Final Report

May 5, 2022

Team LumberHack

Sponsor:

Dr. Andrew J. Sánchez Meador

Mentor:

Melissa D. Rose

Team Members:

Matthew Flanders

Jenna Pedro

Thomas Whitney

Colin Wood

Version: 1.0

Table of Contents

Introduction

Forest ecosystem health is at the center of many large-scale environmental problems that the world faces today. 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, while fire exclusion has left more combustible material on forest floors than would be natural. Add climate change to all of this, and you have a greatly increased risk of catastrophic wildfires, droughts, and many other ecosystem-disrupting phenomena. Ensuring that our forests are healthy is more important than ever.

Scientists and forest managers engage in a variety of tasks to restore forests or maintain healthy forests. One of the most fundamental steps in any such project is surveying, whereby researchers gain an understanding of a forest's structure, density, age, combustible material, and other derived information such as predisposition to wildfire. The scale of such projects ranges from a few acres of land to many square miles of forests. For larger projects especially, surveying can become resource intensive. Methods to automate surveys have been invented, notably the various LiDAR (light detection and ranging) tools. In essence, these technologies create physically accurate, three-dimensional representations of forests, and with the proper processing can yield the same measurements that a group of surveyors might collect, but do so more quickly and more cost effectively.

Dr. Andrew J. Sánchez-Meador, the sponsor of this project, is an Associate Professor at the School of Forestry, as well as the Executive Director at the Ecological Restoration Institute, both at Northern Arizona University. Dr. Sánchez-Meador's research focuses on using quantitative forest ecology to study forest ecosystems and guide restoration projects. His recent research applies the practical interpretation of complex data sets, such as those generated by LiDAR, to address important questions in forest ecosystem research. By using the remote sensing technologies discussed above, Dr. Sánchez-Meador and other ecologists are able to collect and interpret large complex data-sets in order to e.g. model individual tree growth or capture indicators of forest health.

Two roadblocks currently prevent LiDAR technologies from being used ubiquitously in the forest research community. The first is the simple fact that the data generated by LiDAR technologies is complex and vast. Much work has been done to develop algorithms to glean useful statistics from LiDAR points clouds efficiently and accurately. But there is a lot of progress yet to be made. In this project we implement algorithms to measure tree diameter and angle for all trees in a point cloud, using statistical methods to improve accuracy, and parallelization to improve performance. Because tree diameter and lean angle are important fundamental statistics to any forest survey, we thus address all three potential shortcomings: applicability, accuracy, and efficiency.

The second roadblock is the knowledge gap that exists between many forest researchers and what it takes to use many LiDAR processing tools. To use most of these tools a significant level of programming ability is required, which is simply not realistic for many researchers who specialize in ecology, biology, etc. To bridge this gap, a main goal of this project was to wrap an intuitive graphical user interface around the LiDAR processing tools in the program.

Process Overview

To develop the project, the team adopted a highly collaborative approach, both within the development team, and with the sponsor, Dr. Sánchez-Meador. During weekly team-only meetings, progress by each of the team members was discussed, and next steps were planned out. Tasks were divided among team members, but not to the extent that any one team member ever had to tackle an extensive amount of work alone. For more complicated tasks, two team members would collaborate directly. Every other week, the team met with Dr. Sánchez-Meador. In the beginning, these meetings were critical to bringing everyone up to speed on the many forest ecology and LiDAR technicalities. Throughout the project, because Dr. Sánchez-Meador has plenty of programming experience, he was able to assist with difficult hang-ups. A back-and-forth between development and then discussion and refinement with Dr. Sánchez-Meador was instrumental in the development process.

The life-cycle development process roughly followed the following steps. First, during requirements acquisition, the team spent time understanding Dr. Sánchez-Meador's research context and his goals of the project, while communicating our specific strengths and weaknesses in order to arrive at requirements that would be useful and achievable. Next, during design, the team researched the domain-level and functional requirements to arrive at plans for implementation. These plans were then successively refined using risk and feasibility analyses. Next, during development, the team sought to implement the key functional requirements in the order in which they would be required by a hypothetical user. Much of the back-and-forth with Dr. Sánchez-Meador occurred during this stage. Finally, during testing, all of the functionality involved in generating numerical output were tested to ensure accuracy. Unit testing was employed, with a focus on boundary inputs to the data processing functions.

The team used Github to collaborate on source code, and team members submitted pull requests for significant changes and new contributions. The team used Trello and Github issues to track and coordinate tasks.

Each team member shared an equal part in development, but when possible specialized where prior experience existed. In addition, each team member assumed a unique role with unique responsibilities. Colin Wood was the Team Leader, the responsibility of which was to coordinate task assignments and ensure work is progressing, run meetings, and make initial efforts to resolve conflicts. Thomas Whitney was the Website Manager, whose responsibility was to keep the Capstone website updated and make sure that the site was professional and easy to use. Jenna Pedro was the Release Manager, who coordinated project versioning and branching, reviewed and cleaned up commit logs for accuracy, readability, and understandability, and ensured that any build tools could quickly generate a working release. Matthew Flanders was the Architect, who was primarily responsible for ensuring that core architectural decisions were followed during implementation, and the Meeting Manager, whose responsibility it was to create meeting agendas and reserve locations and times for meetings.

Requirements

Requirements acquisition involved a collaborative process between the team and the sponsor, whereby Dr. Sánchez-Meador informed the team of his research context and desires for the project, and the team explained their technical strong suits and the capstone timeline. A list of functional, non-functional, and environmental requirements that found common ground were agreed upon.

The functional requirements were as follows:

- A Shiny application, which is an R-based web application framework, was to be implemented. Shiny excels in interactive data visualization contexts and Dr. Sánchez-Meador had previous experience with the software. Shiny simplified much of the user interface design by bundling various web technologies such as HTML, CSS, and JavaScript.

- A data upload ability. Researchers should be able to conveniently use their LiDAR point cloud data with the developed app. Multiple files should be uploadable.
- Data cleaning and normalization functionality. These steps are fundamental to all LiDAR processing pipelines, and as such should be implementable as the first step in our application.
- Tree segmentation. Before statistics can be computed for each tree in a LiDAR points cloud of a forest, individual trees must be segmented from one another.
- Tree characteristic measurement. This requirement was the most important of the project, and generates the data (e.g. number of trees, average tree diameter, location of each tree) that is useful to the user.

The non-functional requirements were as follows:

- An intuitive user interface. Because a key goal of the project was to make LiDAR processing accessible to the non-technical user, a clear and easy to understand interface to the program was important.
- Accuracy of tree statistics. This is a basic assumption of any user, and all else is rendered pointless if the reported statistics are inaccurate. See the process overview section for methods the team used to ensure accuracy.

Finally, the environmental requirements were as follows:

- The R programming language. The R programming language is used widely in the scientific community, and thus lends itself well to collaboration from other researchers. Also, the lidR package, from which our package borrows much functionality, is written in R.
- The GeoSLAM algorithm. GeoSLAM is a proprietary software that processes the raw lidar output from the mobile lidar machine. GeoSLAM must be run as the very first step in the processing pipeline, otherwise the lidar output is unusable. This restricts the pipeline in that all input to our application must have already been run through

GeoSLAM, or another proprietary tool if the mobile lidar data is from another manufacturer's mobile lidar scanner.

Architecture and Implementation

The conceptual architecture of the project is perhaps more difficult to understand than in the average case because it is intertwined with concepts from LiDAR processing that may be unfamiliar to a new developer. Figure 1 and the paragraph that follows attempt to combine explanations of the software architecture and the structure of a typical LiDAR workflow.

Figure 1: Key architectural components

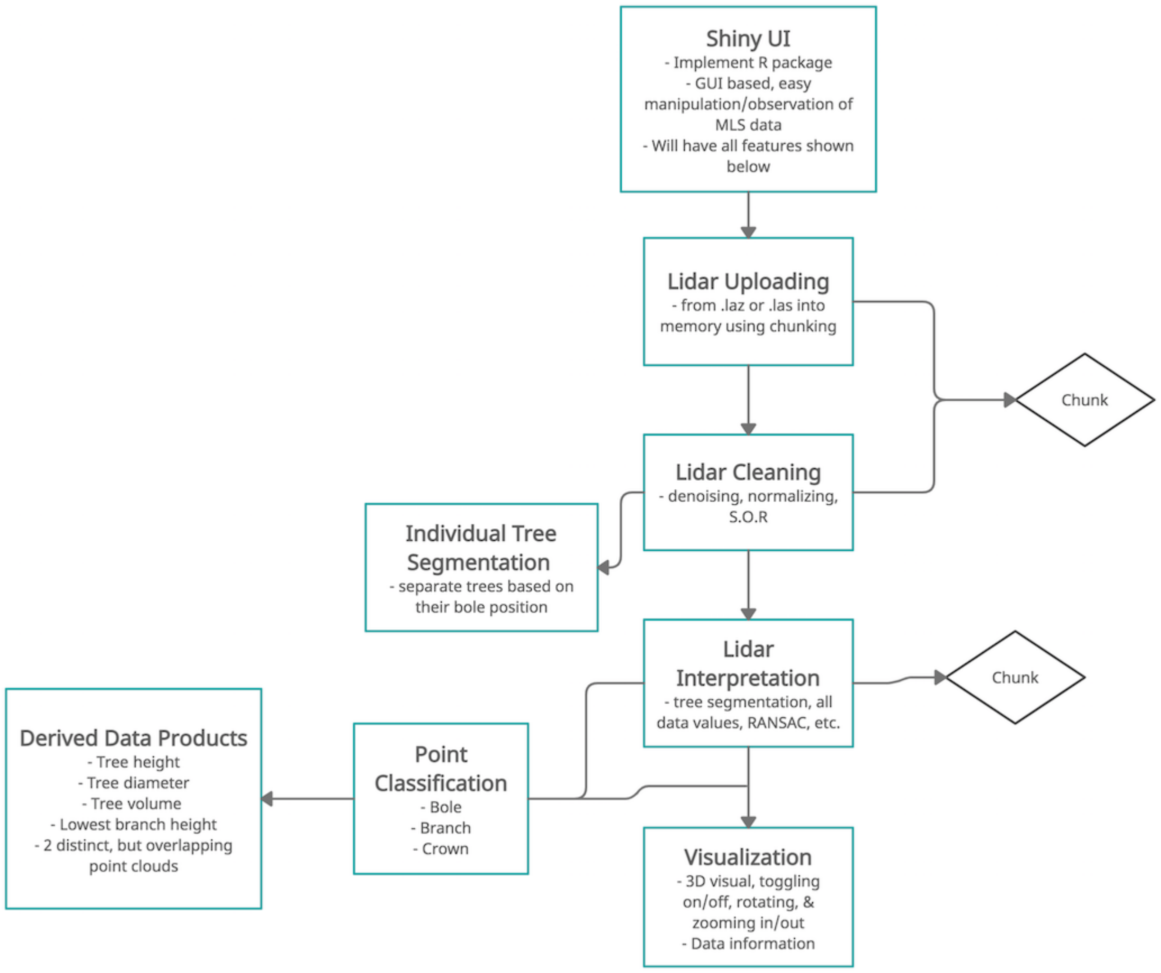
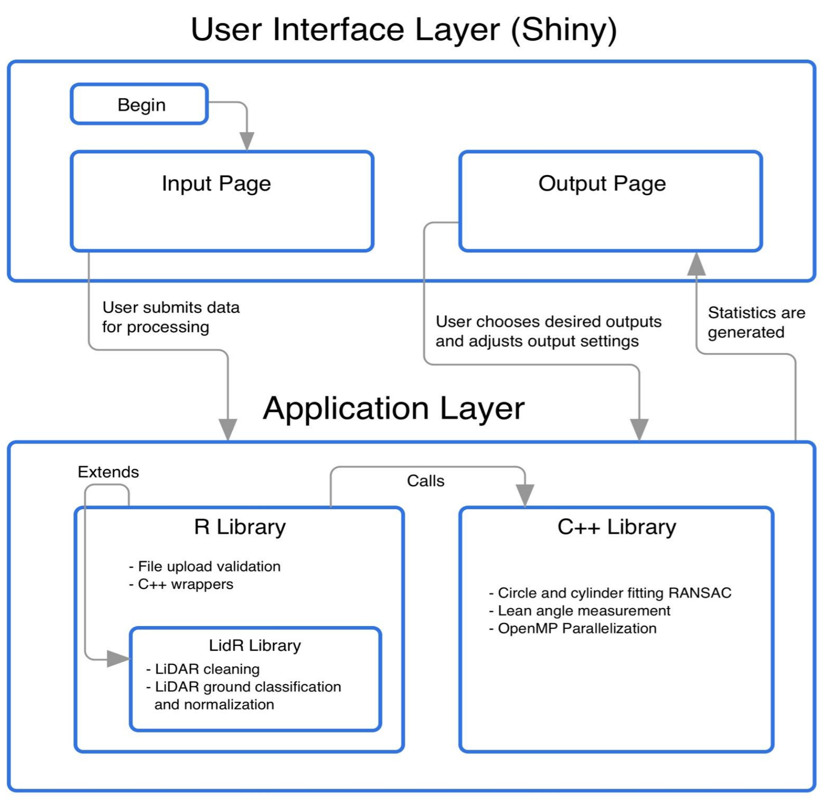

Shiny UI
- Implement R package
- GUI based, easy manipulation/observation of MLS data
- Will have all features shown below
Lidar Uploading
- from .laz or .las into memory using chunking
Chunk
Lidar Cleaning
- denoising, normalizing, S.O.R
Individual Tree Segmentation
- separate trees based on their bole 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

Figure 1 shows an overview of the project's main subcomponents and briefly lists the responsibilities and/or features of each. Beginning at the top, the Shiny UI is exclusively what the user will interact with. Thus it must handle all input and display all output. Inputs are accepted through the various pre-made options made available by Shiny and include file upload and text input. Shiny itself performs no processing; once data are collected by Shiny they are sent to one of the various R routines for processing. The Lidar Uploading box represents the first step in a user's pipeline and occurs when a user selects the file(s) they want to analyze. Once these files are uploaded, the next step is Lidar Cleaning, wherein the raw LiDAR point cloud data is cleaned and normalized. What exactly this means is relatively complicated and beyond the scope of this document, but a good resource is the following manual: https://r-lidar.github.io/lidRbook/index.html, specifically chapters 2 through 5. Once the LiDAR data have been cleaned and normalized, tree segmentation can begin. Once individual trees have been segmented from one another, point classification can begin, whereby subelements of the tree are detected. These elements may include the bole, branches, and crown, but at the time of writing, only an approximately 50cm slice at breast height (1.35m) is classified. This classified segment of each tree is passed to the Derived Data Products box, where the tree's diameter at breast height, (x, y) coordinate position, and angles of lean are calculated. These statistics are then passed back up to Shiny for display, here represented as the Visualization box.

With a basic understanding of the high-level structure, a more detailed explanation of the interaction between software subcomponents should be possible. Figure 2 shows the interactions at the application layer between software environments, and the communication between the application layer and the user interface layer. Beginning at the user interface layer, which consists solely of Shiny, there are in essence two views: the "input page", where a user uploads files and sets processing parameters, and the "output page", where the generated statistics are displayed to the user. The software pipeline thus begins once the contents from the input page are submitted for processing. This passes control to the R library, via simple function calling within the Shiny source code. The R library includes source code developed by the team and the lidR package, which is used to implement the LiDAR cleaning and normalization steps as described above. The lidR package may call C++ routines behind the scenes, but this is abstracted even from the developer; the outputs are validated once returned from lidR routines and there is no interference from the project's software in lidR's implementations. Once the cleaning and normalization steps have been completed, a DBSCAN (density based spatial clustering algorithm) implementation segments individual trees. An R function to create a csv file containing points and the tree ID to which they belong is then called, and the location of this file passed to the appropriate C++ routines. The R and C++ libraries interface via Rcpp, an R package designed to integrate the two languages. Rcpp documentation can be found here: https://cran.r-project.org/web/packages/Rcpp/index.html. Within the C++ library, various functions exist to fit circles to two-dimensional data and cylinders to three-dimensional data, with the goals of calculating trunk diameter, (x, y) position in the scan, and angles of lean for each tree. This situation lends itself well to parallelization, which the team planned to implement using OpenMP, but at the time of writing the C++ library routines were still single-threaded. C++ routines pass their return values directly back to the calling context in the R library, thanks to the Rcpp interface.

Figure 2: Layered design architecture



Testing

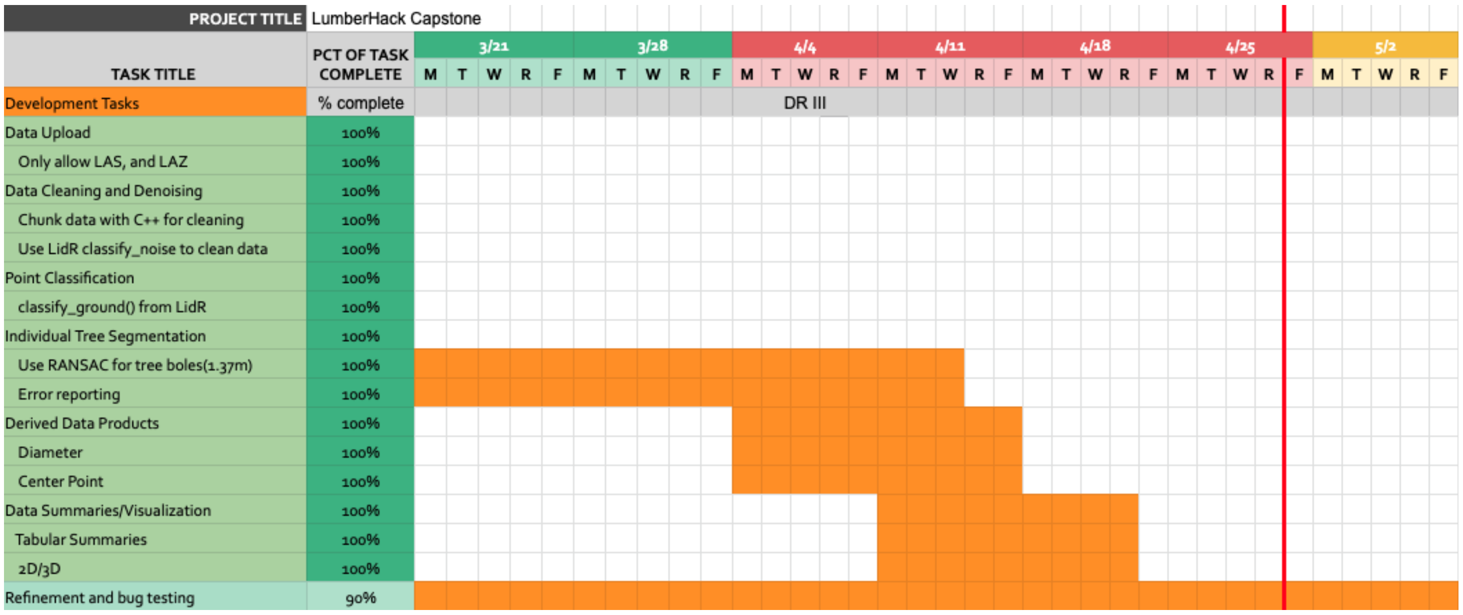
In order to meet specified requirements, software testing will be used. The first type of testing covered in this document is the unit test. Unit testing looks at individual functions of a piece of software to test the correctness of isolated chunks of code. In the context of this application, the team will be testing how the major modules of the application function and how each unit related to a module will be tested. Each of these units will be tested individually to ensure that they are working as intended. Our unit testing covers the main modules of our application including file uploading, data cleaning, data plotting, slicing the tree with dbscan, fitting circles using RANSAC and passing data to and from C++, and lastly the data summary results will be tested. By unit testing each of these, we are able to verify that they work on their own as they should. Next we will outline our integration testing.

Integration testing serves the purpose of ensuring that the major subcomponents of a software package interact with each other as expected. In this case, there are three major subcomponents. The first is the Shiny front-end, which is responsible for accepting input from the user, displaying output to the user, and driving the other two subcomponents. The second major subcomponent is the lidR package, which contains much of the functionality needed to parse, clean, and normalize the incoming lidar data. The third major subcomponent consists of the C++ library, which is responsible for various computationally intensive processes throughout the workflow. With our integration testing, it is important to verify that each of the major components of the project are working together and transferring data as needed. Lastly, we will outline our usability testing.

The purpose of usability testing is to ensure that end users of the product are able to effectively interact with the software and access its features. The goal of usability testing is to improve end user experiences by making changes to the user interface or by adjusting the workflow of the product so that it becomes more intuitive for the end users. The intended end users are forestry professionals, or forestry students, who are working with lidar data and may have experience working with other lidar specific software such as cloud compare or R packages such as lidR. While some of these users may be technologically proficient in working with lidar data the product aims to be straightforward enough so that end users with little experience are able to effectively interact with the product.

To evaluate the usability of the product the team plans to observe users as they interact with the user interface and then ask them to fill out a questionnaire at the end of the testing session. The questionnaire will have a section for each activity that will ask them to them rate their experience on a scale of one to ten as well as have them answer more open-ended questions where they will be prompted to elaborate on what they liked about the activity, what they did not like about the activity, and what they would like to see changed. The team is handing the application and usability testing documents off to our client to gain insight and information into how end users are using the product. Each of these testing strategies have been important to creating a future-proof, usable end product.

Project Timeline



PROJECT TITLE	LumberHack Capstone																																			
TASK TITLE	PCT OF TASK COMPLETE	3/21					3/28					4/4					4/11					4/18					4/25					5/2				
		M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F
Development Tasks	% complete													DR III																						
Data Upload	100%																																			
Only allow LAS, and LAZ	100%																																			
Data Cleaning and Denoising	100%																																			
Chunk data with C++ for cleaning	100%																																			
Use LidR classify_noise to clean data	100%																																			
Point Classification	100%																																			
classify_ground() from LidR	100%																																			
Individual Tree Segmentation	100%																																			
Use RANSAC for tree boles(1.37m)	100%	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧																	
Error reporting	100%	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧																	
Derived Data Products	100%											🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧															
Diameter	100%											🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧															
Center Point	100%											🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧															
Data Summaries/Visualization	100%																🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧											
Tabular Summaries	100%																🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧											
2D/3D	100%																🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧											
Refinement and bug testing	90%	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧

Throughout the semester, the team has worked through many major development tasks. The tasks have been split up into 6 major phases; data upload, data cleaning and denoising, point classification, individual tree segmentation, derived data products, data summaries and visualization, and refinement and bug testing. Last semester, the team mostly focused on learning about mobile LiDAR and Shiny. However, the team was able to accomplish data uploading where users can upload only .las or .laz file types up to 10GBs. This semester, the team started with working through chunking the data with C++ for data cleaning and using classify_noise method form LiDAR for denoising. The team then moved onto point classification, where classify_ground() was used from LiDAR. Lastly, individual tree segmentation with RANSAC and error reporting took the most time implementing. However, it has been accomplished as well as deriving data products on a table and having 3D visualization. As we are nearing the end of the semester, the team is starting to wrap up on refining and bug testing the web application.

Future Work

There were some originally envisioned features that were not implemented owing to the unforeseen complexity of complete LiDAR processing pipelines. More detailed point classification, whereby various parts of a single tree are separately classified, including tree bole, crown, trunk, and branches, was originally intended. At the time of writing however, only a slice of the tree trunk within the bole area is classified. Classification of the other segments of a tree would yield additional information relevant to forest research, such as tree health (relative crown sizes, broken branches), and predisposition to fire (low hanging branches, overlapping tree crowns).

Another envisioned but unimplemented feature is the longitudinal comparison of LiDAR point clouds. This would allow for the detection of change in important characteristics for both individual trees and entire forests over time. This would be useful to researchers in a variety of ways, such as monitoring young forests for healthy development or detecting increased risk of fire susceptibility over time.

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. More modern approaches that use lidar are implemented for the purposes of ALS monitoring where a top down view of the forest is provided and not a ground level view. These modern approaches to forest monitoring also require technical knowledge that can be challenging for non-technical individuals to use, or the software is poorly optimized for processing data.

This project has addressed these problems by providing a more efficient method of forest monitoring. First it will require less people for data collection by using mobile LiDAR to create a point cloud of the forest. From the point cloud the project can 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.

Appendix A:

- Hardware:
  - Package was developed in Windows and macOS platforms, with Windows being the primary development platform for developing parallel code using openMP.
  - Technical specifications during development: ram size of 32 GB and 4 CPU cores / 8 logical processors
  - Advised to have at least 16 GB of ram for loading of large las files and a leat a dual core processor for multithreaded operations. A CPU with 8 logical processors is recommended but this can be adjusted. Number of threads to use in parallel can be adjusted in ransac_circle_fit.cpp for scalability.
- Toolchain:
  - R and R Studio: required for running shiny apps and to build the package.
  - R packages: shiny, lidR, rgl, spanner, dplyr, Rcpp, rospca.
    - Shiny provides the web application interface
    - lidR provides functionality needed to load and process lidar data
    - Rgl provides 3d visualization of data
    - Rcpp allows C++ code to be developed used in the package
    - Rospca a robust version of PCA analysis that is used for point projections in ransac cylinder fitting

    - Rtools needs to be installed for package development using C++ code as it installs a compiler needed to compile C/C++ code.
- Setup:
    - Installation from .tar or .zip: if installing the package from a tar or zip file all that is needed is to open RStudio and on the toolbar navigate to Tools -> Install Packages. From there change the install from option to "Package Archive File (.zip; .tar.gz)". Once installed, load the package by calling library(mobileLidar) then calling the app() function to launch the shiny app.
    - For installing the package from source code, first download the repository from github: https://github.com/tommywhitney/mobile_lidar. Open the mobileLidar.Rproj file in Rstudio, be sure that Rcpp is being used by inputting "usethis::use_rcpp()" into the console, then document the project by moving to the Build tab in RStudio and selecting Document, alternatively the package can be documented using the shortcut "Ctrl+Shift+D". After documentation clean and rebuild the package by again navigating to the build tab and selecting Clean and Rebuild.

- Production Cycle:

Once setup is complete further development of the package can begin. To develop R scripts to be used in the app create a new R script file in the /R package directory and name it accordingly. Include documentation for the newly defined R script so that users can understand parameters and what the function does, see roxygen2 documentation for information on how to document. At a minimum "#' @export" must be included to properly build the package and include the new function. For development of C++ code, create a new C++ file in the /Src directory. Include //' @export and //[[Rcpp::export]] just above the function declaration to have the package build correctly and include the newly created C++ code. If code contains openMP pragmas to perform parallel operations include // [[Rcpp::plugins(openmp)]] above the previous two lines to allow the use of openMP compiler directives.

After code has been developed it can now be compiled into a package. First save all work and then restart the RStudio session. RStudio can be closed and reopened or it can be reset using the "Ctrl+Shift+F10" hotkey to restart. Once RStudio has reset and the package is open, document the build by moving to the Build tab in RStudio and selecting Document, or by using the shortcut “Ctrl+Shift+D”. Once the project is documented Clean and rebuild the package by navigating to the build tab and selecting Clean and Rebuild. After the package is built load all required packages by typing “devtools::load_all(".")” into the RStudio console or by using the hotkey “Ctrl+Shift+L” and then launch the app with app() to see the app and resulting changes to the build.

Once a new version of the package is ready to be distributed it can be shared through github and downloading the source code with the same steps outlined in the setup section above. Or it can be built as a source package or a binary package. A source package will result in a .tar.gz file that users can install in the same way that is outlined in the setup section above. A binary package is platform specific so building on Windows results in a .zip that Windows users can install, and building in macOS results in a .tgz that macOS users can install. Once files have been created that can be installed by opening RStudio and navigating to Tools -> Install Packages and then selecting to install from “Package Archive File (.zip; .tar.gz)” and selecting the appropriate file.