CPSC 4050 Group Project Report

29 April 2020

Table of Contents

Project Goals

In short, the main goal of this project was to create an interactable application that creates fractal canopies. We wanted users to be able to control varying parameters related to the creation of the canopy, with all of these having the capability of being altered dynamically at run-time. It was one of our goals to allow users ample and quick experimentation abilities; in other words, we wanted users to be able to choose certain canopy values to modify from a list and see the results of these modifications quickly. As a result, by using our application, users should have the ability to take part in an educational (and hopefully fun) experience relating to fractals.

Project Description

Task Overview

The most important task that we had to address at the start was the actual creation of the fractal canopy. In order to carry out this task, we needed a line-drawing function along with other functions that would determine where to draw the lines in order to create the fractal. Therefore, we created an implementation of Bresenham's Line Algorithm as was explored in the first programming assignment of this class to address the first need. We then worked on the logic behind creating the canopy, which resulted in two recursive functions, branchLeft and branchRight. Another function, rotateBranch, was implemented as well in the vector2 class. This class made it easier to interact with points, as we were able to store x and y values in a single instance of that class instead of having to keep track of them separately.

After we were able to draw fractal canopies correctly, we then implemented translations along the X, Y, and Z axes. This task required changing to perspective projection, and after completion, users were able to translate up and down, left and right, and in and out. The Z axis translations were especially interesting, since it afforded the ability to zoom in really close to the tree and examine the smaller fractal branches in greater detail.

Our next big group of tasks was to allow dynamic interactivity with the canopy. The following modifiable parameters that we implemented as are follows: increase/decrease angle between left and right branches, alter the tree base location (either have the tree grow upwards or downwards), change the tree's color, increase/decrease the tree color gradient, increase/decrease the branch reduction value, increase/decrease the minimum branch length

for left and right branches, increase/decrease the branch thickness for left and right branches, increase/decrease the thickness for the trunk, and reset all transformations.

All of these interactions are carried out using keyboard keys, and the exact mappings are printed in the console on the start of the program (and can be seen in the Screen-Grabs section below). Furthermore, every task that deals with either increasing or decreasing a value or moving something one way along with the opposite way has something in common. This similarity is best explained through an example. Say you want to increase the branch reduction factor of the canopy. In order to do this, you hit the 'd' key. However, in order to get the opposite effect (meaning *decreasing* the branch reduction factor), you have to input 'D'. This feature of using the capital version of a key to do the opposite effect of one certain parameter is implemented wherever possible in our program to cut down on the number of keys needed.

Finally, we implemented the rather fun ability to change the tree color. There are several different colors to toggle between. The color of the tree is a gradient between two of the RGB color channels, with the unused RGB channel being set to zero. The gradient is based off of the pixel length of each branch.

Tools and Libraries Used

In this project, in addition to OpenGL, we utilized the Glu and Glut utility libraries. We also used the following C++ libraries: math.h, stdlib.h, and stdio.h.

Workflow

We all worked from one central repository hosted on Github. We utilized different branches (assigned to each team member) that were merged to master upon completion. We also used Clemson's School of Computing virtual desktop to test our program execution.

We first outlined all tasks that would need to be completed. We then organized these tasks into a table with columns Task, Additional Info, Assigned To, and Completed. We then went about assigning ourselves tasks and updating the table whenever necessary. This way, we were always able to avoid having a situation in which more than one group member was concurrently working on the same exact task as another. It also allowed us to gauge our progress to ensure that the assignment was completed on time.

Challenges

Most of this project went pretty smoothly. However, after setting up the fractal canopy, adding all of the user customization took a while. There were so many different parameters we had to add and it required a lot of trial and error to get all of them working correctly. We worked through this challenge by splitting up the parameters so each member of the group worked on different parts so no one got overwhelmed. The recursive nature of the tree was also a challenge. This made it difficult to visualize how the tree would turn out, and it also took a lot of trial and error to get it right.

Strategies Used

One strategy that we used that we touched on briefly before is the class vector2. Vector2 allowed us to easily keep track of x and y points in one instance instead of dealing with them separately. This class made implementing the branchLeft, branchRight, and rotateBranch functions easier.

Lastly, our use of recursion was essential in creating the fractal canopy. Implementing functions that called themselves while always including an exit condition made the fractal creation process much simpler.

Learnings

One thing we learned from testing out our completed application was how interesting the shapes of fractals could be. For example, we discovered that increasing/decreasing the angle of the branches resulted in shapes that definitely do not resemble a tree. Nonetheless, it was still an educational experience to see exactly how the fractal warped when, say, the right angle was increased by a large amount.

Likewise, the ability to zoom in on the fractal provided more insight into the properties of fractal canopies. Viewing the canopy from a distance can sometimes make it hard to believe that the branches at the top of the tree are still repeating the same pattern. Getting a closer look at these branches, though, resolves any doubts that we had.

Screen-Grabs

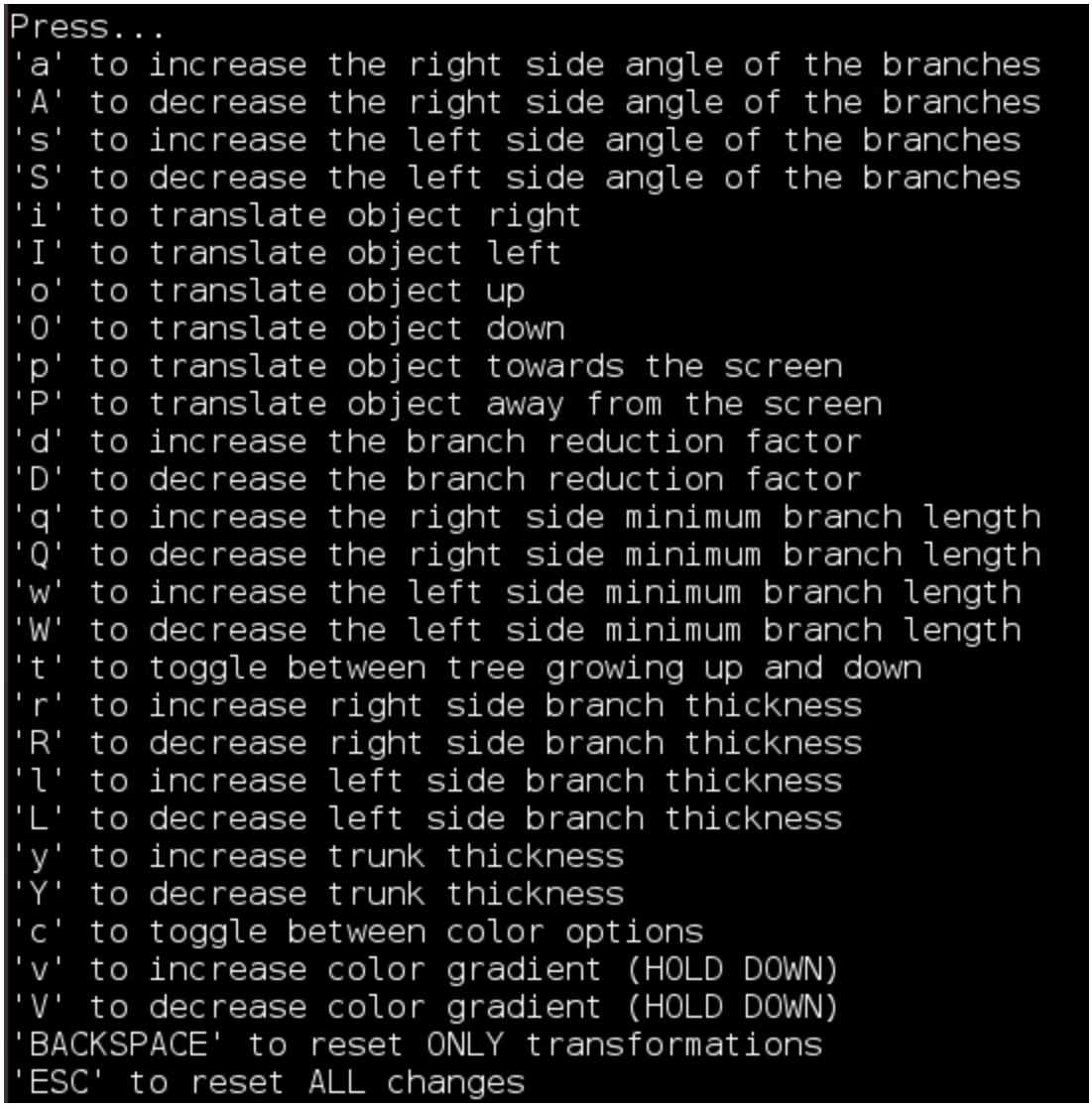

Figure 1: The menu of all key options printed to the terminal for the user

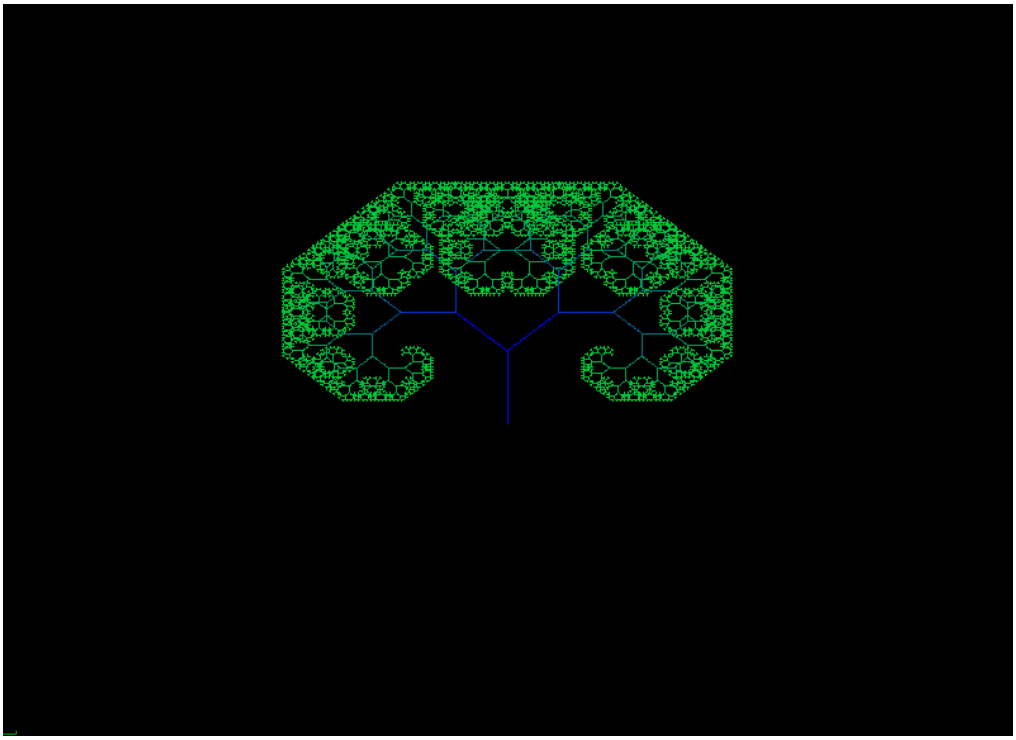

Figure 2: The original fractal canopy without any user input

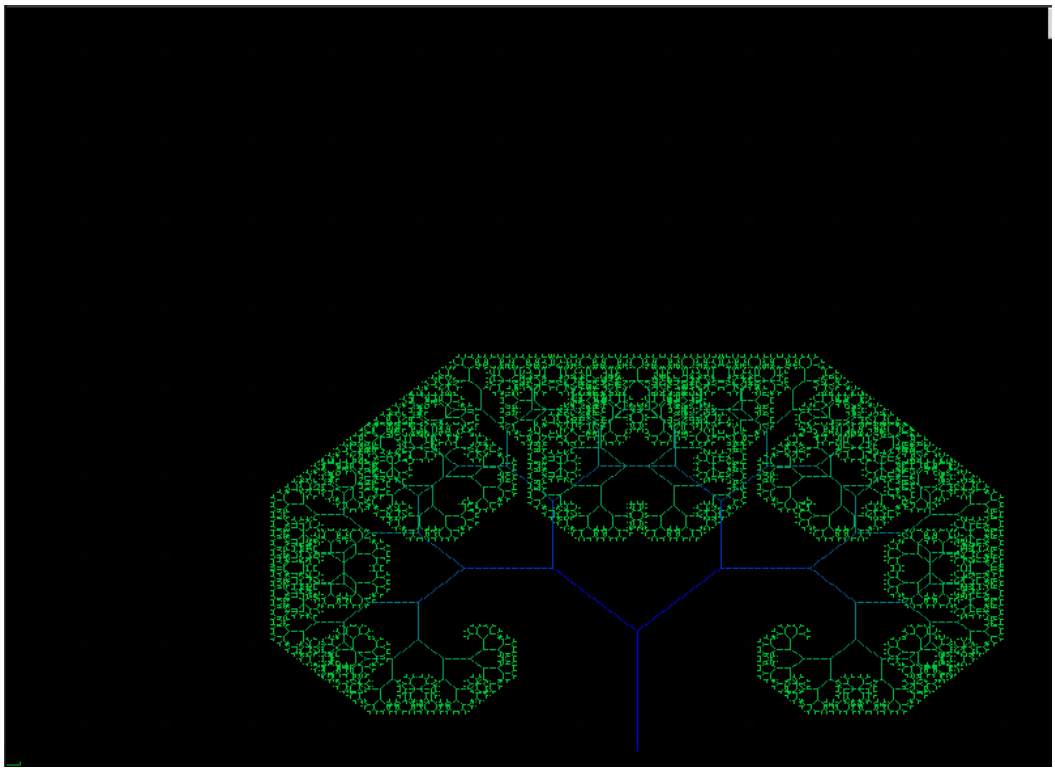

Figure 3: The fractal canopy with translations along the X, Y, and Z axis applied

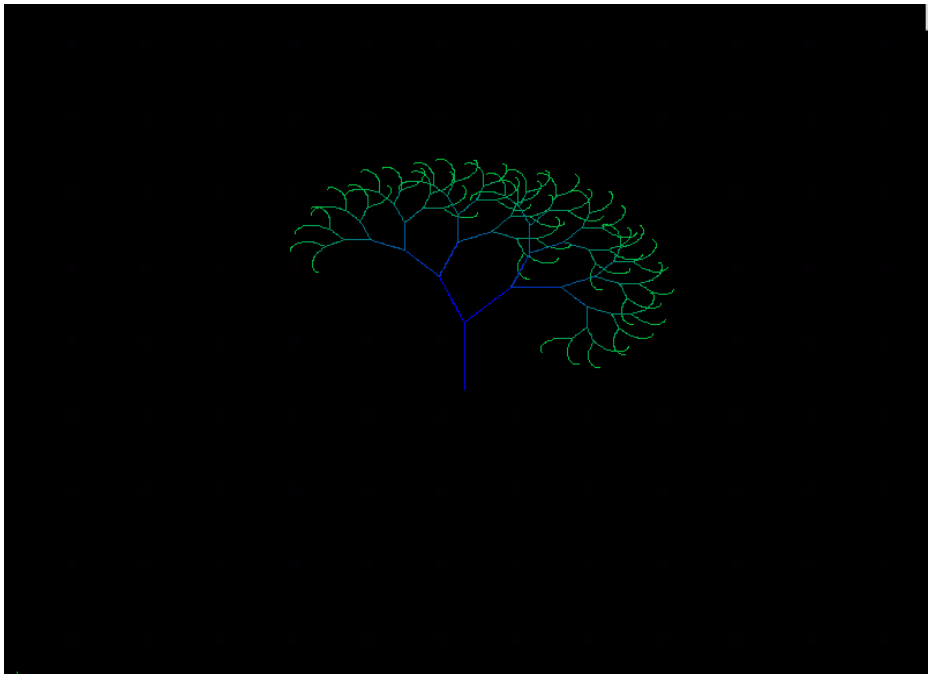

Figure 4: The fractal canopy with the right side minimum branch length increased and the left side angle decreased

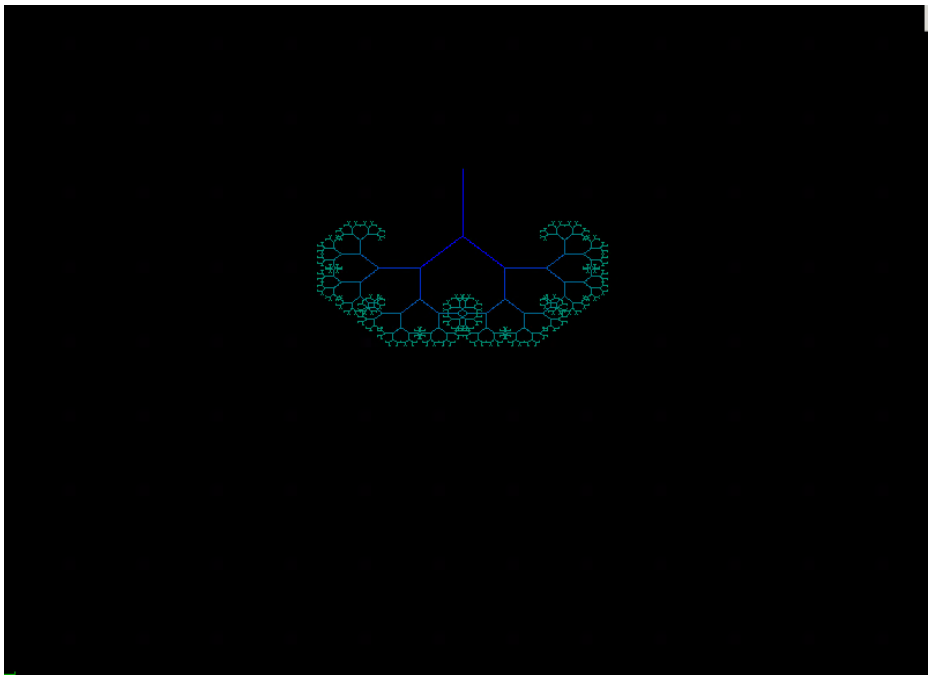

Figure 5: The fractal canopy with the branch reduction factor decreased and the fractal canopy growing from above

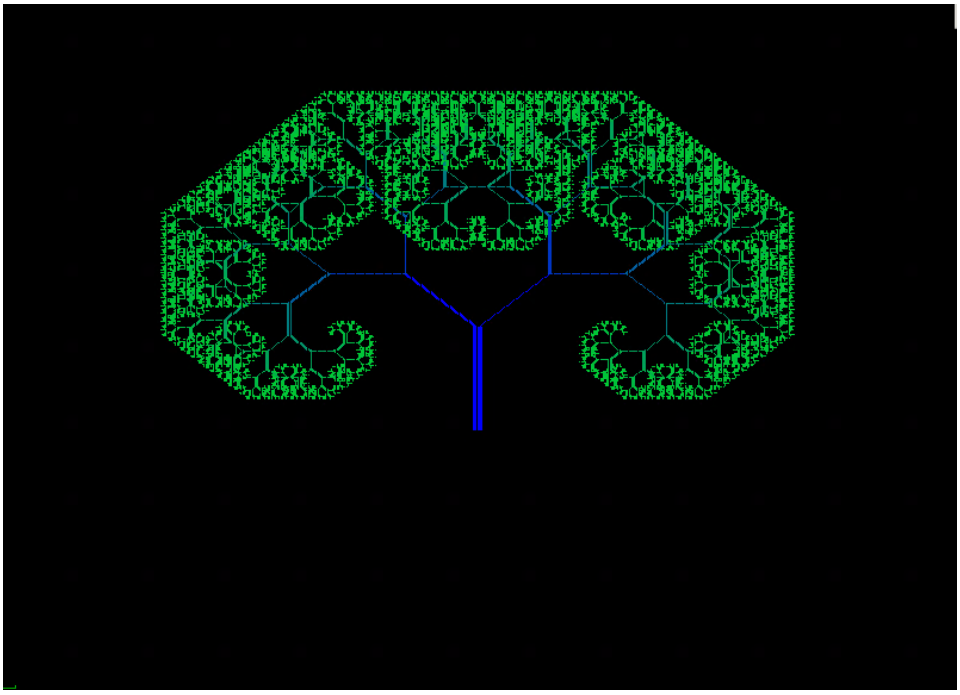

Figure 6: The fractal canopy zoomed in, with the left branches and trunk's thickness increased

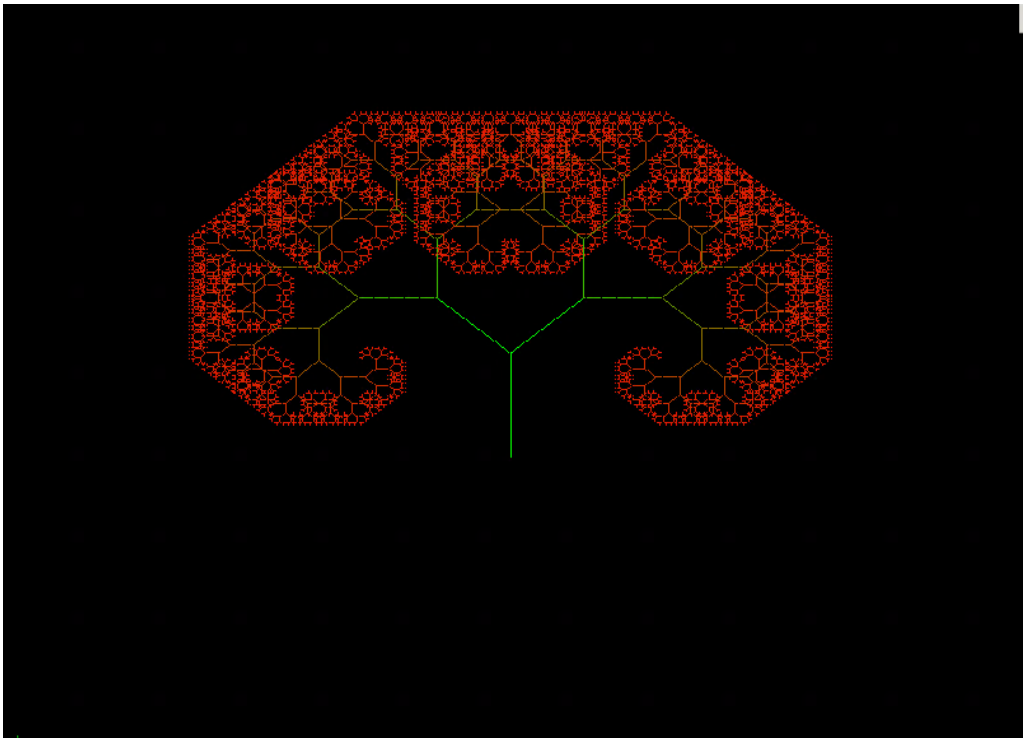

Figure 7: The fractal canopy zoomed in, with the color changed and the color gradient decreased

Results

After its completion, our application was able to handle all the requirements we set forth in the project topic writeup. Nothing was omitted from the writeup, but there were a few changes. For example, originally we were going to have all parameters inputted through the terminal. Changing these parameters would require the app to essentially restart to get new input, meaning it would be more time-consuming for users to fine-tune the canopy. So, instead, we made it such that all of these parameters could be changed dynamically with key presses. As a result, users can experiment and test out different parameters with the canopy without having to type in the exact values for every single value each time they want something changed.

We also modified how the tree would change color. Originally, we would have the user input a hexadecimal value that would apply to the entire tree. However, we decided to also add some dynamic capabilities to this feature. Now, users can choose a base color (by clicking ‘c’) and then increase or decrease the gradient of that color in the tree. This implementation made it such that it is easier (and frankly, more visually interesting) to see how the branches split off into sub-branches when compared to having a flat color.

Conclusion

In summary, we created an application that draws fractal canopies and allows the user to customize many factors of the fractal canopy. The user does so through different key presses dynamically during runtime. Users also have the option to revert some of their changes in case there was an error without having to restart the entire program. These capabilities are accomplished through the backspace (reset all transformations) and escape (reset all changes) button. We completed all requirements from the original project writeup and believe this application has the ability to be very educational and enjoyable to its users. We hope users are able to gain a deeper understanding of how fractals work and the impacts that varying parameter values have on its appearance.