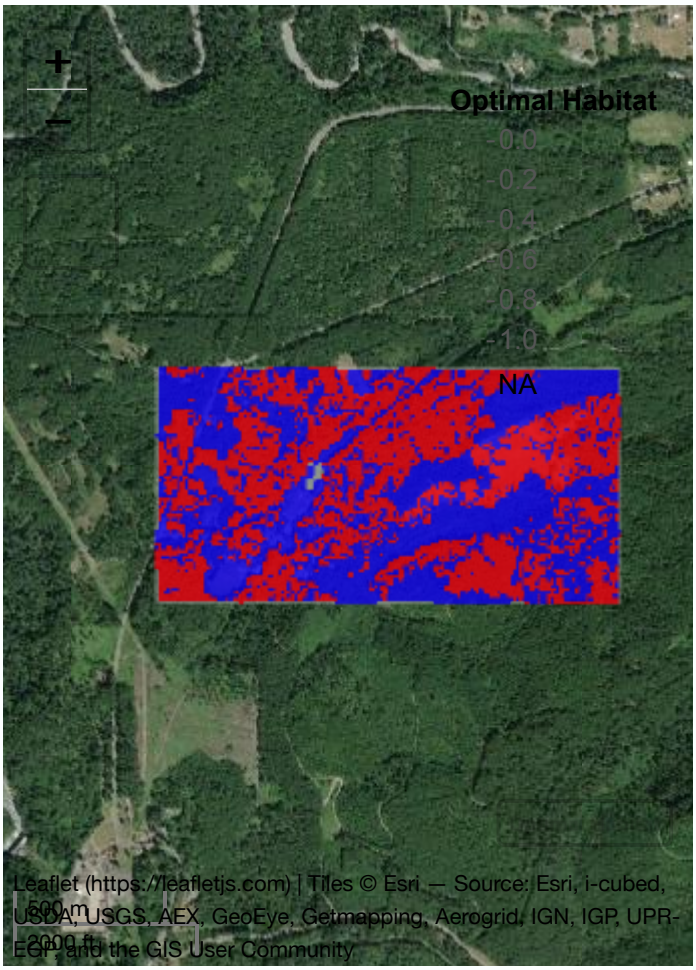
Let's stack everything together into one raster stack and rename the layers. After this we can plot the data using `mapview` a little easier.

Also we can write out our Raster and save it for later

```
#stack the optimal and suboptimal layers with the BA_predict
opt_and_sub <- c(BA_predict, BA_predict_optimal, BA_predict_suboptimal)
names(opt_and_sub) <- c("BA_predict", "BA_predict_optimal", "BA_predict_suboptimal")
#write the raster to a file
writeRaster(opt_and_sub,filename =("Lab 6/LAB6Data/SpottedOwlHabitat.tif"), overwrite
=TRUE)
```

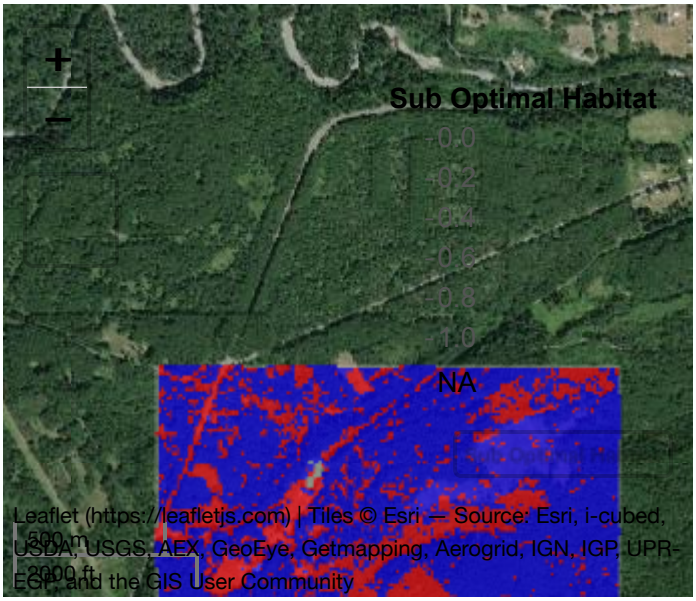
Now let's look at the optimal habitat

```
mapview(raster::raster(opt_and_sub$BA_predict_optimal), map.types = "Esri.WorldImager
y", col.regions = c("blue", "red"), method = "ngb", layer.name = "Optimal Habitat")
```



And the suboptimal habitat

```
mapview(raster::raster(opt_and_sub$BA_predict_suboptimal), map.types = "Esri.WorldIma
gery", alpha = 0.7, col.regions = c("blue", "red"), layer.name = "Sub Optimal Habita
t", method = "ngb")
```



QUESTION 11: While you use mapview() to create maps over Esri.WorldImagry of the optimal and suboptimal habitat rasters think about what the habitat looks like. Does this match what you would describe for optimal and suboptimal habitat?

Go back to where you set the parameters for optimal and suboptimal habitat:

```
BA_predict_optimal <- (BA_predict > 160 & PackCC > 0.7)
BA_predict_suboptimal <- (BA_predict < 160 & PackCC < 0.7)
```

Change these parameters and remake the optimal and suboptimal maps and take screenshots. Do you think changing these parameters makes better representations of the habitat? Why or why not? Also, what else could we do with lidar to improve the habitat model such as more metrics, newer data, or get more field data?