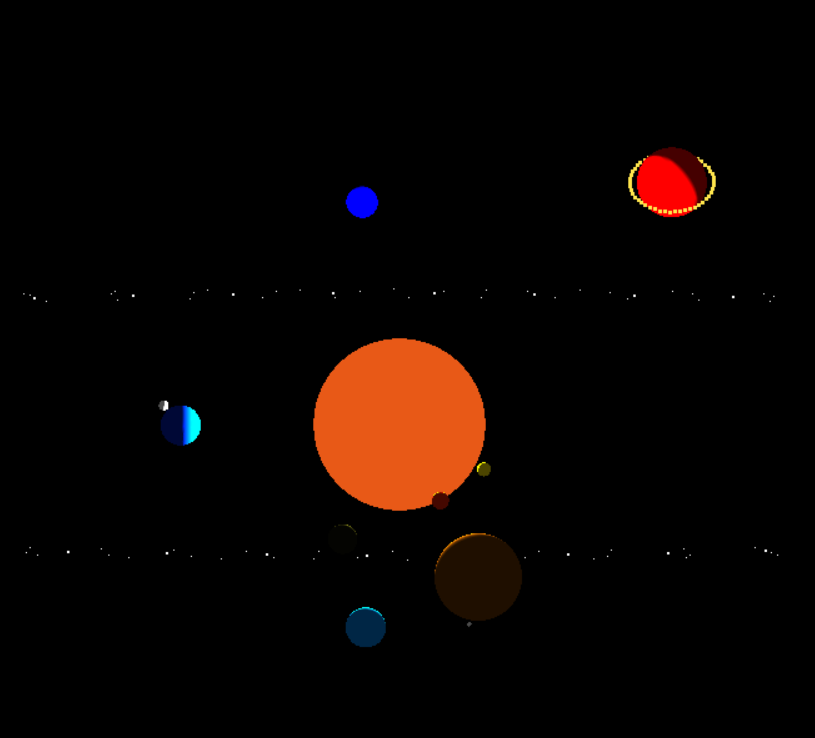
DOCUMENTATIE PENTRU CODUL PRIN CARE ESTE REPREZENTATA MISCAREA PLANETELOR IN JURUL SOARELUI

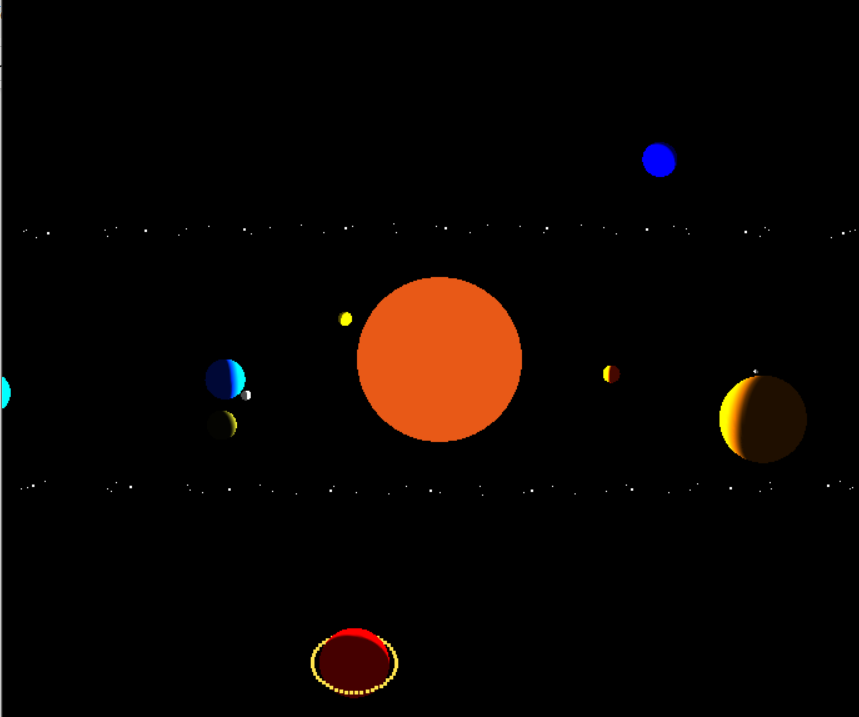
CUPRINS:

1. Descriere generala
2. Biblioteci incluse
3. Macrodefinitii
4. Variabile globale
5. Functii
6. Observatii/ Concluzie

### **Descriere generală:**

Acest cod implementează o simulare simplă a sistemului solar folosind OpenGL și GLUT în limbajul de programare C. Sistemul solar este compus din mai multe planete care orbitează în jurul unei stele centrale (presupusă a fi Soarele).





### **Biblioteci incluse:**

* **stdio.h**: Include funcții pentru intrare/ieșire standard.
* **stdlib.h**: Oferă funcții pentru gestionarea memoriei și alte funcții utile.
* **GL/glut.h**: Biblioteca pentru OpenGL Utility Toolkit, care facilitează crearea de aplicații OpenGL.
* **math.h**: Biblioteca pentru funcții matematice.
* **time.h**: Oferă funcții pentru manipularea timpului.
* **windows.h**: Specific pentru sistemele de operare Windows.

### **Macrodefiniții:**

* **PI**: Definirea constantei π.

### **Variabile globale:**

* **angleMoon, angleEarth, angleAstroid, angleMars, angleMercury, angleVenus, angleJupiter, angleSaturn, angleUranus, angleNeptune**: Unghiurile de rotație ale diferitelor planete și a asteroidului în sistemul solar.
* **sx, sy, sz**: Factorii de scalare pentru stea (Soare) în coordonatele x, y, și z.
* **planet1**: Identificatorul de afișare pentru planeta1 (în prezent neutilizat).
* **black, white, blue, er, yellow, qAmb, qDif, qSpec, qPos**: Definirea unor culori și proprietăți ale materialelor pentru iluminare.
* **sc[8]**: Factori de scalare pentru orbitele planetelor.

### **Funcții:**

#### **initLighting()**

* + **Descriere:** Inițializează iluminarea pentru scenă.
  + **Utilizare:** Apelată în funcția **main()**.

#### **myinit()**

* + **Descriere:** Inițializează setările pentru OpenGL, cum ar fi culoarea de fundal.
  + **Utilizare:** Apelată în funcția **main()**.

#### **background()**

* + **Descriere:** Desenează un fundal colorat pentru scenă.
  + **Utilizare:** Apelată în funcția **render()**.

#### **orbit()**

* + **Descriere:** Desenează orbitele planetelor în jurul stelei.
  + **Utilizare:** În prezent neutilizată în cod.

#### **render()**

* + **Descriere:** Funcția principală de redare. Desenează planetele și steaua în scenă.
  + **Utilizare:** Apelată în funcția **main()**.

#### **update(int value)**

* + **Descriere:** Actualizează unghiurile de rotație ale planetelor și alte animații pentru a simula mișcarea planetelor în jurul stelei.
  + **Utilizare:** Apelată în funcția **main()** pentru a programa actualizările.

#### **main(int argc, char\*\* argv)**

* + **Descriere:** Funcția principală a programului. Inițializează GLUT și OpenGL, stabilește setările inițiale, și intră în bucla principală de evenimente.
  + **Utilizare:** Punctul de intrare în program.

### **Observații:**

* Se simulează mișcarea de rotație a planetelor în jurul stelei și a unui asteroid în jurul unei planete.
* Mișcarea de translație a stelei (Soare) este influențată de poziția unui satelit (Luna) care orbitează în jurul planetei Pământ.
* Actualizarea continuă a unghiurilor permite animarea mișcării planetelor și a altor elemente în sistemul solar.

Codul sub forma de plain text:

#include<stdio.h>

#include<stdlib.h>

#include <GL/glut.h>

#include<math.h>

#include<time.h>

#include<windows.h>

#define PI 3.14

float angleMoon = 0.0, angleEarth = 0.0, angleAstroid = 0.0,

angleMars = 0.0,

angleMercury = 0.0,

angleVenus = 0.0,

angleJupiter = 0.0,

angleSaturn = 0.0,

angleUranus = 30.0,

angleNeptune = 60.0;

GLfloat sx = 0.2, sy = 0.2, sz = 0.2;

int planet1;

GLfloat black[] = { 0.0f,0.0f,0.0f,1.0f };

GLfloat white[] = { 1.0f,1.0f,1.0f,1.0f };

GLfloat blue[] = { 0.0f,0.0f,0.9f,1.0f };

GLfloat er[] = { 0.0f,5.0f,0.9f,1.0f };

GLfloat yellow[] = { 0.7f,0.2f,0.0f,1.0f };

GLfloat qAmb[] = { 0.1,0.1,0.1,1.0 };

GLfloat qDif[] = { 1.0,1.0,1.0,1.0 };

GLfloat qSpec[] = { .50,.50,.50,.10 };

GLfloat qPos[] = { 0,0,0,0.1 };

GLfloat sc[8] = { 0.295 , 0.40,0.50, 0.60,0.80,1.0,1.05,1.13 };

double ang = 2 \* PI / 300;

double angular = 2 \* PI / 50;

void initLighting()

{

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT7);

glLightfv(GL\_LIGHT7, GL\_AMBIENT, qAmb);

glLightfv(GL\_LIGHT7, GL\_DIFFUSE, qDif);

glLightfv(GL\_LIGHT7, GL\_SPECULAR, qSpec);

}

void myinit()

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glPointSize(1.0);

glLineWidth(2.0);

}

void background()

{

glBegin(GL\_QUADS);

glColor3f(0.0, 0.00, 0.00);

glVertex3f(-01.00, 01.00, 1);

glColor3f(.20, 0.0, 0.70);

glVertex3f(01.00, 1.00, 1);

glColor3f(0, 0.0, 0.0);

glVertex3f(1.00, -1.00, 1);

glColor3f(.70, .10, .20);

glVertex3f(-1.00, -1.00, 1);

glEnd();

}

void orbit()

{

glColor3f(0.3, 0.5, 0.5);

int i = 0;

for (i = 0;i < 8;i++) {

glPushMatrix();

if (i == 5)

{

glRotatef(45, 1.0, 0.0, 0.0);

}

else

{

glRotatef(63, 1.0, 0.0, 0.0);

}

glScalef(sc[i], sc[i], sc[i]);

glBegin(GL\_POINTS);

double ang1 = 0.0;

int i = 0;

for (i = 0;i < 300;i++)

{

glVertex2d(5\*cos(ang1), sin(ang1));

ang1 += ang;

}

glEnd();

glPopMatrix();

}

}

void render(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

background();

//orbit();

glLoadIdentity();

glPushMatrix();

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glPushMatrix();

glColor3f(0.7, 0.5, 0.3);

glScalef(sx, sy, sz);

glLightfv(GL\_LIGHT7, GL\_POSITION, qPos);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_EMISSION, yellow);

glutSolidSphere(1, 50, 50);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_EMISSION, black);

glPopMatrix();

glScalef(0.2, 0.2, 0.2);

glPushMatrix();

glRotatef(angleMercury, 0.0, 1.0, -0.5);

glTranslatef(1.5, 0.0, 0.0);

glColor3f(1.0, 0.9, 0.0);

glScalef(0.08, 0.08, 0.08);

glutSolidSphere(1, 50, 50);

glPopMatrix();

glPushMatrix();

glRotatef(angleVenus, 0.0, 1.0, -0.5);

glTranslatef(2.0, 0.0, 0.0);

glColor3f(0.9, 0.1, 0.0);

glScalef(0.1, 0.1, 0.1);

glutSolidSphere(1, 50, 50);

glPopMatrix();

glPushMatrix();

glRotatef(angleEarth, 0.0, 1.0, -0.5);

glTranslatef(2.5, 0.0, 0.0);

glColor3f(0.0, 0.1, 0.7);

glScalef(0.23, 0.23, 0.23);

glutSolidSphere(1, 50, 50);

glPushMatrix();

glRotatef(angleMoon, 0.0, 0.1, 0.05);

glTranslatef(1.3, 0.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glScalef(0.5, 0.5, 0.5);

glutSolidSphere(0.5, 50, 50);

glPopMatrix();

glPopMatrix();

glPushMatrix();

glRotatef(angleMars, 0.0, 1.0, -0.5);

glTranslatef(-3.0, 0.0, 0.0);

glColor3f(0.05, 0.05, 0.01);

glScalef(0.17, 0.17, 0.17);

glutSolidSphere(1, 50, 50);

glPopMatrix();

glPushMatrix();

glColor3f(3.30, 3.30, 3.30);

glRotatef(63, 1.0, 0.0, 0.0);

int j = 0, i = 0, div = 90;float siz = 2;

float scl[4] = { 3.3,3.4,3.35,3.2 };

for (j = 0;j < 4;j++)

{

glPushMatrix();siz -= 0.3;

glPointSize(siz);

glScalef(scl[j], scl[j], scl[j]);

glBegin(GL\_POINTS);

double ang1 = 0.0 - angleAstroid, a = (2 \* PI) / div;

for (i = 0;i < div;i++)

{

glVertex2d(5\*cos(ang1), sin(ang1));

ang1 += a;

}

div += 10;

glEnd();

glPopMatrix();

}

glPopMatrix();

glPushMatrix();

glRotatef(angleJupiter, 0.0, 1.0, -0.5);

glTranslatef(-4.0, 0.0, 0.0);

glColor3f(0.4, 0.2, 0.0);

glScalef(0.5, 0.5, 0.5);

glutSolidSphere(1, 50, 50);

glPushMatrix();

glRotatef(angleMoon, 1.0, -0.5, 0.0);

glTranslatef(0.0, 0, 1.1);

glColor3f(1.0, 1.0, 1.0);

glScalef(0.1, 0.1, 0.1);

glutSolidSphere(0.5, 50, 50);

glPopMatrix();

glPopMatrix();

glPushMatrix();

glRotatef(angleSaturn, 0.0, 1.0, -1.0);

glTranslatef(-5.0, 0.0, 0.0);

glColor3f(0.9, 0.0, 0.0);

glScalef(0.4, 0.4, 0.4);

glutSolidSphere(1, 50, 50);

glPushMatrix();

glRotatef(45, 1.0, 0.0, 0.0);

glPointSize(3);

glColor3f(5.0, 3.0, 1.0);

glScalef(1.2, 1.2, 1.2);

glBegin(GL\_POINTS);

double ang1 = 0.0;

i = 0;

for (i = 0;i < 50;i++)

{

glVertex2d(cos(ang1), sin(ang1));

ang1 += angular;

}

glEnd();

glPointSize(2);

glPopMatrix();

glPopMatrix();

glPushMatrix();

glRotatef(angleUranus, 0.0, 1.0, -0.5);

glTranslatef(5.2, 0.0, 0.0);

glColor3f(0.0, 0.5, 0.9);

glScalef(0.23, 0.23, 0.23);

glutSolidSphere(1, 50, 50);

glPopMatrix();

glPushMatrix();

glRotatef(angleNeptune, 0.0, 1.0, -0.5);

glTranslatef(-5.7, 0.0, 0.0);

glColor3f(0.0, 0.0, 0.9);

glScalef(0.2, 0.2, 0.2);

glutSolidSphere(1, 50, 50);

glPopMatrix();

glPopMatrix();

glFlush();

}

void update(int value) {

if ((angleMoon >= 0 && angleMoon < 180))

{

sx -= 0.0003;sy -= 0.0003;sz -= 0.0003;

}

else { sx += 0.0003;sy += 0.0003;sz += 0.0003; }

angleMoon += 2;

if (angleMoon > 360) {

angleMoon -= 360;

}

angleEarth += 0.7;

if (angleEarth > 360) {

angleEarth -= 360;

}

angleMercury += 2;

if (angleMercury > 360) {

angleMercury -= 360;

}

angleVenus += 0.9;

if (angleVenus > 360) {

angleVenus -= 360;

}

angleMars += 0.5;

if (angleMars > 360) {

angleMars -= 360;

}

angleJupiter += 0.2;

if (angleJupiter > 360) {

angleJupiter -= 360;

}

angleSaturn += 0.1;

if (angleSaturn > 360) {

angleSaturn -= 360;

}

angleUranus += 0.05;

if (angleUranus > 360) {

angleUranus -= 360;

}

angleNeptune += 0.02;

if (angleNeptune > 360) {

angleNeptune -= 360;

}

angleAstroid += 0.002;

if (angleAstroid > 360) {

angleAstroid -= 360;

}

glutPostRedisplay();

glutTimerFunc(20, update, 0);

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(0, 0);

glutInitWindowSize(700, 700);

glutCreateWindow("Solar System");

initLighting();

myinit();

glutDisplayFunc(render);

glutTimerFunc(25, update, 0);

glutMainLoop();

return 0;

}