Computer Graphics (UCS505)

Project Report on Solar System

Submitted by

Kulpreet Singh (101803186) 3CO9

Sajjal Tiwari (101803187) 3CO9

Vishrut Agrawal (101853038) 3CO9

Submitted to

Mr. Ashwani Kumar

Computer Science and Engineering Department

TIET, Patiala

Feb-June 2021

Table of Content

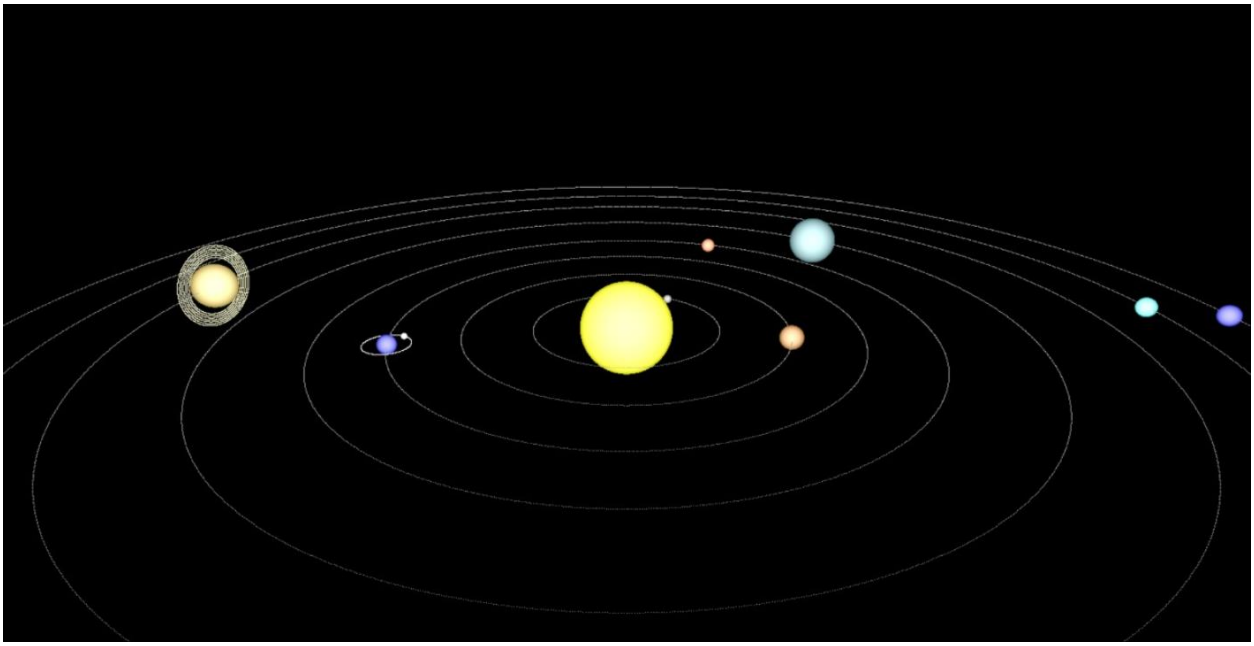
S. No.	Description	Page No.
1.	Introduction	**1**
2.	Computer Graphics Concept Used	**2**
3.	User Defined Functions	**3**
4.	Code	**4**
5.	Output	**13**

Introduction of the Project

Our solar system consists of our star, the Sun, and everything bound to it by gravity — the planets Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, and Neptune, dwarf planets such as Pluto, dozens of moons, and millions of asteroids, comets and meteoroids. Beyond our own solar system, we have discovered thousands of planetary systems orbiting other stars in the Milky Way.

This is a 3D solar system implementation with OpenGL and C++. We have tried to keep it simple because this project demo is only for educational purposes. It contains the sun, the planets, our moon, the planet's orbit, and some asteroids. It is programmed in Visual Studio 2019. For this demo, we have used the Glut library which is interop between OpenGL and C++.

The goal of the project is to create a program in OpenGL that will accurately animate our Solar System.



Computer Graphics Concept Used

- Glut Library (#include <GL/glut.h>) - The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input.

- glColor3f() - The GL stores both a current single-valued color index and a current four-valued RGBA color. glColor sets a new four-valued RGBA color. glColor has two major variants: glColor3 and glColor4. glColor3 variants specify new red, green, and blue values explicitly and set the current alpha value to 1.0 (full intensity) implicitly. glColor4 variants specify all four color components explicitly.

- glPushMatrix() - There is a stack of matrices for each of the matrix modes. In GL_MODELVIEW mode, the stack depth is at least 32. In the other modes, GL_COLOR, GL_PROJECTION, and GL_TEXTURE, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode.glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.

- glPopMatrix() - glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.

- glutInitWindowPosition() - The initial value of the initial window position GLUT state is -1 and -1. If either the X or Y component to the initial window position is negative, the actual window position is left to the window system to determine. The initial value of the initial window size GLUT state is 300 by 300. The initial window size components must be greater than zero.

- glutDisplayFunc() - glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and (if no overlay display callback is registered) the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state).

User-Defined Functions

- renderScene() : This is the main function that will render the solar system. It will set the initial viewport and orbits of all the planets first then one by one will initialize sun and other planets with different color, rotating axis and size.

- mouse() : This function is used to bind left click to start rendering the scene by calling the renderScene fucntion and right click to pause the rendering.

- keyboard() : This function will bind esc key to exit the rendering.

- keyboardSpecial() : This will set UP ARROW key to ZOOM IN functionality and DOWN ARROW to ZOOM OUT functionality by increasing and decreasing the eyeY and eyeZ values.

- changeColour() : This function is used to change the color of the specific planet.

- setupMaterials() : This function will the initialize the light position , model and material type , model.

Code

```
#include<iostream>
#include <math.h>
#include<GL/glut.h>

#define PI 3.14159265358979323846264338327795

static int mercuryRadius = 200;
static int venusRadius = mercuryRadius + 150;
static int earthRadius = venusRadius + 150;
static int marsRadius = earthRadius + 150;
static int jupiterRadius = marsRadius + 200;
static int saturnRadius = jupiterRadius + 200;
static int uranusRadius = saturnRadius + 150;
static int neptuneRadius = uranusRadius + 150;

static float mercury = 360.0 / 87.97;
static float venus = 360.0 / 224.70;
static float earth = 360.0 / 365.26;
static float mars = 360.0 / 686.98;
static float jupiter = 360.0 / 4332.82;
static float saturn = 360.0 / 10755.7;
static float uranus = 360.0 / 306971.5;
static float neptune = 360.0 / 601900.3;

static float mercuryAngle = 0.0f;
static float venusAngle = 0.0f;
static float earthAngle = 0.0f;
static float marsAngle = 0.0f;
static float jupiterAngle = 0.0f;
static float saturnAngle = 0.0f;
static float uranusAngle = 0.0f;
static float neptuneAngle = 0.0f;

float eyeX = 0.0f;
float eyeY = 800.0f;
float eyeZ = 1700.0f;

GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };
GLfloat mat_specular[] = { 0.0, 0.0, 0.0, 0.03 };
GLfloat mat_shininess[] = { 50.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat model_ambient[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
void setupMaterials()
{
   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
   glLightfv(GL_LIGHT0, GL_POSITION, light_position);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, model_ambient);
}

void changeColour(GLfloat r, GLfloat g, GLfloat b, GLfloat A) {
   model_ambient[0] = r;
   model_ambient[1] = g;
   model_ambient[2] = b;
   model_ambient[3] = A;

   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, model_ambient);

}
void changeSize(int w, int h)
{
   if (h == 0) h = 1;
   float ratio = 1.0 * w / h;

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();

   glViewport(0, 0, w, h);
   gluPerspective(45, ratio, 1, 5000);
   glMatrixMode(GL_MODELVIEW);

}

float angle = 0.0f;
float j = 0.0f;

void renderScene(void) {

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();

   gluLookAt(
      0.0f, eyeY, eyeZ,
      0.0f, 0.0f, 0.0f,
      0.0f, 1.0f, 0.0f);
   changeColour(1.0, 1.0, 1.0, 1.0);
```

```
glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / mercuryRadius)
{
   glVertex3f(cos(i) * mercuryRadius, 1.0, sin(i) * mercuryRadius);
}

glEnd();

glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / venusRadius)
{
   glVertex3f(cos(i) * venusRadius, 1.0, sin(i) * venusRadius);
}

glEnd();

glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / earthRadius)
{
   glVertex3f(cos(i) * earthRadius, 1.0, sin(i) * earthRadius);
}

glEnd();

glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / marsRadius)
{
   glVertex3f(cos(i) * marsRadius, 1.0, sin(i) * marsRadius);
}

glEnd();

glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / jupiterRadius)
{
   glVertex3f(cos(i) * jupiterRadius, 1.0, sin(i) * jupiterRadius);
}

glEnd();

glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / saturnRadius)
{
   glVertex3f(cos(i) * saturnRadius, 1.0, sin(i) * saturnRadius);
```

```
}

glEnd();

glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / marsRadius)
{
   glVertex3f(cos(i) * marsRadius, 1.0, sin(i) * marsRadius);
}

glEnd();

glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / uranusRadius)
{
   glVertex3f(cos(i) * uranusRadius, 1.0, sin(i) * uranusRadius);
}

glEnd();

glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / neptuneRadius)
{
   glVertex3f(cos(i) * neptuneRadius, 1.0, sin(i) * neptuneRadius);
}

glEnd();


//sun
glPushMatrix();
glRotatef(angle, 0.0f, 1.0f, 0.0f);
changeColour(1.0, 1.0, 0.0, 0.0);
GLUquadric* sun;
sun = gluNewQuadric();
gluSphere(sun, 100, 100, 100);

   glPopMatrix();

//mercury
glPushMatrix();
glRotatef(mercuryAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(mercuryRadius, 0.0, 0.0);
changeColour(66 / 255.0, 61 / 255.0, 57 / 255.0, 0.0);
GLUquadric* Mercury;
```

```
Mercury = gluNewQuadric();
gluSphere(Mercury, 10, 100, 100);

glPopMatrix();

//venus
glPushMatrix();
glRotatef(venusAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(venusRadius, 0.0, 0.0);
changeColour(140.0 / 255.0, 70.0 / 255.0, 0.0, 0.0);
GLUquadric* Venus;
Venus = gluNewQuadric();
gluSphere(Venus, 25.8, 100, 100);

glPopMatrix();

//earth
glPushMatrix();
glRotatef(earthAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(earthRadius, 0.0, 0.0);
changeColour(0.0, 0.007, 0.686, 0.0);
GLUquadric* Earth;
Earth = gluNewQuadric();
gluSphere(Earth, 20.6, 100, 100);

//moon
glRotatef(5 * earthAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(50, 0.0, 0.0);
changeColour(0.7, 0.7, 0.7, 0.0);
GLUquadric* Moon;
Moon = gluNewQuadric();
gluSphere(Moon, 7, 100, 100);

glPopMatrix();

//*
glPushMatrix();
glRotatef(earthAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(earthRadius, 0.0, 0.0);
changeColour(1.0, 1.0, 1.0, 1.0);
glBegin(GL_POINTS);
for (double i = 0; i < 2 * PI; i += PI / 1000)
{
   glVertex3f(cos(i) * 50, 0.0, sin(i) * 50);
}
```

```
glEnd();
glPopMatrix();

//mars
glPushMatrix();
glRotatef(marsAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(marsRadius, 0.0, 0.0);
changeColour(170 / 255.0, 60 / 255.0, 5 / 255.0, 0.0);
GLUquadric* Mars;
Mars = gluNewQuadric();
gluSphere(Mars, 18.4, 100, 100);

glPopMatrix();

//jupiter
glPushMatrix();
glRotatef(jupiterAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(jupiterRadius, 0.0, 0.0);
glRotatef(angle + angle * 4, 0.0f, 1.0f, 0.0f); // rotating on its own axis
changeColour(53 / 255.0, 107 / 255.0, 112 / 255.0, 0.0);
GLUquadric* Jupiter;
Jupiter = gluNewQuadric();
gluSphere(Jupiter, 64.5, 100, 100);

glPopMatrix();

//saturn
glPushMatrix();
glRotatef(saturnAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(saturnRadius, 0.0, 0.0);
glRotatef(angle + angle * 4, 0.0f, 1.0f, 0.0f); // rotating on its own axis
changeColour(165 / 255.0, 138 / 255.0, 38 / 255.0, 0.0);
GLUquadric* Saturn;
Saturn = gluNewQuadric();
gluSphere(Saturn, 54, 100, 100);

glPopMatrix();

//saturn rings
glPushMatrix();
glRotatef(saturnAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(saturnRadius, 0.0, 0.0);
glRotatef(80.0f, 0.0f, 1.0f, 0.0f); // rotating the circles
changeColour(163 / 255.0, 149 / 255.0, 79 / 255.0, 0.0);
glBegin(GL_POINTS);
```

```
for (double i = 0; i < 2 * PI; i += PI / 1000)
{
   glVertex3f(cos(i) * 70, sin(i) * 70, 0.0);
}
for (double i = 0; i < 2 * PI; i += PI / 1000)
{
   glVertex3f(cos(i) * 75, sin(i) * 75, 0.0);
}
for (double i = 0; i < 2 * PI; i += PI / 1000)
{
   glVertex3f(cos(i) * 80, sin(i) * 80, 0.0);
}
for (double i = 0; i < 2 * PI; i += PI / 1000)
{
   glVertex3f(cos(i) * 85, sin(i) * 85, 0.0);
}
for (double i = 0; i < 2 * PI; i += PI / 1000)
{
   glVertex3f(cos(i) * 90, sin(i) * 90, 0.0);
}
for (double i = 0; i < 2 * PI; i += PI / 1000)
{
   glVertex3f(cos(i) * 95, sin(i) * 95, 0.0);
}
for (double i = 0; i < 2 * PI; i += PI / 1000)
{
   glVertex3f(cos(i) * 100, sin(i) * 100, 0.0);
}
glEnd();
glPopMatrix();


//uranus
glPushMatrix();
glRotatef(uranusAngle, 0.0f, 1.0f, 0.0f);
glTranslatef(uranusRadius, 0.0, 0.0);
changeColour(53 / 255.0, 198 / 255.0, 198 / 255.0, 0.0);
GLUquadric* Uranus;
Uranus = gluNewQuadric();
gluSphere(Uranus, 22.5, 100, 100);

glPopMatrix();


//neptune
```

```
    glPushMatrix();
    glRotatef(neptuneAngle, 0.0f, 1.0f, 0.0f);
    glTranslatef(neptuneRadius, 0.0, 0.0);
    changeColour(5 / 255.0, 5 / 255.0, 186 / 255.0, 0.0);
    GLUquadric* Neptune;
    Neptune = gluNewQuadric();
    gluSphere(Neptune, 24, 100, 100);

    glPopMatrix();

    angle += 0.2f;
    mercuryAngle += mercury / 10;
    venusAngle += venus / 10;
    earthAngle += earth / 10;
    marsAngle += mars / 10;
    jupiterAngle += jupiter / 10;
    saturnAngle += saturn / 10;
    uranusAngle += uranus / 10;
    neptuneAngle += neptune / 10;

    if (angle == 360)
    {
        angle = 0;
    }

    glutSwapBuffers();

}


void mouse(int button, int state, int x, int y)
{

    switch (button) {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
            glutIdleFunc(renderScene);
        break;
    case GLUT_RIGHT_BUTTON:
        if (state == GLUT_DOWN)
            glutIdleFunc(NULL);
        break;

    }
}
```

```
void keyboard(unsigned char key, int xx, int yy) {
   switch(key) {
   case 27:
      exit(0);
      break;
   }
}

void keyboardSpecial(int key, int xx, int yy) {
   switch (key)
   {
   case GLUT_KEY_UP:
      eyeY -= 10.0f;
      eyeZ -= 10.0f;
      break;
   case GLUT_KEY_DOWN:
      eyeY += 10.0f;
      eyeZ += 10.0f;
      break;
   }

}

void init(void) {
   glClearColor(0.0, 0.0, 0.0, 1.0);
   setupMaterials();
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_CULL_FACE);
   glFrontFace(GL_CCW);
   glShadeModel(GL_SMOOTH);
}

int main(int argc, char* argv[]) {

   //init GLUT and create window
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowPosition(100, 100);
   glutInitWindowSize(1000, 500);

   glutCreateWindow("Solar System");
   init();
```

```
    //callbacks
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);
    glutMouseFunc(mouse);
    glutKeyboardFunc(keyboard);
    glutSpecialFunc(keyboardSpecial);

    //enter GLUT event processing cycle
    glutMainLoop();
    return 0;
}
```

Output

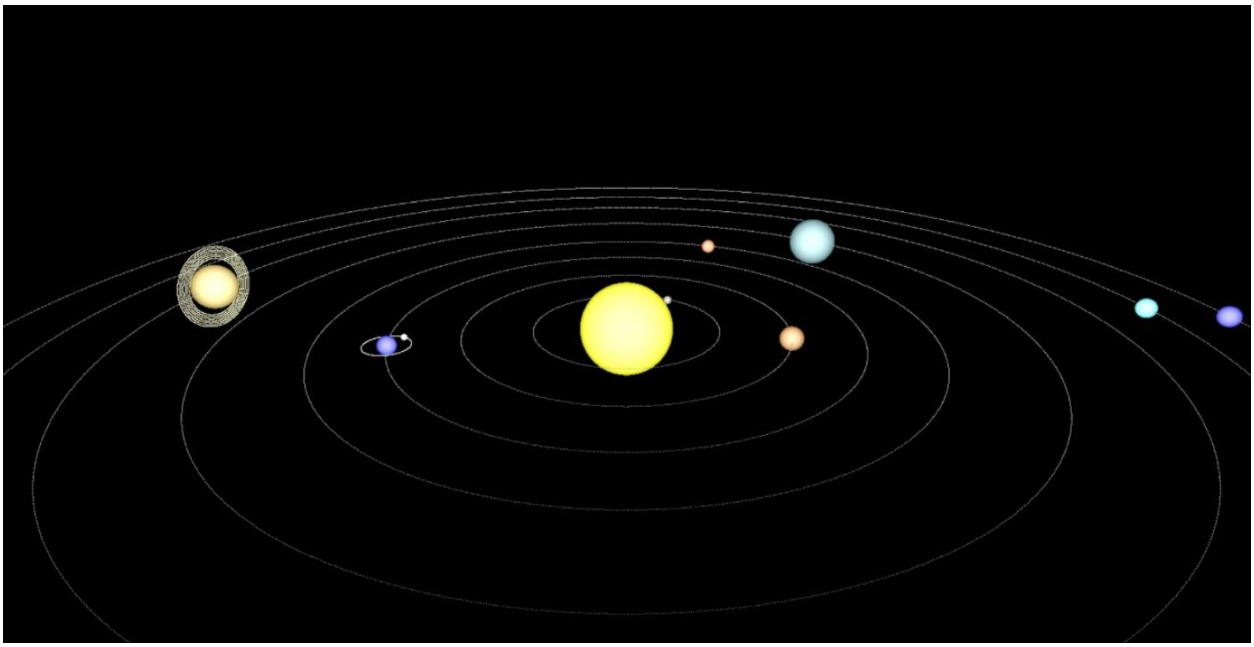


Fig. 1.1