Alexandria University

Faculty of Engineering

Computer Graphics

Course Project

(Solar System Simulation)

Name: Toka Ashraf Abo Elwafa Ahmed

ID: 19015539

Problem Statement

You are required to implement an application that simulates the solar system. Also, you should enable the user from controlling spacecraft to explore the solar system. You are required to use two view ports: One for spacecraft and the other for the whole solar system (See figure 1 for more declaration).

In simulation you need to handle:

• Instantiation of Sun and 8 planets (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune)

• Solar system animation (spinning and rotation of planets around Sun and Moon around Earth)

• Space-craft movement.

• Lighting and emission.

You are free to choose the proper implementation of:

• Planet sizes (make sure it is sensible)

• Planet colors or (textures) (make sure it is sensible)

• Mouse and keyboard interaction (make sure it gives good user experience).

Code Flow:

1- An object (from the class of planet) for each planet of the eight planets is created with its own data (its color of (R, G, B), speed of rotation around the sun, the orbit radius, and the planet radius. There is also an additional one that concerning the planet of

"Saturn" as it has a ring around itself so in case of this planet the arguments of ring radius and ring thickness is added).

Planet Class:

- It initializes all the previous mentioned variables to zero and when making an object from it, each variable takes its new value and put all in an array.

2- Draw Scene Function:

- It divides the screen into two viewports. The left one takes all the screen and has an elevation view of the solar system and the right one takes the right bottom ¼ of the screen and has a plan view of the solar system and the spaceship.
- In the right port, there are the spacecraft(spaceship), and two lines are drawn around the viewport to separate the two viewports.
- In each port a call to the function of drawSolarSystem is made to draw the system.
- gluLookAt built in function is used to determine the camera for each view port.

3- drawSolarSystem Function:

- Firstly, it calls the function of drawSun to put the sun in the solar system, then loop around the array of the planets and draw each planet with its own data stored by function of drawPlanets.
- Secondly, some calculations are made to calculate the moon orbit around the earth by getting the x and y positions of the earth to calculate the values of x, y, and z of the translated moon and make the moon rotation angle is the elapsed time divided by 1000 and multiplied by 50.and finally, draw the moon by the built in function of glutSolidSphere.

4- drawPlanets Function:

- ❖ Firstly, it calculates the planet position using the orbit radius and angle theta (time * speed of the planet) to use it in translating the planet then rotate by theta in z direction and draw the planet using glutSolidSphere.
- ❖ Secondly, drawing the orbits itself using GL_LINE_LOOP and calculating the x and y positions in each angle of the 360 degrees.
- ❖ Finally, drawing the ring around the “Saturn” planet using the built in function of glutSolidTorus that takes the thickness and the radius as parameters to draw the ring.

5- drawSun Function:

- ❖ It draws the sun as described before about the planets then enables the lighting and defines the arrays of ambient, diffuse, specular, and emissive the enables the material of each one on the sun.
- ❖ It also defines the light position, its color(diffuse), ambient, and specular then set these values to the light source of GL_LIGHT0

6- SetUp Function:

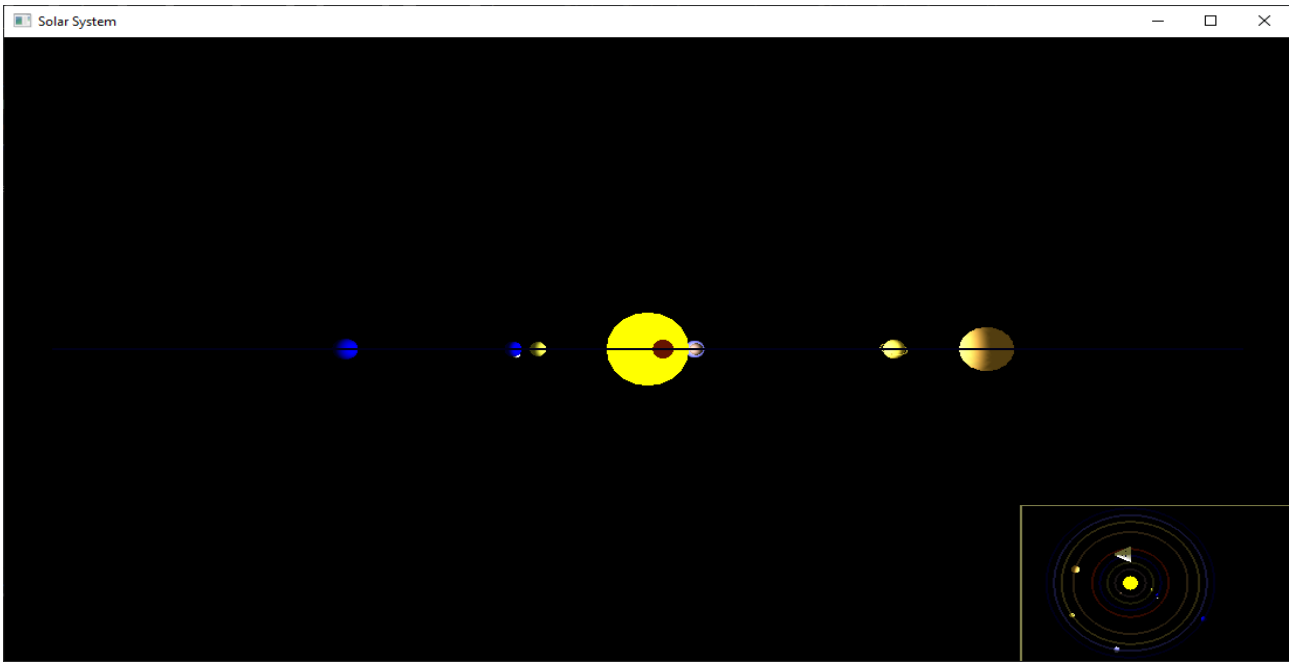
- ❖ It defines the list of the spacecraft the materials of planets colors.

7- Mouse and Keyboard Interaction:

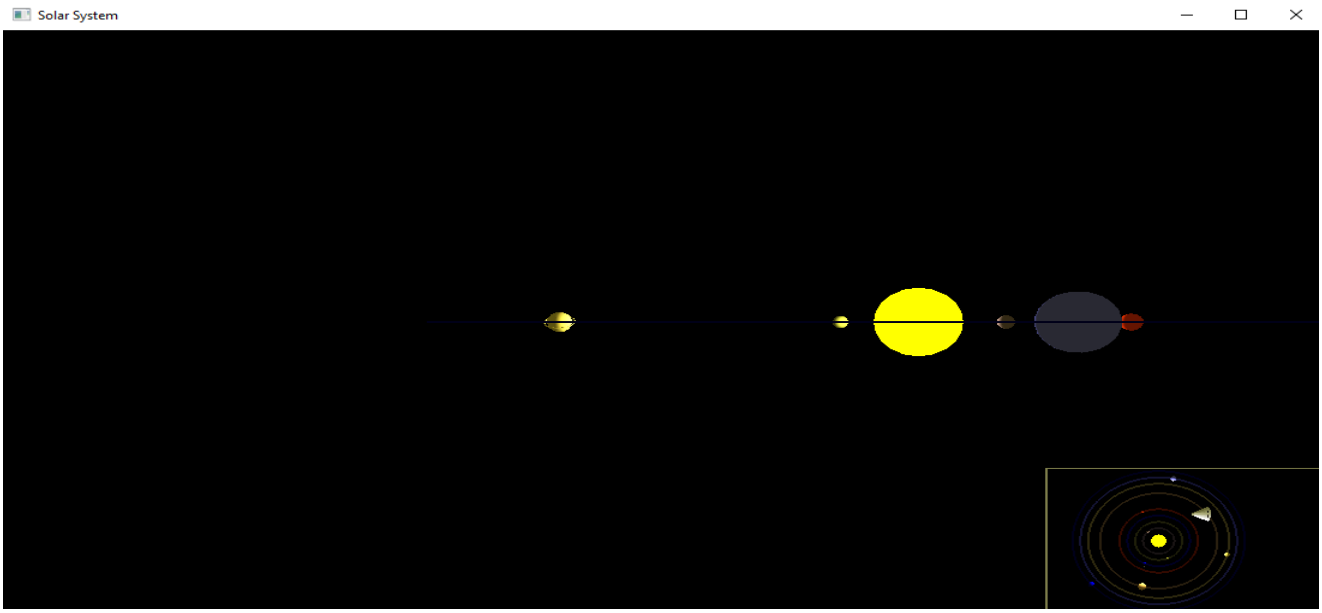
- ❖ When pressing the right arrow key, it increases the value of y in the camera of the left viewport, so the solar system moves to the right. Also, the left arrow key decreases this value and moves to the left.
- ❖ When pressing up arrow key, it decreases the value of z in the camera, so it appears as a zoom in and the down arrow does the opposite so appears as a zoom out.

- ❖ When pressing the left button of the mouse, it increases the value x in the camera and allows the user to see the scene from downside. Also, the right button of the mouse does the opposite so allows the user to see from upside.

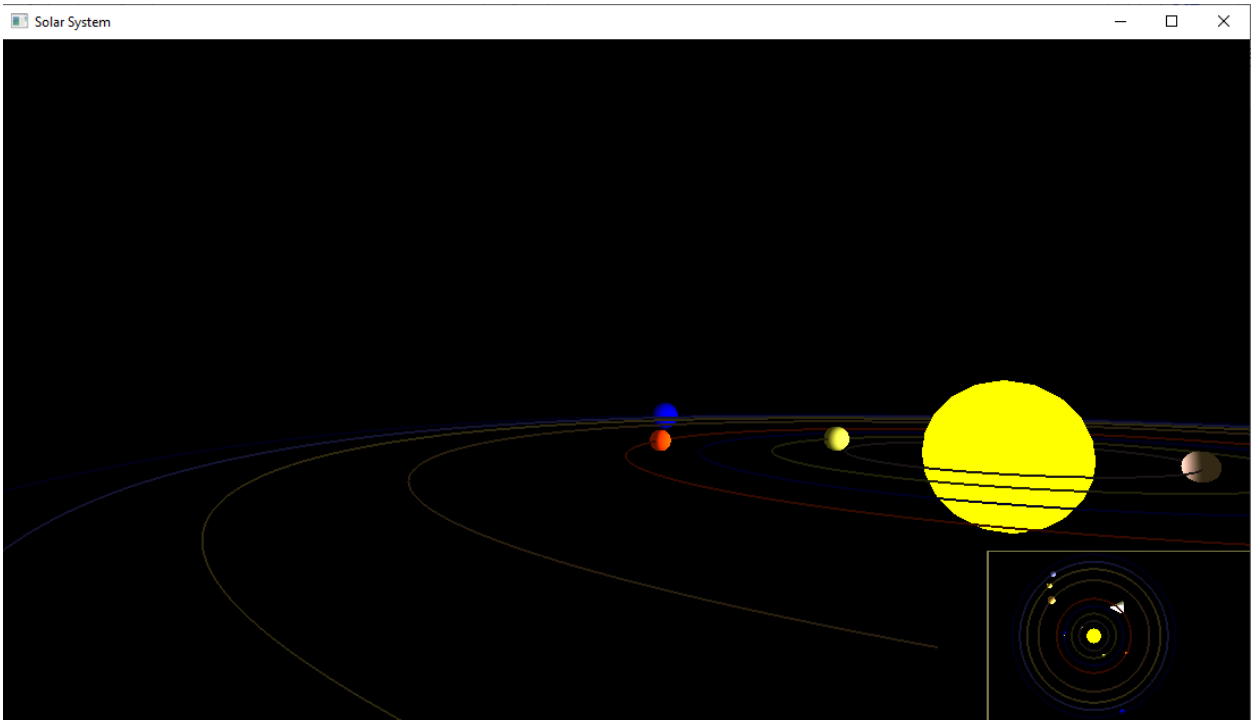
Sample Runs:



After pressing right arrow:



After pressing up arrow and left click of the mouse:



After pressing down arrow and right click of the mouse: