Graphics

Kynan Justis

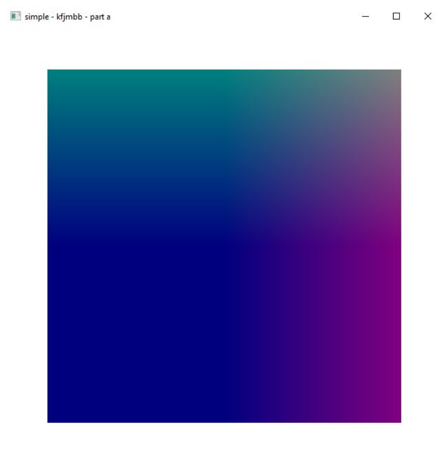
Assignment 3a

03/22/2018

Overview

In this programming assignment, there were three major parts. Part A involved downloaded the sample code and setting it up to compile. Once this was done, a colored, three dimensional cube would be drawn to the screen. Next, in Part B, the task was to add some basic key callbacks to modify the camera's field of view and aspect ratio. The values for the near and far clipping plane were also made to be adjustable through key callbacks. Finally, Part C involved using more callbacks to manipulate the cube or the camera within the window. The cube/camera was made to translate and rotate along the X, Y, and Z directions. For the rotation specifically, a mouse-drag feature had to be added on top of the key callbacks used for rotation in order to allow users to interact via the mouse. Having completed all of that, the last requirement was to allow the camera to zoom in and out, again using key callbacks.

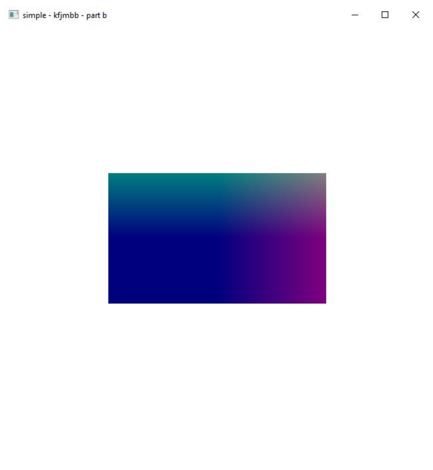
Part A



Similar to previous homework assignments, this part was mostly for setting up the development environment for the rest of the project. Sample code had to be downloaded from the course website and added to the project such that our code editors could compile the program.

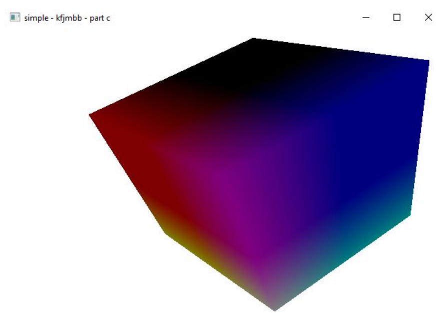
With my environment, it was fairly straightforward since Microsoft Visual Studio was used along with *vcpkg*. Once all of the header files and shader files were added to the project subfolders and *glm* was installed with *vcpkg*, everything compiled successfully. The image above shows the display window after the program is first ran.

Part B



Part B was a little more involved than the previous part. In this section, the task was to implement key callbacks to allow the user to change the field of view and the aspect ratio of the camera. For this program, the field of view can be increased and decreased with the UP and DOWN arrow keys. Similarly, the aspect ratio can be increased and decreased with the RIGHT and LEFT arrow keys. The simplest way to go about implementing this was to maintain global variables for the state of each property. Then, before the cube is rendered, default values that match the original program from Part A are fed in. However, since the globals get hooked into I/O through the key callbacks, their values can be manipulated during the render loop. This ultimately causes the changes to display during run-time after the correct keys are pressed by the user. The same implementation had to be made available for the near and far clipping planes too, however, those are controlled with the “< >” and “[ ]” keys respectively to increase and decrease their values.

Part C



Finally, Part C probably involved the most amount of extra code. More key callbacks had to be implemented in order to allow the cube to be rotated and translated along the X, Y, and Z axes. However, instead of just key callbacks, a mouse-click and mouse-position callback had to be implemented as well. These callbacks would allow the cube to be rotated by dragging the mouse across the screen instead of limiting the user to just the keyboard. On top of these features, the ability to zoom in and out with the camera position itself also had to be made available. To make the program a little easier to use, a *reverse_mode* state was implemented. For example, the keys that control the X/Y/Z translation are mapped to A/S/D respectively, but these only add values in the positive direction. To undo the changes made with these keys, the LEFT SHIFT button can be pressed to activate *reverse_mode*, which allows for subtraction from the current values, thus reversing the translations that were previously applied or to simply allow movement in the opposite direction. This feature works for the X/Y/Z rotation keys as well. On top of that, the ability to reset the entire state of the cube/camera was added through a callback to the R key on the keyboard. This allows the program to essentially reset without having to be re-ran for further testing.

Note

The final version of the code, the full-color result images, and the Visual Studio build instructions can be found online at https://github.com/kfjustis/graphics-3a.