Introduction to Computer Graphics Homework 1

Andres Ponce
彭思安
0616110

October 14, 2019

1 Program Structure

This program initially creates the essential aspects of the display, such as the window position, and title. Within `main`, we then create the other lights, since theu will be at a set posiiton in the scene. The next step is to pass the diplay function pointer to `glutDisplayFunc`, which handles all the alctual creation of elements and translations. We also have other functions handling the keyboard and mouse inputs, however those are handled in separate functions. In `display()`, we handle the creation of the actual objects, in this case the axis and the three spheres.

2 How to implement the rotations

The rotations proved far trickier than expected. The general method to implement transformations involves rotating an object around the origin first, and afterwards moving the object to the position we desire. This simplifies the requirement to remember the vertices after we translate them.

Translating the moon around the Earth involved calculating the rotation of the moon relative to the earth and to the sun. Two translations should be applied to compensate for the fact that the moon is rotating around two different axes, which means there are two differnent axes of rotations.

For the Earth, applying a rotation and a translation put it on the correct path. However, the speed at wich the Earth revolves around the Sun remained too high, so the moon could not rotate around the Earth at the proper speed and with the proper radius.

Again, because of the order in which OpenGL applies the transformations, they need to be specified in the reverse order in which they are to be processed. The incorrect order will result in unexpected behavior, since matrix multiplication is not commutative.

Then how do we know when to push and pop the matrices from the stack? If we think of pushing as 'saving" our work, then when we push it onto the stack we generate another exact copy of the array to use with transformations that are not to be stored for future use. One such transformation is the Earth's axis, which is tilted 23.5 degrees; if we push the matrix after we apply the transformations, then the moon will be tilted by 23.5 degrees as well. Therefore, we apply the transformations to the moon and the axis after we push the earth's matrix, since we rely on the earth's position.

3 How to draw the planets

To draw polygons, there exist different methods. For example, we could specify the vertices of the planet to draw, and afterwards pass the array pointer to OpenGL for display, since we would only need to handle the transformations at that point. In this implementation however, we manually create the sphere at eatch iteration by looping ober the amount of stack and slices for every sphere. We then use `GL_QUAD_STRIP`s to specify the points of each vertex. Depending on the radius of the sphere, we use the classical parametric equations for a sphere. The formulae for the different points are as follows:

$$x = r\cos(\theta)\sin(\phi)$$

$$y = r\cos(\theta)\cos(\phi)$$

$$z = r\sin(\phi)$$

Using these formulas, the coordinates for the points to the `GL_QUAD_STRIP` for each vertex is relatively simple. After the formulas for drawing the individual spheres were resolved, only drawing the spheres on the correct locations remained. This was done by applying the correct translations required by a circular motion of a sphere.

Within the `drawPlanet()` functions, we loop for every stack and for every slice. Then, using sin and cos of the two points, and the two angles θ and ϕ, the coordiantes of the two vertices corresponding to the qaud strip can easily be found. These formulas can be derived easily from geometric relations between sin, cos, and the lengths of the sides of the triangle.

We begin and end the second loop in those functions by using the `glBegin()` and `glEnd()` functions, which notify OpenGL that we are to start declaring `GL_QUAD_STRIP`. We therefore loop in a circle around the center of the sphere, and ceclare each vertex in realtion to the previous one.

After we declare the vertices, we also delcare the normal vectors to those vertices. These normal vectors are set once per vertex; however, OpenGL performs some interpolation on the normals of the vertices to calculate the normal of the whole plane.

We repeat the same procedure for the earth and for the moon, however we change the radii of the circles to match the spec sheet.

As always, multiple ways to draw the planet exist, such as storing the values of the vertices in an array, and merely passing that array as the parameter to the function that draws the spheres. This method may use less resources, since the same calculations are not being performed every time the sphere is drawn. However, for practical purposes both appraches deliver the correct result.