CREATING A SMALL SOLAR SYSTEM IN C++

DH2323 Computer Graphics and Interaction

Project Report

Isabella Johannesson
(TIMTM) *Interactive Media Technology*
ijohanne@kth.se

Moa Winberg
(TIMTM) *Interactive Media Technology*
moawin@kth.se

ABSTRACT

This paper describes the implementation of a small solar system in c++ using OpenGL and GLUT on macOS. The objects used are spheres and 3D vectors. The transformations made consists of a number of translations and rotations for the objects and camera view. The Phong Reflection Model was added to make the environment look more realistic, and keyboard input was added to make the system more interactive to the user. The paper also discusses what could be improved in future work.

1. INTRODUCTION

The goal of this project was to create a small solar system in c++ using OpenGL and GLUT. The solar system should contain the sun and more stars, and the earth rotating around the sun. The sun and the earth should be 3D objects and the user should be able to control the camera angle.

The process of the work is described in more details on the blog found at:
https://computergraphics.blogg.se/

2. BACKGROUND

2.1 OpenGL

Solar system simulations are in computer graphics usually modelled using OpenGL. OpenGL are prominent in cases where you want to create interactive application that render high-quality color images composed of 3D geometric objects and image.

2.2 GLUT

GLUT(OpenGL Utility Toolkit) is the window system independent toolkit for writing OpenGL programs. The library contains a number of functions that draw simple geometrical primitives, such as sphere, cube, cone, and torus. [1]

2.3 Transformations

The goal in such systems is often to implement some sort of transformation for solely creating a life-like orientation. In the OpenGL SuperBible, Wright [2] presents that modelling transformation is used for object manipulation, and allows you to: rotate objects, move them about, and even stretch, shrink and warp them.

2.4 Lightning

Wright [2] presents different methods of how to simulate light, using simple diffuse lightning, the point light diffuse shader, the ADS light model and Phong Shading. The Phong Shading is appropriate when we are working with spheres, as we interpolate surface normals between

vertices.

2.5 Related Work

There are several works done on the same topic, but as we mentioned Richard S. Wright, his project is of interest. He is behind the Bisque software, Seeker, a 3D solar system simulator. The application uses a custom OpenGL to generate a realistic and animated view of the earth more than 60 times a second. This includes atmospheric effects, the sun's reflection in the water, and even stars in the background.

3. IMPLEMENTATION AND RESULT

3.1 Getting started

The idea which we specified before starting the implementation included using SDL2, glm such as in the labs, and possibly openGL to implement the project. However, it turned out when using that SDL2 was unnecessary to use since we render the window and objects by OpenGL.

The IDE used was Xcode since we both use macOS. OpenGL and GLUT already exist on macOS all we had to do to get started was to include the frameworks in our Xcode project. The first step from this was to set up the main function [3], initiating the other functions, and render a window.

3.2 Main objects

The objects were two spheres for the sun and earth, and 3D vectors for the stars. They were initiated by separate functions called in the display() function. These were initiated in two separate functions, called in our display() function.

In OpenGL, spheres can be created in different ways. We started with the function glutSolidSphere() [4]. However, if adding texture to the sphere, it has to be created as a quad object first and then sent to the function gluSphere().

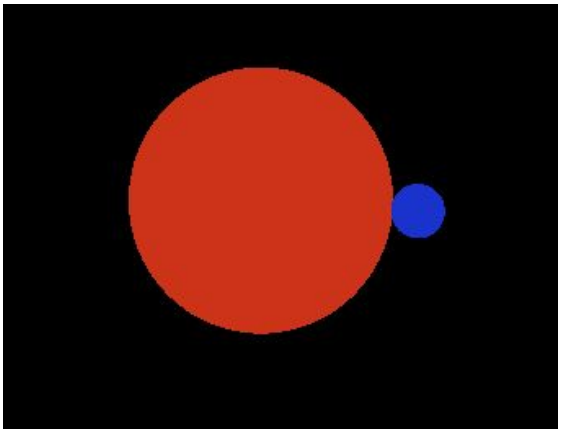

Image 1. The two spheres.

The code for creating 3D vector stars was taken from the source code in Lab 1 render track. The values for x, y were randomly chosen between -1 and 1, and between 0 and 1 for z to spread out the objects and value for color. They were then rendered with OpenGL by GL_POINTS with vertices in each dimension.

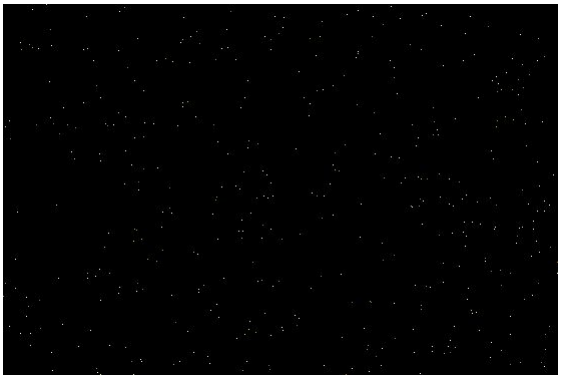

Image 2. The stars.

3.3 Transformations

The earth was moved a radius distance to the left of the sun in x and z. It was then

rotated in a circular motion around the sun by first translating a radius away from the sun, rotating around the y-axis, and then translated back to the original position using the OpenGl built-in functions glTranslatef() and glRotatef(). The angle in which the rotation was made was updated with 1 in the idle() function to create the circulation. In order to create depth in the view we had to enable GL_DEPTH_TEST otherwise it would have been seen as the earth rotated in front of the sun and not around.

The camera position in z-led changed with keyboard input (arrow up/arrow down). In order to achieve this the whole view changed in z-led with a translation depending on which key was pressed. The rotation changed with keys arrow left/arrow right and the function gluLookAt().

The code to handle the keypresses in the function processSpecialKeys() and rotation of the camera with gluLookAt() and the new coordinates were *entirely* from the tutorial [5]. The code used was short and precise and rewriting it felt unnecessary for this task.

3.4 Lighting

The light source was added with OpenGL's built-in function glLightfv() and material light was added with glMaterialfv [6]. We used the Phong Reflection model with ambient, diffuse, and specular light added together. The light source was positioned in origo in order to make the light come from the sun.

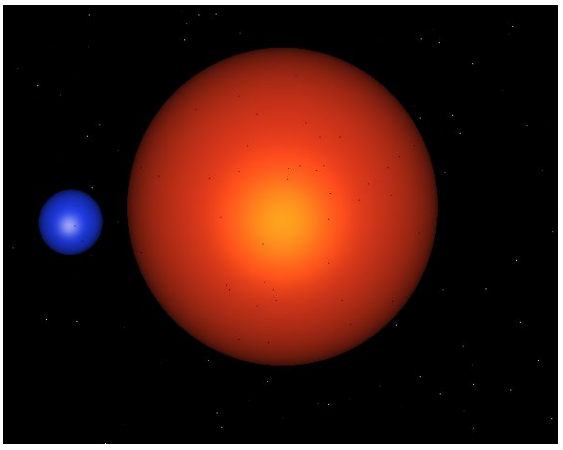

Image 3. The final system.

4. DISCUSSION

This project led us to explore how to use OpenGL with GLUT, which none of us had worked with before. We also learned how to use translation and rotation in more detail. The most challenging part was to create the rotation of the earth around the sun before we found out we needed to do the depth test. Before doing the depth test we had to rotate the view on the x-axis to see the objects from 'above' in order to find out how they actually rotate in relation to each other.

The challenges were solved by Googling the problem and reading tutorials and answers by others with the same problem. We also glanced at the previous labs we had made.

From the start, we had the ambition to add texture to the spheres, but it turned out to be too challenging for this project so we skipped it. If adding texture, we would have had to add code to read the texture/image file in c++, converting it to a GLUT texture and then add it to the spheres with OpenGL.

5. FUTURE WORK

The solar system created in this project only contains the sun and the earth. The first improvement in future work would be to add all our planets and their moons in order to resemble a real solar system. If possible also implement an interactive timeline to see the state of the solar system at a selected time.

In order for the system to be more realistic, the texture for the objects should be added as well as animation for the stars. As described previously adding textured turned out to be more tricky than we thought from the start. With more time and work it would definitely have been possible. The stars could be animated such as in lab 1 where we add an update function which is called based on a timer and then updated the values for the star's each dimension. Another thing that could have been improved in order to make the simulation more realistic is to base the angle, on which the earth rotation is made, by correct time values, since it now only is changed by the integer 1.

For the project to include more user interaction, the rotation and zoom could be connected to the mousepad instead of keyboard input, such as the view following the motion of the user's mousepad. Clicking the objects could also display some information about them if one would like a more informative system to work with, for instance as an educational program.

6. REFERENCES

[1] The OpenGL Utility Toolkit (GLUT) Programming Interface. Retrieved May 13 2020, from, http://www.opengl.org/resources/libraries/glut/spec3/spec3.html

[2] Richard S.Wright Jr. 2011. OpenGL Super Bible: Comprehensive Tutorial and Reference 5th ed. Pearson Education, Inc.

[3] erjan123. (2008, February 1). A Simple OpenGL Window with GLUT Library. Retrieved May 8, 2020, from, https://www.codeproject.com/Articles/23365/A-Simple-OpenGL-Window-with-GLUT-Library

[4] Carmine. (2015, October 19). Draw Sphere. Retrieved May 8, 2020, from, https://community.khronos.org/t/draw-sphere/73994/3

[5] Keyboard Example: Moving the Camera. (2014, June 18). Retrieved May 8, 2020, from https://www.lighthouse3d.com/tutorials/glut-tutorial/keyboard-example-moving-around-the-world/

[6] M. S. Rakha. Adding lighting to a solid sphere - C openGL examples. (n.d.). Retrieved May 8, 2020, from, https://www.codemiles.com/c-opengl-examples/adding-lighting-to-a-solid-sphere-t9125.html