Computer Graphics Project Progress Report

Our project vision was to create a tool that helps visualize popular graph algorithms, especially some of the more complicated details that are hard to capture in standard textbook diagrams (e.g. time). In line with this vision, our main goal was to build an interactive and insightful program that visualizes breadth-first-search, depth-first-search, and shortest-path problems on two-dimensional graphs. We also had a goal to gain an intermediate understanding of how graphics work, how to create shapes and form larger objects, and how to handle user interaction.

In our plan, we proposed allowing the user to create and delete nodes and edges to make their own graph. However, after reviewing the feedback on our project plan as well as discussing and demonstrating the matter during the show-and-tell, we have decided to make this feature a stretch feature because it is not central to the core idea of our project vision: Illustrating graph algorithms. Likewise, we have shifted exploring different visualization techniques to a more central task. We have also scrapped the stretch task of allowing users to write their own algorithms because it is not computer graphics oriented. Additionally, we decided to lump implementing lighting effects into the "exploring different visualization techniques" task, though, looking at the course schedule, we may not cover the material soon enough for us to implement it correctly and timely. Further, we have added tasks for visualizing backtracking of the algorithm, implementing controllable animation speed, and having randomly generated graphs. The inspiration for these additional tasks comes from plan feedback, peer suggestions during the show-and-tell, and from gaining a better understanding after we began working.

Thus far, we have utilized the concepts we learned in class in completing the following tasks as listed on our initial plan: Represent the graph structure visually, add functionality for selecting start and end nodes, implement graph algorithms, illustrate the path of the algorithms, and implement panning and zooming. The resulting product from these completed tasks can be seen in the Appendices. Before any algorithm is run (see Appendix A), the nodes appear as red circles and the edges appear as black lines between the red circles. Clicking on a node (see Appendix B) will toggle it as the start node, colored blue. In the case of shortest-path, shift-clicking a node will toggle it as the end node, colored purple (see Appendix B). To run an algorithm, the user can press <b> for BFS, <d> for DFS, or <s> for shortest-path. Once an algorithm has begun (see Appendix C), the start node turns green, indicating it has been reached. From the start node, the edge on which the algorithm is currently traveling incrementally turns cyan in the direction of travel. Once an edge has been fully traversed, it remains cyan, and once a node has been reached it turns green. After the algorithm is finished (see Appendix D), the path remains on the screen until the user presses <q>, at which point the graph will go back to the way it was before the algorithm began. Pressing the arrow keys will translate the graph in the direction of the arrow, and holding <shift> while doing so will perform finer translations. Scrolling up or down on the mouse wheel will zoom in or out, respectively. This functionality provides us with a very basic graph algorithm visualizer and we intend to build upon it to make it more insightful.

Our next steps are visualizing backtracking, adding interactive animation speed, exploring different visualization techniques, and generating random graphs. We will also revisit customizable graphs if we have time. Our updated timeline is below:

Green: Completed Blue: Updated from plan Red: New addition

Start Date	End Date	Milestone	Description
2/16	3/02	Represent graph structure visually	Create two-dimensional shapes to represent both the nodes and edges.
2/16	3/02	Add functionality for selecting start and end nodes	Create event handlers for when the user clicks on a node.
3/02	3/09	Implement graph algorithms	Write code for BFS, DFS, and shortest path algorithms.
3/09	3/16	Illustrate the paths of the algorithms	Show the algorithm traveling along edges and reaching nodes.
3/02	3/16	Implement panning and zooming	Allow users to move around the graph in three dimensions to gain different perspectives.
3/16	3/28	Visualize backtracking	Show the algorithm go back to previously visited nodes, but indicate the node has been visited
3/16	3/28	Add interactive animation speed	Allow the user to speed up or slow down the animation speed
3/23	4/04	Generate random graphs	Create functionality that randomly generates graphs with given number of nodes and edges
3/28	4/13	Explore different visualization techniques	Test implementation of three-dimensional effects, various color and lighting gradients, and 3D planar surfaces

4/13	4/18	Allow graph customization (Stretch)	Test functionality of adding nodes and edges to see if it is feasible.
4/13	4/18	Produce a beta version of our project	Finish adding core features so that we can debug and make any needed modifications.
4/18	4/25	Produce final version of our project	Finish bug fixes and modifications for submission and presentation

As shown in the above timeline, we are on track with the original timeline we laid out in our plan. We have, however, added much work, so the coming weeks will be more strenuous than before. This should be fine because we have a much better understanding of our project since we began and we have laid the foundation for all the graph code, making it much easier for us to work with high-level representations of objects.

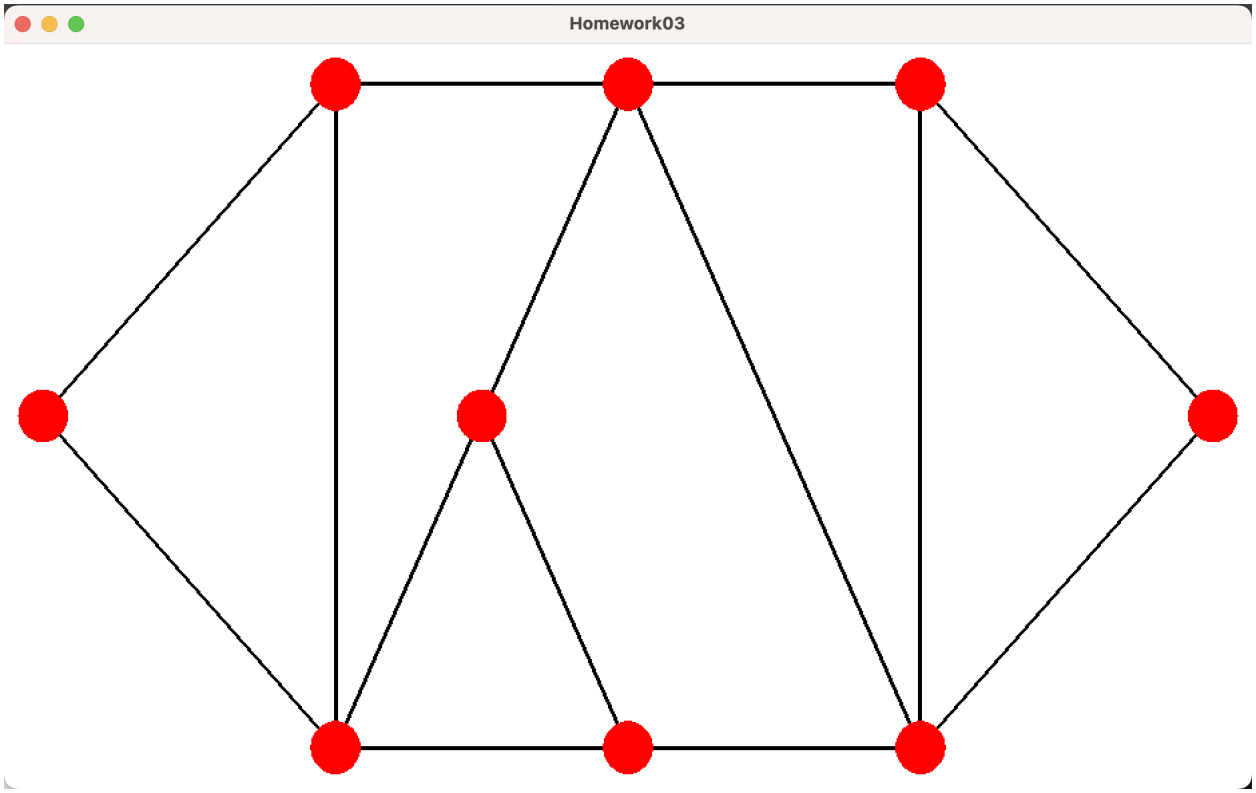
So far, we have found success in mirroring the work we have done across all of the homeworks within our project. To begin, we repurposed the gradle scripts from the homeworks to compile our code into executable files. We then copied the template used in the homeworks and tested it to make sure it ran. With the project running correctly, we used the material we learned in class and our homeworks as a guide for writing the code. We used the circle drawing function we learned in class to draw nodes and we used the GL_LINES mode to draw the edges. We then incorporated the Application-View-Model-MouseHandler-KeyHandler file architecture used in homework 3 into our project to allow for input handling of node selection and panning and zooming. Finally, we used the properties of vectors we learned in class to draw the partially traveled edges. With the basic structure and knowledge we have borrowed from homeworks and learned in class, filling in the project-specific details is much easier than starting from scratch.

One of the earliest decisions we had to make in our project was how to represent graphs as objects. We knew a graph could be represented as a collection of nodes and edges, but the biggest challenge we have had to face thus far has been how to represent an edge. We considered two main possibilities: A pair of two numbers, or a pair of two node objects. In the former, the numbers would represent the positions within the graph's node array of the nodes that are connected. The drawback to this is, if a node was deleted, the indices of all nodes to the right of it would shift down, making any edges that reference the old positions incorrect. In the latter, edges would just hold the addresses of two node objects. The drawback to this, however, is that when we access the graph, we are actually accessing a deep copy of the graph, to be in line with good coding practice. This means if we try to delete or update a node in the graph, we do not have the actual address of the node, just a copy of all of its attributes. This is ultimately the method we went with, and we created an equalsTo() function for the node class that checks if

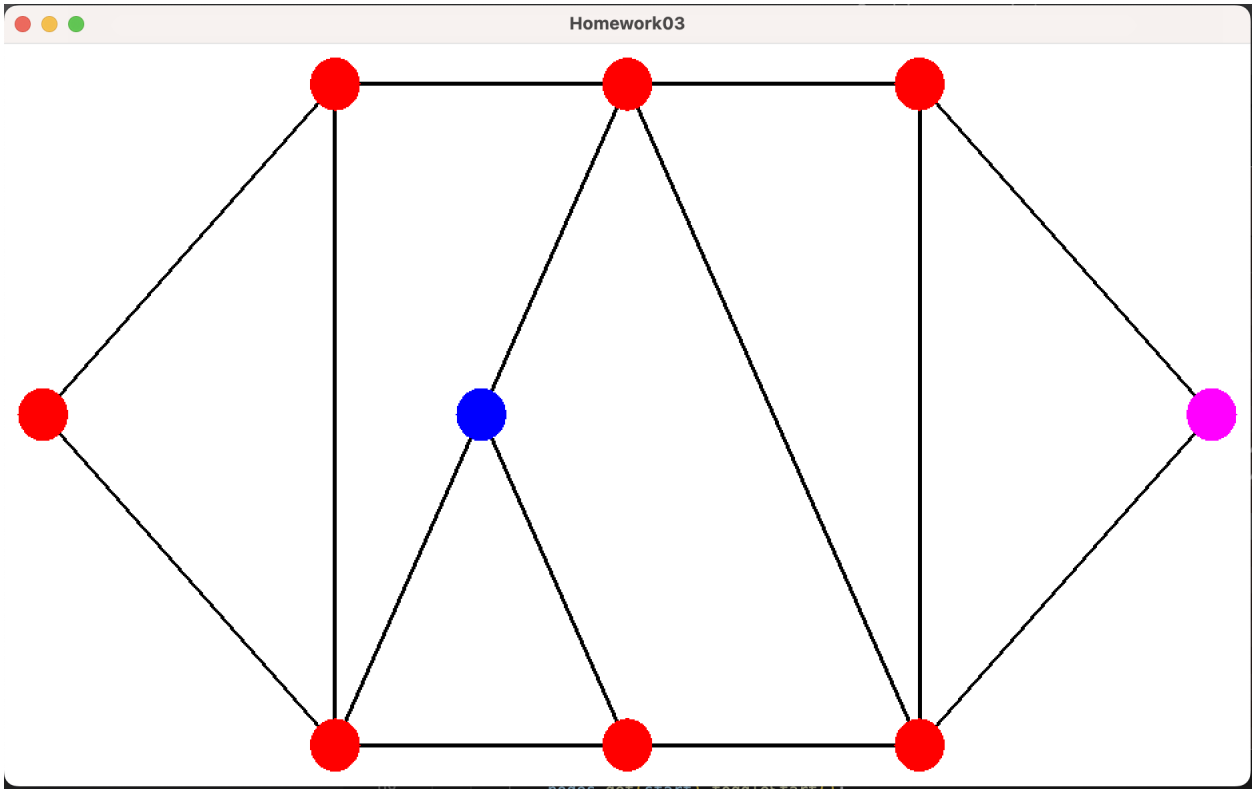
two nodes are in the same position, and if they are, returns true. We made this decision with graph customization in mind, but now that we have decided to put that to the side, it may make more sense to opt for the first method.

Our progress up to this point has been encouraging toward reaching our goals. Already, we have a basic working graph algorithm visualizer, but we intend to improve upon it still. Additionally, we have utilized the knowledge of graphics, shape drawing, and interaction handling we learned from class and homework to build the program, which was our other main goal. We will continue to follow the updated timeline and keep our goals in mind as we progress.

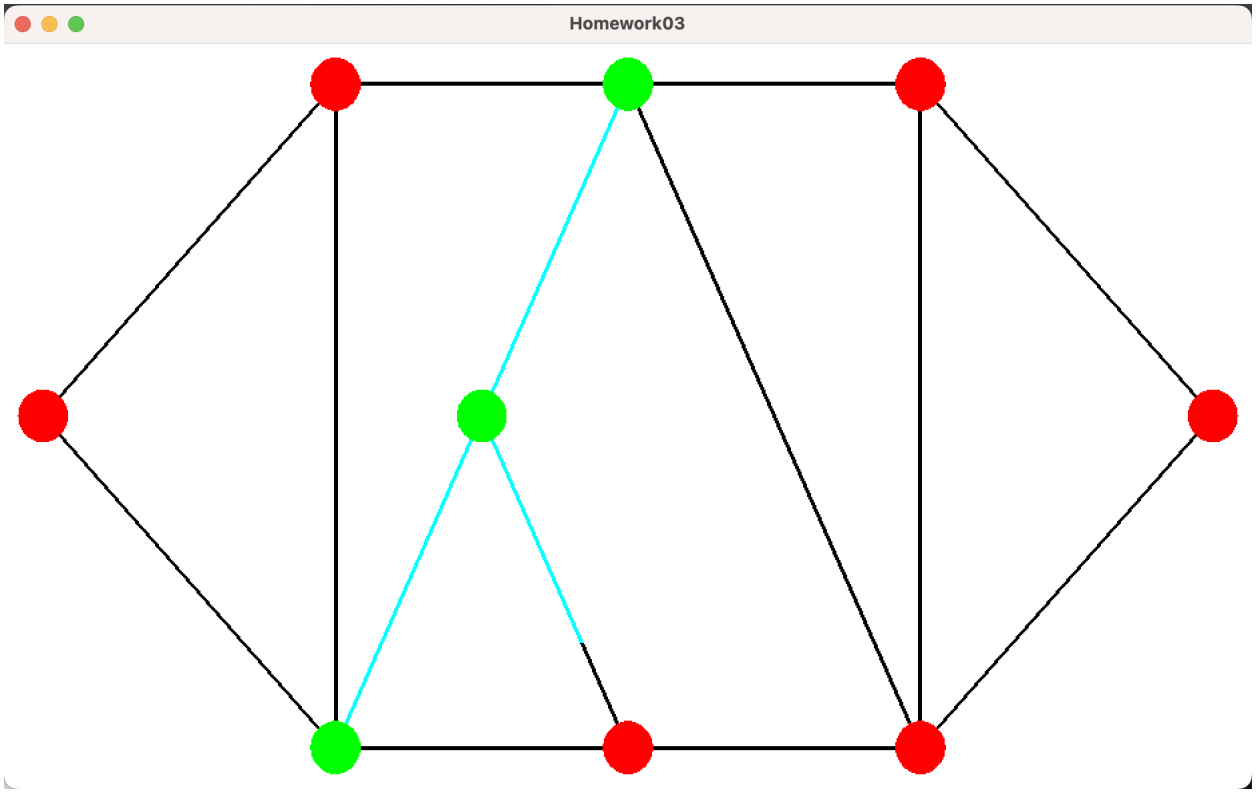
Appendix A



Appendix B



Appendix C



Appendix D