Programming Exercise 5 : simulate the running of stars

Introduction

In this exercise, you will implement a program to simulate the running of solar system in that the Moon rotates around the Earth and the Earth rotates around the Sun. All three planets are spheres that could load the textures from a plain image files if you like. This exercise doesn't simulate the real trajectory of these objects because it was not the target of it. This exercise make you familiar with the basic transformations in OpenGL and 3D worlds. Before starting on this programming exercise, we strongly recommend watching the lectures and the "Viewing" chapter in OpenGL redbook . We also recommend you to look at the Robin's tutorial about transformation.

To get started with the exercise, you will need to download the starter codes. There is a skeleton file "***SolarSystem.cpp***" which provided all necessary code for you to start, you just need to finish the function *display()* to simulate the rotations between the Earth, the Moon, and the Sun .

To run these programs ,the glut enviroment should be setup. You can get the glut from glut-3.7.6-bin.zip (117 KB)(http://user.xmission.com/~nate/glut/glut-3.7.6-bin.zip) ,unzip it and copy three files to different locations on your window as following:

- glut.h – This is the file you'll have to include in your source code. The common place to put this file is in the gl folder which should be inside the include folder of your system. for example ,your vc2010 include folder "*c:\program files\Microsoft Visual Studio 10.0\VC\include\gl*" or "*c:\program files(x86)\Microsoft Visual Studio 10.0\VC\include\gl*" in(my computer)
- glut32.lib (Windows version) – This file must be linked to your application so make sure to put it your lib folder. example ,your vc2010 lib folder "*c:\program files\Microsoft Visual Studio 10.0\VC\lib*" or "*c:\program files(x86)\Microsoft Visual Studio 10.0\VC\lib*" in(my computer)
- glut32.dll (Windows) – You could place the dll file in your exe's folder.for example,the sustem folder: "*c:\windows\system32*" or "*c:\windows\sysWOW64*" in my computer

Simulate the running of solar system

In the skeleton code, a function " void DrawSphere(double dRadius) " is provided to draw a star such as the Moon or The Sun.

To animation the process of the running of the solar system, we should update the image of the solar system continuously that we draw the solar system using their new positions and orientations at different time instant. You could use a timer to

trigger the new rendering of the solar system after a short time interval. We generally instead re-draw the solar system in OpenGL callback function IdleFunc which do display when the machine is idle.

glutIdleFunc(& IdleProc);

Because the relative position between two centers of two Stars are fixed, we just update the rotating angle between them. We do this work in the idle) .After that OpenGL function "glutPostRedisplay();" is called to trigger the re-draw of scene using these updated rotating angles. Following is the *IdleProc ()*:

```
void IdleProc(){
 g_fSunSelfAngle += 3.0f;
 if( g_fSunSelfAngle > 360 )
      g_fSunSelfAngle = 0.0f;
 g_fEarthCommonAngle += 3.0f;
 if( g_fEarthCommonAngle > 360 )
      g_fEarthCommonAngle = 0.0f;
 g_fEarthSelfAngle += 9.0f;
 if( g_fEarthSelfAngle > 360 )
      g_fEarthSelfAngle = 0.0f;
 g_fMoonCommonAngle += 12.0f;
 if( g_fMoonCommonAngle > 360 )
      g_fMoonCommonAngle = 0.0f;
 g_fMoonSelfAngle += 3.0f;
 if( g_fMoonSelfAngle > 360 )
      g_fMoonSelfAngle = 0.0f;
 Sleep(30);
 glutPostRedisplay();
}
```

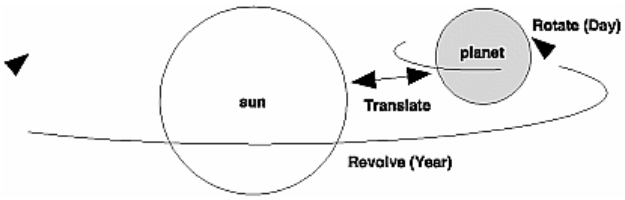

figure 1 the solar system

To determine the order of modeling transformations, visualize what happens to the local coordinate system (see figure 1). An initial glRotate*() rotates the local coordinate system that initially coincides with the grand coordinate system. Next, glTranslate*() moves the local coordinate system to a position on the planet's orbit; the distance moved should equal the radius of the orbit. Thus, the initial glRotate*() actually determines where along the orbit the planet is (or what time of year it is). A second glRotate*() rotates the local coordinate system around the local axes, thus determining the time of day for the planet. Once you've issued all these

transformation commands, the planet can be drawn.

You should use OpenGL api "glPushMatrix()" and " glPopMatrix()" to make the transformation correctly that the self-rotation of the Sun do not influence the rotation of the Earth around the Sun, etc. A example code is provided here to help you make your solar system work correctly:

```
void didpaly(){
        //.....
         glPushMatrix(); {   // Sun
         glColor4f( 1f, 1f, 1f, 1f );
         gluSphere( qobj0, 0.8f, 10, 10) ;

            glPushMatrix(); {// Moon
                glRotatef( angleMonth, 0.0f, 1.0f, 0.0f );
                glTranslatef (0.8f, 0.0f, 0.0f );
                gluSphere( qobj0, 0.1f, 10, 10 ) ;
            }
            glPopMatrix(); //Moon

        } glPopMatrix(); // sun
}
```

In this code, the Moon rotates around the Sun, but they have no self-rotation. In your solar system, you should make the earth revolves (rotates) around the Sun and itself revolves (rotates) on its own axis. The Moon should have this similar process.

Your **task** is to finish the function *display()* to simulate the rotations between the Earth, the Moon, and the Sun.

If you like, you could texture mapping the Stars so that you could check if the Earth is self-rotating!

All parts of this programming exercise are due 2012/11/8 PM at 23:59