Technological Feasibility Analysis

November 1, 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
1.1 Overview	3
1.2 Problem	3
1.3 Solution Vision	4
Technological Challenges	4
Technological Analysis	4
3.1 Lidar Data Upload	4
3.2 Cleaning the Data	8
3.3 Data Parsing	10
3.4 Data Visualization	13
Technology Integration	16
Conclusion	17
Works Cited	19

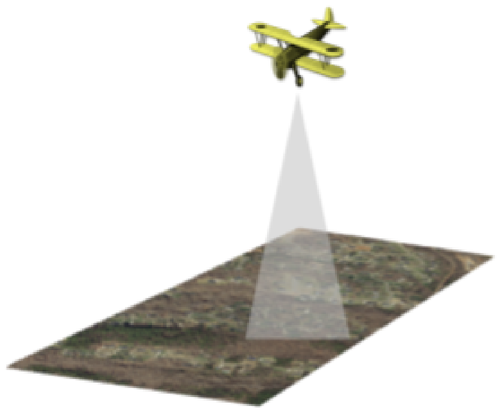
Introduction

Overview

Forest ecosystem health is at the center of many large-scale environmental problems that the world faces today. Over the last century of human expansion and development, 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, droughts lasting longer and happening more frequently, as well as invasive species taking over forests. With climate change advancing, forest ecosystems are in worse shape than ever before. Scientists and forest managers work on restoration projects and look for ways to increase the pace of these efforts with little time to plan and consider the long term effects they may have.

Problem

Due to how large forests are, observing single trees can be a lot of work. Light Detection and Ranging or lidar has been one of the most useful resources for gathering data over large areas of forests. Lidar is a remote sensing method that uses light pulses combined with GPS data to create precise, three-dimensional information about the shape and characteristics of what is being scanned. Airborne lidar has been the traditional method for surveying. A lidar sensor is attached to an airplane or helicopter and allows for a large area of land to be surveyed. Typically with airborne lidar you get more land coverage than other forms, but the number of data points is low and the forest is viewed from above. With mobile lidar, data is taken from the forest floor. The sponsor for the project, Dr. Andrew J. Sánchez Meador works with mobile lidar to get forest data as seen from the perspective of someone walking through the forest. Current methods for analyzing lidar data and pulling useful information from the data are


Example of airborne LiDAR

resource intensive and hard to use. These methods are designed for airborne lidar and have not been able to quickly and effectively analyze mobile lidar data.

Solution Vision

This project will serve as a resource for researchers and ecologists to analyze their own mobile lidar data. The team plans to create a web application that allows users to upload lidar data which will be processed and then return useful information on the data. Useful information includes tree height, tree diameter, lowest branch height, crown size, and the ability to distinguish trees from smaller shrubbery. These statistics provide important information for ecological restoration projects and are important in maintaining healthy forests.

In the early stages of this project it is key to identify technological challenges and candidates to address them. Candidates will be identified and analyzed in their ability to address the technological challenges of the project based on desired characteristics. In the following sections each of these challenges will be discussed in depth. At the end of each subsection in *Technology Analysis* a candidate will be selected to address a technological challenge in the and explained why it is considered to be the best choice.

Technological Challenges

- Capability for users to upload lidar data
- Capability for users to clean the data uploaded
- Capability for users to parse useful information such as ecological attributes (i.e. Tree height, tree diameter, lowest branch height, crown size, etc.) from the raw lidar output
- The project will need a way to allow users to visualize their data with useful characteristics (i.e. Tree height, tree diameter, lowest branch height, crown size, etc.)

Technology Analysis

Lidar Data Upload

Lidar data files can range from a few hundred Megabytes to a few Gigabytes in size, therefore data must be able to be loaded in a way that won't slow down performance. The loading of data needs to be efficient and scalable for different hardware resources available such as CPU core count as well as GPU core count and memory size. A strong candidate to address this challenge would need to be fast at loading data, be able to chunk the data into smaller portions and only load in what is needed if necessary, as well as utilize multiple cores on the CPU and provide GPU support.

Candidates

C++

C++ was designed by Bjarne Stroustrup in 1985 to address the features not present in the C programming language and the lack of performance in programming languages that had these features. With these two features in mind Stroustrup was able to create C++, an extension of C that has the performance features present in C with the additional features such as classes.

C++ is fast and efficient due to being a compiled language, and has a low performance overhead by allowing the compiler to assume that the programmer has taken care to address possible safety concerns, such as memory leaks and proper memory allocation. C++ supports multithreaded programming that can speed up performance as well as GPU support that can allow for greater memory and processing performance. Some downsides to using C++ can be the difficulty in implementing code and the need to build extra features from scratch as needed. Library support for the loading and processing of lidar data is somewhat limited, the two main libraries found are libLas and LASTools. For these reasons C++ is one of the candidates that will be evaluated in lidar data uploading.

R

R was developed by Ross Ihaka and Robert Gentleman in 1993 where it focuses on the statistical analysis of data as well as the graphical representation of the results. With the statistical support that is provided it is commonly used by statisticians in the field of academics.

R has strong data management capabilities that allows for effective reshaping of data, this makes it a good candidate when handling large data sets. R also has great community support that has produced numerous packages that are able to load and process lidar data. Some of these packages are notability lidR and liDAR, but there are many others as well . With R being an interpreted language performance can be an issue, but this can be improved by installing the Rcpp package that will allow code to be written and compiled in C++ which can increase the performance. The speed increase of using C++ does bring with it some of the downsides of programming with C++ as mentioned before. In addition to the data management and performance increasing packages in R there is also support for parallel programming and GPU utilization to support varying hardware systems that the software will run on. For these reasons R is in the candidate pool to be evaluated on the loading of lidar data.

Python

______First released in 1991 by Guido Van Rossum, Python is an interpreted language that places greater emphasis on ease of use for programmers rather than the performance of the software. Being an interpreted language it does have some disadvantages with regards to performance, but with many libraries and plug-ins available there are options available that can improve the performance of Python. One such option is Cython, with this library code is able to be written normally using Python and then compiled in C. This compilation step and other features allows for C++ or C code to be developed and executed at any time within Python code. In addition to the libraries and plug-ins available that improve performance there are libraries that support the loading and processing of lidar data as well. As with the other candidates Python is also able to run in parallel on multiple CPU cores and utilize the GPU to improve performance based on the hardware in the system. With Python being capable of loading lidar data, scaling with CPU and GPU hardware, and its performance capabilities it is the third candidate to be included in the analysis of lidar data uploading.

Analysis

To see what candidate may be the best choice a sample mobile lidar file was downloaded from the Geoslam website. Each programming language is evaluated on how fast it can load the sample LAS file and how easy it is to install needed packages and write code. The characteristic that is valued most in solving this technical challenge is speed so it will be given a weight of 0.8 in the total score and ease of use will be given a weight of 0.2 in determining the total score.

C++ was difficult to set up and install the required libraries, there was a general direction in the documentation for set up and install but seemed outdated with broken links and outdated information for the libLas library. The LASTools library was easier to set up and provided a toolset that was easy to use. Due to the difficulty of installation and ease of use C++ scores a 1 for this criteria. When loading the sample file C++ was able to load the data what seemed almost immediately and then take about 30 seconds on average to visualize the point cloud. With the speed of loading the data C++ scores a 5. Using the weighted scoring system gives C++ a total score average of 4.2.

R and RStudio installation was fast and easy with the use of an installation wizard, and package installation was also easy with the built-in package manager in RStudio. Writing the code was fast and easy with examples provided in the package vignettes. For these reasons R scores highly in the ease of use criteria, receiving a score of a 5. Using the lidR package to load the lidar data took a significant amount of time. With R taking on average 1 minute to load the data it receives a score of a 2 for the speed and efficiency criteria. From the scoring weights R receives a total average score of 2.6.

Python was also easy to set up and program with easy to follow installation wizards, and package managers such as pip to install the needed packages. Programming Python code was simple to do as well. For these reasons Python scores highly for ease of use receiving a score of a 5. Using the laspy package to load the sample las file it took Python on average 7 seconds to load the data. For this reason Python scores well in speed and efficiency, receiving a score of a 4.

Chosen Approach

The table below outlines the results of each candidate and how well they scored on the desired characteristics needed. With speed and efficiency being given a weight of 0.8 and ease of use being given a 0.2 in determining the final score.

	Speed / Efficiency	Ease of use	Average Score	Ranking
C++	5	1	4.2	1
R	2	5	2.6	3
Python	3	5	3.4	2

Reviewing the candidates performance on the desired characteristics it would appear that the C++ programming language is able to accomplish both speed and scalability better than R and Python. Although it brings challenges in setting up and implementing code, the speed benefits outweigh the difficulty in programming and setting up.

Cleaning the Data

Once the lidar data is uploaded and ready to work with, the next step is to clean the data. Cleaning the data is an important step in ensuring that the results of an analysis are accurate. The first step in cleaning is to denoise the data. This includes looking for outliers or other non normal data points. For example, a lidar scan could accidentally scan a bird that flew by and the data may include points that affect the accuracy of other data points. These inaccuracies must be accounted for and removed. The next step is to run a statistical outlier removal filter to remove outliers and other noise. The data after this point should be cleaned and normalized in a way that will allow for further analysis to be accurate.

Candidates

When choosing the best candidate it is important for it to meet specific criteria. Most importantly, the data must be able to be denoised and cleaned with the software. The cleaning process should also be able to be done quickly and efficiently. It is important that the candidate can process a lot of data at a time without needing a lot of user interaction. Maintainability is another important factor that will help in deciding which candidate is the best fit.

CloudCompare

CloudCompare was developed in 2003 by Daniel Girardeau-Montaut in collaboration with Telecom ParisTech and the R&D division of Électricité de France. It is a 3D point cloud processing software that can handle an endless amount of scalar fields per point cloud. To visualize per-point scalar fields in an efficient way, there is a dynamic color rendering system to help users. It is a popular piece of software for research on forestry related ecological projects.

CloudCompare has a built in user interface that allows the software to be learned quickly. CloudCompare was tested using sample lidar data and its built-in cleaning functions. CloudCompare has a statistical outlier removal function that will remove points when provided with certain criteria for determining outliers. It also includes a filter to reduce noise using a nearest neighbor algorithm. CloudCompare does a very good job at cleaning and denoising lidar data, however it lacks in speed due to the user having to manually clean their data with these functions. There is no automation in CloudCompare.

<u>LidR</u>

LidR is a package developed by Jean-Romain Roussel with contributors such as the sponsor of the project, Dr. Andrew J. Sánchez Meador. LidR is an R package for visualizing and analyzing airborne lidar data. The lidR package has many powerful tools for reading and displaying lidar data and computing useful information from the data. The package is currently being maintained and is still receiving updates.

For the purposes of cleaning lidar data, lidR has many useful built in functions. The lidR package implements its own noise segmentation algorithm, *classify_noise*. This function works similarly to the CloudCompare tool by identifying outliers using a statistical outlier removal method. When testing the function with sample lidar data, it works similarly to CloudCompare and outputs similar results. The cleaning and denoising works well and runs quickly while also allowing for easy automation. Automation will be important for speeds when processing lots of data. In terms of maintainability, lidR has lots of documentation with examples to assist in learning the functions of the package.

Analysis

To choose a best candidate for cleaning lidar data, the important criteria to consider are how well the software cleans/denoises, the speed/efficiency of the software, and the maintainability of the software. For cleaning/denoising, CloudCompare has built in tools such as Statistical outlier removal and a filter noise tool. These tools are built into a GUI that allows for easy user interaction. The cleaning works well in the software and outputs a denoised point cloud as expected. In terms of speed/efficiency, CloudCompare works well for processing one set of data but is not very expandable to larger sets of data. The software is quick for one data set but is lacking in efficiency due to not having the ability to automate cleaning. For maintainability, CloudCompare is open source however learning and understanding the back end code would be a large, time consuming task.

The other candidate to compare against the criteria is the lidR package. For cleaning/denoising, lidR includes the function *classify_noise* that works similarly to the built in CloudCompare tool. It works by classifying outliers and removing those points from the point cloud to create a clean set of data. The function works well and returns a cleaned point cloud with outlying points removed. Next is the speed/efficiency of lidR. LidR works quickly and was able to clean a sample data set in a few seconds. LidR allows for the cleaning process to be automated which is important for scalability and efficiency when working with multiple data sets. It can be programmed to clean multiple sets of data with little user interaction needed. Last is maintainability. LidR includes lots of documentation that is built into the package. Each function includes examples of how it can be used and what the output will look like. There will still be a bit of a learning curve, however the documentation will assist greatly.

Chosen Approach

Criteria	Cleaning/ Denoising	Speed/ Efficiency	Maintainability	Average Score	Ranking
CloudCompare	5	3	2	3.33	2
lidR	5	5	4	4.66	1

The table above outlines how each candidate fits each criteria on a scale from 1-5. At this stage in the project, the team believes that the lidR package will be the best tool for cleaning lidar data after it has been uploaded. It does a great job in cleaning and denoising data with its built in functions. These functions are fast and allow for scaling the project to allow for multiple lidar data sets to be processed at a time. The documentation behind lidR is a great resource for learning the package and being able to maintain the code in the future. The lidR package fits the criteria best and will be used for cleaning lidar data in this project.

Data Parsing

Once the raw lidar data is fully processed, a method of parsing it into a meaningful representation is necessary. This step is perhaps the most crucial in the entire pipeline, because it will be computationally intensive, involve novel algorithms and strategies, and its output will constrain all downstream interpretation of the lidar input, including any quantitative data and data visualization. The technologies involved in this step are also the most likely to be changed, in part or in full, as the project progresses and the team gains insight into the strengths and weaknesses of each technology.

These technologies will need to extract, for each tree in the sample, its coordinate location, height, diameter, angle relative to ground, lowest branch height, and crown size. Given these primitive data, more complex information such as forest density, or estimates of forest age or wildfire risk, for example, can be determined. These data will additionally be used to construct 2-D or 3-D model visualizations which will be displayed in the web application.

Candidates

Currently, the team recognizes three possible ways to accomplish these goals, or at least a subset of them. The first involves using CloudCompare, a 3-D point cloud processing suite that is well established in the field of lidar visualization, and has a multitude of analysis options. The second would be an extension of the lidR software package, an open source tool written in R but which was designed specifically for airborne, not mobile, lidar data. The third possibility is to forgo reliance on either of these tools, and come up with a solution mostly from scratch, possibly reusing small portions of code from relevant open source projects.

Analysis

The CloudCompare software suite is polished, reliable due to its large number of users, and offers both a graphical user interface and an application programming interface. Although it is an open source project and plugins are allowed to be contributed, there is no guarantee that a developed plugin will be officially accepted into the software, meaning the team’s work might not eventually be usable by researchers. The plugin creation process has somewhat strict guidelines, although because CloudCompare is widely used and actively maintained, bugs in the plugin system should be minimal. Creating a CloudCompare plugin is likely to be the second most efficient option. This is because, although CloudCompare has well-optimized processing tools, any integration into its ecosystem will incur significant overhead.

The lidR software package is smaller scale than CloudCompare, but well documented and actively maintained. It lacks a sophisticated graphical user interface for testing and experimenting, but does offer certain built-in visualization tools. The project is accepting of outside contribution, and also has an established plugin workflow, which is likely to be more flexible and amenable to the project development than CloudCompare's, given lidR's more research-focused context. This approach would likely result in the most efficient solution: LidR already has many necessary tools, but being a smaller package than CloudCompare will have less overhead and be more nimble with developmental additions. A further plus here is that the project's client co-authored the lidR package and would be a valuable resource for questions concerning the internals--a resource not available with CloudCompare.

Development from scratch, the third option, does away with the central process of the other two approaches, namely plugin development. This has upsides and downsides. An upside is that there will be no constraints imposed by a plugin development framework which might make some requirements difficult or impossible to obtain, or constrain the efficiency of development. The obvious downside is that, without such a plugin framework, development might become too scattered or get bogged down in boilerplate implementation. This approach is likely to result in the least efficient implementation. Although there are no constraining factors, given the complexity of the problem, optimization may be put on the back burner while the team simply focuses on achieving a working solution.

Chosen Approach

Criteria	Ease of Development	Efficiency	Effectiveness	Average Score	Ranking
CloudCompare	4	4	3	3.67	2
LidR	4	5	4	4.33	1
De novo	2	3	4	3	3

At this preliminary stage, it is apparent that an extension of the lidR package is the best goal moving forward. In reality, some combination of all three of the above strategies might be implemented. For example, a plugin to the lidR package that involves significant from-scratch development of algorithms and makes various calls to the CloudCompare API for existing features.

Data Visualization

Once the team pulls useful information from the data, finding a way to allow users to visualize data with useful characteristics is the last step. For example, how are users going to connect visualization with the team's web application? Once the team has pulled information from the data like ecological attributes (i.e. Tree height, tree diameter, lowest branch height, crown size, etc.), the product will then be able to visualize the point cloud, the resulting tree attribute information, and allow the user to interact with the data and statistics. For the team's ideal solution, the following are the key characteristics of the project. First, once the data has been processed the results need to be easily understandable. Second, the team would want to have an effective and simple to use interface with 3D visualizations. Lastly, be able to interact with the visualization such as rotating, toggling on or off, and zooming in or out.

Candidates

CloudCompare

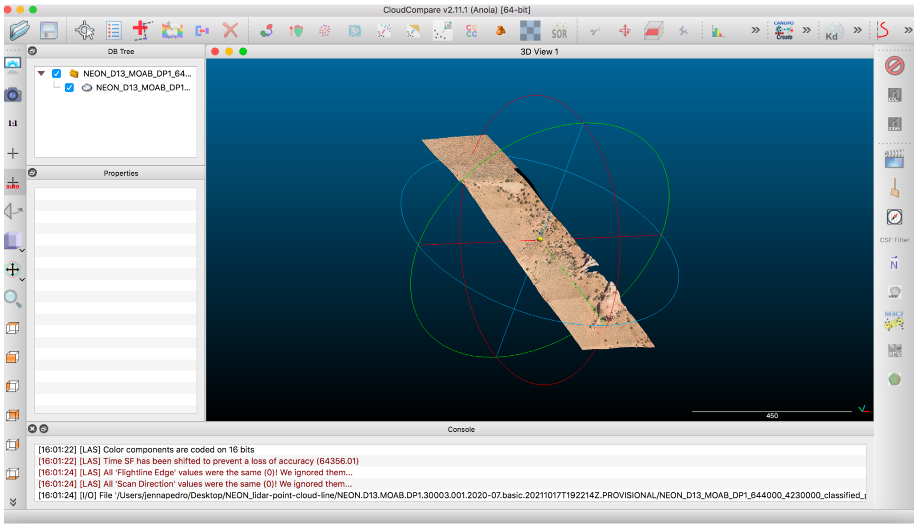
To address this issue, the team have three possible approaches which are CloudCompare, Unity, and Unreal Engine. The first candidate for consideration is the software CloudCompare. CloudCompare is an open source 3D point cloud processing and editing software that has been a recommendation from the team's mentor and client for viewing lidar data. CloudCompare has an interactive user interface that allows for easy manipulation and formatting of point cloud data and is used by researchers in ecological projects, archaeologists, and foresters.

Unity

Unity was released in June 2005 and developed by Unity Technologies. It is a game engine that can be used to create 3D and 2D games, as well as interactive simulations and other experiences. For 2D visualization, Unity uses advanced 2D world renderer and importation of sprites. For 3D visualization, Unity allows mipmaps, texture compression, resolution settings for each platform that the game engine supports, full screen processing effects, and many other features. Besides using it in the video gaming industry, it can also be used in film, engineering, construction, automotive, architecture, and U.S Armed Forces.

Unreal Engine

Unreal engine first launched in 1998 as the first-person shooter game in Unreal and was developed by Epic Games. It is a creation tool used for game development, 3D visualization, linear film and television content creation, automotive and architectural visualization, broadcast and live event production, training and simulation, and other real-time applications. Other than video games and film, unreal engine can also be used by non-creative fields because it has many features available and for their virtual reality tool to explore graphics and other visualizations.



```
[16:01:22] [LAS] Color components are coded on 16 bits
[16:01:22] [LAS] Time SF has been shifted to prevent a loss of accuracy (64356.01)
[16:01:24] [LAS] All 'Flightline Edge' values were the same (0)! We ignored them...
[16:01:24] [LAS] All 'Scan Direction' values were the same (0)! We ignored them...
[16:01:24] [I/O] File '/Users/jennapedro/Desktop/NEON_lidar-point-cloud-line/NEON.D13.MOAB.DP1.30003.001.2020-07.basic.20211017T192214Z.PROVISIONAL/NEON_D13_MOAB_DP1_644000_4230000_classified_
```

Analysis

The team's criteria in choosing which candidate fits best for the project are speed and efficiency, having a learning curve, security, and maintainability. The first criteria the team wants to check for in each candidate is speed and efficiency. To check the speed and efficiency of each candidate, the team downloaded a sample lidar point data set, downloaded CloudCompare, Unity, and Unreal Engine, and uploaded that data sample. When using CloudCompare, the team opened the .laz sample data and it took about 2 seconds to upload the data alongside its visual. When using Unity and Unreal Engine, it took quite a while to install the plugins and also a lot of storage. In Unity, the user would have to get .pcx and .ply files and this entire process took about 10 minutes to complete. In unreal engine, the user has to download the lidar point cloud support extension and its supported .laz files. It was quite fast to open the file, but took quite a while to get the lidar support plugin.

The second criteria the team wants to check for in each candidate is if it has a learning curve. To test this criteria, the first step was to open each software and compare their tools and the ease of finding their usages. CloudCompare was very easy to use, the user just opens the data file that wants to be seen from the device and it pops up. Although there are many tools around, the user can just place the mouse over that button and it says what it is. Unity is quite difficult to use for beginners because users have to search around the program to see what it does. Unreal engine is also difficult to use for beginners because again the tools aren't in one place to play around with, rather it has to be searched for in a content browser.

The third criteria the team wants to check for in each candidate is its security. Security is important because some researchers don't want to openly share their lidar data. To check this, the team dug into the security features that each candidate provides. As of right now, CloudCompare has no documentation of any security features on their website besides their CloudCompare forum. However, CloudCompare is based on the files the user uploads from their own device and doesn't need to make an online account unlike Unity and Unreal Engine. When using CloudCompare, security is based on the user's own device. Unity developed security practices and tools to maintain a high-level of security for their customers. They have made a public version of Unity's internal Secure Software Development Lifecycle(SSDLC), which provides a reusable reference framework that others could contribute to and help to improve it. Epic Games' Unreal Engine has a privacy policy where they provide security services for any of their websites, stores, applications, game developer tools, and other products.

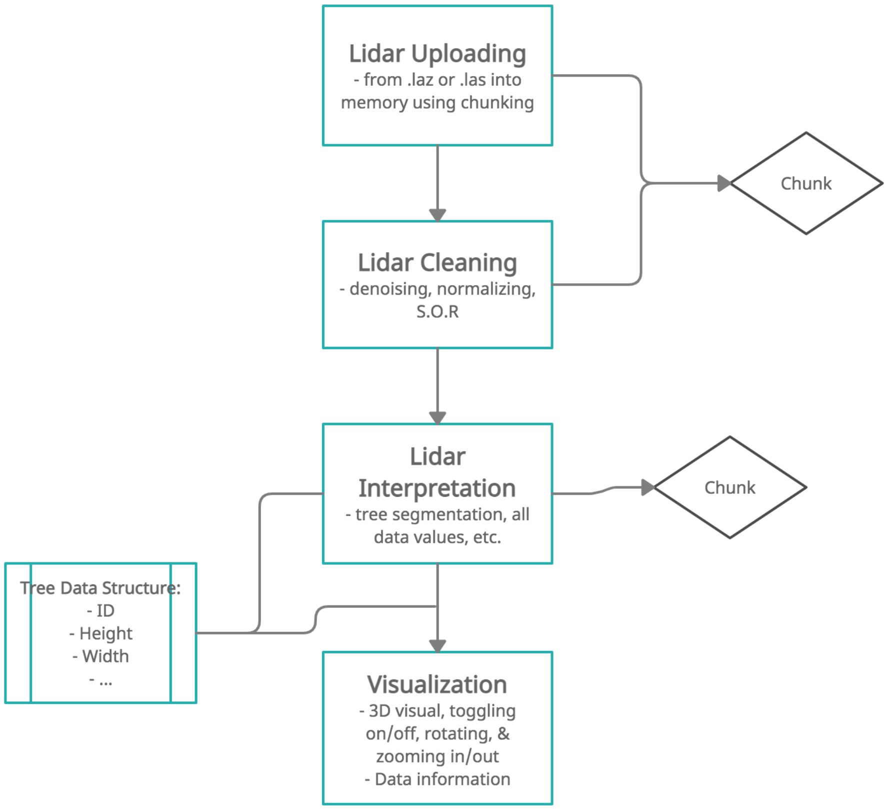
The team's last criteria is maintainability. How well could users understand the software or code 5 years from now? For CloudCompare, users have to upload their own files to view it and the tools are very easy to use. In order to maintain it, just organize files on the device and it will still be easy to view it in CloudCompare. When first getting started on Unity and Unreal Engine, users have to create an account. So, to ensure users could still see their data or code later, they have to ensure that they remember their account. Users could also create different projects on it and have that data organized based on what project it is.

Chosen Approach

Criteria	Speed/ Efficiency	Learning Curve	Security	Maintainability	Average Score	Ranking
CloudCompare	5	5	3	3	4	1
Unity	3	2	5	5	3.75	2
Unreal Engine	1	2	5	5	3.25	3

Above is a table that shows the team's desired characteristics for this challenge, the candidates that have been nominated, their score ranging between 1 and 5, and their rankings. The candidates are then ranked based on their pros and cons to choose which one will best "solve" the specific challenge of data visualization. This leads us to choose CloudCompare as it is the most promising solution. Although it may lack assurance with security, it is all based on the user's device(e.g virus, security, etc.) and all the pros outweigh the rest of the candidates with it being fast and efficient, easy to use, and maintainable.

Technology Integration


Lidar Uploading
- from .laz or .las into memory using chunking
Chunk
Lidar Cleaning
- denoising, normalizing, S.O.R
Lidar Interpretation
- tree segmentation, all data values, etc.
Chunk
Tree Data Structure:
- ID
- Height
- Width
- ...
Visualization
- 3D visual, toggling on/off, rotating, & zooming in/out
- Data information

In order to build the system and implement all of the requirements, the team will need to bring each of the technologies discussed above together. The project consists of four large pieces that will need to be able to work together to create a fully functioning piece of software. The following diagram illustrates how all of these pieces will come together.

The four conceptual steps will be integrated as follows. Users input their lidar files into the team's R package which runs on their local machine. Because the resulting data is memory intensive, this process occurs in chunks, allowing a little at a time to be processed. Thus, Step 1: Uploading directly interfaces with Step 2: Cleaning. Once these two steps are finished computing in tandem, a fully cleaned and denoised lidar data set exists on the user's local machine. Next, the user passes this dataset into Step 3: Processing. This also occurs via chunking. Tree segmentation is performed and the relevant statistics for each tree (height, diameter, coordinate position, etc.) are computed. This parsing results in a collection of data structures, each representing a tree in the sample. The data structures are then passed to Step 4: Visualization. Visualization involves migrating the processed data into the web application, reporting to the user all relevant computed statistics, and their aggregations, as requested/specified by the user. Either a passive or interactive (e.g. 3D) graphical representation will also be presented within the web application at this step.

Conclusion

For forest researchers to keep up with the rapid pace of restoration work that is necessary to maintain and protect forest ecosystems, a more efficient and effective means of processing and understanding the collected data is desirable. After assessing the project needs the team identified challenges in data loading, cleaning, parsing, and visualization. For each of these challenges a pool of candidates was identified that were able to address the challenges and then evaluated based on the outcome of performance of the desired characteristics.

For data uploading it was found that C++ was the best candidate due to the speed at which it was able to load sample data. For the challenge of data cleaning it was found that the lidR package within R was the best choice, with how well it was able to clean the data and with how fast it performed it would be the chosen candidate in cleaning lidar data. Next in the pipeline is data parsing, where it was found that extending upon the built in lidR package functionality is the best candidate due to it being highly flexible, easy to use, and fast and efficient in parsing data. Lastly for the challenge of data visualization CloudCompare was found to be the most qualified candidate with high scores in ease of use and speed and efficiency. Using these candidates does look to be the most promising in addressing each challenge. Moving forward each candidate will continue to be explored and evaluated as the project grows into a complete solution.

Works Cited

- Airborne Lidar. (2020, January 17). *GeoSpatial World.* Retrieved October 17, 2021, from https://www.geospatialworld.net/blogs/do-you-know-types-of-lidar/.
- Roussel, Jean-Romain. "LidR: R Package for Airborne Lidar Data Manipulation and Visualisation for Forestry Application." *GitHub*, https://github.com/Jean-Romain/lidR.
- Wikimedia Foundation. (2021, June 6). *CloudCompare*. Wikipedia. Retrieved October 17, 2021, from https://en.wikipedia.org/wiki/CloudCompare.
- Technologies, U. (2021, October 5). *Wondering what Unity is? Find out who we are, where we've been and where we're going*. Unity. Retrieved October 17, 2021, from https://unity.com/our-company.
- *Frequently asked questions*. Unreal Engine. (n.d.). Retrieved October 17, 2021, from https://www.unrealengine.com/en-US/faq?active=general.
- NEON (National Ecological Observatory Network). Discrete return LiDAR point cloud, RELEASE-2021 (DP1.30003.001). https://doi.org/10.48443/6e8k-3343. Dataset accessed from https://data.neonscience.org on October 17, 2021
    - Data set used to test criteria
- Technologies, U. (n.d.). *Security*. Unity. Retrieved October 17, 2021, from https://unity.com/security.
- *Please read the epic games privacy policy*. Epic Games. (n.d.). Retrieved October 17, 2021, from https://www.epicgames.com/site/en-US/privacypolicy.