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)
```

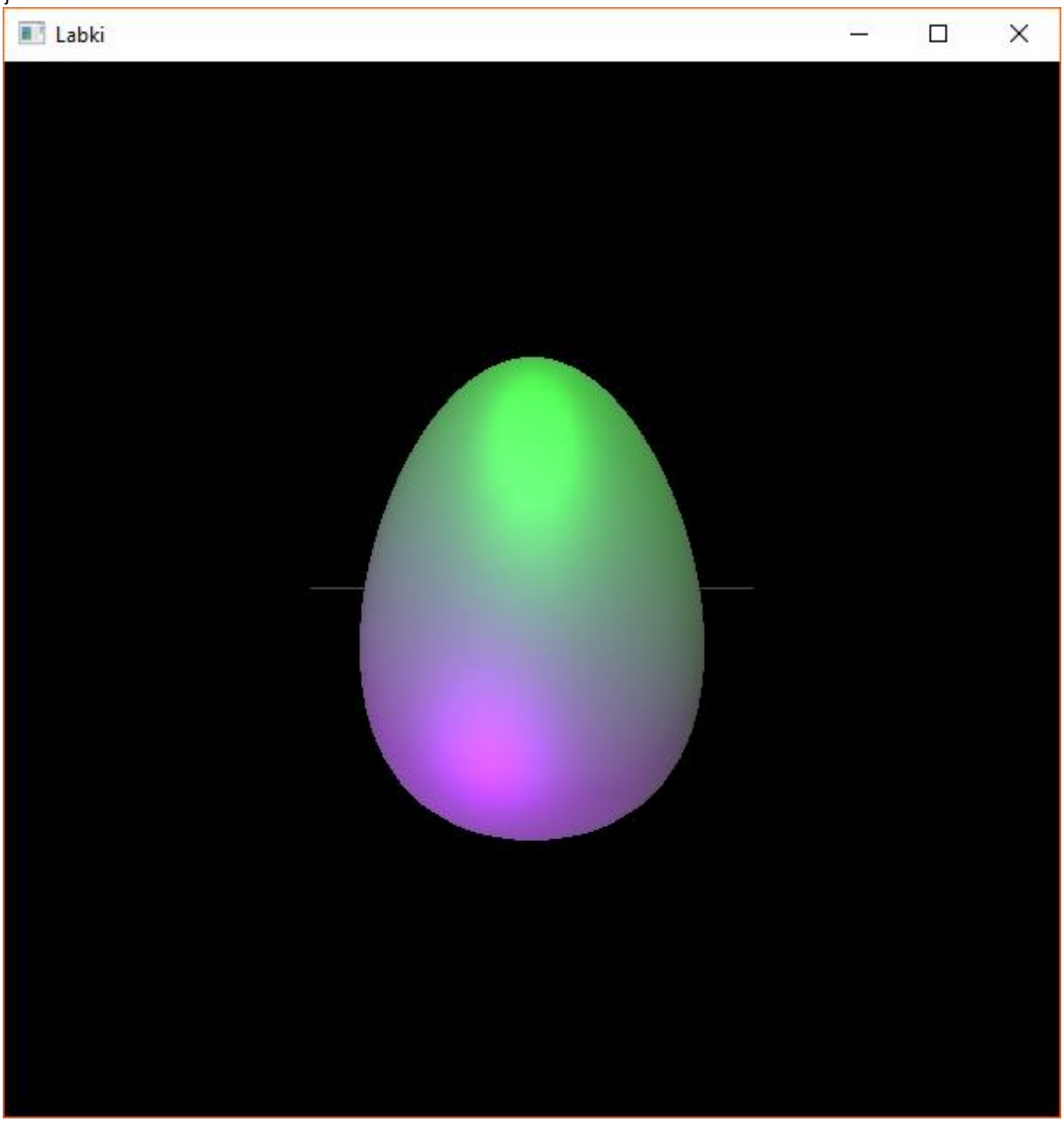
```
	{

		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[] = { 0.5, 0.0, 0.8, 1.0 };
	// składowe intensywności świecenia źródła światła powodującego
	// odbicie dyfuzyjne Id = [Idr,Idg,Idb]

	GLfloat light_specular[] = { 0.5, 0.0, 0.8, 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

        // Definicja źródła światła 2


        GLfloat light_ambient1[] = { 0.1, 0.1, 0.1, 1.0 };
        // składowe intensywności świecenia źródła światła otoczenia
        // Ia = [Iar,Iag,Iab]

        GLfloat light_diffuse1[] = { 0.0, 0.8, 0.0, 1.0 };
        // składowe intensywności świecenia źródła światła powodującego
        // odbicie dyfuzyjne Id = [Idr,Idg,Idb]

        GLfloat light_specular1[] = { 0.0, 0.8, 0.0, 1.0 };
        // składowe intensywności świecenia źródła światła powodującego
        // odbicie kierunkowe Is = [Isr,Isg,Isb]

        GLfloat att_constant1 = 1.0;
        // składowa stała ds dla modelu zmian oświetlenia w funkcji
        // odległości od źródła

        GLfloat att_linear1 = 0.05;
        // składowa liniowa dl dla modelu zmian oświetlenia w funkcji
        // odległości od źródła

        GLfloat att_quadratic1 = 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_ambient1);
	glLightfv(GL_LIGHT1, GL_DIFFUSE, light_diffuse1);
	glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular1);
	glLightfv(GL_LIGHT1, GL_POSITION, light_position1);

	glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, att_constant1);
	glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, att_linear1);
	glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, att_quadratic1);

	/*****************************************************************************
**/
	// 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
```

}


Labki