



POLITECHNIKA WROCŁAWSKA
Instytut Informatyki, Automatyki i Robotyki
Zakład Systemów Komputerowych

Grafika komputerowa i kom.czow.-komp.

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 7

TEMAT ĆWICZENIA

Open GL

Mini projekt układu słonecznego

Wykonał:	Łukasz Mieczyski
Termin:	WT/P 7.30-10.30
Data wykonania ćwiczenia:	18.01.2022
Data oddania sprawozdania:	025.01.2022
Ocena:	

Uwagi prowadzącego:

1. Wstęp

Celem ćwiczenia było wykonanie modelu układu słonecznego, wykorzystując wiedzę nabytą we wcześniejszych laboratoriach. Układ słoneczny będzie składał się ze Słońca oraz 8 planet. Jedynym źródłem światła w układzie będzie Słońce. Każda planeta będzie posiadać własną teksturę.

2. Implementacja

Planety

Każda planeta posiada swoje parametry, które wzorowane były na rzeczywistych wymiarach.

```
//klasa opisująca planetę
class Planet {
public:

    float R, distanceFromSun, orbit, orbitSpeed, axisTilt, axisRotation;

    Planet(float _R, float _distanceFromSun, float _orbit, float _orbitSpeed, float _axisTilt, float _axisRotation) {
        R = _R; //promień planety
        distanceFromSun = _distanceFromSun; //odległość od Słońca
        orbit = _orbit; //obwód orbity
        orbitSpeed = _orbitSpeed; //prędkość planety po orbicie
        axisTilt = _axisTilt; //kąt odchylenia od osi oy
        axisRotation = _axisRotation; //prędkość obrotowa
    }
};

Planet sun(5.0, 0, 0, 0, 0, 0); //Sun
Planet mer(1.0, 7, 0, 4.74, 02.11, 0); //Mercury
Planet ven(1.5, 11, 0, 3.50, 177.0, 0); //Venus
Planet ear(2.0, 16, 0, 2.98, 23.44, 0); //Earth
Planet mar(1.2, 21, 0, 2.41, 25.00, 0); //Mars
Planet jup(3.5, 28, 0, 1.31, 03.13, 0); //Jupiter
Planet sat(3.0, 40, 0, 0.97, 26.70, 0); //Saturn
Planet ura(2.5, 49.5, 0, 0.68, 97.77, 0); //Uranus
Planet nep(2.3, 57.6, 0, 0.54, 28.32, 0); //Neptune
```

Obraz 1. Klasa Planet oraz utworzenie każdej z planet i nadanie jej odpowiednich parametrów

```

void calculateDaysAndHours()
{
    sunRotate = 360 / (24*27.0);
    mercuryRotate = 360 / 2111.0;
    venusRotate = 360.0 / (243 * 24);
    earthRotate = 360.0 / 24;
    marsRotate = 360.0 / 24.617;
    jupiterRotate = 360.0 / 9.9;
    saturnRotate = 360.0 / 10.3;
    uranusRotate = 360.0 / 10.7;
    neptunRotate = 360.0 / 15.8;

    mercuryN = (int)(0.24 * 365);

    venusN = (int)(0.61521 * 365);

    earthN = (int)(1 * 365);

    marsN = (int)(1.88089 * 365);

    jupiterN = (int)(11.8622 * 365);

    saturnN = (int)(29.4577 * 365);

    uranusN = (int)(84.0153 * 365);

    neptunN = (int)(164.788 * 365);
}

```

Obraz 2. Funkcja obliczająca prędkości obrotowe i długości orbit

Obliczone prędkości obrotowe oraz długości orbit są wzorowane na wartościach rzeczywistych.

```

void makeEllipse(float R, int N)
{
    float theta = 2 * 3.1415926 / float(N);
    float c = cosf(theta);
    float s = sinf(theta);
    float t;

    float x = 1;
    float y = 0;

    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < N; i++)
    {
        glVertex2f(x * R, y * R);
        t = x;
        x = c * x - s * y;
        y = s * t + c * y;
    }
    glEnd();
}

```

Obraz 3. Funkcja tworząca orbitę

```

void drawOrbits(void)
{
    glPushMatrix();
    glColor3f(1.0, 1.0, 1.0);
    glTranslatef(0.0, 0.0, 0.0);
    glRotatef(90.0, 1.0, 0.0, 0.0);
    makeEllipse(7, mercuryN);
    makeEllipse(11, venusN);
    makeEllipse(16, earthN);
    makeEllipse(21, marsN);
    makeEllipse(28, jupiterN);
    makeEllipse(40, saturnN);
    makeEllipse(49.5, uranusN);
    makeEllipse(57.6, neptunN);

    glPopMatrix();
}

```

Obraz 4. Funkcja rysująca orbity dla każdej planety

```

void translate()
{
    sun.axisRotation -= sunRotate;
    if ( sun.axisRotation > 360.0)
        sun.axisRotation -= 360.0;

    mer.orbit += mer.orbitSpeed/10;
    mer.axisRotation += mercuryRotate;

    if (mer.orbit > 360.0)
        mer.orbit -= 360;

    if(mer.axisRotation>360.0)
        mer.axisRotation -= 360;

    ven.orbit += ven.orbitSpeed / 10;
    ven.axisRotation += venusRotate;

    if (ven.orbit > 360.0)
        ven.orbit -= 360;
    if (ven.axisRotation > 360.0)
        ven.axisRotation -= 360;

    ear.orbit += ear.orbitSpeed / 10;
    ear.axisRotation += earthRotate;

    if (ear.orbit > 360.0)
        ear.orbit -= 360;

    if (ear.axisRotation > 360.0)
        ear.axisRotation -= 360;
}

```

Obraz 5. Część funkcji translate()

```

mar.orbit += mar.orbitSpeed / 10;
mar.axisRotation += marsRotate;

if (mar.orbit > 360.0)
    mar.orbit -= 360;
if (mar.axisRotation > 360.0)
    mar.axisRotation -= 360;

jup.orbit += jup.orbitSpeed / 10;
jup.axisRotation += jupiterRotate;

if (jup.orbit > 360.0)
    jup.orbit -= 360;
if (jup.axisRotation > 360.0)
    jup.axisRotation -= 360;

sat.orbit += sat.orbitSpeed / 10;
sat.axisRotation += saturnRotate;

if (sat.orbit > 360.0)
    sat.orbit -= 360;

if (sat.axisRotation > 360.0)
    sat.axisRotation -= 360;

```

Obraz 6. Część funkcji translate()

```

ura.orbit += ura.orbitSpeed / 10;
ura.axisRotation += uranusRotate;

if (ura.orbit > 360.0)
    ura.orbit -= 360;
if (ura.axisRotation > 360.0)
    ura.axisRotation -= 360;

glutPostRedisplay(); //odświeżenie zawartości aktualnego okna

```

Obraz 7. Część funkcji translate()

Funkcja translate() jest odpowiedzialna za poruszanie się planet po orbitach i ich obrót wokół własnych osi. Korzysta z prędkości, które są podane przy tworzeniu planet (Obraz 1.) oraz obliczonych prędkości obrotowych (Obraz 2.). W przypadku wykrycia przekroczenia przez zmienne wartości 360.0, kąty są wracają do wartości startowych. Zapewnia to płynne przejście animacji.

```

GLUQuadric* quadric;
quadric = gluNewQuadric();
//Sun
glPushMatrix();
glRotatef(sun.orbit, 0.0, 1.0, 0.0);
glTranslatef(sun.distanceFromSun, 0.0, 0.0);

glPushMatrix();
glRotatef(sun.axisTilt, 1.0, 0.0, 0.0); //przesunięcie względem osi OX
glRotatef(sun.axisRotation, 0.0, 1.0, 0.0); //obrot wokół osi OY
glRotatef(90.0, 1.0, 0.0, 0.0);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, sunTexture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
gluQuadricTexture(quadric, 1);
gluSphere(quadric, sun.R, 20.0, 20.0);
glDisable(GL_TEXTURE_2D);
glPopMatrix();
glPopMatrix();
glPopAttrib(); //ściągnięcie informacji o oświetleniu ze stosu

```

Obraz 8. Część funkcji rysującej planety

Rysowanie planet odbywa się za pomocą funkcji gluSphere(). Dodatkowo planety są przemieszczane o odpowiednie wartości, wynikające z odległości od Słońca. Planety są również obracane względem osi OX, dzięki czemu są odpowiednio przekrzywione. Funkcja glRotatef() wykorzystywana jest również do osiągnięcia obrotu wokół własnej osi. Funkcje glPushMatrix() oraz glPopMatrix()

powodują, że przesunięcia poszczególnych planet nie oddziałują na siebie oraz pozwalają każdej planecie przypisać inną teksturę. Wszystkie planety posiadają analogiczny kod ze zmienionymi wartościami.

Tekstury

```
GLuint loadText(const char* file_name)
{
    GLbyte* pBytes;
    GLint ImWidth, ImHeight, ImComponents;
    GLenum ImFormat;
    pBytes = LoadTGAImage(file_name, &ImWidth, &ImHeight, &ImComponents, &ImFormat);
    GLuint textureId;
    glGenTextures(1, &textureId);
    glBindTexture(GL_TEXTURE_2D, textureId);

    glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat, GL_UNSIGNED_BYTE, pBytes);

    free(pBytes);
    return textureId;
}
```

Obraz 9. Funkcja do wczytywania tekstur korzystająca z funkcji LoadTGAImage() podanej w laboratorium

```
void MyInit(void)
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    sunTexture = loadText( "sun.tga");
    merTexture = loadText( "mercury.tga");
    venTexture = loadText("venus.tga");
    earTexture = loadText("earth.tga");
    marTexture = loadText("mars.tga");
    jupTexture = loadText("jupiter.tga");
    uraTexture = loadText("uranus.tga");
    nepTexture = loadText("neptune.tga");
    satTexture = loadText("saturn.tga");
}
```

Obraz 10. Przypisanie każdej zmiennej odpowiedniej tekstury

```

GLUQuadric* quadric;
quadric = gluNewQuadric();
//Sun
glPushMatrix();
glRotatef(sun.orbit, 0.0, 1.0, 0.0);
glTranslatef(sun.distanceFromSun, 0.0, 0.0);

glPushMatrix();
glRotatef(sun.axisTilt, 1.0, 0.0, 0.0);    //przesunięcie względem osi OX
glRotatef(sun.axisRotation, 0.0, 1.0, 0.0); //obrót wokół osi OY
glRotatef(90.0, 1.0, 0.0, 0.0);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, sunTexture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
gluQuadricTexture(quadric, 1);
gluSphere(quadric, sun.R, 20.0, 20.0);
glDisable(GL_TEXTURE_2D);
glPopMatrix();
glPopMatrix();
glPopAttrib(); //ściągnięcie informacji o oświetleniu ze stosu

```

Obraz 11. Fragment funkcji rysującej planety

Dla każdej planety wczytujemy inną teksturę, są one nakładane na sfery podczas rysowania. Jest to możliwe dzięki funkcji `glBindTexture()`, która wiąże ze sobą teksturę i odpowiednią etykietę.

Oświetlenie

```

glPushAttrib(GL_LIGHTING_BIT); //wrzucenie aktualnych wartości oświetlenia na stos

GLUQuadric* quadric;
quadric = gluNewQuadric();
//Sun
glPushMatrix();
glRotatef(sun.orbit, 0.0, 1.0, 0.0);
glTranslatef(sun.distanceFromSun, 0.0, 0.0);

glPushMatrix();
glRotatef(sun.axisTilt, 1.0, 0.0, 0.0);    //przesunięcie względem osi OX
glRotatef(sun.axisRotation, 0.0, 1.0, 0.0); //obrót wokół osi OY
glRotatef(90.0, 1.0, 0.0, 0.0);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, sunTexture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
gluQuadricTexture(quadric, 1);
gluSphere(quadric, sun.R, 20.0, 20.0);
glDisable(GL_TEXTURE_2D);
glPopMatrix();
glPopMatrix();
glPopAttrib(); //ściągnięcie informacji o oświetleniu ze stosu

glPushAttrib(GL_LIGHTING_BIT); //ściągnięcie informacji o oświetleniu ze stosu

```

Obraz 12. Fragment funkcji rysującej planety


```
glPushAttrib(GL_LIGHTING_BIT); //wrzucenie informacji o źródle światła
```

```
GLfloat mat_ambient[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat mat_shininess = { 50.0 };  
  
GLfloat light_position1[] = { 0.0, 0.0, 0.0, 1.0 };  
GLfloat light_ambient[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat light_diffuse1[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat light_specular1[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat att_constant = { 0.01 };  
GLfloat att_linear = { 0.005 };  
GLfloat att_quadratic = { 0.000001 };  
GLfloat light_ambient2[] = { 0.1, 0.1, 0.1, 1.0 };
```

Obraz 13. Parametry dla materiału i źródła światła

```
GLfloat light_specular2[] = { 0.1, 0.1, 0.1, 1.0 };  
  
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);  
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);  
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);  
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);  
  
glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient2);  
glLightfv(GL_LIGHT1, GL_DIFFUSE, light_diffuse1);  
glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular2);  
glLightfv(GL_LIGHT1, GL_POSITION, light_position1);  
glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, att_constant);  
glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, att_linear);  
glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, att_quadratic);  
glEnable(GL_LIGHTING);  
glEnable(GL_COLOR_MATERIAL);  
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);  
  
//maskuj...
```

Obraz 14. Ustawienie źródła światła i materiałów

Oświetlenie w projekcie zostało rozdzielone na dwa segmenty. Pierwszy obejmuje tylko Słońce, dla którego zdefiniowano domyślne ustawienia światła (Obraz 12.). Drugi segment obejmuje resztę planet, dla których zdefiniowano źródło światła w środku układu słonecznego oraz zmieniono poszczególne parametry (Obraz 13. i 14.).

Obsługa myszy

```
void MouseButtonState(int btn, int state, int x, int y) {  
  
    // Aktualizacja pozycji kursora  
    xMousePosition = x;  
    yMousePosition = y;  
  
    // Zmiana stanu zmiennej określającej naciśnięcie guzika  
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {  
        buttonState = 1;  
    }  
    else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {  
        buttonState = 2;  
    }  
    else  
        buttonState = 0;  
}  
  
void MousePosition(GLsizei x, GLsizei y) {  
  
    // Obliczenie różnicy w pozycji myszy  
    xMousePositionDifference = x - xMousePosition;  
    yMousePositionDifference = y - yMousePosition;  
  
    // Zapisanie aktualnej pozycji myszy  
    xMousePosition = x;  
    yMousePosition = y;  
  
    // Odświeżenie widoku  
    glutPostRedisplay();  
}
```

Obraz 15. Funkcje odpowiedzialne za obsługę myszy

```

if (buttonState == 1) {
    thetaU += xMousePositionDifference * xAngleInPixels;
    thetaV += yMousePositionDifference * yAngleInPixels;
}

// Jeżeli wciśnięty jest RMB
// Zmiana pozycji obserwatora, w zakresie {4.0, 95.0}
// Zależnie od różnicy w pozycji kursora myszy
else if (buttonState == 2) {

    observerPosition[2] += yMousePositionDifference * yAngleInPixels;

    if (observerPosition[2] <= 4) {
        observerPosition[2] = 4;
    }

    if (observerPosition[2] >= 95.0) {
        observerPosition[2] = 95.0;
    }
}

// Określenie aktualnej pozycji obserwatora
gluLookAt(observerPosition[0], observerPosition[1], observerPosition[2], 0.0, 0.0, 0.0, 0.0, 5.0, 0.0);

// Obrót wg osi x i y, zależny od
// Aktualnej wartości kąta theta
glRotatef(thetaU, 0.0, 1.0, 0.0);
glRotatef(thetaV, 1.0, 0.0, 0.0);

```

Obraz 16. Część funkcji RenderScene()

Część funkcji pokazana na Obrazie 16. za zadanie obliczenie nowych wartości dla przesunięcia obserwatora oraz obrotu wokół odpowiednich osi.

3. Wnioski

Udało się zaimplementować układ słoneczny zgodnie z instrukcjami. Jedyne czego nie udało się osiągnąć to orbity w kształcie elips. Największe problemy w projektowaniu układu sprawiło mi użycie kilku różnych tekstur, dopiero po czasie udało się to osiągnąć dzięki funkcją `glPushMatrix()` i `glPopMatrix()`.