Gavin Dassatti
CS 4250
Final Project Report
12/10/2021

Abstract:

One of the most effective ways to learn and fully understand a given process is to visualize, in real-time, what exactly the process is doing, and witness the individual steps the process takes to achieve its end goal. I wanted to somehow implement this method of learning into something that I have struggled with in the past as a student studying computer science. The ultimate goal for my final project was to develop an application that helps it's users understand the concepts and processes behind some of the most important sorting algorithms. In the paragraphs below, I will explain the technical approach I took to achieve this objective, and explain the individual features that were implemented to make this application as useful as possible.

Introduction:

The ability to comprehend the differences and similarities between different sorting algorithms and know when to apply a certain one to a given problem is something that takes time and practice to possess. In my several years studying topics in computer science I've tried my best to gain this expertise, but from my experience it can be difficult to actually understand what a certain sorting algorithm is doing just by analyzing the code that comes with it. Checking out the time and space complexity of these algorithms can help you to grasp what is actually going on, but actually being able to visualize the decisions being made by the algorithm during the sorting process is a game changer. This is where the motivation came from when I was deciding on what I wanted to develop for my final project. My overall goal for this project was to develop a visualization tool that anyone could use to help them further their understanding of some of the most important sorting algorithms. Also, in the process I figured I would also learn some things about these algorithms that I didn't know before by being the one who actually implements them and creates the visualization.

Technical Approach:

Building a sorting algorithm visualizer from the ground up was quite a task. At first I wasn't exactly sure where to start. My first task after getting my main function and glut function implemented was to create a function myInit() that generated a vector of size n and filled it with random integers from [1 … m]. This task was quite simple. All I did was first initialize the size of the array n, and then loop through the vector and fill it with random integers. Then I placed the function call in my main function before the window is created.

Because I was going to be developing this program in a 3D setting, my vision for this project was to display n (where n is the size of the array being sorted) stacks of cubes on a line where the line was perpendicular to the eye of the camera. I utilized the cube class that I tinkered with in my minecraft project to be able to scale and draw the varying amount of cubes that were going to be needed to make the program work. I decided I was also going to place a (3, n + 2) grid below the stacks to make it look a little better like the stacks were placed on top of something. In my Init() function, after getting the locations of all my shader variables and initializing the vertex array, I initialized points, a 4D vector of type cube* and size(n + grid), to store all of the information about the cubes that was going to be sent to the shaders. I then looped through a vector of type cube and initialized (n + grid) cubes and stored them in this vector so I can access them when I need to move them around and draw them in the display() function.

My display() function is where I do all of the transforming, scaling and drawing of the cubes as well as position the camera. So the first thing I did was initialize the at, eye, and up matrices for the Lookat function to orient the camera in the world. I gave the camera a default position upon startup, but by messing with the arrow and 'awsd' keys, the user is able to move around the world, zoom in & out, and check out the visualization from any viewpoint they please by setting global values to be incremented or decremented in the glut keyboard function. After the camera position is initialized, the next thing I did was draw the grid that the stacks were going to be placed on top of. So I looped through the (2 + n * 3) cubes and placed them up and down the x-axis in their correct position. Next was drawing the actual stacks. At first I thought I was going to place V[i] cubes in each stack to make the height of each stack correspond to the element at V[i], but I came up with a much better way of setting the height of each stack. All I had to do for each stack was draw one cube, transform it to it's correct position on the grid, and then scale it's y value by V[i]. Finishing the display function allowed me to visualize what the grid was going to look like and what the array was going to look like as stacks.

From here, I began implementing my first sorting algorithm which was bubble sort. I decided to create my bubble sort function that sorted V[] as a standalone function and then utilize the glutIdle function to call it when the user clicked on the bubble sort option in the drop down menu I made. When the user clicks the bubble sort option, a bool variable bubble_sort that is initially set to false becomes true. Then In the idle function all I had to do was check if bubble_sort was true and if it was, the function call bubbleSort(V) would start sorting the array, and then I would set bubble_sort back to false after the function call so it doesn't keep looping. Then to actually visualize what was going with the sorting of V[], I created a function color_compare that would quickly highlight the two stacks that were being compared in red right before the elements swapped. This made the visualization so much better because you're actually able to follow the algorithm as it goes. Then to display the swap, all I had to do was call my display() function right after the swap was executed, and because V was passed by reference the stacks would update on the screen immediately until they were sorted from tallest to shortest. Also, I added a delay call right after the display function in order to slow down the visualization enough to where the user could actually see what's going on. The user is also able to increment or decrement this value with the '+' and '-' keys to slow down or speed up the execution. And for the six other sorting algorithms I implemented after this (quick-sort, merge-sort, heap-sort, selection-sort, insertion-sort, and shellsort) I used the same exact process as I did for bubble sort. This process allowed me to implement a visualization for a total of seven sorting algorithms which was honestly more than I was expecting to get when I first proposed this project.

I also added a few other features to make the program as user-friendly as possible. Before each sorting algo function call in the idle function I keep track of the time so when the sort is finished it will display how long it took to complete in the terminal. I thought that this would be a cool feature to add to the application because users can compare the running times of the different sorts. Also, in the drop down menu, there are options to reset the array back to its original state and also generate a new random array. I thought the reset would be a good feature because then you can directly compare different algorithms by resetting the array everytime and getting run-time of each algo.

Future Enhancements:

One thing I really would have liked to implement into the visualizer was some type of light source, but I unfortunately was unable to figure how to get it working. I was having trouble setting up my shaders to account for lighting and I'm pretty sure I noticed I would of had to change around my cube class and I unfortunately didn't have time to go down that road, but if I ever start working on it again in the future that will be the first thing I will want to implement.

Another I would have liked to implement in the project was some type of audio that would play everytime two stacks got swapped. I tried using a library called irrKlang to play a split second of a quick swap like sound in the swap function I used but I wasn't able to get it working on the stocker machines and the documentation was also not very good. I tried out a few other solutions but I was unsuccessful, but I hope one day I could add audio into the program.

Conclusion:

Overall I had a great experience with working on this project. I had seen algorithm visualizers on the internet before, but never a 3D one so I thought it was cool that I was building something that I've never seen before. I learned a lot about the sorting algorithms I implemented which was a plus. I had to figure out exactly where to place the compare_color and functions in each implementation, which made me realize what is actually going on a little bit more in each one. I also learned a lot about camera movement and orientation during the development process as well. When I started implementing the functionality to allow the user to move the camera around I played around with the placement of the rotate and lookat functions when multiplying them together and it kind of opened my eyes and helped me understand how the camera view model actually works. If I could start over, one thing I would do differently would be to add all of the necessary components for a light source to the program during the early stages of the development process. Once I was basically done with my application and I decided I wanted to try to add a light source it was very difficult to do because I would have had to change around so many things and keep track of so many moving parts at once which caused me to not be able to figure out how to add one. Overall I had a great time with this project and I'm glad I learned the things I did in the development process.

References

I got all of the implementations for the sorting algorithms in my program from the sources below:

- https://www.geeksforgeeks.org/bubble-sort/
- https://www.geeksforgeeks.org/quick-sort/
- https://www.geeksforgeeks.org/merge-sort/
- https://www.geeksforgeeks.org/heap-sort/
- https://www.geeksforgeeks.org/selection-sort/
- https://www.geeksforgeeks.org/insertion-sort/
- https://www.geeksforgeeks.org/shellsort/