pwr**POLITECHNIKA WROCŁAWSKA**

**Instytut Informatyki, Automatyki i Robotyki**

**Zakład Systemów Komputerowych**

**Wprowadzenie do grafiki komputerowej**

**Kurs: INEK00012L**

**Sprawozdanie z ćwiczenia nr**

**TEMAT ĆWICZENIA :**

**OpenGL – Oświetlenie scen 3-D**

|  |  |
| --- | --- |
| **Wykonał:** | **Paweł Biel** |
| **Termin:** | **WT TN 13:15 – 16:15** |
| **Data wykonania ćwiczenia:** | **7.11.2017** |
| **Data oddania sprawozdania:** | **21.11.17** |
| **Ocena:** |  |

|  |
| --- |
| **Uwagi prowadzącego:** |

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// PLIK ŹRÓDŁOWY: Source.cpp

//

// OPIS: Oświetlanie jajka 3D z dwóch różnych źródeł

//

//

// AUTOR: Paweł Biel

//

// DATA 5.11.2017

// MODYFIKACJI:

//

// PLATFORMA: System operacyjny: Microsoft Windows 10.

// Kompilator: Microsoft Visual C++ v2017.

//

// MATERIAŁY Nie wykorzystano.

// ŹRÓDŁOWE:

//

// UŻYTE BIBLIOTEKI Nie używano.

// NIESTANDARDOWE

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <windows.h>

#include <gl/gl.h>

#include <gl/glut.h>

#include <math.h>

#include <cstdlib>

#include <iostream>

#define PI 3.14159265

typedef float point3[3]; // Definicja typu przechowującego współrzędne X,Y,Z punktu

const int N = 60; // Poziom szczegółowości - rysowana figura składa się z N^2 punktów lub wierzchołków trójkątów

static GLfloat viewer[] = { 0.0, 0.0, 10.0 }; // Położenie obserwatora

static GLfloat p = 1.0; // Współrzędna Y skrócenia kamery

static GLfloat theta\_x = 0.0, theta\_y = 0.0, theta\_zoom = 10.0, theta\_x1 = 0.0, theta\_y1 = 0.0 ;

static GLfloat pix2angle; // Przelicznik pikseli na stopień

static GLint status = 0; // Stan wciśnięcia przycisków myszy:

// 0) żaden przycisk nie jest wciśnięty,

// 1) wciśnięty został lewy przycisk,

// 2) wciśnięty został prawy przycisk

static int x\_pos\_old = 0; // Poprzednia pozycja X,Y kursora myszy

static int y\_pos\_old = 0;

static int delta\_x = 0; // Różnica w położeniu bieżącym i poprzednim

static int delta\_y = 0;

GLfloat light\_position1[] = { 0.0, 0.0, 10.0, 1.0 }; // położenie źrodeł światła

GLfloat light\_position[] = { 0.0, 0.0, 10.0, 1.0 };

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Przeliczenie wspolrzednych dwu wymiarowych na wspolrzedne trzy wymiarowe dla x, y, z

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

float x(int i, int j, float n)

{

float u = i / (n - 1);

float v = j / (n - 1);

float xx = ((-90 \* pow(u, 5)) + (225 \* pow(u, 4)) - (270 \* pow(u, 3)) + (180 \* pow(u, 2)) - 45 \* u)\*cos(PI \* v);

return xx;

}

float y(int i, int j, float n)

{

float u = i / (n - 1);

float v = j / (n - 1);

float yy = (160 \* pow(u, 4)) - (320 \* pow(u, 3)) + (160 \* pow(u, 2));

return yy;

}

float z(int i, int j, float n)

{

float u = i / (n - 1);

float v = j / (n - 1);

float zz = ((-90 \* pow(u, 5)) + (225 \* pow(u, 4)) - (270 \* pow(u, 3)) + (180 \* pow(u, 2)) - 45 \* u)\*sin(PI \* v);

return zz;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Wywołanie funkcji rysującej jajko

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void rysuj\_jajko()

{

point3 points[N][N]; // Macierz przechowująca współrzędne N^2 punktów z których składa się obraz

point3 normal[N][N]; // Macierz współrzędnych wektorów normalnych do wierzchołków trójkątów

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

{

float u = (float)i / (float)(N - 1);

float v = (float)j / (float)(N - 1);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Rzutowanie punktow o wsporzednych 3d na macierz

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

points[i][j][0] = x(i, j, N);

points[i][j][1] = y(i, j, N);

points[i][j][2] = z(i, j, N);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Obliczenie wektorów normalnych

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

float xu = (-450 \* pow(u, 4) + 900 \* pow(u, 3) - 810 \* pow(u, 2) + 360 \* u - 45)\*cos(PI \* v);

float yu = 640 \* pow(u, 3) - 960 \* pow(u, 2) + 320 \* u;

float zu = (-450 \* pow(u, 4) + 900 \* pow(u, 3) - 810 \* pow(u, 2) + 360 \* u - 45)\*sin(PI \* v);

float xv = PI \* (90 \* pow(u, 5) - 225 \* pow(u, 4) + 270 \* pow(u, 3) - 180 \* pow(u, 2) + 45 \* u)\*sin(PI \* v);

float yv = 0;

float zv = -PI \* (90 \* pow(u, 5) - 225 \* pow(u, 4) + 270 \* pow(u, 3) - 180 \* pow(u, 2) + 45 \* u)\*cos(PI \* v);

if (i < (N / 2))

{

normal[i][j][0] = (yu \* zv - zu \* yv);

normal[i][j][1] = (zu \* xv - xu \* zv);

normal[i][j][2] = (xu \* yv - yu \* xv);

}

else

{

normal[i][j][0] = -(yu \* zv - zu \* yv);

normal[i][j][1] = -(zu \* xv - xu \* zv);

normal[i][j][2] = -(xu \* yv - yu \* xv);

}

// Zamiana na wektory jednostkowe

float len = sqrt(pow(normal[i][j][0], 2) + pow(normal[i][j][1], 2) + pow(normal[i][j][2], 2));

for (int k = 0; k<3; k++)

normal[i][j][k] /= len;

}

glColor3f(1.0f, 0.0f, 1.0f);

// Wyświetlenie wszystkich trójkątów

for (int i = 0; i < N - 1; i++)

for (int j = 0; j < N - 1; j++)

{

glBegin(GL\_TRIANGLES);

glNormal3fv(normal[i][j]); // Naniesienie wektora normalnego do punktu

glVertex3f(points[i][j][0], points[i][j][1] - 5, points[i][j][2]); // Naniesienie punktu

glNormal3fv(normal[i + 1][j]);

glVertex3f(points[i + 1][j][0], points[i + 1][j][1] - 5, points[i + 1][j][2]);

glNormal3fv(normal[i + 1][j + 1]);

glVertex3f(points[i + 1][j + 1][0], points[i + 1][j + 1][1] - 5, points[i + 1][j + 1][2]);

glEnd();

glBegin(GL\_TRIANGLES);

glNormal3fv(normal[i][j]);

glVertex3f(points[i][j][0], points[i][j][1] - 5, points[i][j][2]);

glNormal3fv(normal[i][j + 1]);

glVertex3f(points[i][j + 1][0], points[i][j + 1][1] - 5, points[i][j + 1][2]);

glNormal3fv(normal[i + 1][j + 1]);

glVertex3f(points[i + 1][j + 1][0], points[i + 1][j + 1][1] - 5, points[i + 1][j + 1][2]);

glEnd();

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Funkcja rysująca układ współrzędnych

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Axes(void)

{

point3 x\_min = { -5.0, 0.0, 0.0 };

point3 x\_max = { 5.0, 0.0, 0.0 };

// początek i koniec obrazu osi x

point3 y\_min = { 0.0, -5.0, 0.0 };

point3 y\_max = { 0.0, 5.0, 0.0 };

// początek i koniec obrazu osi y

point3 z\_min = { 0.0, 0.0, -5.0 };

point3 z\_max = { 0.0, 0.0, 5.0 };

// początek i koniec obrazu osi y

glColor3f(1.0f, 0.0f, 0.0f); // kolor rysowania osi - czerwony

glBegin(GL\_LINES); // rysowanie osi x

glVertex3fv(x\_min);

glVertex3fv(x\_max);

glEnd();

glColor3f(0.0f, 1.0f, 0.0f); // kolor rysowania - zielony

glBegin(GL\_LINES); // rysowanie osi y

glVertex3fv(y\_min);

glVertex3fv(y\_max);

glEnd();

glColor3f(0.0f, 0.0f, 1.0f); // kolor rysowania - niebieski

glBegin(GL\_LINES); // rysowanie osi z

glVertex3fv(z\_min);

glVertex3fv(z\_max);

glEnd();

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Funkcja określająca co ma być rysowane (zawsze wywoływana gdy trzeba

// przerysować scenę)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void RenderScene(void)

{

GLfloat R = 15;

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

glLoadIdentity();

gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, p, 0.0);

Axes();

if (status == 1) // Jeśli wciśnięto lewy przycisk myszy

{

// Obliczenie położenia pierwszego źródła światła

theta\_x += delta\_x \* pix2angle / 30.0;

theta\_y += delta\_y \* pix2angle / 30.0;

light\_position[0] = R\*cos(theta\_x)\*cos(theta\_y);

light\_position[1] = R\*sin(theta\_y);

light\_position[2] = R\*sin(theta\_x)\*cos(theta\_y);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

}

else if (status == 2) // Jeśli wciśnięto prawy przycisk myszy

{

// Obliczenie położenia pierwszego źródła światła

theta\_x1 += delta\_x \* pix2angle / 30.0;

theta\_y1 += delta\_y \* pix2angle / 30.0;

light\_position1[0] = R\*cos(theta\_x1)\*cos(theta\_y1);

light\_position1[1] = R\*sin(theta\_y1);

light\_position1[2] = R\*sin(theta\_x1)\*cos(theta\_y1);

glLightfv(GL\_LIGHT1, GL\_POSITION, light\_position1);

}

// Wywołanie funkcji rysującej jajko

rysuj\_jajko();

glFlush();

// Przekazanie poleceń rysujących do wykonania

glutSwapBuffers();

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Funkcja obługująca zdarzenia myszy

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Mouse(int btn, int state, int x, int y)

{

// Jeśli wciśnięto lewy przycisk myszy

if (btn == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{

x\_pos\_old = x;

y\_pos\_old = y;

status = 1;

}

// Jeśli wciśnięto prawy przycisk myszy

else if (btn == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)

{

y\_pos\_old = y;

status = 2;

}

// Jeśli zwolniono przyciski myszy

else

status = 0;

}

void Motion(GLsizei x, GLsizei y)

{

delta\_x = x - x\_pos\_old;

x\_pos\_old = x;

delta\_y = y - y\_pos\_old;

y\_pos\_old = y;

glutPostRedisplay();

}

void MyInit(void)

{

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

// Kolor czyszczący (wypełnienia okna) ustawiono na czarny

GLfloat mat\_ambient[] = { 1.0, 1.0, 1.0, 1.0 };

// współczynniki ka =[kar,kag,kab] dla światła otoczenia

GLfloat mat\_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };

// współczynniki kd =[kdr,kdg,kdb] światła rozproszonego

GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

// współczynniki ks =[ksr,ksg,ksb] dla światła odbitego

GLfloat mat\_shininess = { 20.0 };

// współczynnik n opisujący połysk powierzchni

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Definicja źródła światła

GLfloat light\_ambient[] = { 0.1, 0.1, 0.1, 1.0 };

// składowe intensywności świecenia źródła światła otoczenia

// Ia = [Iar,Iag,Iab]

GLfloat light\_diffuse[] = { 1.0, 1.0, 0.0, 1.0 };

// składowe intensywności świecenia źródła światła powodującego

// odbicie dyfuzyjne Id = [Idr,Idg,Idb]

GLfloat light\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

// składowe intensywności świecenia źródła światła powodującego

// odbicie kierunkowe Is = [Isr,Isg,Isb]

GLfloat att\_constant = 1.0;

// składowa stała ds dla modelu zmian oświetlenia w funkcji

// odległości od źródła

GLfloat att\_linear = 0.05;

// składowa liniowa dl dla modelu zmian oświetlenia w funkcji

// odległości od źródła

GLfloat att\_quadratic = 0.001;

// składowa kwadratowa dq dla modelu zmian oświetlenia w funkcji

// odległości od źródła

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Ustawienie parametrów materiału i źródła światła

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Ustawienie patrametrów materiału

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);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Ustawienie parametrów źródła

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glLightf(GL\_LIGHT0, GL\_CONSTANT\_ATTENUATION, att\_constant);

glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, att\_linear);

glLightf(GL\_LIGHT0, GL\_QUADRATIC\_ATTENUATION, att\_quadratic);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Ustawienie parametrów dla drugiego źródła światła

glLightfv(GL\_LIGHT1, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT1, GL\_SPECULAR, light\_specular);

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);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Ustawienie opcji systemu oświetlania sceny

glShadeModel(GL\_SMOOTH); // właczenie łagodnego cieniowania

glEnable(GL\_LIGHTING); // właczenie systemu oświetlenia sceny

glEnable(GL\_LIGHT0); // włączenie źródła o numerze 0

glEnable(GL\_LIGHT1); // włączenie źródła o numerze 0

glEnable(GL\_DEPTH\_TEST); // włączenie mechanizmu z-bufora

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

// Funkcja ma za zadanie utrzymanie stałych proporcji rysowanych

// w przypadku zmiany rozmiarów okna.

// Parametry vertical i horizontal (wysokość i szerokość okna) są

// przekazywane do funkcji za każdym razem gdy zmieni się rozmiar okna.

void ChangeSize(GLsizei horizontal, GLsizei vertical)

{

pix2angle = 360.0 / (float)horizontal;

glMatrixMode(GL\_PROJECTION);

// Przełączenie macierzy bieżącej na macierz projekcji

glLoadIdentity();

// Czyszcznie macierzy bieżącej

gluPerspective(100.0, 1.0, 1.0, 30.0);

// Ustawienie parametrów dla rzutu perspektywicznego

if (horizontal <= vertical)

glViewport(0, (vertical - horizontal) / 2, horizontal, horizontal);

else

glViewport((horizontal - vertical) / 2, 0, vertical, vertical);

// Ustawienie wielkości okna okna widoku (viewport) w zale?no?ci

// relacji pomiędzy wysokością i szerokością okna

glMatrixMode(GL\_MODELVIEW);

// Przełączenie macierzy bieżącej na macierz widoku modelu

glLoadIdentity();

// Czyszczenie macierzy bieżącej

}

// Główny punkt wejścia programu. Program działa w trybie konsoli

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(600, 600);

glutCreateWindow("Labki");

// Uruchomienie obsługi zdarzeń wciśnięcia klawisza, przycisku myszy

// oraz jej przesunięcia, wybranymi funkcjami

glutMouseFunc(Mouse);

glutMotionFunc(Motion);

glutDisplayFunc(RenderScene);

// Określenie, że funkcja RenderScene będzie funkcją zwrotną

// (callback function). Bedzie ona wywoływana za każdym razem

// gdy zajdzie potrzba przeryswania okna

glutReshapeFunc(ChangeSize);

// Dla aktualnego okna ustala funkcję zwrotną odpowiedzialną

// zazmiany rozmiaru okna

MyInit();

// Funkcja MyInit() (zdefiniowana powyżej) wykonuje wszelkie

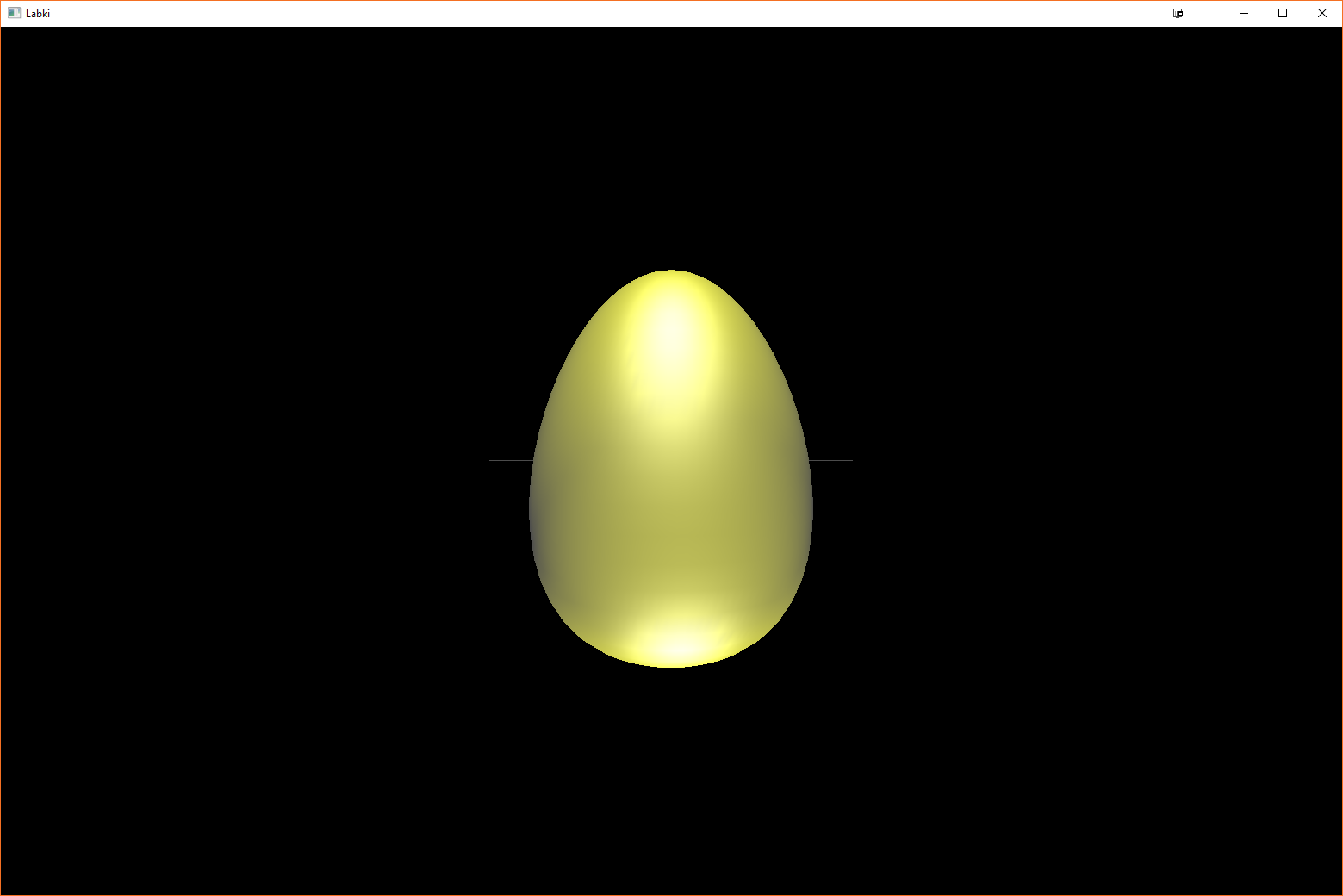
// inicjalizacje konieczne przed przystąpieniem do renderowania

glEnable(GL\_DEPTH\_TEST);

// Włączenie mechanizmu usuwania powierzchni niewidocznych

glutMainLoop();

// Funkcja uruchamia szkielet biblioteki GLUT

}

