Final Product Report

8 May 2020
Version 1

Sponsors

Dr. Kiona Ogle

Dr. Michael Fell

Mentor

Isaac Shaffer

TreeViz

Riley McWilliams
Qi Han
Haitian Tang
Daniel Rustrum
Alex Bentley

Contents

1 - Introduction

1.1 Background

Climate change is a natural cycle of the Earth that impacts many ecosystems. However, due to the recent and rapid growth in industrialism, climate change is now being more affected by artificial causes than natural ones. A severe increase in the amount of carbon dioxide (CO2) emitted into the atmosphere by human-made technology is dangerously speeding up the effects of climate change on the planet.

Trees play an extremely important role in Earth's climate behavior. They cover around 50% of Earth's land area and contain upwards of 90% of the global vegetation carbon. This means that a lot of CO2 that humans emit is being taken in by trees and used for their growth. Observing tree growth patterns can lead to a better understanding of how trees and climate interact with each other. These observations can also be used to predict the effects of climate change in the future.

To better understand how certain factors affect trees, Dr. Kiona Ogle and Dr. Michael Fell of Ogle Labs developed a simulation that shows a tree's growth over time. The simulation is called the Allometrically Constrained Growth and Carbon Allocation (ACGCA) model. It uses over 30 input parameters to run the simulation, which calculates the state of the tree over time. The output of the model contains useful information such as the tree's height, trunk radius, the carbon in the leaves and trunk, etc. The model was purely used for research by Dr. Ogle, Dr. Fell, and Ogle lab associates, and is not being used for profit. However, they wanted to expand the use of the model to anyone with an internet connection, for free, allowing everyday people to learn about tree growth from it.

1.2 Problem

The issue was that the ACGCA model had a very small user base. This means that very few people were using the model to learn about tree growth. The specific problems that the model has are as follows.

- Not available online

  The ACGCA model was not available online. This limited the use to only those acquainted with Dr. Ogle or Dr. Fell.

- Unfriendly user input

  The model's input was command line based, which required the user to have programming experience to run it.

- Required knowledge in biology

  The user needed to know what each of the 30+ input parameters are, which requires experience in tree biology.

- Unfriendly output
  The model only outputs raw numerical data, which can be useful for biologists, but no one else. It was not visually appealing nor informative to non-researchers.

- User information
  Our clients wanted a way to keep track of what kind of people are using their model. They wanted to know information about the user such as if they are a student, teacher, researcher, etc and their general location.

- Low budget
  Our clients did not have a grant for this project and had to pay for everything themselves.

The combination of these factors created a user environment that was unfriendly and required knowledge in specific fields. The fact that the model was not available online also contributed heavily to its low usage.

1.3 Solution

Dr. Ogle and Dr. Fell wanted to expand their audience so that anyone can utilize and learn from the ACGCA model. More specifically, they envisioned the model being used in a classroom setting where students can experiment with the inputs. This way, students can learn how certain factors affect tree growth. While students are the primary demographic of focus, researchers were also considered during development as the original demographic of the model.

Our clients also wanted a way to track how the model is being used. They were interested in knowing if more students versus researchers versus everyday people are using it. They were also interested in the general location of the users. While mandatory, this information is only used for analytical purposes and does not infringe on anyone's privacy.

Team TreeViz was tasked with making the model more user-friendly. To do this, we created a website that is hosted online for anyone to use. The website runs the model and significantly increases the accessibility to it by overhauling the input and output. Specific features that solved the project’s problems are as follows.

- Available online
  The website is hosted online so that anyone with internet access can use it. A wrapper and handles the passing of inputs and outputs of the model between the user and a server.

- User-friendly input
  The 30+ input parameters were converted to text boxes and sliders instead of a command line. This allows anyone to use it, rather than only researchers.

- Help boxes

  Each parameter on the input page has a help box that explains the parameter in more detail. This allows non-biologists to get a better understanding of what they do.

- Tree visualization

  The output is a rendered tree visualization. This is significantly more intuitive than the raw data and allows many more users to learn from the model. The raw data is also available to those who wish to use it.

- User surveys

  Users will submit a survey with some basic information so that our clients can see how the model is being used. This helps our clients develop the product further, allowing the prioritization of certain user demographics for future updates.

- Local desktop development

  To avoid costly solutions to our clients, the biggest change we made was developing the wrapper to be used on a local server, rather than hosting it online with the website.

With this solution, we provided our clients the means to broaden their audience. Now more people can use the ACGCA model to learn about tree growth. The website may be used in labs for researchers to conduct experiments and study climate change. It could also be used in a classroom where students are learning together about how trees grow..

1.4 Purpose of this Document

The purpose of this document is to efficiently get future developers for this project up to speed. It details all of the components necessary for understanding how to effectively develop this project further. Appendix A includes critical information on exactly how TreeViz initially developed the project, with what tools, and how to install everything you need to jump right into development.

2 - Process Overview

In our development process, we used the weekly task report to keep track of all of the tasks that we needed to do for the upcoming week and the tasks that have been completed during the past week. Every week we met with our mentor once to check if our task report is appropriate or not. We had group meetings two times a week on average to talk about some ideas about current tasks and future tasks. We also used the group meetings to do some group coding so that when we had difficulty we could help each other. We used a google doc to keep track of the meeting content so that when we forget something about the meeting we can go back and check. We used Github and the Github Desktop application for version control. We pushed our changes to it when we made sure that the current version of the local project can run properly. The benefit of using it is that when something in the project went wrong we could quickly find where it was wrong and revert the changes.

The process that we used for our development is similar to the waterfall method. We decided how we were going to complete the project during the first semester and planned very carefully. In the second semester, we kept developing the project as we planned. During the development process, we kept in touch with our clients, meeting every week. We showed our project progress to them to get suggestions and feedback. We keep improving our product in order to satisfy our clients' requirements.

The roles in the team are quite clear. Riley was the team leader and the coder for the tree visualization. Alex took the role of the recorder and worked on the database, model wrapper, and web interface. Daniel designed the structure of the project in the first semester and mainly worked on the Amazon Web server in the latter half of the developing process. Tang was the backend coder who mainly helped Alex on the model wrapper. Han was the frontend coder of the project. He mainly took care of the web pages to make sure everything on the frontend web pages looks fine and works well.

3 - Requirements

Our clients' requirements were divided into functional and non-functional requirements.

3.1 Functional Requirement

3.1.1 Web Application Functional Requirement

The product must be a web application that provides different ways for users to input data to the **ACGCA** model and get the results. These requirements include the functions that will give users access to the functionality of the model. They also include functions that will give the user a better experience when interacting with the model.

3.1.2 Survey Functional Requirement

Survey requirements are the requirements that pertain to the mandatory surveys that users need to fill out to gain access to the visualization. The surveys are necessary because our client would like to be able to gather data on who uses the model and why. They would also like to know how often people use the model and their affiliation, i.e., students, teachers, independent users, etc.

3.1.3 Maintenance Functional Requirement

Maintenance requirements are the functions the product needs in order to be easily maintainable in the future. Implementing these requirements will allow future developers to continue the project with minimal resistance.

3.2 Non-functional Requirement

3.2.1 Internet Accessible

Our clients want their product to be accessible for school students or researchers from different places all over the world. Thus, our product needs to be available online.

4 - Architecture and Implementation

This section will only be referencing, *Figure 1*, below. There are four independent systems that communicate with each other and each system has their own completely different functions that contribute to the overall program. These systems are connected over a network connection and communicate via ReST API calls. The four systems are the website, Amazon Web Server (AWS), Firebase, and the ACGCA wrapper. A very broad understanding of the interactions between systems is that the website communicates with AWS and Firebase; the ACGCA wrapper interacts with AWS; and AWS serves as a connection between the Website and ACGCA wrapper. As we dive deeper into the details of the individual systems, the broader picture will also be clearer.

The website is the system that the user interacts with. It provides the user with a more understandable Graphical User Interface. Starting from the first component, authentication, the user can log in, which uses the authentication services from Firebase, and validates if the user has an account or not. If the user authentication succeeds, then the user is redirected to the visualization page. However, if the authentication fails, then the user can either try again or create a new account. Once the user creates a new account they are routed to the survey page, which they have to complete, before being redirected to the visualization page.

Once on the visualization page, the user fills out form elements. Once they are filled out and the user hits run, the inputs are formatted into a JSON object and sent to AWS to be processed. When an input is sent to AWS, the website gets a unique ID for requesting the output. The website queries the ReST API on AWS to get the respective output from the ACGCA model. If an empty JSON object is sent back, then then the website waits for a period of time then queries again.

When the website receives the output, it is displayed as a visualization for the user. There are a few options for the user to choose from for how they wish to see the data: a 3D tree model, a visualization of the tree rings, and the raw numerical data.

Firebase is used to store information from the survey and login. As a result of this, the architecture is very simple and concise, containing only two components. Those components being a Survey Database and User Authenticator. The Survey Database allows administrators to review stored user surveys. The User Authenticator provides the means to validate users for login.

AWS has two main roles: host the website, and provide a means to connect users to an instance of the ACGCA wrapper. Starting with the hosting role, when a user goes to acgca.org (the domain name for the website) the Domain Name Server (DNS) on AWS will direct the user to an S3 bucket which stores the static website. The static website is then served to the users' web browser and to be rendered.

There is a ReST API hosted and built within AWS that passes data between the user's browser and the ACGCA wrapper. When the user sends an input to the API, a Run ID is generated which acts as a unique

ID for that instance of running the model. This Run ID is passed to both the user's browser and into the input queue with its correlating input that it was generated for. The ACGCA wrapper requests the ReST API for an input to be processed; the API will send the inputs as well as the Run ID. If there are no inputs in the Input Queue then an empty JSON object is sent instead.

Once an output is created, it is sent back to the ReST API along with the Run ID. The Output is then sent to DynamoDB, a collection, as a key-value pair. The key being a Run ID and the value being the output. When the user's browser requests the output using the Run ID, it will check DynamoDB if a key with a strictly equivalent value exists. If the key doesn't exist, it will give back an empty JSON object. However, if it does exist, it gives back the output which is the value that correlates with the key.

The ACGCA wrapper wraps the ACGCA model and provides extra functionality to the model such as get and post requests and data serialization. How the ACGCA wrapper works, is that it is an endless loop of grabbing inputs, processing those inputs into outputs, then sending the outputs back to the ReST API. After the wrapper gets the input, it serializes the data. This means it converts the json data that the API sends, into data that the ACGCA model can run. However, if the wrapper receives an empty JSON object it will wait a predetermined amount of time before trying to get more input. The outputs are also serialized into something that the ReST API can work with.

Overall, as the user progresses through the website, network calls to external services are made and data is passed to and from those services. An important part and core to this project, architecturally, is that the input data is able to be passed into the model without eachother even knowing the existence of the other.

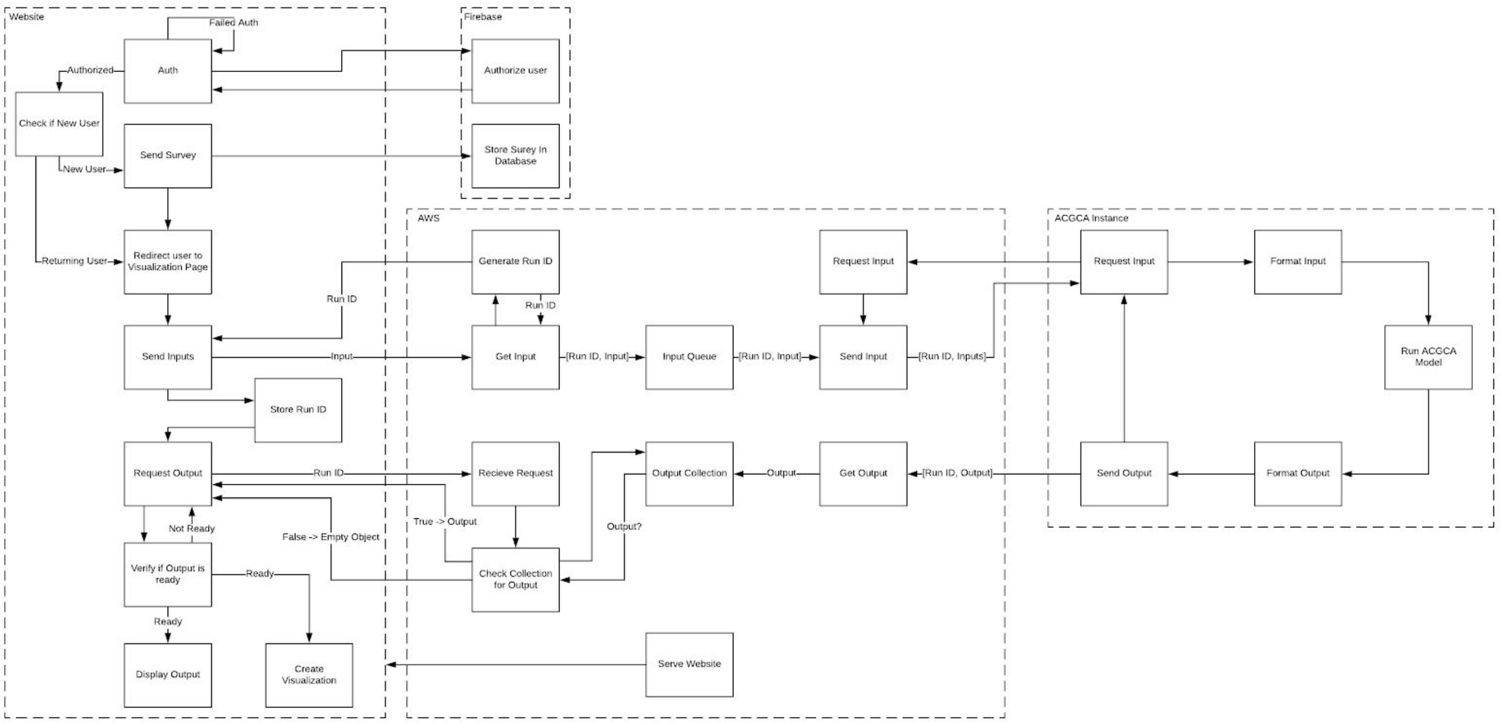


Figure 1: Architecture Diagram

5 - Testing

This section details our testing strategy and results. We focused on usability testing as this program is designed for use by the everyday person.

For usability testing, we created a Google form that had users perform 2 tests:

Test 1 -

The first test asked the users to follow the link to the website and attempt to visualize a tree with no further instruction. We asked them to rate how long they were on each page for and how difficult it was to figure out how to use each page. Then we asked them for any feedback that they can think of so far.

Test 2 -

The second test asked the users to follow step-by-step instructions to navigate through the website. If they got stuck during the first test, then this test would make sure they got to the end goal of visualizing a tree. Then users were asked to write their thoughts about each page and suggestions they have.

From these tests, we gathered that the sign-up and survey pages are very user-friendly. However, the visualization page needs some work. The biggest issue was the testers wanted visual feedback from the website that the model is running in the background, rather than a still page. Testers also asked for more default tree sets. Another problem was that when users refreshed the visualization page, it redirected them back to the login page, which was troublesome for many testers.

6 - Project Timeline

We divided our timeline into two parts: spring semester, fall semester.

Fall Semester

During the fall semester, as shown in *Figure 2*, we focused on discussing requirements with our clients to get a better understanding of what they need. We finished technological feasibility research and requirements specification.

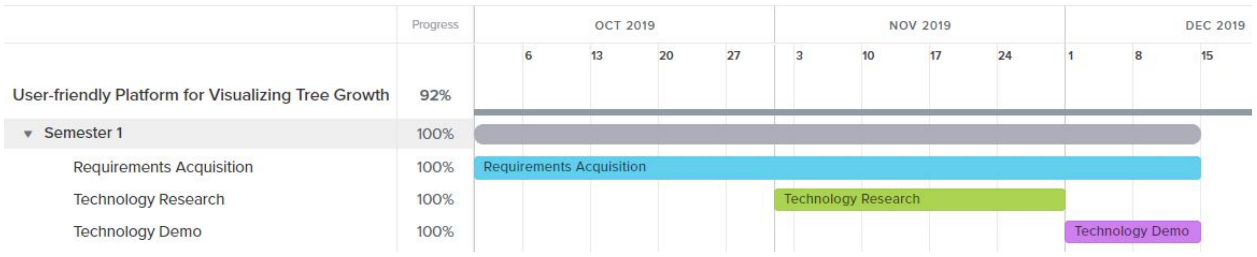


Figure 2: Fall Semester

Spring Semester

During the spring semester, as shown in *Figure 3*, we finished our basic product and ran the alpha prototype the week before spring break. After spring break, we focused on adding details to our product including website and tree visualization improvement. At the beginning of May, we delivered our final product to our clients.

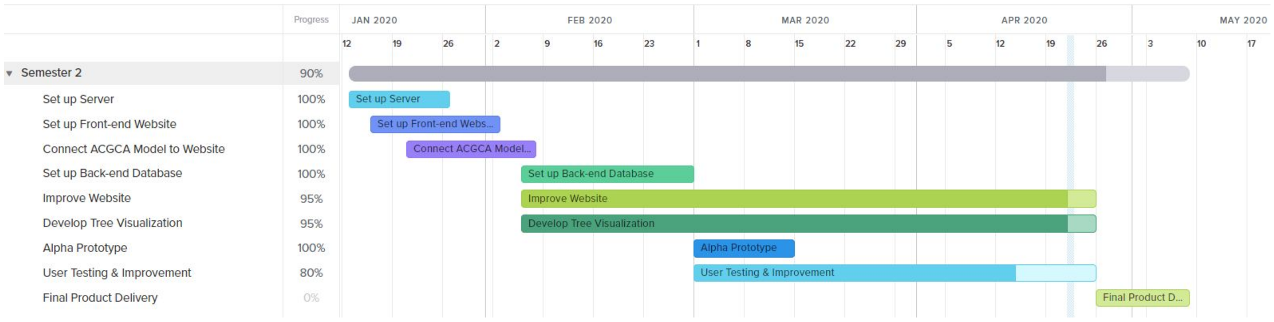


Figure 3: Spring Semester

7 - Future Work

The product that we have left the client has a lot of potential future for development. There are still many things that could be implemented within the system to work more for the client. The team was able to complete all of the required specifications from the client, so work will not need to focus on the primary functions of the product.

Future Goals

Some of the goals that could be implemented revolve around refinement of the webpage because the development team spent most of the time of the initial development cycle to build a low-cost and efficient backend for the product. Some updates that could be made to the website include:

- Better visualization that is more detailed
- Real time updates to the inputs and the outputs of the model
- Include a FAQ for the operation of the visualization software
- More detailed .csv file that is downloaded from the outputs of the model
- A loading icon that lets the user that the website is waiting for a response from AWS
- More detailed help boxes for the model's input
- Arrows to indicate that menus can be opened with a click
- Rerunning the visualization without reloading the webpage
- Allow the light variable to change over time

The only thing that proved to be a problem with the back-end of the system is that AWS's DynamoDB has a limit for the amount of data that can be stored within a single call. This means that the amount of data that can be sent at a given time is severely limited, and the amount of time in years that can be run maxes out at 450, which is not ideal for this type of model. To fix this, multiple messages need to be sent from the model and retrieved from the website to break up the large data package, or the size of the database on AWS needs to be increased.

8 - Conclusion

Climate change has been drastically increasing in speed in recent years. Trees play an important role in it by containing the majority of global vegetation carbon. To better understand how tree growth is affected by various factors, our clients, Dr. Kiona Ogle and Dr. Michael Fell, developed a simulation that calculates the growth of a tree over time: the ACGCA model. However, the model was only used by researchers in their lab because it was not available online and not very user-friendly. Dr. Ogle and Dr. Fell hope to expand its use to everyday people by making it more user-friendly and available online.

TreeViz worked for a year to lay the foundation of this project, gathering the requirements, designing an architecture, and implementing the plan into code, leaving our clients with a functioning alpha prototype. We developed an online, user-friendly web application that will allow the everyday person to learn about tree growth from their model. Some specific features include:

- Hosted online for anyone to use.
- User-friendly control panel that replaces the command-line based input.
- 3-dimensional rendered tree of the output of the model, providing a more intuitive way to see how the inputs may affect the growth of a tree.

With their user-friendly platform for visualizing tree growth, Dr. Ogle and Dr. Fell can expand their audience from just a few researchers in their lab to any of the millions of people in the world that are interested in tree growth. With this product, hopefully more people can become educated on tree growth and, in turn, seek out to learn more about climate change. Researchers may also use this product to study tree growth and its effect on the climate.

TreeViz was glad to have worked on this project for so long, and are proud of what we built for our clients.

9- Glossary

ACGCA - Allometrically Constrained Growth and Carbon Allocation mode. The model built by our clients to simulate the growth of trees over time.

ReST API - Representational State Transfer Application Programming Interface. A system used to send and receive data between components of a system.

AWS - Amazon Web Services.

DynamoDB - A fully managed NoSQL database managed by Amazon.

API - Application Programming Interface.

DNS - Domain Name System. Used to create a URL that is easy to navigate to and is of our choosing (e.g. acgca.org)

Firebase - Mobile and web application development platform developed by Google.

10 - Appendix A

10.1 - Hardware

The hardware that the team used for development was a combination of Windows and macOS environments. The development team eash used their personal computers for development. A combination of development environments was essential for the development team to learn the differences between machines when developing. The machines that were used for the development of each part of the system are as follows:

Website

All team members had a hand in developing the website so everyone's machine was used; 4 Windows machines and 2 macOS machines. Each individual's machines had different hardware specifications with 1 machine using an AMD CPU and every other machine using an Intel CPU. The work that the development team did was mainly CPU intensive tasks. There were three dual core processors and three quad core processors, which proved to be more than enough processing power for development.

Model Wrapper

The wrapper that was created for the client's model was developed primarily on a macOS machine with a quad core 2.5 GHz i7 with 16 GB of RAM.

Visualization

The visualization for the website was developed on a Windows machine that had a quad core 2.8 GHz i7 with 16GB of RAM.

Minimum Specifications

The minimum hardware specifications for the further development of the entire system are very low. VueJS requires Node.js version 8.9 or higher which only requires a 64 bit architecture. To run the wrapper for the model, the operating system should be Unix based, which includes Linux builds as well as macOS, and should be running Python 2. The minimum requirements for the further development of the product are relative to the type of development to be done. If development for the product requires more processing power, then the minimum requirements will be much higher, but for the software that we have used for the initial development, the requirements are low.

10.2 - Toolchain

This section details all of the software tools that we used to develop the project.

For version control, we used GitHub Desktop at https://desktop.github.com/. This application makes it easier to commit/push/pull/merge changes when working on the project.

For our package manager, we used Node.js. This allowed us to easily download all the necessary tools without having to push them to GitHub. The package.json file was the only file that was pushed to GitHub, which contains all the information on which tools we used.

To use Node.js, simply install it at https://nodejs.org/en/. Then go to the root directory of the project and type "npm install". This will install all of the tools we used as saved in the package.json file. If you wish to download more tools, type "npm install tool-name" and it will download it on your machine and add it to the list of dependencies in the package.json file. Make sure to push the updated package.json to GitHub so that other developers have the updated dependency. The repository will ignore the dependency itself to save space, but all they have to do is type "npm install" to download any new dependencies listed in the package.json file.

10.2.1 - Website

For the website we use Vue.js to complete the project.
Link: https://vuejs.org/index.html

Vue.js - An open-source JavaScript framework that was developed by Evan You, who was an old Angular team member. The goal of Vue.js was to create a new framework that combined the best approaches to front-end web development

10.2.2 - Visualization

For the visualization, we used Three.js, which is embedded in Vue.js on the website.
Link: https://threejs.org/

Three.js - A 3D graphics framework for JavaScript. It allowed us to render 3D trees and 2D rings based on the model's output, quickly giving users a visualization of the tree that was simulated from their inputs.

10.2.3 - Server

For the server, we used AWS, which hosts the website and provides communication between the ACGCA Instance and the User.
Link: https://aws.amazon.com/

AWS - A cloud service platform by amazon. It allowed us to host the website on it, have a DNS server for that website, and be able to pass data to and from the ACGCA instance, just by connecting the services they offer together.

10.2.4 - Database

For the database to store the user information, we used Google Firebase which allows an environment for the client to navigate easily as well as provide easy storage from the website.
Link: https://firebase.google.com/

Real Time Database - Firebase's real time database is a fast and responsive database that was used to store the survey information that the user supplies before they can access the visualization. The built in methods that are included in the firebase package made development easy as well as provides an environment for analysis and support from Google.

10.2.4 - Wrapper

For the Wrapper, we used python and requests, which wraps the ACGCA instance allowing us to request and send data, run the model, and to serialize and format the data passed.
Links: https://www.python.org/, https://requests.readthedocs.io/en/master/

Python - A general purpose programming language. We used it to format and serialize the data; it also allowed us to call the ACGCA model which is written in C.

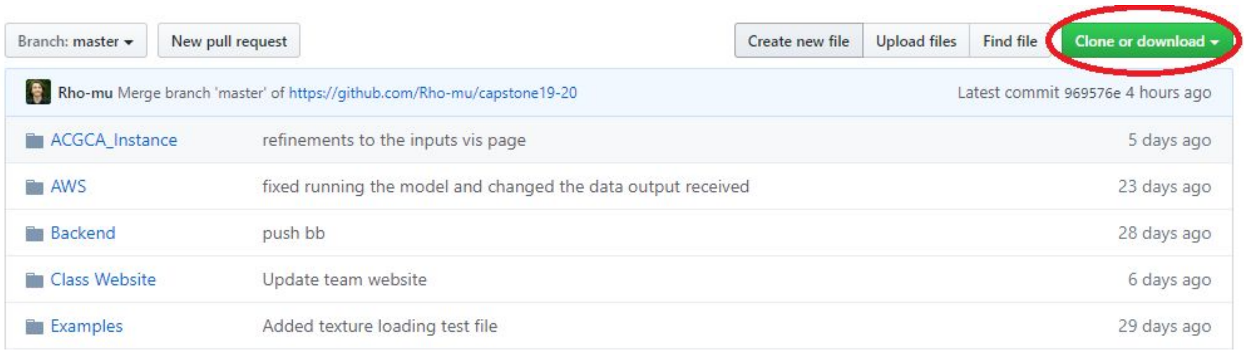
Requests - A module for python. It allowed us to call the server to get input data and send output data.

10.3 - Setup

There are two things that need to be done to start development on the project: run the ACGCA model wrapper, and run the website. The steps for setting up both of these are as follows.

10.3.2 Local Website Setup

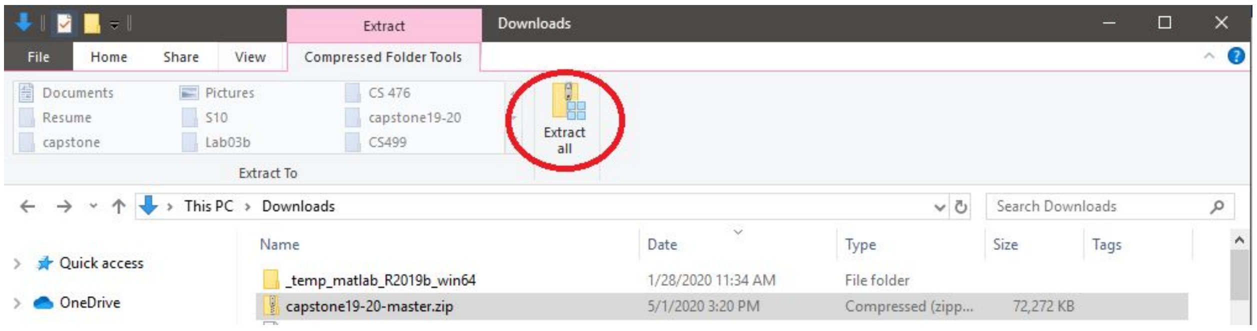
1. Go to the GitHub repository and download it. https://github.com/Rho-mu/capstone19-20



Rho-mu Merge branch 'master' of https://github.com/Rho-mu/capstone19-20 Latest commit 969576e 4 hours ago

ACGCA_Instance	refinements to the inputs vis page	5 days ago
AWS	fixed running the model and changed the data output received	23 days ago
Backend	push bb	28 days ago
Class Website	Update team website	6 days ago
Examples	Added texture loading test file	29 days ago

2. Extract the downloaded folder to a directory of your choosing.



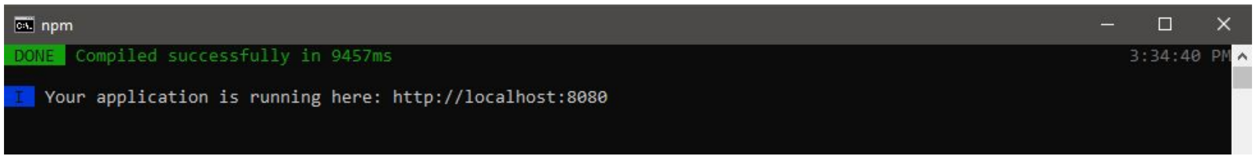
Name	Date	Type	Size	Tags
_temp_matlab_R2019b_win64	1/28/2020 11:34 AM	File folder		
capstone19-20-master.zip	5/1/2020 3:20 PM	Compressed (zipp...	72,272 KB	

3. Navigate to the new location of the capstone19-10 folder with the command line.

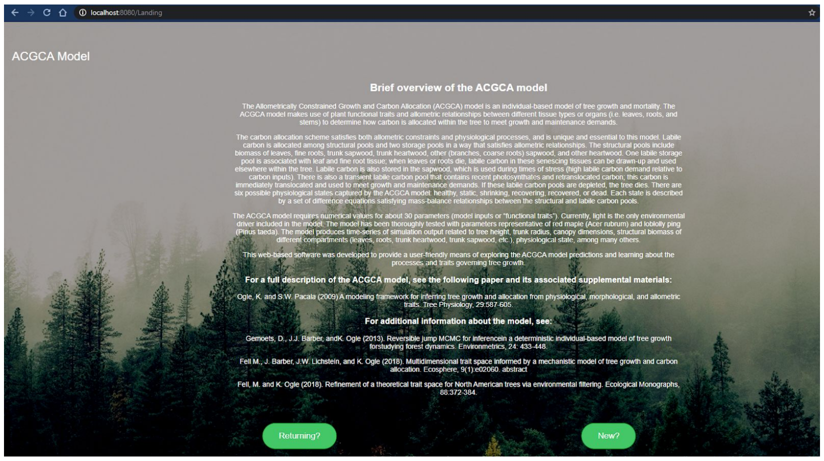
4. Type “cd capstone” to navigate to the capstone folder.

5. Type “npm install” to install all dependencies. Note that this only has to be redone when a new dependency is installed. For example: if you want to use a framework called test.js, you would type “npm install test.js” to install it. It will automatically update the package.json file so that when another developer types npm install, test.js will also be installed.

6. Once it’s done installing, type “npm run dev” to run a development server. If nothing went wrong, you should be able to go to http://localhost:8080 and view the project website.



```
npm
 DONE  Compiled successfully in 9457ms                                    3:34:40 PM

 I  Your application is running here: http://localhost:8080
```



Brief overview of the ACGCA model

The Allometrically Constrained Growth and Carbon Allocation (ACGCA) model is an individual-based model of tree growth and mortality. The ACGCA model makes use of plant functional traits and allometric relationships between different tissue types or organs (i.e. leaves, roots, and stems) to determine how carbon is allocated within the tree to meet growth and maintenance demands.

The carbon allocation scheme satisfies both allometric constraints and physiological processes, and is unique and essential to this model. Labile carbon is allocated among structural pools and two storage pools in a way that satisfies allometric relationships. The structural pools include biomass of leaves, fine roots, trunk sapwood, trunk heartwood, other (branches, coarse roots) sapwood, and other heartwood. One labile storage pool is associated with leaf and fine root tissue; when leaves or roots die, labile carbon in these senescing tissues can be drawn-up and used elsewhere within the tree. Labile carbon is also stored in the sapwood, which is used during times of stress (high labile carbon demand relative to carbon inputs). There is also a transient labile carbon pool that contains recent photosynthates and retranslocated carbon; this carbon is immediately translocated and used to meet growth and maintenance demands. If these labile carbon pools are depleted, the tree dies. There are six possible physiological states captured by the ACGCA model: healthy, static, shrinking, recovering, recovered, or dead. Each state is described by a set of difference equations satisfying mass-balance relationships between the structural and labile carbon pools.

The ACGCA model requires numerical values for about 30 parameters (model inputs or "functional traits"). Currently, light is the only environmental driver included in the model. The model has been thoroughly tested with parameters representative of red maple (Acer rubrum) and loblolly ping (Pinus taeda). The model produces time-series of simulation output related to tree height, trunk radius, canopy dimensions, structural biomass of different compartments (leaves, roots, trunk heartwood, trunk sapwood, etc.), physiological state, among many others.

This web-based software was developed to provide a user-friendly means of exploring the ACGCA model predictions and learning about the processes and traits governing tree growth.

For a full description of the ACGCA model, see the following paper and its associated supplemental materials:

Ogle, K. and S.W. Pacala (2009) A modeling framework for inferring tree growth and allocation from physiological, morphological, and allometric traits. Tree Physiology, 29:587-605.

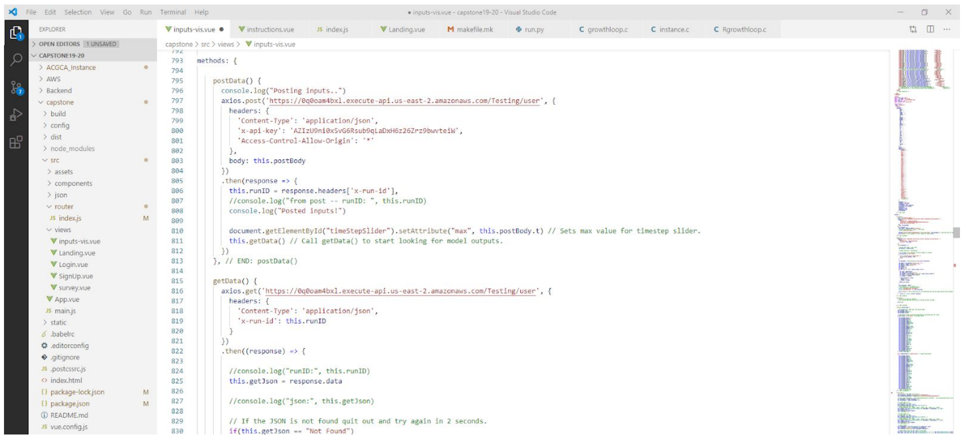
For additional information about the model, see:

Gemoets, D., J.J. Barber, andK. Ogle (2013). Reversible jump MCMC for inferencein a deterministic individual-based model of tree growth forstudying forest dynamics. Environmetrics, 24: 433-448.

Fell M., J. Barber, J.W. Lichstein, and K. Ogle (2018). Multidimensional trait space informed by a mechanistic model of tree growth and carbon allocation. Ecosphere, 9(1):e02060. abstract

Fell, M. and K. Ogle (2018). Refinement of a theoretical trait space for North American trees via environmental filtering. Ecological Monographs, 88:372-384.

7. Now that you have the website up and running, it’s time to set up the development environment. Open your text editor of choice and navigate to the project folder. This screenshot shows VSCode, but any text editor will do.



```
methods: {

  postData() {
    console.log("Posting inputs..")
    axios.post('https://0q0oam4bxl.execute-api.us-east-2.amazonaws.com/Testing/user', {
      headers: {
        'Content-Type': 'application/json',
        'x-api-key': 'AZIzU9ni0x5vG6Rsub9qLaDxH6z26Zrz9bwvteiW',
        'Access-Control-Allow-Origin': '*'
      },
      body: this.postBody
    })
    .then(response => {
      this.runID = response.headers['x-run-id'],
      //console.log("from post -- runID: ", this.runID)
      console.log("Posted inputs!")

      document.getElementById("timeStepSlider").setAttribute("max", this.postBody.t) // Sets max value for timestep slider.
      this.getData() // Call getData() to start looking for model outputs.
    })
  }, // END: postData()

  getData() {
    axios.get('https://0q0oam4bxl.execute-api.us-east-2.amazonaws.com/Testing/user', {
      headers: {
        'Content-Type': 'application/json',
        'x-run-id': this.runID
      }
    })
    .then((response) => {

      //console.log("runID:", this.runID)
      this.getJson = response.data

      //console.log("json:", this.getJson)

      // If the JSON is not found quit out and try again in 2 seconds.
      if(this.getJson == "Not Found")
```

8. Start coding! Any changes you save will automatically be updated on the website without needing to be refreshed. This is one of the benefits of using Vue.js.

10.3.1 ACGCA Model Wrapper Setup

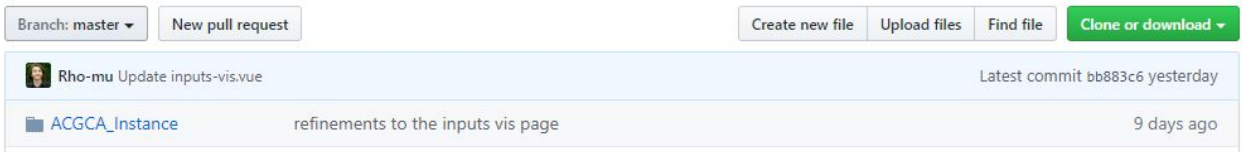
There are a couple things to consider before setting up the model wrapper.

- Am I working on just the website?
  If you are only working on the website and there is already an instance of the model running somewhere, then there is no need to set up the model on your local machine.

- Am I working on the wrapper/ACGCA model code?
  If you are working on the wrapper or ACGCA model, then you may want to have debug/log statements print to the terminal. In this case, other instances of the model besides your own can intercept your inputs, and you will not be able to see your log statements. The only way around this is to have the only running instance of the model on your machine. This guarantees that all inputs will be picked up by your instance of the model.

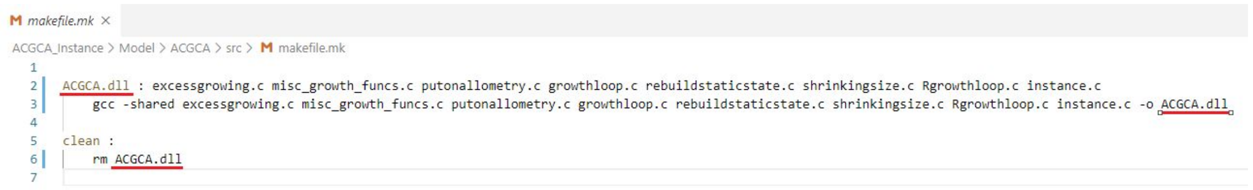
Considering those two things, here are the steps for setting up an instance of the model wrapper:

1. Download the ACGCA_Instance folder from the Github repository. If you have already downloaded the whole repository then you can skip this step.



2. Go to the ACGCA_Instance directory in the command line and type in the following command to enter the src folder. "cd Model/ACGCA/src"

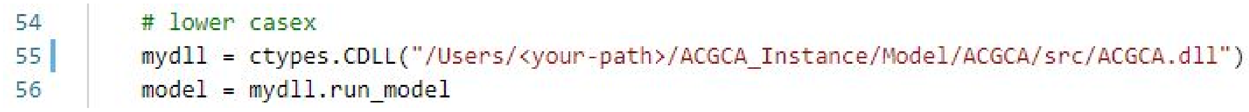
3. Open makefile.mk at "ACGCA_Instance/Model/ACGCA/src" and check if you have ACGCA.dll or ACGCA.so. If you are on Windows, you should have ACGCa.dll. If you are on Mac/Linux, you should have ACCGA.so. If you have the wrong ACGCA type in your makefile.mk, simply change all instances of ACGCA.xx to the one you need. The screenshot below shows all three instances.



```

ACGCA.dll : excessgrowing.c misc_growth_funcs.c putonallometry.c growthloop.c rebuildstaticstate.c shrinkingsize.c Rgrowthloop.c instance.c
	gcc -shared excessgrowing.c misc_growth_funcs.c putonallometry.c growthloop.c rebuildstaticstate.c shrinkingsize.c Rgrowthloop.c instance.c -o ACGCA.dll

clean :
	rm ACGCA.dll

```

4. Once you have the correct ACGCA.xx, type "make -f makefile.mk" in the command line to compile the ACGCA model into a dll/so. If you are on Windows, you will need to use a C compiler to run this command, such as MinGW (http://www.mingw.org/).

5. Open run.py in “ACGCA_Instance/Model/ACGCA/src” and change the line shown in the screenshot below to your path to ACGCA.dll/ACGCA.so.



```
# lower casex
mydll = ctypes.CDLL("/Users/<your-path>/ACGCA_Instance/Model/ACGCA/src/ACGCA.dll")
model = mydll.run_model
```

This line links the wrapper with the compiled model.

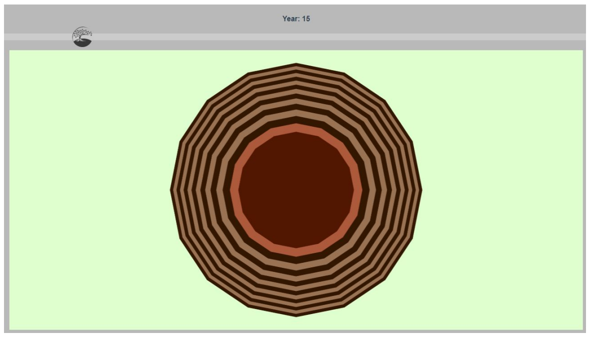
6. Now you should be able to run an instance of the ACGCA model. Type “python run.py” to run the instance. You can do this on multiple terminals to run multiple instances. You will need Python installed for this part (https://www.python.org/downloads/).

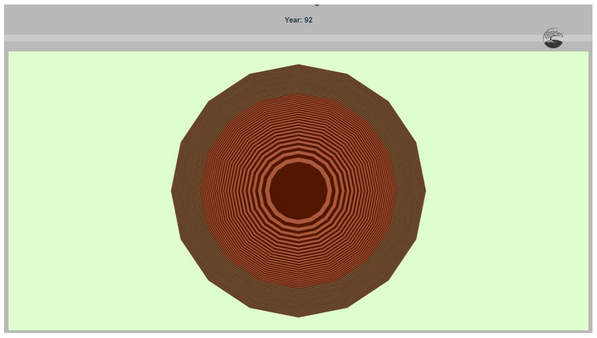
10.4 - Production Cycle

The production cycle starts with having everything setup as detailed in section 10.3. To update the project, you need to edit the code, check the website to make sure there are no errors, and create a build version to upload to AWS. To explain this, we will be going through a specific example of changing how the tree rings are displayed.

10.4.1 - Edit

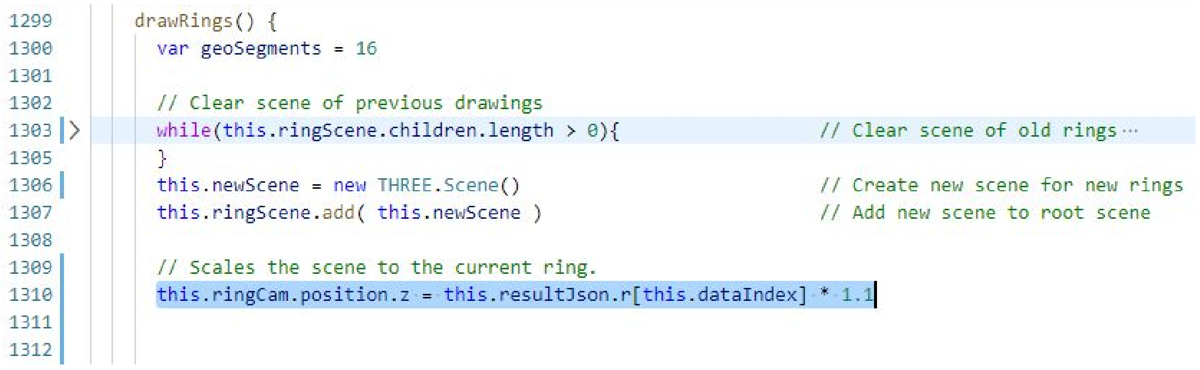
As you can see from the screenshots below, the size of the tree rings visualization changes depending on the current radius of the tree. This means that whether you have a small radius or a large radius, the size of the full circle stays the same. We want to change this so that the scene scales with the maximum radius of the tree such that the rings stay the same size instead of scaling to fill the scene.





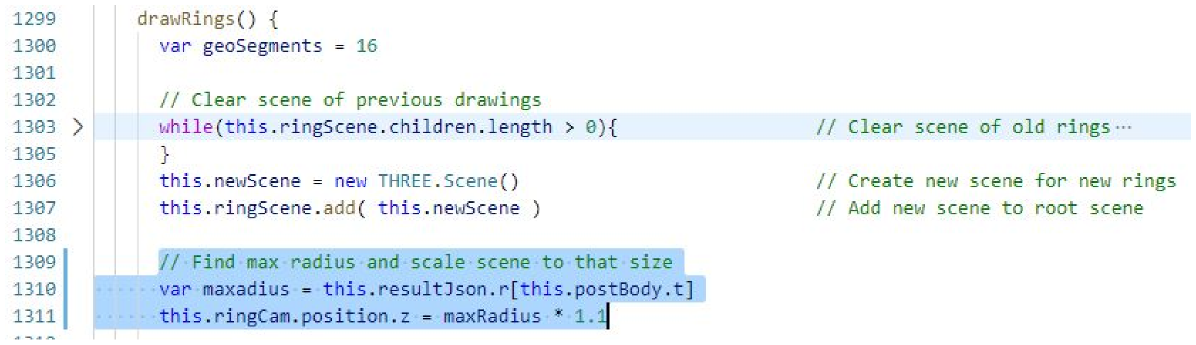
Here are the steps to make this change:

1. First, we need to find the code that scales the scene based on the current radius. The way the scene scales is by moving the camera further back.



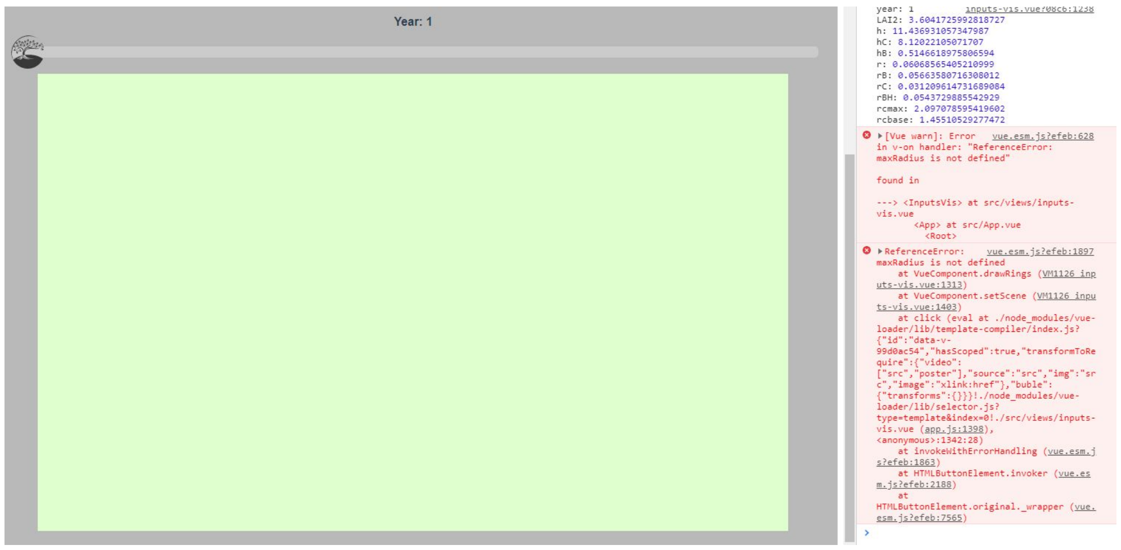
```
1299      drawRings() {
1300        var geoSegments = 16
1301
1302        // Clear scene of previous drawings
1303 >      while(this.ringScene.children.length > 0){            // Clear scene of old rings…
1305        }
1306        this.newScene = new THREE.Scene()                    // Create new scene for new rings
1307        this.ringScene.add( this.newScene )                  // Add new scene to root scene
1308
1309        // Scales the scene to the current ring.
1310        this.ringCam.position.z = this.resultJson.r[this.dataIndex] * 1.1
1311
1312
```

2. We will replace this line of code with new code that gets the radius of the final ring and scales the scene to that instead.



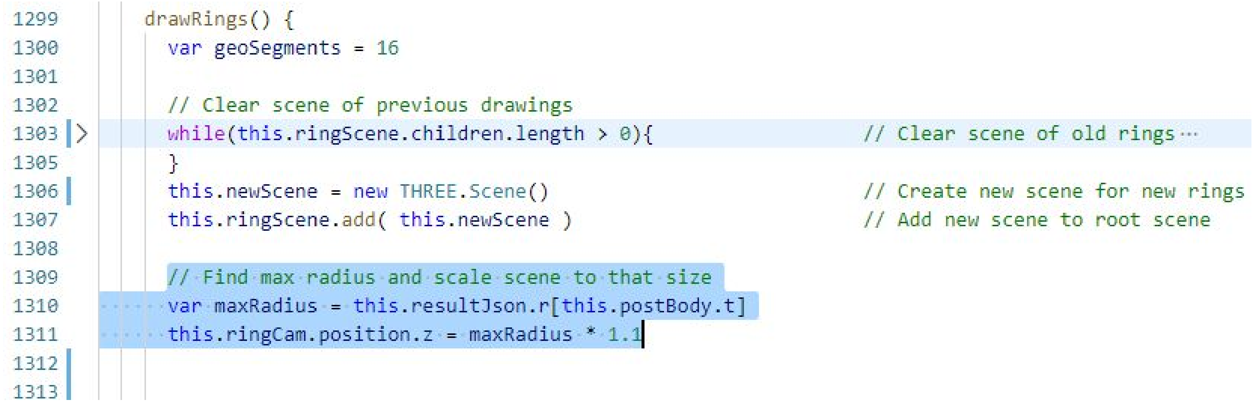
```
1299      drawRings() {
1300        var geoSegments = 16
1301
1302        // Clear scene of previous drawings
1303 >      while(this.ringScene.children.length > 0){            // Clear scene of old rings…
1305        }
1306        this.newScene = new THREE.Scene()                    // Create new scene for new rings
1307        this.ringScene.add( this.newScene )                  // Add new scene to root scene
1308
1309        // Find max radius and scale scene to that size
1310        var maxadius = this.resultJson.r[this.postBody.t]
1311        this.ringCam.position.z = maxRadius * 1.1
```

3. After changing the code and saving the file, we check the website and the browser's console for errors. As you can see from the screenshot, it seems that there is a typo in the new code that we wrote. On line 1310, maxadius should be maxRadius.



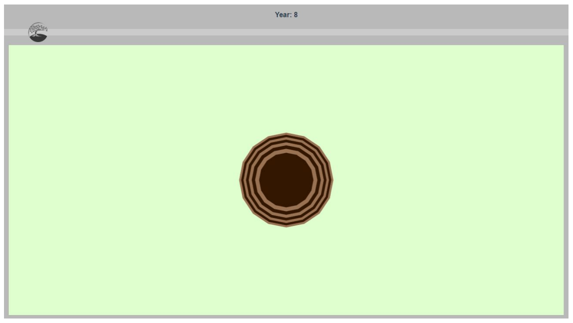
```
year: 1          inputs-vis.vue?08c6:1238
LAI2: 3.6041725992818727
h: 11.436931057347987
hC: 8.12022105071707
hB: 0.5146618975806594
r: 0.06068565405210999
rB: 0.05663580716308012
rC: 0.031209614731689084
rBH: 0.0543729885542929
rcmax: 2.097078595419602
rcbase: 1.45510529277472
▶[Vue warn]: Error   vue.esm.js?efeb:628
in v-on handler: "ReferenceError:
maxRadius is not defined"

found in

---> <InputsVis> at src/views/inputs-
vis.vue
       <App> at src/App.vue
         <Root>
▶ReferenceError:    vue.esm.js?efeb:1897
maxRadius is not defined
    at VueComponent.drawRings (VM1126 inp
uts-vis.vue:1313)
    at VueComponent.setScene (VM1126 inpu
ts-vis.vue:1403)
    at click (eval at ./node_modules/vue-
loader/lib/template-compiler/index.js?
{"id":"data-v-
99d0ac54","hasScoped":true,"transformToRe
quire":{"video":
["src","poster"],"source":"src","img":"sr
c","image":"xlink:href"},"buble":
{"transforms":{}}}!./node_modules/vue-
loader/lib/selector.js?
type=template&index=0!./src/views/inputs-
vis.vue (app.js:1398),
<anonymous>:1342:28)
    at invokeWithErrorHandling (vue.esm.j
s?efeb:1863)
    at HTMLButtonElement.invoker (vue.es
m.js?efeb:2188)
    at
HTMLButtonElement.original._wrapper (vue.
esm.js?efeb:7565)
>
```

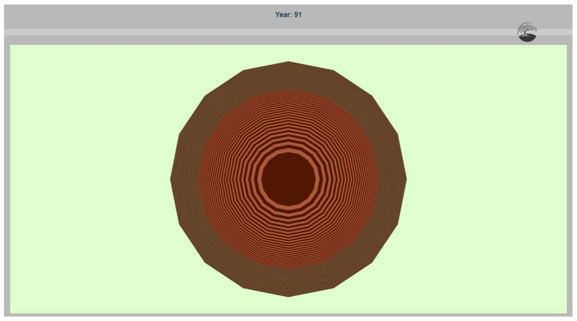
4. So, we go back and fix this and check the website again to make sure everything’s working.



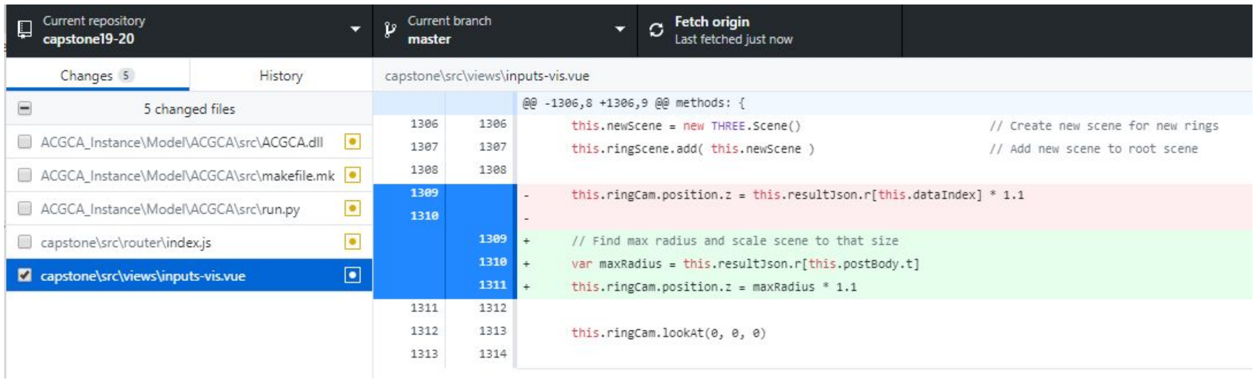
```
  drawRings() {
    var geoSegments = 16

    // Clear scene of previous drawings
    while(this.ringScene.children.length > 0){               // Clear scene of old rings···
    }
    this.newScene = new THREE.Scene()                        // Create new scene for new rings
    this.ringScene.add( this.newScene )                      // Add new scene to root scene

    // Find max radius and scale scene to that size
    var maxRadius = this.resultJson.r[this.postBody.t]
    this.ringCam.position.z = maxRadius * 1.1


```

The screenshots below show that the new code works and is making the tree rings appear to grow outward.





5. Now that the change has been made and the code is working, we need to commit the change to the repository and push it to GitHub.



ACGCA_Instance\Model\ACGCA\src\ACGCA.dll
ACGCA_Instance\Model\ACGCA\src\makefile.mk
ACGCA_Instance\Model\ACGCA\src\run.py
capstone\src\router\index.js
capstone\src\views\inputs-vis.vue

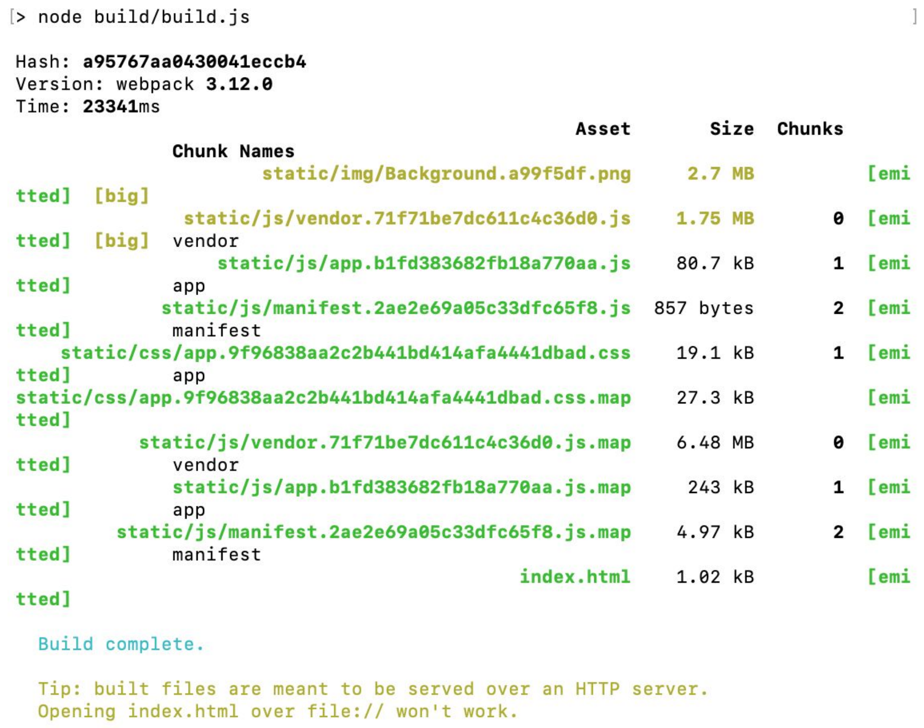
capstone\src\views\inputs-vis.vue

```
@@ -1306,8 +1306,9 @@ methods: {
        this.newScene = new THREE.Scene()                      // Create new scene for new rings
        this.ringScene.add( this.newScene )                    // Add new scene to root scene

-       this.ringCam.position.z = this.resultJson.r[this.dataIndex] * 1.1
-
+       // Find max radius and scale scene to that size
+       var maxRadius = this.resultJson.r[this.postBody.t]
+       this.ringCam.position.z = maxRadius * 1.1

        this.ringCam.lookAt(0, 0, 0)

```

6. That ends the process for editing code. The next section details how to create a build version of the website and push it to AWS.

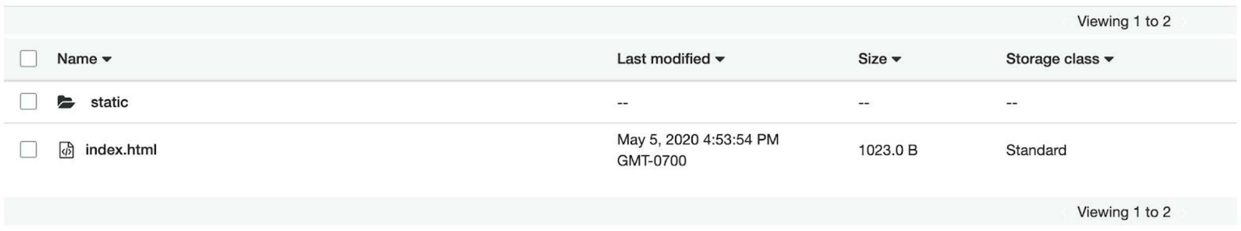
10.4.2 - Build

In order to build the product for development there is only one command that needs to be run. Within the same directory where the webpage files are located (capstone19-20/capstone/src) run the following command in your terminal, “npm run build”. The result should resemble the screenshot below.



```
> node build/build.js

Hash: a95767aa0430041eccb4
Version: webpack 3.12.0
Time: 23341ms
                                                     Asset       Size  Chunks
              Chunk Names
                      static/img/Background.a99f5df.png     2.7 MB          [emi
tted]  [big]
               static/js/vendor.71f71be7dc611c4c36d0.js    1.75 MB       0  [emi
tted]  [big]  vendor
                  static/js/app.b1fd383682fb18a770aa.js    80.7 kB       1  [emi
tted]         app
             static/js/manifest.2ae2e69a05c33dfc65f8.js  857 bytes       2  [emi
tted]         manifest
    static/css/app.9f96838aa2c2b441bd414afa4441dbad.css    19.1 kB       1  [emi
tted]         app
static/css/app.9f96838aa2c2b441bd414afa4441dbad.css.map    27.3 kB          [emi
tted]
           static/js/vendor.71f71be7dc611c4c36d0.js.map    6.48 MB       0  [emi
tted]         vendor
              static/js/app.b1fd383682fb18a770aa.js.map     243 kB       1  [emi
tted]         app
         static/js/manifest.2ae2e69a05c33dfc65f8.js.map    4.97 kB       2  [emi
tted]         manifest
                                             index.html    1.02 kB          [emi
tted]

  Build complete.

  Tip: built files are meant to be served over an HTTP server.
  Opening index.html over file:// won't work.
```

Once the build is complete, we will upload the files to AWS. Navigate to https://s3.console.aws.amazon.com/s3/home?region=us-east-2#, and open the S3 bucket that says “acgca.org”. Once there click the button that says “Upload”. Once redirected there will be a page that allows you to drag and drop files in. The files will be located within the directory where we ran “npm run build”, which contains the necessary files for the web page. The S3 bucket that we have navigated to will show the files listed in the screenshot below.



Viewing 1 to 2

Name	Last modified	Size	Storage class
static	--	--	--
index.html	May 5, 2020 4:53:54 PM GMT-0700	1023.0 B	Standard

Viewing 1 to 2

We will find these same files inside the “dist” folder within the “src” directory for the web page. We need to take our new “static” and “index.html” from our new build and replace them in AWS. Once the files have been uploaded they will be viewable under the “acgca.org” URL.