# Software Design Document

March 2, 2021

Team LumberHack

**Sponsor:**

Dr. Andrew J. Sánchez Meador

**Mentor:**

Melissa D. Rose

**Team Members:**

Matthew Flanders

Jenna Pedro

Thomas Whitney

Colin Wood

## Table of Contents

Introduction 3

Problem Statement

Solution Overview

Implementation Overview 7

Architectural Overview 8

Module and Interface Descriptions 10

Implementation Plan 15

Conclusion 16

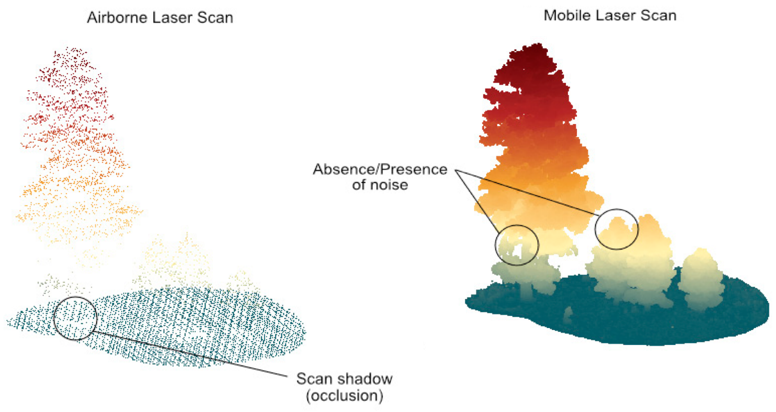
## Introduction

Forest ecosystem health is at the center of many large-scale environmental problems. Over the last century, forest management policies have heavily focused on timber production and fire exclusion. The focus on production has led to high tree densities and more combustible material in forests. This has resulted in a greatly increased risk of catastrophic wildfires, longer lasting and more frequent droughts , as well as increased recruitment of invasive species taking over forests. With climate change advancing, forest ecosystems are facing greater threats than ever before. Scientists and forest managers work on restoration projects to try and bring forests back to a healthy state. These projects include forest surveying, thinning, and cleaning of high danger, dense areas. The scale of these projects ranges from a few acres of land to many square miles of forests. With better surveying tools and analysis, restoration efforts are quicker and yield more useful information. The sponsor for this project is Dr. Andrew J. Sánchez-Meador, an Associate Professor at the School of Forestry, as well as the Executive Director at the Ecological Restoration Institute at Northern Arizona University. Dr. Sánchez-Meador's research focuses on using quantitative forest ecology for multi-scale forest ecology and restoration projects. His recent research applies practical interpretations of complex data sets to analyze individual tree growth and provide new ways of visualizing data for science communication. By using remote sensing technologies such as light detection and ranging, or lidar, Dr. Sánchez-Meador and other ecologists are able to collect and interpret large complex data-sets to model individual tree growth and indicators of forest health.

*Ponderosa Pine Forest*
*Flagstaff, AZ*

## Problem Statement

Due to forests covering large spatial areas over difficult terrain, collecting observations for an individual tree can be challenging. Lidar instead serves as a way to collect informative-data for individual trees over large forested areas. Lidar is a remote sensing method that uses light pulses combined with Global Positioning System (GPS) data to create precise, three-dimensional point clouds replicating the shape and characteristics of trees and their surrounding environment. Airborne lidar scanning (ALS) has been the traditional method for surveying forests, utilizing a lidar sensor attached to the bottom of an airplane or helicopter. ALS covers a larger range than other forms of lidar, such as mobile lidar, but the number of data points is low (around 10 points per square meter). With mobile lidar scanning (MLS), data is instead taken from the perspective of the forest floor. Mobile lidar can have a couple thousand points per square meter which results in denser, higher-resolution point clouds.



Current tools are lacking in a few ways such as:

- No focus on mobile lidar
- Steep and complex learning curves
- Programming knowledge is usually required
- Visualization can be improved upon
- No graphical user interfaces (GUI's)

With MLS gaining popularity, researchers such as Dr. Sánchez-Meador need a tool that is built to effectively evaluate MLS data and will yield informative ecological characteristics.

## Solution Vision

The envisioned solution of this project will serve as a resource for researchers and ecologists to analyze their own MLS data. As required by the sponsor of the project, this program will be created in the R programming language with C++ backend support. As part of the final product, the team plans to create an application with the following features:

- Data Upload
    - Users will be able to upload only .laz and .las files.
    - Users will be able to upload as many files as they would like, under 10GB.
- Data Cleaning and Denoising
    - Once data is uploaded, it will be cleaned and normalized using a statistical outlier removal algorithm (SOR).
- Individual Tree Segmentation
    - After the uploaded data has been cleaned, the user can run a tree segmentation algorithm to separate trees based on their bole position.
- Point Classification
    - Individual points in the point cloud will be classified based on their position on a tree such as bole, branch, or crown.
- Derived Data Products
    - Users will then be able to select specific trees and obtain relevant measurements such as tree height, tree diameter, and lowest branch height.
    - Users will also be able to see how two temporally distinct but overlapping point clouds match up
- Data Summaries/Visualization
    - The GUI will also display statistics in both a 3D interactive point cloud, as well as on a table.

The final product will be a single portable tool that any ecologist can access and use. With a straightforward GUI, the team plans for users to be more focused on their research than on learning a new piece of software. The project will focus on creating a MLS specific application to support the growing number of researchers utilizing MLS. The features explained above will be wrapped into a Shiny R application that will be portable and accessible from the web.

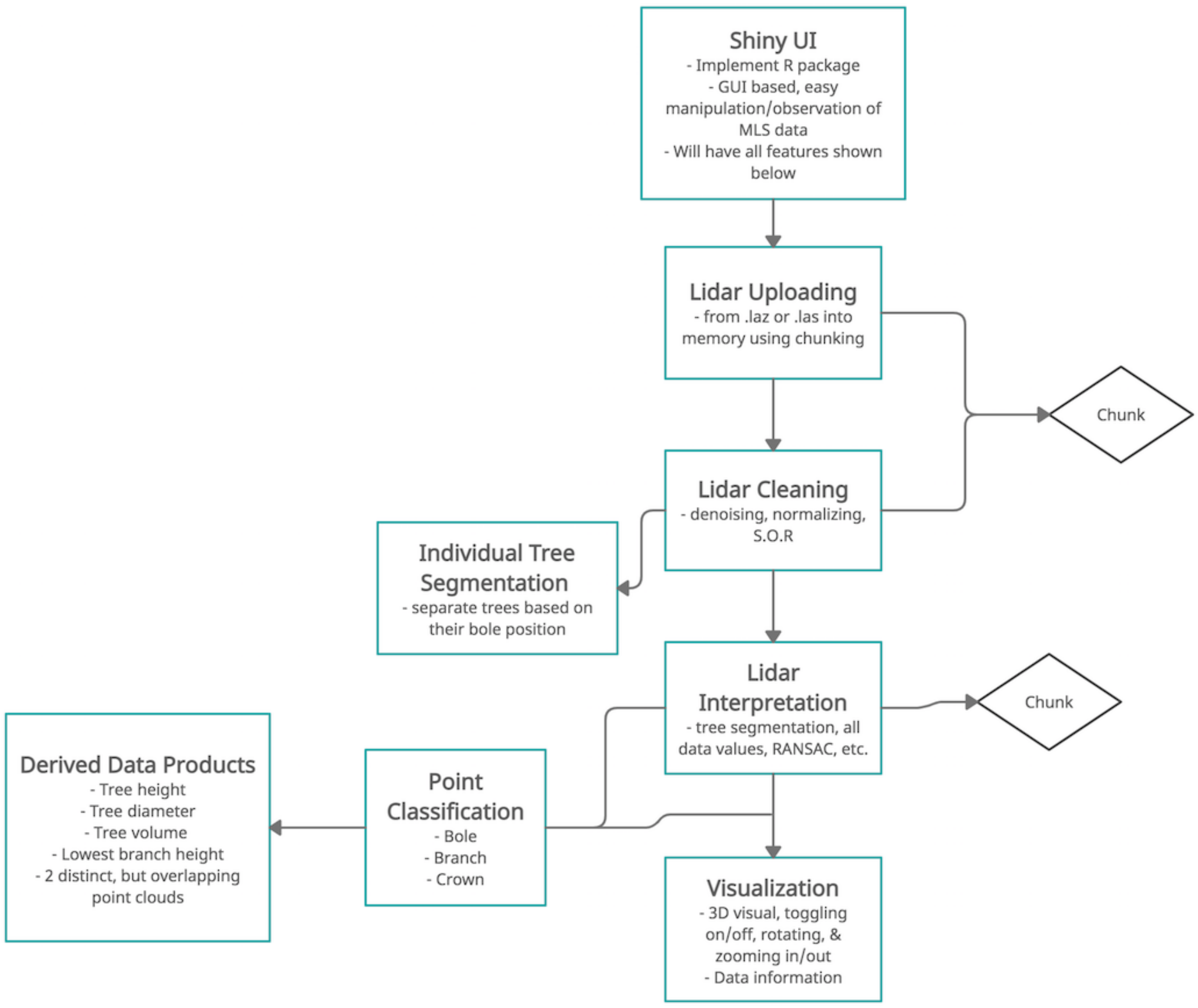

Shiny UI
- Implement R package
- GUI based, easy manipulation/observation of MLS data
- Will have all features shown below
Lidar Uploading
- from .laz or .las into memory using chunking
Chunk
Lidar Cleaning
- denoising, normalizing, S.O.R
Individual Tree Segmentation
- separate trees based on their bole position
Lidar Interpretation
- tree segmentation, all data values, RANSAC, etc.
Chunk
Derived Data Products
- Tree height
- Tree diameter
- Tree volume
- Lowest branch height
- 2 distinct, but overlapping point clouds
Point Classification
- Bole
- Branch
- Crown
Visualization
- 3D visual, toggling on/off, rotating, & zooming in/out
- Data information

*Fig. 1 Solution Vision Workflow*

## Implementation Overview

To implement the web app that will take user inputted LIDAR files, process them, and provide meaningful results, several different technologies will be used. The general approach is that of a layered pattern. At the top will be the user interface layer, at which the user performs all inputs and receives all outputs. Beneath that, the application layer will perform the meaningful computational tasks, including LIDAR cleaning and processing and generating visualizations. At the bottom is the database layer, where the LIDAR data, parsed C++ objects, and derived statistics will be stored.

The core technology implemented in the user interface layer is Shiny, an R-based web application framework. In this case, the user will interact only locally with the Shiny app–it will not be served over the internet. The user will download and locally serve the Shiny app as part of the software installation process. At the application layer, multiple technologies will be implemented. Functions written in the R programming language and either developed by the team or borrowed from the existing lidR package, will be the most common means of implementing core functionality. The lidR package provides many methods for common LIDAR processing tasks. These R functions in turn will sometimes call functions developed in C++ to perform the most computationally demanding tasks. Once all data have been processed and all statistics have been computed, outputs will be generated primarily in R (more specifically in Shiny), and a three-dimensional visualization will be displayed using WebGL, a plugin-free JavaScript API that allows 2-D and 3-D graphics to be displayed in most modern browsers. The use of WebGL requires no additional installation or setup by the user. Thus, an R interpreter, C++ compiler, and web browser are the core tools needed to run the application. After installing the requisite R packages, and the R, C++, and JavaScript source code that comprise the application, the user will have all that is necessary for full functionality.

## Architectural Overview

Below is an architectural diagram that overviews the application's main functions. The diagram is divided into the three layered sections discussed above: interface, application, and database layers.

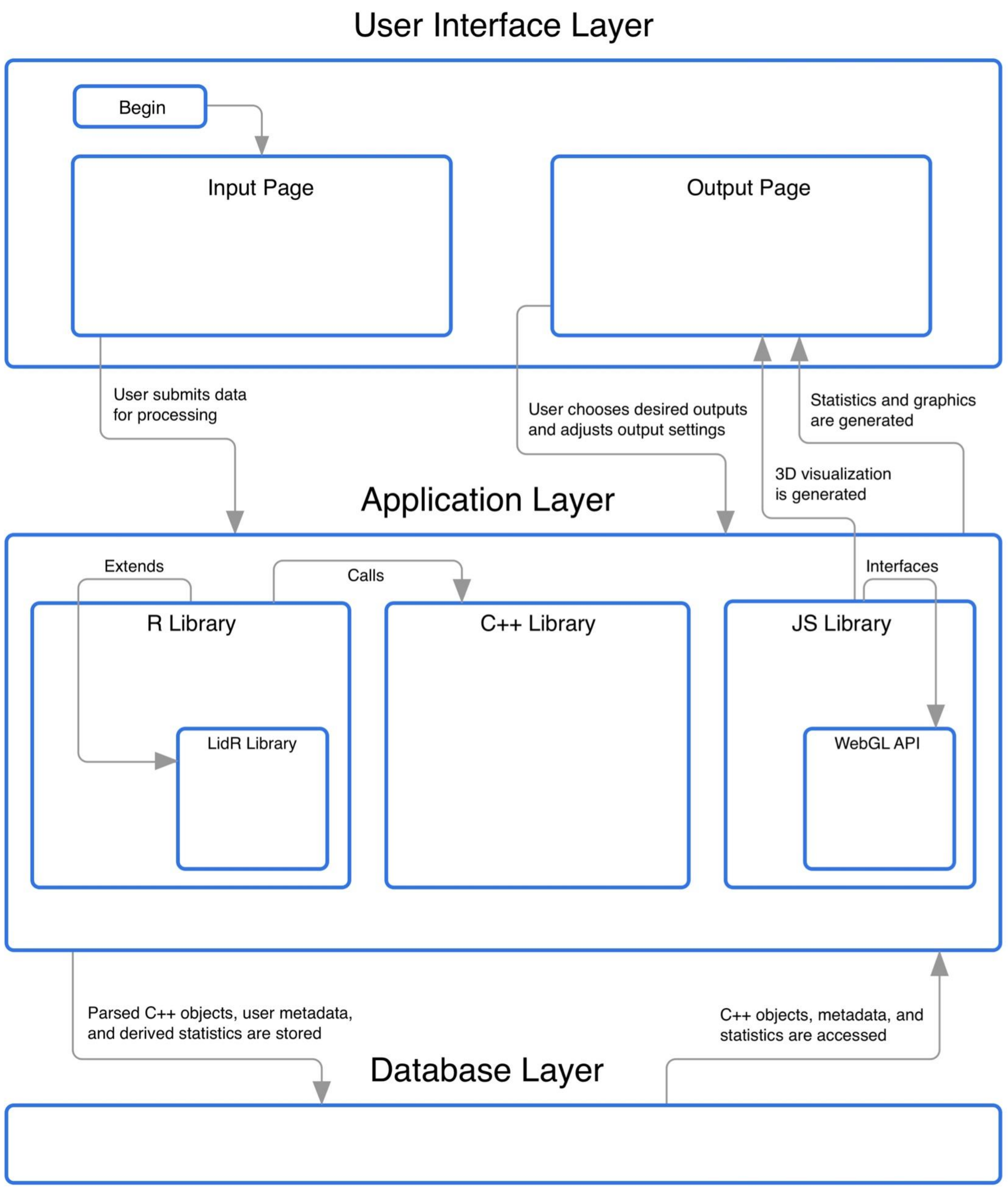

User Interface Layer
Begin
Input Page
Output Page
User submits data for processing
User chooses desired outputs and adjusts output settings
Statistics and graphics are generated
3D visualization is generated
Application Layer
Extends
Calls
Interfaces
R Library
C++ Library
JS Library
LidR Library
WebGL API
Parsed C++ objects, user metadata, and derived statistics are stored
C++ objects, metadata, and statistics are accessed
Database Layer

*Fig 2. Architectural Diagram*

Beginning at the user interface layer, there will be one of two main views displayed to the user at any given time. The first is the input view, where the user has the ability to upload LIDAR files, parameterize the cleaning, filtering, and processing of those files, and adjust the usage of their system's resources. These options will be implemented in essence as HTML forms, with dropdowns, radio buttons, and other input methods as appropriate. The second view, the output page, is displayed to the user once processing has completed and the requested statistics and visualizations have been generated. What is displayed on this page will vary according to what the user selects, and is not necessarily static–options to adjust the statistics or visualizations will be available.

At the application layer, the R library is responsible for responding to user input, carrying out all of the LIDAR processing functionality, and most of the statistics and graphics generation. R functions will call C++ functions for efficiency where appropriate. The JavaScript library is responsible for generating the 3-D visualization.

The database layer is responsible for structuring and storing the C++ objects that represent the parsed LIDAR data and the user's metadata. Entities stored in the database can be exported to JSON for persistence between runs of the program.

Information will flow in a cyclical manner through the layers, often beginning at the top with user input, progressing to the application layer for processing, optionally reaching down to the database for stored information, and returning to the user interface layer for display to the user. Various means will exist for the user to send information to the application layer, including forms, file upload, and buttons, each with a distinct purpose. Information will flow back to the user primarily through statistical and graphical output displayed on the output page. These outputs can be modified, repeating the information cycle. Shiny sits at the interface between the user interface layer and the application layer, and will be involved in virtually all transmission of information between the two.

## Module and Interface Descriptions

### Shiny User Interface

The Shiny R package will serve as a wrapper for the rest of the program. This means that the application is being built around the Shiny framework. Shiny works for projects of any size, however, for larger projects like this one it is important to stay organized. The team is developing each module in its own R file to allow for expansion without losing readability and easy file navigation.

Shiny allows for functions to be run and parameters to be set through a user interface. Drop down menus allow for file upload, buttons, text fields, and radio buttons will allow for user input and parameter modification. These parameters will allow for users to modify the functions being used to interpret the data set. The Shiny UI module is what ties all of the other modules together. Everything from the data set itself to the functions that are modifying it will be based around the Shiny module. We have built in file uploading as well as point classification from the lidR package into the Shiny UI. We have also built an embedded 3 dimensional point cloud viewer that allows for panning through a point cloud. Later on, we will build a table into the Shiny UI that allows for 2 dimensional viewing of data as well. The Shiny UI will tie the application together and give the user a straightforward workflow.

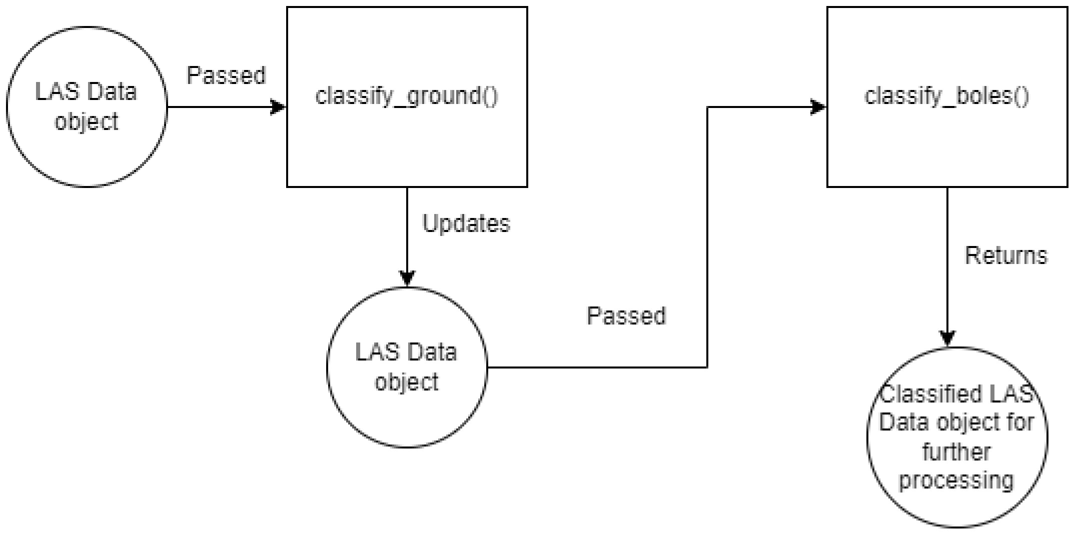
### Point Classification

The project will include the function to classify specific points in the lidar point cloud. Point classifications will build off of one another, meaning the order in which points are classified is important for accuracy and efficiency. The order of these responsibilities of this function and the order of each classification is as follows:

1. First, the ground will be classified. This will be done by implementing the function classify_ground() from the lidR package. This function works by turning the point cloud upside down, and then uses a cloth simulation filter to identify the ground. Cloth simulation filtering works by acting as if a cloth is laid on the upside down point cloud, any points that touch this cloth are identified as ground points. These points can then be removed from the data set for further classification.

2. Next, the bole of the tree will be classified by implementing a RANSAC algorithm to identify the cylindrical structure of a tree bole. The RANSAC algorithm will randomly select points with replacement to identify a best-fitting cylinder to fit over the bole of the tree. The cylinder will identify the position, verticality, and diameter of the tree which will be returned for data visualization in a further step. Once tree boles have been classified, they will be removed from the data set.

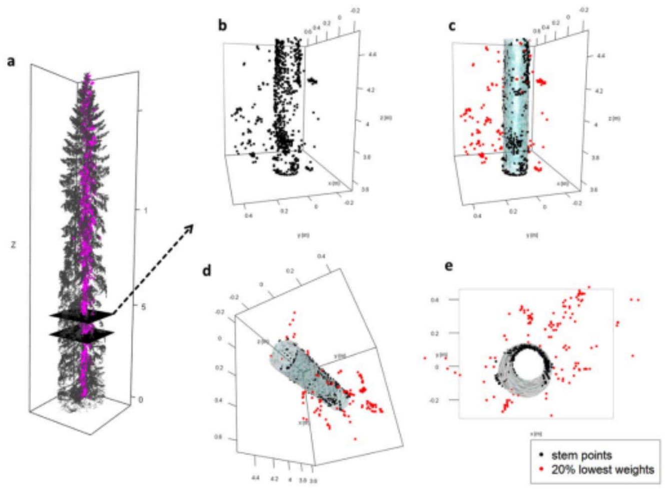
With the ground and bole classified, we will be able to identify individual trees. The RANSAC algorithm will return a position and radius of each individual tree that will be visualized in a further step. To use the classification, users will click an event listener button that will run all of the point classifications. The diagram below explains how the different point classifications are related and how this component of the system integrates with the rest of the project.



Point classification serves as a baseline for much of the other functions of the project. Accurate point classification allows for the data to be processed and interpreted more accurately and efficiently in the other modules below.

## Cylinder Shape Fitting With Error Reporting

In order to derive data products like tree diameter and individual tree segmentations, the team will be using cylinder shape fitting with the tree center size at 1.37m. This will use RANSAC (Random Sample Consensus) for tree boles with attribute and error reporting. RANSAC extracts random individual sets of points from the point data. Even with a high level of outliers present, it is still a reliable process. The following steps will be done in order to implement this function:



1. First, the trees need to segment individually and this will be done with the function segment_trees() from the lidR package. This by default will give each tree an ID number to distinguish between distinct individuals. The way that segment_trees works is by:
    a. Use Li et al. 2012 with segment_trees(las, li2012(R = 3, speed_up = 5))
2. Then, as we use RANSAC for the cylinder shape fitting:
    a. A slice of the bole will be taken and starts separating points from each other
    b. For a good cylinder fit, there needs to be about 15 to 30 random points and this process will be done about 10 to 20 times, which will be done by repeating the process.
    c. These points should outline a 1.37m sample circle and the tree location is defined with the projection of the circle's center
3. Lastly, for error reporting, about a 20% margin of error and 80% accuracy would be used for the team's software.

## Calculate Differences Between Ecological Pertinent Attributes

A function that this application will include is having the ability to calculate differences between ecological attributes including height, volume, above-ground biomass, and individual tree location across two individual trees in time. Having the ability to calculate these attributes will allow users to clearly visualize the MLS data and get a summary of these attributes. In order for this function to work, the following steps must be taken:

1. First, the data has to be denoised and cleaned. This will be done by having the data chunked with C++ then implementing the function classify_noise() from the lidR package. The way that classify_noise() works is by:
   a. Add 20 artificial outliers
   b. Use SOR (Statistical Outliers Removal) with classify_noise(las, sor(15,7))
   c. Use IVF (Isolated Voxels Filter) with classify noise(las, ivf(5, 2))
   d. Remove outliers using filter_poi(las, Classification != LASNOISE)
2. Next, we have to classify the ground using classify_ground() from the lidR package as well. Afterwards, we classify the trees and segment them individually with segment_trees(). The way that classify_ground() works is by:
   a. Use CSF(Cloth Simulation Filter)
   b. Classify ground with classify_ground(las, csf)
   c. Normalize height with normalize_height(las, tin())
   d. Decimate points with decimate points(las, random_per_voxel(1,1))
3. Lastly, the height-above-ground method will be used to calculate tree height. The height above ground method is calculated by subtracting the height value of the highest point in the tree point cloud from the height of the ground surface. The volume of the tree is computed by the stem curve. The stem curve is automatically derived from the point cloud, between the minimum height and height of the ground surface. A method to estimate above-ground biomass of large trees is by three-dimensional tree modeling of the MLS point clouds. To get accurate measurements, cylinder shape fitting using RANSAC is needed.

## Calculations of Decomposed Eigenvalues and Hough Transformation

Another key function of the program will involve using the coordinate values of points within a cloud to define relationships between them. These relationships can define a grouping of points as being linear, planar, or spherical in nature. To do so; the program will perform the following:

1. By first taking a group of points within a set distance of one another.

2. For a group of neighboring points principal component analysis (PCA) is performed to reduce the dimensionality of the points.
3. From the reduced dimensionality of the matrix eigenvalues will then be calculated.
4. Once eigenvalues are calculated; 3D shapes can be described by calculating linearity, planarity, and sphericity.
5. These values will aid in point classification to describe groups of points as being part of the ground, bole, branch, or other.

Once points have been classified a Hough Transformation will be performed on the points classified as bole in order to find the center and diameter of the tree, and can be done in the following steps:

1. At each point in a bole section a circle is traced around it.
2. Once all circles have been traced evaluate the overlapping circles to identify if at least 80% of circles all overlap the same point.
3. If less than 80% overlap the same point, increase the radius and retrace circles for each point and repeat step 2.
4. Once 80% of points overlap the same point, that point can be used as the center of the cross section.
5. After the center has been identified it can be used to locate the tree and the resulting circle can also be used for the diameter of the tree.

## Registration of Two Temporally Distinct, but Overlapping Point Clouds

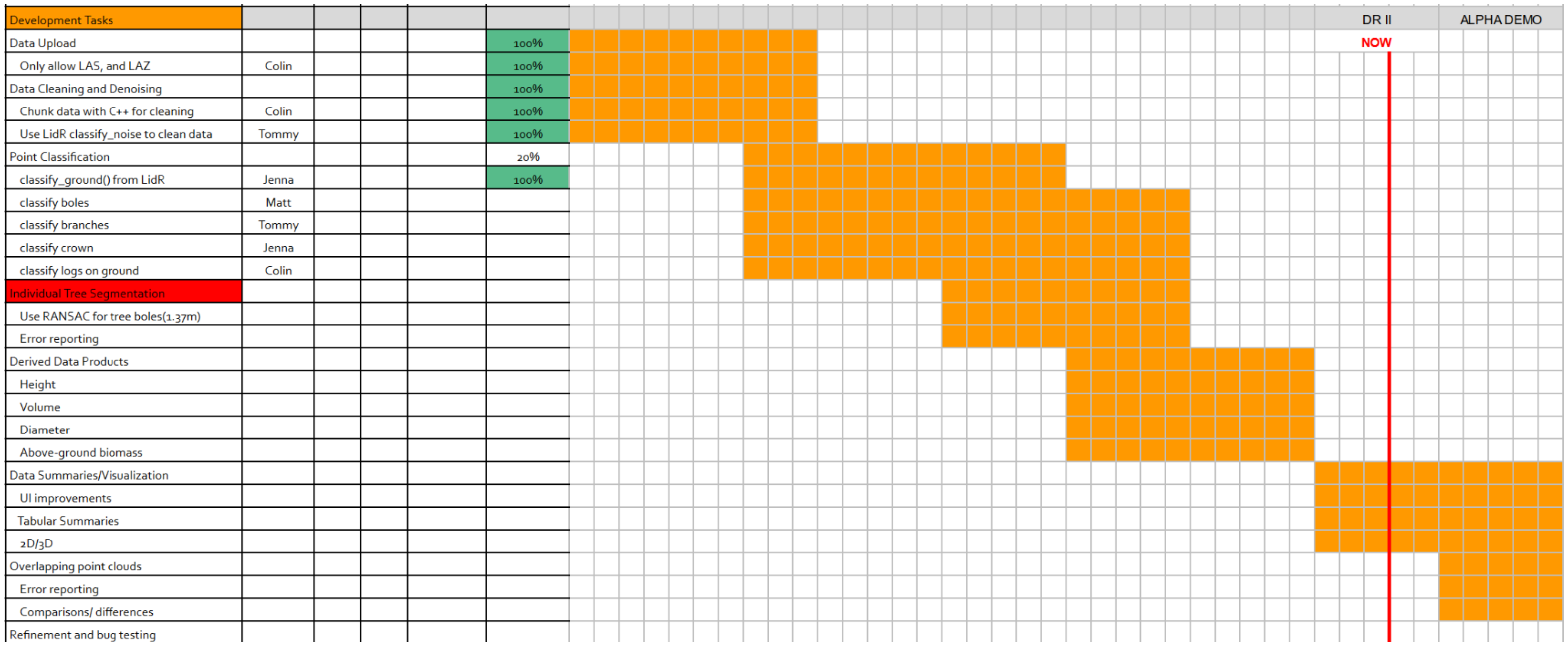
A final function will allow users to view and explore the changes that have occurred over time for two overlapping point clouds. To perform this the program will do the following:

1. The user will first visually align the two point clouds to the best of their ability by dragging and dropping the two point clouds onto one another in the Shiny interface.
2. Once point clouds have been roughly aligned; an iterative closest point (ICP) algorithm will fine tune the alignment of the two point clouds.
3. Then, taking the difference in results of the calculated ecological pertinent details of the two point clouds, a tabular summary can be constructed that describes the changes in the local environment. To be included in the summary table will be error reporting to take in account changing variables such as tree sway or RANSAC accuracy.

4. The resulting summary table and overlapping point clouds will be shown as output in the Shiny UI.

## Implementation Plan

Each major development task has been broken down into smaller tasks. The tasks have been organized in order on how it will be developed and laid out about a week or two apart and hopefully done before the Alpha Demo. The team has slowly started to split up some of the tasks for now, but are still open to collaborating for the most part. For the first couple of weeks, the team has completed 'Data Uploading' that only allows .las and .laz files. Then the second task that has been completed is 'Data Cleaning and Denoising' that needs to be chunked with C++ and use classify_noise from the lidR package. The team has also classified the ground with classify_ground also from the lidR package, segment trees, and slice the tree at 1.37m. The next two major tasks that the team will start to work on before Alpha Demo are creating an interface with C++ and RANSAC Cylinder Shape Fitting. With the C++ interface, the team will be able to load our point cloud R dataframe into a C++ object and so that it can later be explored back into R. Our client really wants the team to focus on using RANSAC for tree boles and have error reporting. Below is the team's Gantt chart with development tasks, and the red line is where the team is right now. Our app will streamline the mobile lidar workflow for forestry researchers and work towards improving forest ecosystem health.



| Development Tasks | | | | | |
|---|---|---|---|---|---|
| Data Upload | | | | | 100% |
| Only allow LAS, and LAZ | Colin | | | | 100% |
| Data Cleaning and Denoising | | | | | 100% |
| Chunk data with C++ for cleaning | Colin | | | | 100% |
| Use LidR classify_noise to clean data | Tommy | | | | 100% |
| Point Classification | | | | | 20% |
| classify_ground() from LidR | Jenna | | | | 100% |
| classify boles | Matt | | | | |
| classify branches | Tommy | | | | |
| classify crown | Jenna | | | | |
| classify logs on ground | Colin | | | | |
| Individual Tree Segmentation | | | | | |
| Use RANSAC for tree boles(1.37m) | | | | | |
| Error reporting | | | | | |
| Derived Data Products | | | | | |
| Height | | | | | |
| Volume | | | | | |
| Diameter | | | | | |
| Above-ground biomass | | | | | |
| Data Summaries/Visualization | | | | | |
| UI improvements | | | | | |
| Tabular Summaries | | | | | |
| 2D/3D | | | | | |
| Overlapping point clouds | | | | | |
| Error reporting | | | | | |
| Comparisons/ differences | | | | | |
| Refinement and bug testing | | | | | |

## Conclusion

Through the experience gathered by Dr. Sánchez-Meador as the executive director of the Ecological Restoration Institute, it has been found that there is a lack of efficiency in the current methods used to monitor forest health. Traditional methods typically involve groups of three individuals manually measuring trees, recording data, and then later uploading collected data to a computer for processing. This method is time consuming, labor intensive, does not provide a complete picture of forest structure, and the results are difficult to interpret. Other modern approaches that use lidar are implemented for the purposes of ALS monitoring where an aerial view is provided as opposed to a ground level view that researchers are familiar with.  These modern approaches to forest monitoring also require technical knowledge that can be challenging for non-technical individuals to use, or the required software is poorly optimized for processing data.

This project intends to solve these problems by providing a more efficient method of forest monitoring. First it will require less people for data collection by using MLS to create a high-resolution point cloud of the forest. From the point cloud, the project will then provide an easy to use interface via a Shiny app for the upload, cleaning, interpretation, and visualization of lidar data. From the functionality of the app, users will be provided with a tool that can effectively derive and display pertinent ecological data, as well as see ground level views of the forest structure all within an easy navigable web app. Once the application is in the hands of forestry researchers they will be able to streamline their data collection and be able to make informed decisions on how to treat and improve forest ecosystem health.