

ESRM433/SEFS533

Lab 4

Objectives:

- Create Canopy Height Models (CHMs) & Digital Surface Models (DSMs)
- Individual Tree Detection (ITD)

Data and Software:

- Las tiles downloaded previously for lab 3.
- Washington DNR Lidar Portal to download las tile and meta data for Arboretum
- RStudio
- ArcGIS Pro (Optional)

What you will turn in:

- Submission Template in PDF or DOCX file format submitted via Canvas.
-

Welcome to Lab 4 for ESRM433/SEFS533!

In this lab you will create and assess canopy height models (CHMs), then use these models in an individual tree detection (ITD) routine. CHMs are a primary product of LiDAR for use in forest measurement. It is very important that you understand how they are made and how various parameters affect their characteristics. ITD is a more niche field of study; however, it is good to understand how it works and what its limitations are.

Some definitions (<https://www.neonscience.org/chm-dsm-dtm-gridded-lidar-data>):

- [Digital Terrain Model](#) - This product represents the elevation of the ground.
- [Digital Surface Model](#) - This represents the elevation of the tallest surfaces at that point. Imagine draping a sheet over the canopy of a forest, the DEM contours with the heights of the trees where there are trees but the elevation of the ground when there is a clearing in the forest.
- [Canopy Height Model](#) - This represents the difference between a Digital Terrain Model and a Digital Surface Model and gives you the height of the objects (in a forest, the trees) that are on the surface of the earth.

PART 1 Mapping tiles and clipping data



Hopefully, you still have the las tiles downloaded from lab 3. To continue, you will need to have the two tiles located in a folder LAB3/TILES. There is no need to move them to a new lab 4 folder.

Start RStudio. I recommend just loading the R script from lab 3 and then resaving it as LAB4.R. There are several lines of code that will be the same.

Set your working directory (setwd) and you will need to have the packages lidR, rgl, sf, & mapview downloaded (install.packages) and running (library) in your RStudio instance. You don't need the gstat, but I am keeping it in my list of packages at the top of my script just so I know that it is needed to run some line of script.

ESRM433/SEFS533

Lab 4

Assuming that you still have the las tiles  `q47123g8201.laz`  `q47123g8206.laz` located in LAB3/TILES, run the familiar command:

```
ctg <- readLAScatalog("LAB3/TILES/") # looks at the folder TILES and loads all las files
```

Lets use mapview to plot where our tiles are in the world. Run:

```
?mapview  
plot(ctg, mapview = TRUE) #places tiles onto satalite imagery  
#This is awesome to check to make sure the projection is correct, and it is not!
```

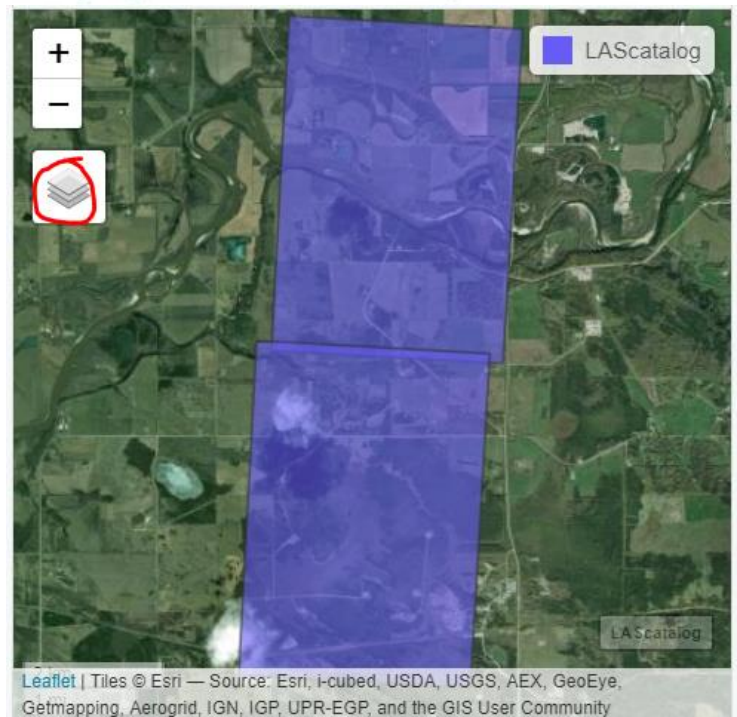
Remember, the ? pulls up documentation about the function. This is the best place to look for the required syntax and the possible options for every R command.

You can use the layers option to select what basemap you want to display.

We know that our tiles should be on the Olympic peninsula, not way up in Canada. This is the affect of the units being displayed as meters when they should be feet.

You can query what Coordinate Reference System is being used for the catalog and you already know how to deal with CRS issues.

```
projection(ctg)
```



It has the known issue of meters being used instead of feet. Change the CRS to EPSG 2927

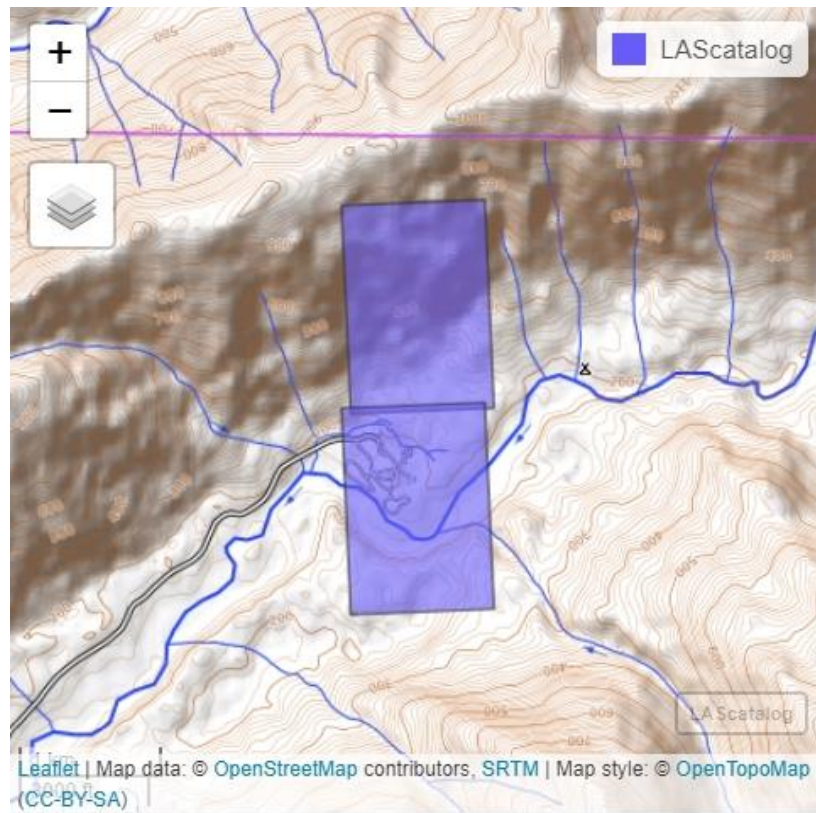
```
st_crs(ctg) <- 2927
```

Check to make sure the catalogs crs was changed:

```
projection(ctg)
```

It looks good, so plot the tiles again using mapview:

```
plot(ctg, mapview = TRUE) #places tiles onto satalite imagery
```



Looks good! Remember, you can use any base map you want. You can define what base map to use in the line of R code. For example, to display our tiles using Esri world imagery (the top map shown here). You would run:

```
plot(ctg, mapview = TRUE, map.types = "Esri.WorldImagery")
```

To see what base map options are available beyond the 5 default ones that are listed, you can use the ?mapview help command, and then search “base maps” to find a web site that lists all of the options for base maps.



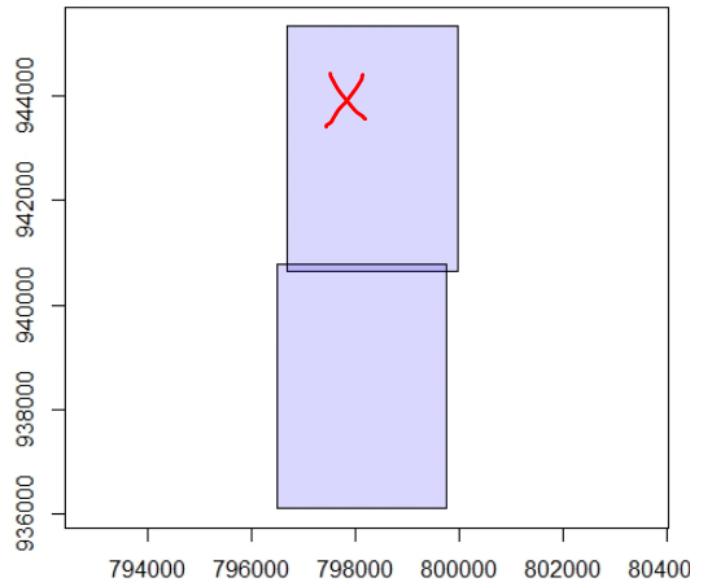
QUESTION 1: Use ?mapview command to track down the website that lists all of the base map options for mapview. Pick a base map that is not one of the 5 default (Carto.DB, OpenStreetMap, OpenTopoMap, or Esri.WorldImagery) and plot your tiles. Give the website address, the base map you used, and include a screen shot of your plot.

Make a basic plot of your tiles with the x and y axis displayed using:

```
plot(ctg) #quick way to plot the position of the tiles, we want a clip
```

We want to make a clip of the forest on the hillside to use to practice making digital surface models (DSM). The project wide DSM is available for download, just like the DTM, but we are going to make our own.

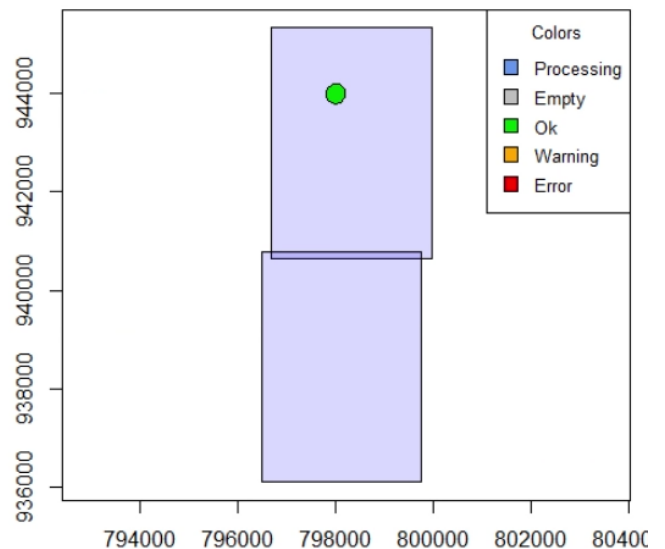
The red x is a good location for us to practice with and it is approximately located at 798000, 944000. You could plot this point in a GIS program like ArcGIS or QGIS, with those numbers, as long as you specified it was EPSG: 2927.



Let us clip a 200 ft radius circle at that location. Run:

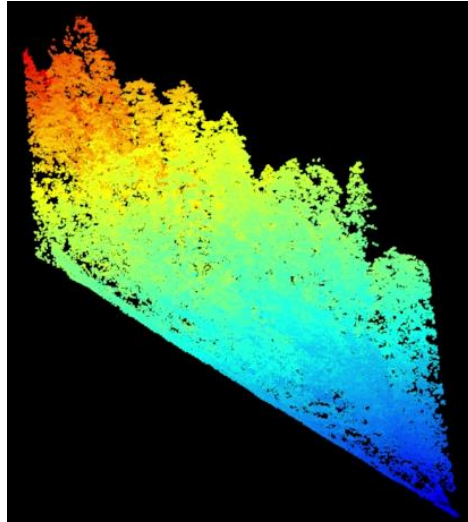
```
lasclip <- clip_circle(ctg,798000,944000,200) #creates a circle clip of 200units
```

A cool feature of processing within a las catalog in lidR, is that it gives you a graphic display about what it is processing, and how the processing is doing. In this case we see our clip, and that the processing was "OK"



Take a look at the point cloud within the clip:

```
plot(lasclip)
```



PART 2 Rasterizing Canopy Height Models

Part 2 of this lab was largely converted from the lidR wiki page found here:

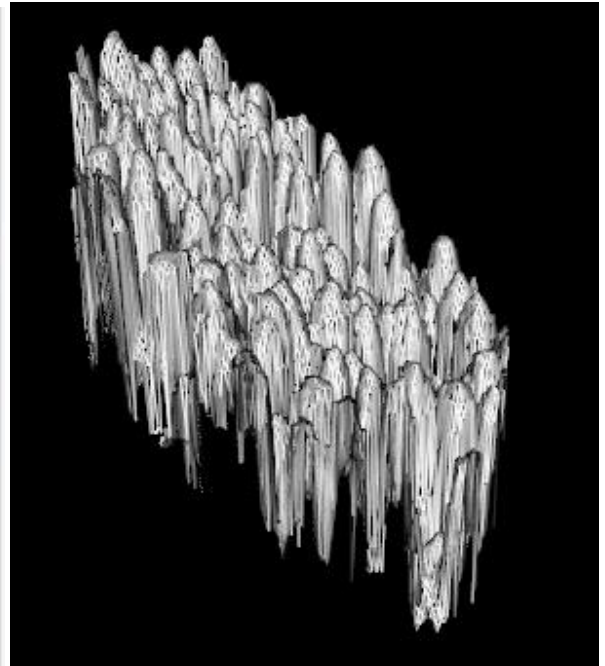
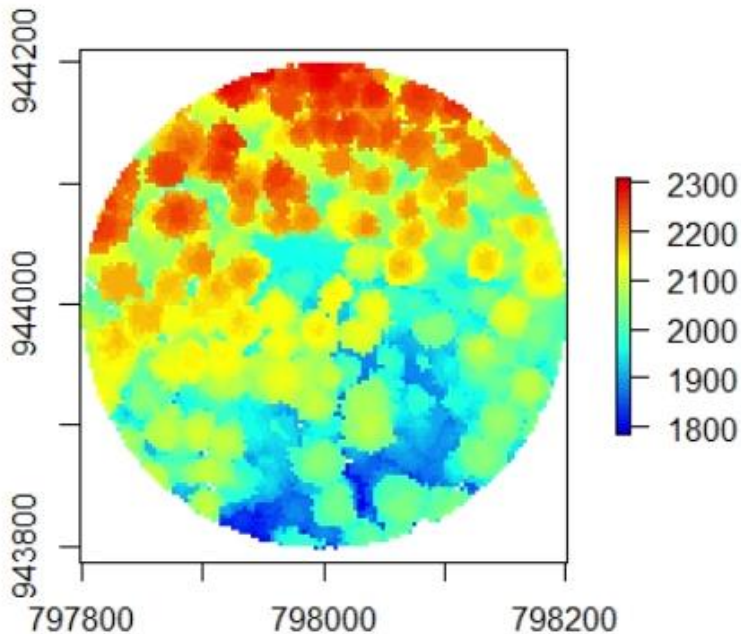
<https://github.com/Jean-Romain/lidR/wiki/Rasterizing-perfect-canopy-height-models>

updated lidR book is here: <https://r-lidar.github.io/lidRbook/>

We have our clip and it is on a significant slope... We are interested in the trees but any height information derived from the point cloud now, will also include height due to elevation. Lets take a look at a raster output of the point cloud using the maximum Z value for each pixel. This is essentially the same thing as outputting a raster in cloud compare. Run:

```
chmHILL <- rasterize_canopy(lasclip, 3.3 ,p2r())  
plot (chmHILL, col=height.colors(50))  
plot_dtm3d(chmHILL)
```


Lab 4



It certainly looks cool, we can absolutely see where there are trees, but we can't get tree height here because we don't know exactly what the slope of the hill is, and even if we knew the slope, there are variations in slope so a singular value won't give us a realistic model of the hill.

We need to "normalize" the point cloud. You have seen this term previously when you have checked out point clouds in lidR. So far, all your point clouds have not been normalized. We are going to normalize this clip so we can get information about the trees with the slope values (Z values) "normalized".

We are going to use the function "normalize_height". Take a moment to read the description and scan the help documentation. Run:

```
?normalize_height
```

normalize_height uses ground classified points to create a DTM, and then subtracts that value for all points above. Lasnormalize is dependent on the las file having classified ground points, OR, you provide a DTM raster. For this step, we will only provide our clip of lidar data that does have ground points classified and let the function first create a DTM, and then subtract those values from all non-ground points. This is all done in one simple line of code:

```
NORMclip <- normalize_height(lasclip,tin())
plot(NORMclip)
```

Lab 4

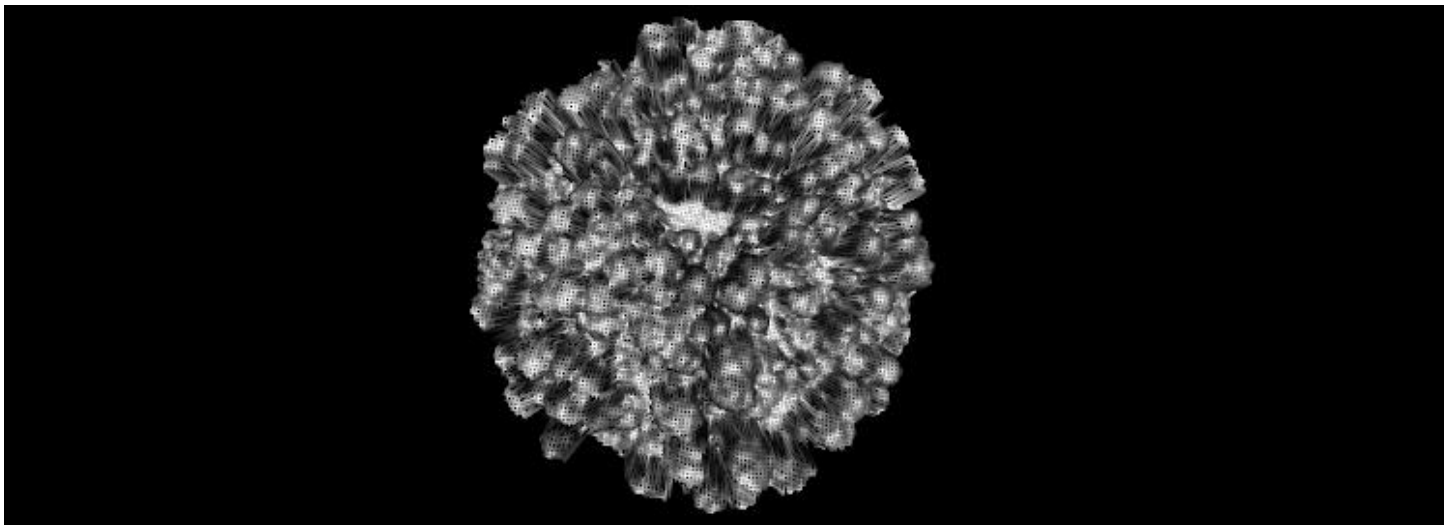
Note that we defined 'tin'. normalize_height could also use knn1dw or kriging.

QUESTION 2: Include a screenshot of your plot(NORMclip).

QUESTION 3: Run rasterize_canopy on your NORMclip. Use all the same values you did in the rasterize_canopy step above, but change lasclip to NORMclip, and change chmHILL to chmNORM. Include 2 screenshots. One of your plot(chmHILL), and one of your plot(chmNORM). You can use any colrramp you want (i.e. col=height.colors(50)). Comment on the similarities and the differences. What does the change in the numbers on the legend bar indicate?

Make sure you also check out:

`plot_dtm3d(chmNORM)`



Let us explore grid_canopy more. Make sure you check out the documentation in ?grid_canopy. The basic arguments in what you have run so far are

```
chmNORM <- rasterize_canopy(NORMclip, 3.3 ,p2r())
```

`chmNORM` = The name of the output object we are creating

`rasterize_canopy` The function that we are using that looks at the points with the highest Z value within a defined range.

`(NORMclip,` The las data that will be used

`3.28,` The resolution of the output. In this case, 3.23 ft or 1 m

`p2r())` The algorithm used. This can be p2r, dsmtin, or pitfree

Lab 4

QUESTION 4: Using the `?rasterize_canopy` help documentation, what are the descriptions for `p2r`, `dsmtin`, and `pitfree`?

When you create a DEM, cells with no value (empty cells) can cause major issues. There are a few methods that can be used to try and fill those voids. You can decrease the resolution of the output, but this effectively erases a lot of data that we likely want to retain. You can use different algorithms to fill the voids, but the more sophisticated the algorithm, the more computationally intensive it is to run. Let us start by looking at the output DSM created by decreasing the spatial resolution, and by applying the `p2r` algorithm. Run:

```
chm1 <- rasterize_canopy(NORMclip, 1, p2r())
plot(chm1,col=height.colors(50), main="Res1, p2r0")
plot_dtm3d(chm1)
chm2 <- rasterize_canopy(NORMclip, 3.3, p2r())
plot(chm2,col=height.colors(50), main="Res3.3, p2r0")
plot_dtm3d(chm2)
chm3 <- rasterize_canopy(NORMclip, 1, p2r(3))
plot(chm3,col=height.colors(50), main="Res1, p2r3")
plot_dtm3d(chm3)
```

Take a moment to compare the outputs and especially the `dtm3d` outputs.

QUESTION 5: Include screenshots of the 2D plot outputs of “Res3.28, p2r0” and “Res1,p2r3”. How are they similar and how are they different? What is `p2r3` doing to fill the holes?

Let us check out the outputs from using `dsmtin` and `pitfree`. Run:

```
chm4 <- rasterize_canopy(NORMclip, 1, dsmtin())
plot(chm4,col=height.colors(50), main="dsmtin")
plot_dtm3d(chm4)
chm5 <- rasterize_canopy(NORMclip, 1, pitfree(c(0,2,5,10,15), c(0,1), subcircle = 2))
plot(chm5,col=height.colors(50), main="pitfree")
plot_dtm3d(chm5)
```

QUESTION 6: Include screenshots of the 3D `plot_dtm3d` outputs of “dsmtin” and “pitfree”. How are they similar and how are they different? They both use Delaunay triangulation to create the DSM, what is `pitfree` doing to eliminate the spikey “pits” that are present in `dsmtin`?

The `pitfree` algorithm will likely generate a consistently better DSM from point cloud data, but the processing power required to run it across an entire acquisition is potentially prohibitively large. It is possible to smooth the spikes from a “spikey” DSM using the focal function from the `raster` package. Check out:

ESRM433/SEFS533

Lab 4

?focal

The smoothing happens by running an image kernel across the image. Think of a small moving window with varying sizes, but typically a 3x3pixel window or 5x5pixel window. This is the same thing that filters in image processing programs do. Have your “sharpened” an image you took on your phone? It was likely done using an image kernel. It is beyond the scope of this lab to go indepth on image kernels but more information can be found here:

<https://setosa.io/ev/image-kernels/>

Run:

```
chmSmooth = focal(chm4, w = matrix(1,5,5), fun = mean)
plot(chmSmooth, col = height.colors(50), main = "Smoothed")
plot_dtm3d(chmSmooth)
```

Chm4 should be your DSM produced from ‘dsmtin’. Make sure to read the ?focal documentation. The matrix values are the size and values for the smoothing window, and ‘fun’ is the function used (i.e. mean, median, min, max). Run a few combinations of matrix sizes and different functions until you find a combination that produces a relatively smooth DSM that captures the locations and shapes of the trees reasonably well, without “pits”.

QUESTION 7: Include a screen shot of your output from plot and plot_dtm3d of your chmSmooth. What values did you use for the smoothing? Why?

PART 3 Tree Detection and Segmentation

Part 3 of this lab was largely converted from the lidR online book. Refer to the book for a more extensive exploration of the tools.

We will first be detecting individual trees within our point cloud. We will be using locate_trees. Take a moment to read the documentation for the tool

?locate_trees

Familiarize yourself with the arguments and also the paper that the algorithm lmf was derived from. The most important argument in lmf is the window size (ws).

Create an object ttops for the tree tops detected in our original lasclip.

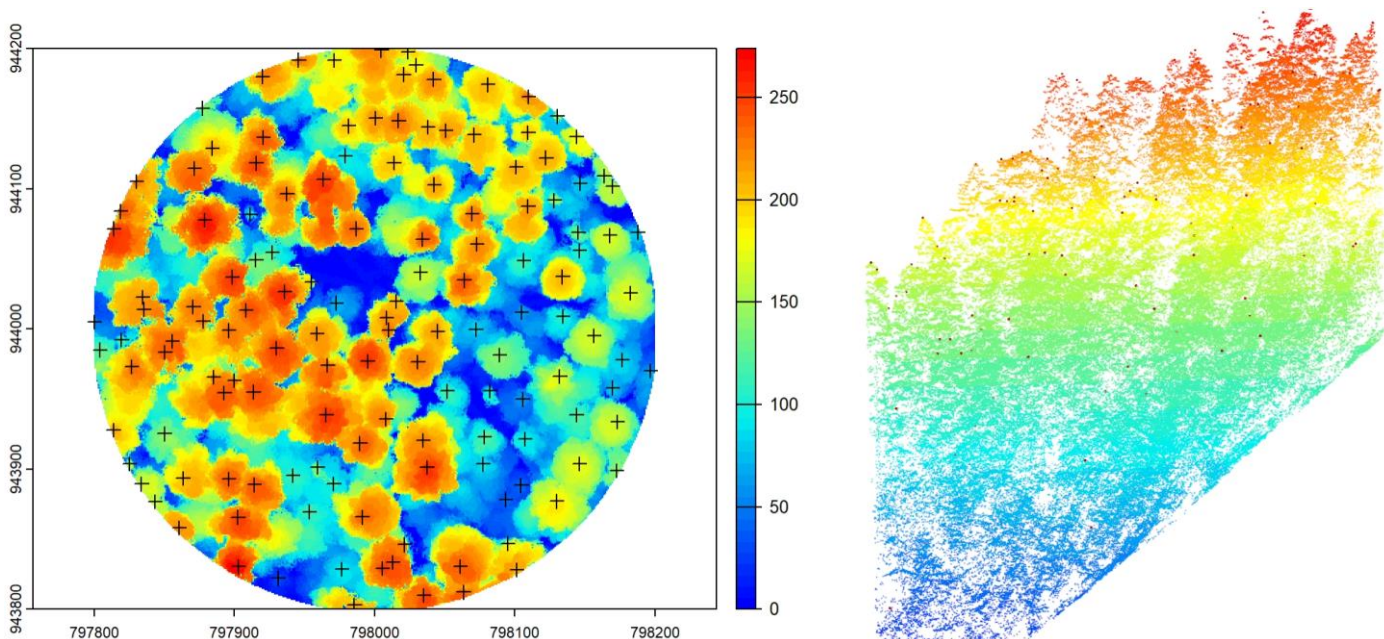
```
ttops <- locate_trees(lasclip, lmf(ws=15))
plot(chm5, col= height.colors(50))
plot(sf::st_geometry(ttops), add=TRUE, pch=3)
```

The line of code sf::st_geometry is just calling the sf package to overly the ttops location information on our chm5 canopy height model. Pch=3 is just defining the symbol used.

You can also display the ttops in 3D on the original lidar clip.

Lab 4

```
Tops<-plot(lasclip,bg="white",size=2)
add_treetops3d(Tops,ttops)
```



As you can imagine, the window size can impact the results dramatically. Play around with a few different `ws` values and remember that our point cloud units are feet.

QUESTION 8: Include a screen shot of the 2D output (like the upper left image) using a different `chm` and/or different `ws` value for `locate trees`. Find a combination that you feel is the best and report the values you used.

That is all we are going to cover for individual tree detection. Tree segmentation is a bit different.

Will be using the `segment_trees` and `crown_metrics`. As always, check out the documentation for both tools

```
?segment_trees
?crown_metrics
```

When looking through the `?segment_trees` documentation, note that there are many algorithms that can be used for tree segmentation.

QUESTION 9: Read the descriptions for all the algorithm options in `?segment_trees`. 4 of the options are named after an author and year. What are those papers? Copying and pasting the citation from RStudio is acceptable.

Feel free to try out all of the different segmentation algorithms. We will mostly be focused on `li2012` and `dalponte2016` but there isn't a single best algorithm to use. Each forest and tree type will likely perform differently with different segmentations. Let's perform a segmentation on our `NORMclip`.

ESRM433/SEFS533

Lab 4

Also check out the documentation for the `filter_poi` tool. This is super helpful for filtering points in the dataset

?filter_poi

```
lasT<-segment_trees(NORMclip, li2012(R = 15,hmin = 6, speed_up = 30))  
lasT<-filter_poi(lasT,!is.na(treeID))  
plot(lasT,bg="white",size=4, color="treeID")
```

Make sure you understand what the inputs for the `li2012` algorithm are. Also remember that the units for the examples in the documentation are assuming that your units for the point cloud are meters. Are units are feet so multiplying any value given that refers to a unit of measurement you should triple.

The `filter_poi` line is filtering points and removing all points that don't have a `treeID` number associated with them.



QUESTION 10: Experiment with the inputs for `li2012`. You can add additional values beyond just `R` and `speed_up`. You should be able to improve the segmentation beyond the values given. Take a screen shot of the segmentation you prefer and report that values you used.

There is now a list of trees in your point cloud. You can plot individual trees from your point cloud:

```
tree24<-filter_poi(lasT,treeID==24)  
plot(tree24,size=8,bg="white")
```

I picked tree number 24, but your numbers will be different.

QUESTION 11: Try several tree numbers until you find a good one. Include a screenshot of your individual tree. You can color it however you like.

Now that you have done all this processing on a `las` file, you will likely want to save the file so you can open it in a better visualizing program such as `CloudCompare`. You write your outputs like this:

```
writeLAS(lasT,file="Lab4/trees.las")
```

Check your `Lab4` folder and you should now have a new `las` file in there. With points that have a `treeID` as an additional scalar field. Take it into `CloudCompare` to check it out.



ESRM433/SEFS533

Lab 4

There is so much we could do with `segment_trees` and I encourage you to continue playing around with the inputs and reference the `lidR` book.

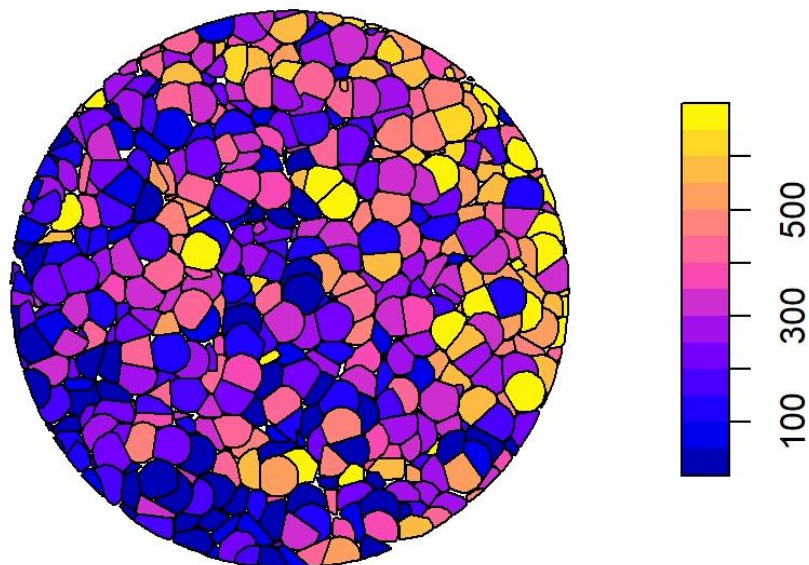
We can also create 2D maps of the crowns using `crown_metrics`.

Using your `lasT` lidar data that you have spent some time on refining the tree segmentation, you can find statistics about your trees such as height (Z) and crown area:

```
crowns <- crown_metrics(lasT, func = .stdtreemetrics, geom = "convex")
summary(crowns)
plot(crowns["convhull_area"], main = "Crown area (convex hull)")
```

The output of the `summary(crowns)` will give you the number of trees identified (Max treeID), the shortest, tallest and mean tree heights (Z), and the max, min, and mean of your tree crown area (`convhull_area`)

Crown area (convex hull)



treeID	Z	npoints	convhull_area
Min. : 1.0	Min. : 3.1	Min. : 4.0	Min. : 0.373
1st Qu.: 117.0	1st Qu.: 121.8	1st Qu.: 114.0	1st Qu.: 130.927
Median : 233.0	Median : 185.6	Median : 293.0	Median : 279.831
Mean : 234.8	Mean : 166.7	Mean : 427.1	Mean : 294.911
3rd Qu.: 351.0	3rd Qu.: 218.6	3rd Qu.: 576.0	3rd Qu.: 434.164
Max. : 480.0	Max. : 273.9	Max. : 2883.0	Max. : 693.909

QUESTION 12: Include a screenshot of your Crown area and stats. They can be similar to mine, but they should be different.

QUESTION 13: Errors in tree identification can be classified as errors of omission (trees that exist but were not identified) and errors of commission (trees that were “made up,” usually because one tree is split into several portions). Do you think your models are more likely to have errors of omission or commission? Do you think smoothing of the canopy height models effects these errors?

ESRM433/SEFS533

Lab 4

Ultimately, there is not one correct way to perform a tree segmentation. Tree density, height, and uniformity may all require a different approach to accurately estimate number of trees. The method used to produce the DSM also has a massive impact on how well trees are detected.

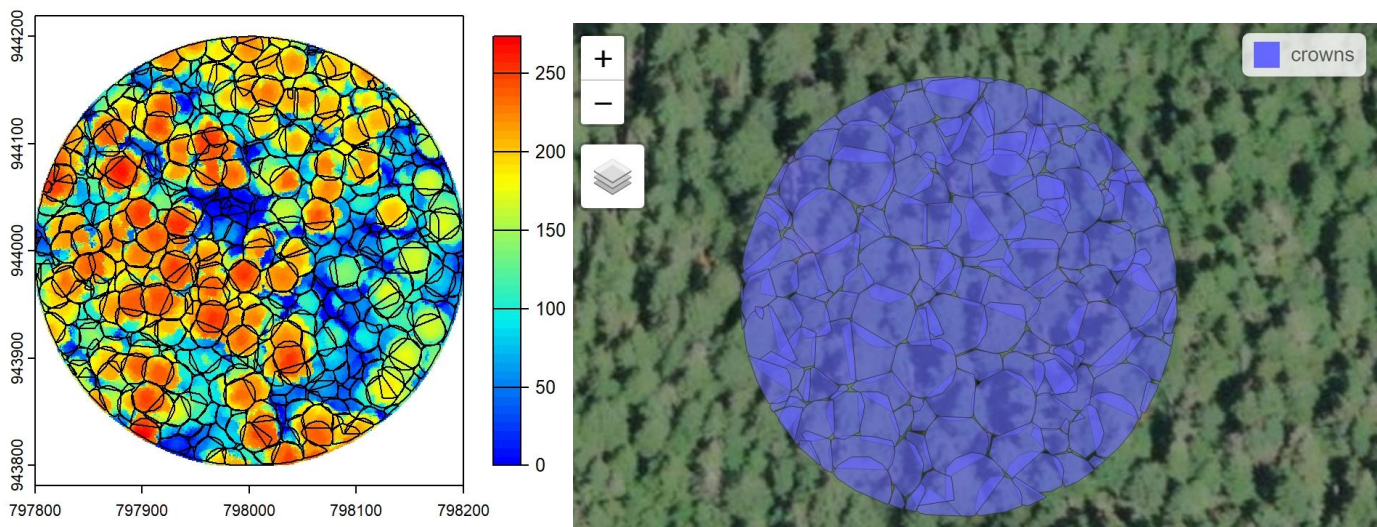
You can plot the hulls of your crowns with your chm:

```
plot(chm5, col= height.colors(50))  
plot(sf::st_geometry(crowns), add=TRUE)
```

You can also plot your hulls on the map!

A lot of refinement needs to happen for the results to be reliable, but this lab is intended to point you in the right direction to apply individual tree detection across an area.

```
mapview(crowns)
```



PART 4 Tree map of the Arboretum

Go to the Washington DNR lidar portal and download the King County 2016 laz tile for the northern portion of the Arboretum. You should download the Metadata to get information about the dataset.



The tile you have should be named q47122f3417_rp.laz

ESRM433/SEFS533

Lab 4

You are going to run the following code. For each line of code you run, you are going to write a short discretion of what that line does. A single sentence will likely be enough, but make sure to say what the input values indicate. Using the code we have already run, here is an example:

174	Reads in the las file q47122f3417_rp.laz using the readLAS command from lidR
-----	--

Run this code and provide comments in the submission template:

```
174 Arborlas<-readLAS("Lab4/q47122f3417_rp.laz")
175 ArborClip<-clip_rectangle(Arborlas,1197000,845300,1198200,846700)
176 ArborNORM <- normalize_height(ArborClip,tin())
177 st_crs(ArborNORM)<-2927
178 ArborCHM <- rasterize_canopy(ArborNORM, 1, p2r(3))
179 ArborT<-segment_trees(ArborNORM, li2012(R = 15,hmin=6, speed_up = 30))
180 ArborT<-filter_poi(ArborT,!is.na(treeID))
181 plot(ArborT,bg="white",size=4, color="treeID")
182 Arborcrowns <- crown_metrics(Arbor,.stdtreemetrics,geom = "convex")
183 mapview(Arborcrowns)
184 plot(ArborCHM,col= height.colors(50))
185 plot(sf::st_geometry(Arborcrowns),add=TRUE)
186 writeLAS(ArborT,file = "Lab4/ArborCrowns.las")
```

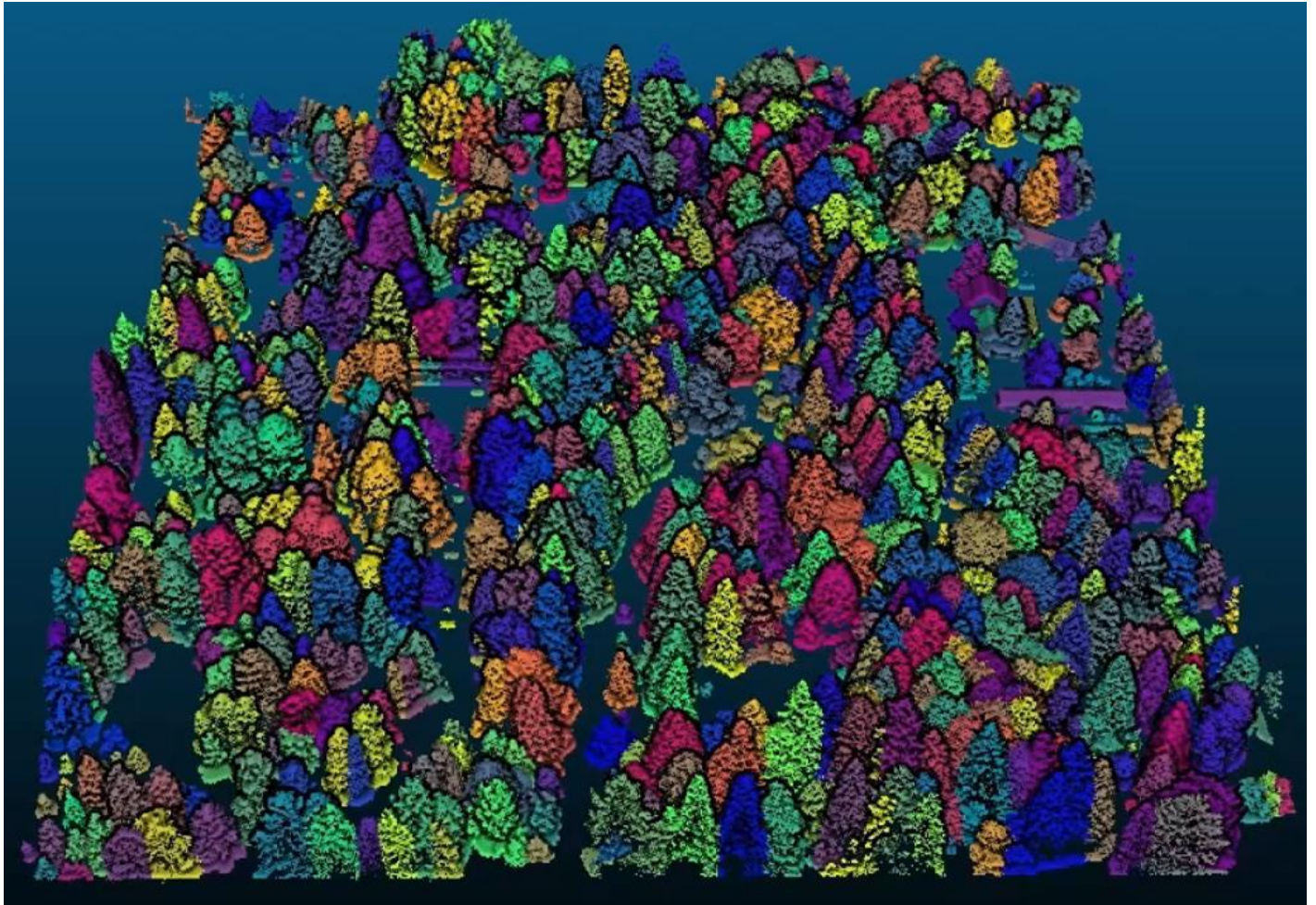
QUESTION 14: Comment each line of code (table in submission template)

QUESTION 15: Include screenshots of your outputs from (3 screenshots total):

- **plot(ArborT...**
- **mapview(Arborcrowns)**
- **plot(ArborCHM...**

You just created a tree map of a portion of the arboretum! You could use both tiles that cover the arboretum to make a complete tree map.

You will likely notice that not all trees have a polygon, and not all polygons are trees. What the algorithm is actually doing is just identifying “tall” objects. We could put some effort into removing buildings in the point cloud if that is deemed necessary.



Individual tree detection for the northern portion of the arboretum rendered with CloudCompare.

GRADUATE STUDENT

You cited several papers in question 9. Write a short paper summary on one of them. One or two paragraphs will suffice.

R Appendix

```
91 ▾ #####
92 ▾ ##### Lab 4 #####
93 ▾ #####
94 install.packages("mapview")
95
96 library(lidR)
97 library(mapview)
98 library(terra)
99
100 setwd("//fs-persona.sefs.uw.edu/student_redirect$/jonbatch/Desktop/ESRM433/")
101
102 help(package=lidR)
103 ctg <- readLAScatalog("Lab3/TILES/")
104
105 ?mapview
106 plot(ctg,mapview=TRUE, map.types = "Esri.WorldImagery")
107
108 projection(ctg)
109 st_crs(ctg) <- 2927
110 projection(ctg)
111
112 plot(ctg)
113 lasclip <- clip_circle(ctg,798000,944000,200)
114 plot(lasclip)
115
116 chmHILL <- rasterize_canopy(lasclip, 3.3 ,p2r())
117 plot (chmHILL, col=height.colors(50))
118 plot_dtm3d(chmHILL)
119
120 ?normalize_height
121 NORMclip<-normalize_height(lasclip,tin())
122 plot(NORMclip)
123
124 chmNORM <- rasterize_canopy(NORMclip, 3.3 ,p2r())
125 plot(chmNORM)
126 ?rasterize_canopy
127
128 chm1 <- rasterize_canopy(NORMclip, 1, p2r())
129 plot(chm1,col=height.colors(50), main="Res1, p2r0")
```

ESRM433/SEFS533

Lab 4

```
130 plot_dtm3d(chm1)
131 chm2 <- rasterize_canopy(NORMclip, 3.3, p2r())
132 plot(chm2,col=height.colors(50), main="Res3.3, p2r0")
133 plot_dtm3d(chm2)
134 chm3 <- rasterize_canopy(NORMclip, 1, p2r(3))
135 plot(chm3,col=height.colors(50), main="Res1, p2r3")
136 plot_dtm3d(chm3)
137
138 chm4 <- rasterize_canopy(NORMclip, 1, dsmtin())
139 plot(chm4,col=height.colors(50), main="dsmtin")
140 plot_dtm3d(chm4)
141 chm5 <- rasterize_canopy(NORMclip, 1, pitfree(c(0,2,5,10,15), c(0,1), subcircle = 2))
142 plot(chm5,col=height.colors(50), main="pitfree")
143 plot_dtm3d(chm5)
144
145 ?focal
146 chmSmooth <- focal(chm4,w=matrix(1,5,5), fun=mean)
147 plot(chmSmooth,col=height.colors(50),main="Smoothed")
148 plot_dtm3d(chmSmooth)
149
150 ?locate_trees
151 ttops <- locate_trees(lasclip,lmf(ws=15))
152 plot(chm5, col= height.colors(50))
153 plot(sf::st_geometry(ttops),add=TRUE, pch=3)
154 Tops<-plot(lasclip,bg="white",size=2)
155 add_treetops3d(Tops,ttops)
156
157 ?segment_trees
158 ?crown_metrics
159 ?filter_poi
160 lasT<-segment_trees(NORMclip, li2012(R = 15,hmin = 6, speed_up = 30))
161 lasT<-filter_poi(lasT,!is.na(treeID))
162 plot(lasT,bg="white",size=4, color="treeID")
163 tree24<-filter_poi(lasT,treeID==24)
164 plot(tree24,size=8,bg="white")
165 writeLAS(lasT,file="Lab4/trees.las")
166
167 crowns <- crown_metrics(lasT,func = .stdtreemetrics,geom = "convex")
```


ESRM433/SEFS533

Lab 4

```
168 summary(crowns)
169 plot(crowns["convhull_area"],main="Crown area (convex hull)")
170 mapview(crowns)
171 plot(chm5, col= height.colors(50))
172 plot(sf::st_geometry(crowns),add=TRUE)
173
174 Arborlas<-readLAS("Lab4/q47122f3417_rp.laz")
175 ArborClip<-clip_rectangle(Arborlas,1197000,845300,1198200,846700)
176 ArborNORM <- normalize_height(ArborClip,tin())
177 st_crs(ArborNORM)<-2927
178 ArborCHM <- rasterize_canopy(ArborNORM, 1, p2r(3))
179 ArborT<-segment_trees(ArborNORM, li2012(R = 15,hmin=6, speed_up = 30))
180 ArborT<-filter_poi(ArborT,!is.na(treeID))
181 plot(ArborT,bg="white",size=4, color="treeID")
182 Arborcrowns <- crown_metrics(Arbor,.stdtreemetrics,geom = "convex")
183 mapview(Arborcrowns)
184 plot(ArborCHM,col= height.colors(50))
185 plot(sf::st_geometry(Arborcrowns),add=TRUE)
186 writeLAS(ArborT,file = "Lab4/ArborCrowns.las")
```