COMPSCI/SFWR 3GC3: Computer Graphics Assignment 2 — 2D Robot and Solar System

Due: October 11th, 2020 at 11:59am (Midnight).
Accepted Late until October 16th, 2020, 12:00pm
This assignment is worth 7.5% of your final grade.

Part A — Robot Program

The purpose of this part of the assignment is to make an user-controlled character in a 2D plane and to write callback functions, and write your own raster algorithms.

Write a simple robot program in C++ using OpenGL/GLUT. The program will draw a user selectable grid with minimum size 10x10 and max size 50x50. A random square is to be filled in with *green*. Right next to it will be a *red* square. The green square represents the body of the robot, and the red square represents the head. As the robot turns the red square *will rotate around the green square in the grid.* The robot will be able to look in 4 directions, up, down, left, right. The green square will be its position.

You will need to *keep track* of both the current position and orientation of the robot. You must implement your own drawing function (i.e., see our discussion of raster algorithms). In other words, you cannot use the built-in primitive types such as GL_LINES, GL_POLYGON, etc, except for points. In all cases, *only use the point* (GL_POINT) primitive type. This means you ***must*** write *your own functions* to draw *lines*, *rectangles,* and any other primitives you need to use, using points. *Note that this is to learn how OpenGL encodes its raster algorithms.*

Your program should include the following functionalities:

a) Drawing the primitive types above with your own drawing routines

b) Pressing “Q” or esc should quit the program

c) Random robot position

d) Reset Robot Position and orientation

e) Left click will open a menu with the commands move and turn. It will then have a 2nd level of menu for right left up down, turn right, turn left.

f) User selected grid size

- This should be done using GLUT menu items (ex. One menu item is 10x10, the next is 20x20, and so on)

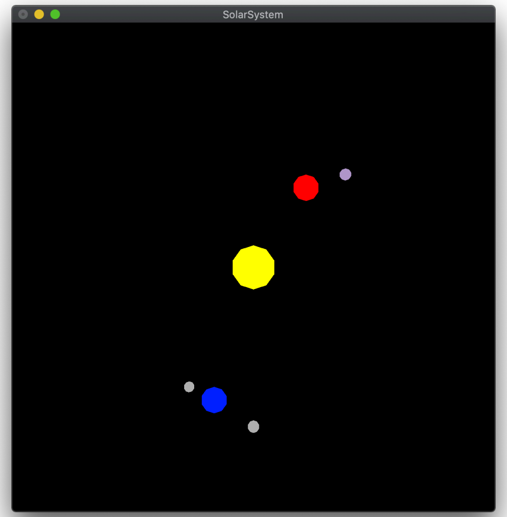
Note: Whenever you scale the grid size, the robot should appear and navigate only within the *bounds of the grid*. The reset and random robot positions should only draw the robot within the bounds of the grid.

Use makefile so that when ‘make robot’ is called, a program called “Robot” will be compiled. Place the makefile and *.cpp files into your *A2* folder.

Part B — Interactive 2D Solar System

The purpose of this program is to create a 2D interactive solar system and to become familiar with matrix hierarchies, transformations, simulations, and more OpenGL/GLUT functionalities.

You will find a SolarSystem.cpp file in your repository under A2. Note that this is written in code that is for a 3D system, however, the transformations and the rest of your code should not affect or use the Z-plane. Running SolarSystem.cpp shows the sun in the middle of the screen.



Write a simple 2D interactive solar system program in C++ using OpenGL/GLUT. When your program starts, it should only have the sun in the centre. The user can add or remove planets and moons. Each planet and moon should have a colour. Each planet will revolve around the sun, and each moon should revolve around their planet. Each moon should have their own orbit around their planet and each planet should be in their own orbit around the sun. Each planet should be smaller than the sun, and the moons should be smaller than the planets they revolve around. And, each planet and moon, should also have their own rotation. You should *only* use the *drawSphere()* to draw the moons and planets. Your solar system should be able to support at least three planets.

Your program must include the following functionalities:

a) Pressing “R” — resets the solar system.

b) Pressing “Q” — quits the program.

c) Pressing “P” — pauses/resumes the simulation.

d) Spacebar — Starts/stops the selection mode. The selection mode can start wherever you choose.

e) Left and Right arrow keys — moves the selection from planet to planet. Selected moon/ planet should be coloured green.

f) Shift + L/R arrow keys — Cycles between selecting the moons of a selected planet.

g) “D” — Deletes selected planet/moon. If no planet/moon is selected, then the last planet is deleted. If the user deletes a planet, then all of its moons will be deleted as well. If the user deletes a planet that is in the middle of the solar system, the other planets should shift over. Note, the sun should not be deleted!

h) “A” — If there are no planets selected, it will add a new planet. If a planet is selected or if a planet’s moon is selected, it will add a new moon to the planet.

Additionally your program must include **two** out of the following basic functionalities:

a) Zoom in + zoom out — Pressing two buttons of your choice should zoom into your solar system, and zoom out.

b) “+”/“-“ — If a planet or moon is selected then their revolution speed should increase/ decrease speed. If no planet is select, the revolution speed for every planet and moon should increase.

c) “<“/“>” — Increases/decreases the size of the selected planet or moon. If no planet or moon is selected, all planets and moon should decrease/increase size.

d) Click to select — the mouse buttons should select/deselect the planet it intersects.

You may instead include **one** advanced functionality (instead of the two basic functionalities):

a) Make it possible for planets or moons to collide. You can decide what happens when they collide, such as: They explode into particles, they change orbits, or the planets and moons are flung out the solar system and destroyed.

b) Camera control — should freely move the camera around the scene. Note, this will require knowledge in 3D graphics. For this, you will need to manipulate the *gluLookAt()* parameters.

The sizes, colours, rotation and revolution speeds are all up to you!

Use makefile from part A so that when ‘make solarsystem’ is called, a program called “SolarSystem” will be compiled. Place your source files into A2 folder and update its makefile.

Update your makefile so that when ‘make’ is called both Robot and SolarSystem will be compiled and generated by using the robot and solarsystem targets.

Have fun and be creative! Don’t be afraid to experiment with other functionalities!

Submission Notes

All of the above functionality is required for Assignment 2.

This assignment is to be **completed individually**!

You have the option of implementing your assignment on your platform of choice, BUT please make sure **your programs can be fully compiled and executed on the VM Image provided in class**.

Submit your source code, readme, any tests you conducted for your library, and any other resources to *A2 folder of your Git repository*. All source code and makefile should be located in A2.

Your readme file should provide details of:

- Any external resources. Please cite and reference any and all websites, online content, external books, and other material that helped in your assignment.
- Any other details relevant to getting your program to run. If the TA cannot get your code to run, your assignment will not be marked!

Familiarize yourself with the department's policy on plagiarism and the university regulations on plagiarism and academic misconduct. **Plagiarism will not be tolerated, and will be dealt with harshly**.