CSCE 4813 – Programming Project 2
Due Date – 02/20/2019 at 11:59pm

1. Problem Statement:

The goal of this project is to create a video game called "BlockWorld" that allows the player to interactively create and display a virtual world by stacking blocks on top of each other. Our ultimate goal is to create models of castles like the one below. For this project, we will focus on: 1) implementing user interaction that allows the player to move around the virtual world, and 2) storing the geometry of the scene in a data structure that is easy to manipulate and display. Later in the semester (after we have learned texture mapping) we can do step 3) display realistic renderings of blocks in the scene.

2. Design:

User Interaction

To keep things simple, we will use keyboard callbacks to control the navigation and actions of the player. When the player types 'F' the game will go into "fly mode" and the player can "fly" around the scene by pressing the 'X', 'x', 'Y', 'y', 'Z', and 'z' keys. Each time type 'X' they go one step in the positive X direction, and each time they type 'x' they go one step in the negative X direction. Similarly, 'Y' and 'y' move the player in the Y direction, and 'Z' and 'z' move the player in the Z direction.

What should we do if the player flies into a block? The easiest answer is nothing. This will allow the player to fly through walls like superman (but with less damage). What should we do if the player flies past the max or min (x,y,z) coordinates of the virtual world? If we do nothing, the player might have trouble finding their way back again. A better solution would be to stop their motion one step away from the max or min (x,y,z) coordinates, like they have bumped into an invisible wall.

Another option would be to have the player's location "wrap around" so they appear on the opposite side of the scene. Either option is fine.
Aside from moving around the virtual world, we need to have some way for the player to change how the BlockWorld model is viewed on the screen. The easiest way to do this is to use keyboard commands again. When the player types 'R' the game will go into "rotate mode". Then when they type the 'X', 'x', 'Y', 'y', 'Z', and 'z' keys, we can increase or decrease the corresponding (x,y,z) rotation angles, and use this information to update the GL_MODELVIEW matrix.

Finally, we need some way for the player to add blocks in the virtual world. When they type '+' we can add a block to the virtual world at the player's current (x,y,z) position. Since the player is allowed to fly anywhere in the scene, there is a good chance that there are no blocks below the current location to "hold up" the new block. To keep our game simple, we can simply ignore gravity and let the player add blocks anywhere they want. When the user types '-' the game should subtract the block at location (x,y,z) from the virtual world. Doing this is more complicated because it requires a search/delete from the block data structure. For this reason, this should be the last user interaction you implement.

Geometric Modeling

To simplify the modeling and display of objects in our virtual world, we will only be using 1x1x1 cubes as building blocks, and restrict their orientations so their faces are aligned with the X,Y,Z axes of the virtual world. By doing so, we now only need to keep track of the integer location (x,y,z) where the center of the block is located. Now we just need to decide how to store this information.

One option is to store the (x,y,z) values for all block centers in arrays or vectors. This way, when the player adds a block, we add the (x,y,z) coordinates to the end of the array or vector and increment the block count. Another option is to use a large 3D array to represent all possible center locations, and store 0's in (x,y,z) locations that are empty, and 1's in (x,y,z) locations that currently have blocks.

To draw the cubes in the scene, your display callback should loop your block coordinate data structure and draw solid colored cubes at the appropriate (x,y,z) locations. You may want to use slightly different colors for the 6 faces so the blocks look more three dimensional.

3. Implementation:

This semester we will be using C++ and OpenGL to implement all of our programming projects. The instructions for downloading and installing a Linux VM and installing OpenGL are posted in README file the "Source Code" page of the class website. Once you have OpenGL installed, you can compile your graphics program using "g++ -Wall firework.cpp -o firework -lGL -lGLU -lglut".

You are encouraged to look at sample OpenGL programs to see how the "main" function and the "display" function are normally implemented. As always, you should break the code into appropriate functions, and then add code incrementally writing comments, adding code, compiling, debugging, a little bit at a time.

Remember to use good programming style when creating your program. Choose good names for variables and constants, use proper indenting for loops and conditionals, and include clear comments in your code. Also, be sure to save backup copies of your program somewhere safe. Otherwise, you may end up retyping your whole program if something goes wrong.

4. Testing:

Test your program with different user inputs until you get some images that look fun/interesting. Take a screen shot of these images to include in your project report. You may also want to show some bad/ugly images that illustrate what happens if there is a problem somewhere. You can discuss how you corrected these problems in your project report.

5. Documentation:

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Be sure to include several output images. Finally, describe any known problems and/or your ideas on how to improve your program. Save this report to be submitted electronically via Blackboard.

6. Project Submission:

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted. When you have completed the tasks above go to Blackboard to upload your documentation (a single docx or pdf file), and all of your C++ program files. Do NOT upload an executable version of your program.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late,
- 20% off if less than 2 days late,
- 30% off if less than 3 days late,
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

7. Academic Honesty Statement:

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are NOT allowed to distribute code to each other, or copy code from another individual or website. Students ARE allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance.

This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a ZERO on the programming project, an F in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.