UCS505

Computer Graphics

Project Report - Solar trajectory and Lunar Eclipse

Submitted by:

101903191 Akshat Sharma

102083033 Rahul Mishra

101903189 Ravnoor Singh

Group:
3CO7

Submitted to:
Dr. Harpreet Singh

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology, Patiala

TABLE OF CONTENTS

1. Introduction.

In this project we've shown a solar and lunar eclipse and simulate it using C++ and Open GL libraries. The simulation consists of two options where 'R' is used to show the condition of solar eclipse and 'S' is used to show the condition of solar eclipse. Other than that, the earth is continuously moving around the sun and the moon is continuously moving around the earth. The flow of the program on various inputs is given below –

[Keyboard] Type 'R' or 'r':
- Solar eclipse will be shown.

[Keyboard] Type 'S' or 's':
- Lunar eclipse will be shown.

[Keyboard] (Up key):
- Increase the speed of the earth around sun and moon around the sun.

[Keyboard] (Down key):
- Decrease the speed of the earth around sun and moon around the sun.

[Keyboard] (Esc key):
- Exit the program.

The scenery is designed according to the time of the day (Modes) which are described as follows –

Initially, the earth is moving around the axis of the earth and the moon is moving around the earth. We have an option to control the speed of the sun and moon. We can control the speed using the up and down key. If we want to see the condition of the solar eclipse, we can press the key 'R' or 'r' and if we want to see the condition of the lunar eclipse, we can press the key 'S' or 's'. On pressing the above keys, the earth and moon will stop and we will be able to see the condition of the sun.

Condition of Solar eclipse:

At a point, the moon comes between the earth and the sun.

Condition of Lunar eclipse:

At a point, the earth comes between the moon and the sun.

We could have added other seven planets to the solar system to the project but we focused on the sun, the moon and the earth as we wanted to elaborate the solar and lunar eclipse.

2. Computer Graphics concepts used.

OpenGL is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering. OpenGL libraries and various GL and GLU commands were used in this project. These are as follows:

- glColor3f(): Set the current colour of objects.
- glBegin() and glEnd(): Delimit the vertices of a primitive or a group of like primitives.
- glVertex3f(): These functions specify a vertex in float value
- glClear(): Clears buffers to preset values.
- glClearColor(): Specifies the red, green, blue, and alpha values used to clear the color buffers.
- glFlush(): Forces execution of OpenGL functions in finite time.
- glMatrixMode(): Specifies which matrix is the current matrix.
- glLoadIdentity(): Replaces the current matrix with the identity matrix.
- gluOrtho2D(): Defines a 2-D orthographic projection matrix.
- glTranslatef: Back off eight units to be able to view from the origin.
- glRotatef: Rotate the model's plane about the x axis by described degrees.
- glutWireSphere: Draw object as a wireframe.
- glPushMatrix: Save matrix state.
- glPopMatrix: Rotate matrix state.

GLUT provides high-level utilities to simplify OpenGL programming, especially in interacting with the Operating System (such as creating a window, handling key and mouse inputs). The following GLUT functions were used in the project:

- glutInit: Initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the main().
- glutInitDisplayMode: Sets the initial display mode.
- glutInitWindowSize: Specifies the initial window width and height, in pixels.
- glutInitWindowPosition: Positions the top-left corner of the initial window at (*x*, *y*).
- glutCreateWindow: Creates a window with the given title.
- glutDisplayFunc: Registers the callback function for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request.
- glutMainLoop: Enters the infinite event-processing loop, i.e, put the OpenGL graphics system to wait for events, and trigger respective event handlers.
- glutSpecialFunc: Sets the special keyboard callback for the current window.
- OpenGLInit: All those functions in your "init" function set some drawing related state.
- glutReshapeFunc: Sets the reshape callback for the current window.
- glutKeyboardFunc: Used to tell the windows system which function we want to process the “normal” key events. By “normal” keys, we mean letters, numbers, anything that has an ASCII code.

User defined functions.

3.

KeyPressFunc: This function is made to call the function based the key pressed.
Key_r: Shows the solar eclipse.
Key_s: Shows the lunar eclipse.
SpecialKeyFunc: Control the up and down key press.
Key_up: Increase the speed of the program.
Key_down: Decrease the speed of the program.
Animate: This function is made to draw the moon, the earth and the sun. Other functions like controlling the speed, showing lunar and solar eclipse are maintained here.

4.

Code.

```
#include <stdlib.h>
#include<string.h>

#include <GL/glut.h> // OpenGL Graphics Utility Library

static GLenum spinMode = GL_TRUE;
static GLenum singleStep = GL_FALSE;
```

```
void OpenGLInit(void);
static void Animate(void );
static void Key_r(void );
static void Key_s(void );
static void Key_up(void );
static void Key_down(void );
static void ResizeWindow(int w, int h);
static void KeyPressFunc( unsigned char Key, int x, int y );
static void SpecialKeyFunc( int Key, int x, int y );

// These three variables control the animation's state and speed.
static float HourOfDay = 0.0;
static float DayOfYear = 0.0;
static float AnimateIncrement = 24.0;  // Time step for animation (hours)



// glutKeyboardFunc is called below to set this function to handle
//                 all normal key presses.
static void KeyPressFunc(unsigned char Key, int x, int y)
{
        switch (Key) {
        case 'R':
        case 'r':
                Key_r();
                break;
        case 's':
        case 'S':
                Key_s();
                break;
        case 27:        // Escape key
                exit(1);
        }
}
```

```
// glutSpecialFunc is called below to set this function to handle
//		all special key presses.  See glut.h for the names of
//		special keys.
static void SpecialKeyFunc(int Key, int x, int y)
{
	switch (Key) {
	case GLUT_KEY_UP:
		Key_up();
		break;
	case GLUT_KEY_DOWN:
		Key_down();
		break;
	}
}


static void Key_r(void)
{
	if (singleStep) {			// If ending single step mode
		singleStep = GL_FALSE;
		spinMode = GL_TRUE;		// Restart animation
	}
	else {
		spinMode = !spinMode;	// Toggle animation on and off.
		DayOfYear = 0;

	}
}

static void Key_s(void)
{
	singleStep = GL_TRUE;
	spinMode = GL_TRUE;
}

static void Key_up(void)
{
	AnimateIncrement *= 2.0;			// Double the animation time step
}

static void Key_down(void)
{
	AnimateIncrement /= 2.0;			// Halve the animation time step

}
```

```
/*
 * Animate() handles the animation and the redrawing of the
 *		graphics window contents.
 */
static void Animate(void)
{
	// Clear the redering window
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	if (spinMode) {
		// Update the animation state
		HourOfDay += AnimateIncrement;
		DayOfYear += AnimateIncrement / 24.0;

		HourOfDay = HourOfDay - ((int)(HourOfDay / 24)) * 24;
		DayOfYear = DayOfYear - ((int)(DayOfYear / 365)) * 365;
	}

	// Clear the current matrix (Modelview)
	glLoadIdentity();

	// Back off eight units to be able to view from the origin.
	glTranslatef(0.0, 0.0, -11);

	// Rotate the plane of the elliptic
	// (rotate the model's plane about the x axis by fifteen degrees)
	glRotatef(15.0, 1.0, 0.0, 0.0);

	// Draw the sun		-- as a yellow, wireframe sphere
	glColor3f(1.0, 1.0, 0.0);
	glutWireSphere(1.86, 15, 15);

	if (!spinMode) {
		int x1 = 0;
		float y1 = 1.94;
		float x2 = 5.6;
		float y2 = -0.4;
		int z1 = 0;


		glColor3f(1.0, 0.0, 0.0);
		glBegin(GL_LINES);
		glVertex3f(x1 , y1, z1);
		glVertex3f(x2, y2, z1);
		glEnd();
```

```
                int x3 = 0;
                float y3 = -2.1;
                float x4 = 5.5;
                float y4 = 0.4;



                glColor3f(1.0, 0.0, 0.0);
                glBegin(GL_LINES);
                glVertex3f(x3, y3, z1);
                glVertex3f(x4, y4, z1);
                glEnd();


        }

        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_POINTS);
        glVertex3f(0, 1.0, -10);

glVertex3f(0, -6.7, -10);

        glEnd();
        // Draw the Earth
        // First position it around the sun
        //              Use DayOfYear to determine its position
        glRotatef(360.0 * DayOfYear / 365.0, 0.0, 1.0, 0.0);
        glTranslatef(6.0, 0.0, 0.0);
        glPushMatrix();                                         // Save matrix state
        // Second, rotate the earth on its axis.
        //              Use HourOfDay to determine its rotation.
        glRotatef(360.0 * HourOfDay / 24.0, 0.0, 1.0, 0.0);
        // Third, draw the earth as a wireframe sphere.
        glColor3f(0.2, 0.2, 1.0);
        glutWireSphere(0.6, 10, 10);
        glPopMatrix();                                      // Restore matrix state

        // Draw the moon.
        //      Use DayOfYear to control its rotation around the earth
        glRotatef(360.0 * 12.0 * DayOfYear / 365.0+180, 0.0, 1.0, 0.0);
        glTranslatef(1.9, 0.0, 0.0);
        glColor3f(1.0, 1.0, 1.0);
        glutWireSphere(0.2, 7, 7);
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_POINTS);
```

```
        glVertex3f(0.0,0.2,0);
        glVertex3f(0.0,-0.2, 0);


        glEnd();



        // Flush the pipeline, and swap the buffers
        glFlush();
        glutSwapBuffers();

        if (singleStep) {
                spinMode = GL_FALSE;
        }

        glutPostRedisplay();            // Request a re-draw for animation purposes

}


// Initialize OpenGL's rendering modes
void OpenGLInit(void)
{

glShadeModel(GL_FLAT);

glClearColor(0.0, 0.0, 0.0, 0.0);
        glClearDepth(1.0);
        glPointSize(5);

        glEnable(GL_DEPTH_TEST);
}

// ResizeWindow is called when the window is resized

static void ResizeWindow(int w, int h)
{
        float aspectRatio;
        h = (h == 0) ? 1 : h;
        w = (w == 0) ? 1 : w;
        glViewport(0, 0, w, h);         // View port uses whole window
        aspectRatio = (float)w / (float)h;

        // Set up the projection view matrix (not very well!)
        glMatrixMode(GL_PROJECTION);
```

```
        glLoadIdentity();
        gluPerspective(60.0, aspectRatio, 1.0, 30.0);

        // Select the Modelview matrix
        glMatrixMode(GL_MODELVIEW);
}


// Main routine
// Set up OpenGL, hook up callbacks, and start the main loop
int main(int argc, char** argv)
{
        // Need to double buffer for animation
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

        // Create and position the graphics window
        glutInitWindowPosition(0, 0);
        glutInitWindowSize(600, 400);
        glutCreateWindow("Solar System Demo");

        // Initialize OpenGL.
        OpenGLInit();

        // Set up callback functions for key presses
        glutKeyboardFunc(KeyPressFunc);
        glutSpecialFunc(SpecialKeyFunc);


// Set up the callback function for resizing windows
        glutReshapeFunc(ResizeWindow);

        // Callback for graphics image redrawing


        glutDisplayFunc(Animate);
        // Start the main loop.  glutMainLoop never returns.
        glutMainLoop();

        return(0);                      // Compiler requires this to be here. (Never reached)
}
```

Output: