POLITECHNIKA WROCŁAWSKA
Instytut Informatyki, Automatyki i Robotyki
Zakład Systemów Komputerowych

Grafika komputerowa

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 4

OPEN GL – oświetlanie scen 3-D

Wykonała:	**Weronika Luźna**
Termin:	**PN/N 16.15-19.15**
Data wykonania ćwiczenia:	**10.01.2016**
Data oddania sprawozdania:	**11.01.2016**
Ocena:	

Uwagi prowadzącego:

1. Cel projektu.

Celem projektu było zaimplementowanie 2 programów z oświetleniem scen 3-D.

2. Oświetlone jajko.

Zaimplementowano na podstawie wytycznych zawartych w instrukcji do laboratorium.

2.1. Kod źródłowy.

```
//*************************************************************************
//
//  PLIK ŹRÓDŁOWY:          oswietlenie1.cpp
//
//  OPIS:                           Program służy do rysowania oświetlonej sceny 3-D.
//
//  AUTOR:                          Weronika Luźna
//
//  DATA                            11 Stycznia 2016 (Versja 1.00).
//
//  PLATFORMA:                      System operacyjny:  Microsoft Windows 8.1.
//                                          Kompilator:         Microsoft Visual Studio 2015
//
//  UŻYTE BIBLIOTEKI Nie używano.
//  NIESTANDARDOWE
//
//*************************************************************************

#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <math.h>
#include <time.h>

//Definicja typu point3 - punkt w przestrzeni 3D
typedef float point3[3];

const float PI = 3.14159265;

point3 **coordinates;
point3 **norm_coordinates;

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

static GLfloat       phi = 0.0,
theta = 0.0;
static GLfloat       pix2angle_x = 0.0,
pix2angle_y = 0.0;
static GLint status = 0;
static int    x_pos_old = 0,
y_pos_old = 0;
static int    delta_x = 0,
delta_y = 0;

int N = 50;
float verLength = 1.0;
float viewerR = 10.0;

static GLfloat angle[] = { 0.0, 0.0, 0.0 };
```

```
//Funkcja wyliczajaca wspolrzedna X
float Calculate_x(float u, float v) {
	float x, a = v*PI;

	x = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) *
cos(a);
	return x;
}

//Funkcja wyliczajaca wspolrzedna Y
float Calculate_y(float u, float v) {
	float y;

	y = 160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2);
	return y - 5;
}

//Funkcja wyliczajaca wspolrzedna Z
float Calculate_z(float u, float v) {
	float z, a = v*PI;

	z = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) *
sin(a);
	return z;
}

//Obliczenie wspolrzednej X wektora normalnego do powierzchni w punkcie
float Calculate_norm_x(float u, float v) {
	float x, a = v*PI;

	float yu = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
	float yv = 0;
	float zu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*sin(a);
	float zv = -PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) +
45 * u)*cos(a);

	x = (GLfloat)(yu*zv - zu*yv);
	return x;
}

//Obliczenie wspolrzednej Y wektora normalnego do powierzchni w punkcie
float Calculate_norm_y(float u, float v) {
	float y, a = v*PI;

	float xu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*cos(a);
	float xv = PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45
* u)*sin(a);
	float zu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*sin(a);
	float zv = -PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) +
45 * u)*cos(a);

	y = (GLfloat)(zu*xv - xu*zv);
	return y;
}

//Obliczenie wspolrzednej Z wektora normalnego do powierzchni w punkcie
float Calculate_norm_z(float u, float v) {
	float z, a = v*PI;

	float xu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*cos(a);
	float xv = PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45
```

```
* u)*sin(a);
	float yu = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
	float yv = 0;

	z = (GLfloat)(xu*yv - yu*xv);
	return z;
}

//Funkcja generujaca siatke puntow, najpierw w 2D, potem w 3D
void Mesh() {
	float stepXY = verLength / N;

	for (int i = 0; i<N + 1; i++) {
		for (int j = 0; j<N + 1; j++) {
			coordinates[i][j][0] = j*stepXY;
			coordinates[i][j][1] = i*stepXY;
		}
	}

	float u, v;
	for (int i = 0; i<N + 1; i++) {
		for (int j = 0; j<N + 1; j++) {
			v = coordinates[i][j][0];
			u = coordinates[i][j][1];
			coordinates[i][j][0] = Calculate_x(u, v);
			coordinates[i][j][1] = Calculate_y(u, v);
			coordinates[i][j][2] = Calculate_z(u, v);

			//Wyliczenie wspolrzednych wektorow normalnych
			float x = Calculate_norm_x(u, v);
			float y = Calculate_norm_y(u, v);
			float z = Calculate_norm_z(u, v);

			if (i < N / 2) {
				norm_coordinates[i][j][0] = x / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
				norm_coordinates[i][j][1] = y / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
				norm_coordinates[i][j][2] = z / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
			}
			if (i > N / 2) {
				norm_coordinates[i][j][0] = -1.0*x / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
				norm_coordinates[i][j][1] = -1.0*y / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
				norm_coordinates[i][j][2] = -1.0*z / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
			}
			//Wektory na "szczycie" jajka
			if (i == N / 2) {
				norm_coordinates[i][j][0] = 0;
				norm_coordinates[i][j][1] = 1;
				norm_coordinates[i][j][2] = 0;
			}
			//Wektory na "dnie" jajka
			if (i == 0 || i == N)
			{
				norm_coordinates[i][j][0] = 0;
				norm_coordinates[i][j][1] = -1;
				norm_coordinates[i][j][2] = 0;
			}
```

```
			}
		}
}

void Egg(void) {
		Mesh();

		glColor3f(1.0, 1.0, 1.0);

		for (int i = 0; i < N; i++) {
			for (int j = 0; j < N; j++) {
				glBegin(GL_TRIANGLES);
				glNormal3fv(norm_coordinates[i][j + 1]);
				glVertex3fv(coordinates[i][j + 1]);
				glNormal3fv(norm_coordinates[i + 1][j]);
				glVertex3fv(coordinates[i + 1][j]);
				glNormal3fv(norm_coordinates[i + 1][j + 1]);
				glVertex3fv(coordinates[i + 1][j + 1]);
				glEnd();

				glBegin(GL_TRIANGLES);
				glNormal3fv(norm_coordinates[i][j]);
				glVertex3fv(coordinates[i][j]);
				glNormal3fv(norm_coordinates[i + 1][j]);
				glVertex3fv(coordinates[i + 1][j]);
				glNormal3fv(norm_coordinates[i][j + 1]);
				glVertex3fv(coordinates[i][j + 1]);
				glEnd();
			}
		}
}

void Axes(void)
{
		point3  x_min = { -2.0, 0.0, 0.0 };
		point3  x_max = { 2.0, 0.0, 0.0 };

		point3  y_min = { 0.0, -2.0, 0.0 };
		point3  y_max = { 0.0,  2.0, 0.0 };

		point3  z_min = { 0.0, 0.0, -2.0 };
		point3  z_max = { 0.0, 0.0,  2.0 };

		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();
}
```

```
void RenderScene(void)
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();

	if (status == 2) {
		viewerR += 0.1* delta_y;
		if (viewerR <= 6.0)
			viewerR = 6.0;
		if (viewerR >= 25.0)
			viewerR = 25.0;
	}

	viewer[0] = viewerR * cos(theta) * cos(phi);
	viewer[1] = viewerR * sin(phi);
	viewer[2] = viewerR * sin(theta) * cos(phi);

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

	Axes();

	glRotatef(angle[0], 1.0, 0.0, 0.0);
	glRotatef(angle[1], 0.0, 1.0, 0.0);
	glRotatef(angle[2], 0.0, 0.0, 1.0);

	Egg();

	glFlush();

	glutSwapBuffers();
}

void spinEgg()
{
	angle[0] -= 0.5;
	if (angle[0] > 360.0) angle[0] -= 360.0;
	angle[1] -= 0.5;
	if (angle[1] > 360.0) angle[1] -= 360.0;
	angle[2] -= 0.5;
	if (angle[2] > 360.0) angle[2] -= 360.0;

	glutPostRedisplay();
}

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

	// Definicja materia³u z jakiego zrobiony jest przedmiot
	//------------------------------------------------------
	GLfloat mat_ambient[] = { 1.0, 1.0, 1.0, 1.0 };
	// wspó³czynniki ka =[kar,kag,kab] dla œwiat³a otoczenia

	GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
	// wspó³czynniki kd =[kdr,kdg,kdb] œwiat³a rozproszonego

	GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
	// wspó³czynniki ks =[ksr,ksg,ksb] dla œwiat³a odbitego

	GLfloat mat_shininess = { 100.0 };
	// wspó³czynnik n opisuj¹cy po³ysk powierzchni
```

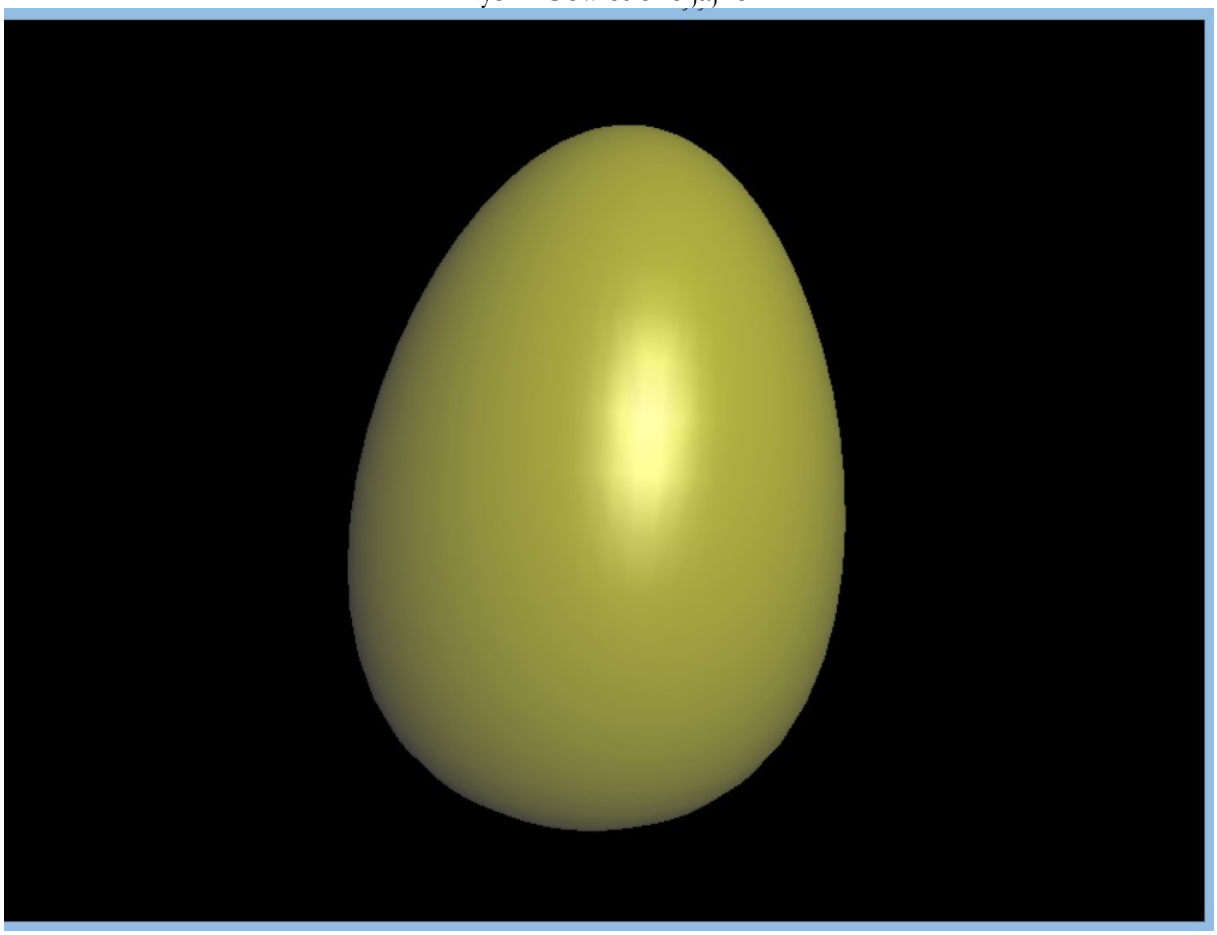
```
	// Definicja Ÿród³a œwiat³a
	//-----------------------------------------------------
	GLfloat light_position[] = { 5.0, 5.0, 10.0, 1.0 };
	// po³o¿enie Ÿród³a

	GLfloat light_ambient[] = { 0.1f, 0.1f, 0.1f, 1.0f };
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a otoczenia
	// Ia = [Iar,Iag,Iab]

	GLfloat light_diffuse[] = { 1.0, 1.0, 0.0, 1.0 };
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a powoduj¹cego
	// odbicie dyfuzyjne Id = [Idr,Idg,Idb]

	GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a powoduj¹cego
	// odbicie kierunkowe Is = [Isr,Isg,Isb]

	GLfloat att_constant = { 1.0 };
	// sk³adowa sta³a ds dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	GLfloat att_linear = { 0.05f };
	// sk³adowa liniowa dl dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	GLfloat att_quadratic = { 0.001f };
	// sk³adowa kwadratowa dq dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	// Ustawienie patrametrów materia³u
	//-----------------------------------------------------
	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 Ÿród³a œwiat³a
	//-----------------------------------------------------
	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 opcji systemu oœwietlania sceny
	//-----------------------------------------------------
	glShadeModel(GL_SMOOTH); // w³aczenie ³agodnego cieniowania
	glEnable(GL_LIGHTING);   // w³aczenie systemu oœwietlenia sceny
	glEnable(GL_LIGHT0);     // w³¹czenie Ÿród³a o numerze 0
	glEnable(GL_DEPTH_TEST); // w³¹czenie mechanizmu z-bufora
}

void ChangeSize(GLsizei horizontal, GLsizei vertical) {
	pix2angle_x = 360.0*0.1 / (float)horizontal; // przeliczenie pikseli na stopnie
	pix2angle_y = 360.0*0.1 / (float)vertical;

	glMatrixMode(GL_PROJECTION);
	// Prze³¹czenie macierzy bie¿¹cej na macierz projekcji
```

```
	glLoadIdentity();
	// Czyszcznie macierzy bie¿¹cej

	gluPerspective(70.0, 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 wielkoœci okna okna widoku (viewport) w zale¿noœci
	// relacji pomiêdzy wysokoœci¹ i szerokoœci¹ okna

	glMatrixMode(GL_MODELVIEW);
	// Prze³¹czenie macierzy bie¿¹cej na macierz widoku modelu

	glLoadIdentity();
	// Czyszczenie macierzy bie¿¹cej
}

void Mouse(int btn, int state, int x, int y) {
	if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
		y_pos_old = y;			// przypisanie aktualnie odczytanej pozycji kursora
						// jako pozycji poprzedniej
		status = 2;			//wciœniêty zosta³ prawy klawisz myszy
	}
	else
		status = 0;		// nie zosta³ wciœniêty ¿aden klawisz
}

void Motion(GLsizei x, GLsizei y) {
	delta_x = x - x_pos_old;	// obliczenie ró¿nicy po³o¿enia kursora myszy
	x_pos_old = x;		// podstawienie bie¿acego po³o¿enia jako poprzednie

	delta_y = y - y_pos_old;	// obliczenie ró¿nicy po³o¿enia kursora myszy
	y_pos_old = y;		// podstawienie bie¿acego po³o¿enia jako poprzednie

	glutPostRedisplay(); // przerysowanie obrazu sceny
}

void main(void)
{
	//Ziarno losowosci
	srand((unsigned)time(NULL));

	coordinates = new point3*[N + 1];
	for (int i = 0; i<N + 1; i++) {
		coordinates[i] = new point3[N + 1];
	}

	//losowość kolorów
	norm_coordinates = new point3*[N + 1];
	for (int i = 0; i < N + 1; i++) {
		norm_coordinates[i] = new point3[N + 1];
	}

	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(800, 600);

	glutCreateWindow("Oswietlone, ruchome jajko");
```

```
	glutDisplayFunc(RenderScene);

	glutReshapeFunc(ChangeSize);

	MyInit();

	glutMouseFunc(Mouse);

	glutMotionFunc(Motion);

	glutIdleFunc(spinEgg);

	glutMainLoop();

	for (int i = 0; i < N + 1; i++) {
		delete[] coordinates[i];
		delete[] norm_coordinates[i];
		coordinates[i] = 0;
		norm_coordinates[i] = 0;
	}
	delete[] coordinates;
	delete[] norm_coordinates;
	coordinates = 0;
	norm_coordinates = 0;
}
```

2.2. Demonstracja działania.

Rys.1. Oświetlone jajko.



3. Oświetlone jajko – dwa ruchome punkty oświetlenia.

Zaimplementowano na podstawie wytycznych zawartych w instrukcji do laboratorium.

3.1. Kod źródłowy.

```
//*************************************************************************
//
//  PLIK ŹRÓDŁOWY:          oswietlenie2.cpp
//
//  OPIS:                           Program służy do rysowania oświetlonej z dwóch miejsc sceny
3-D.
//
//  AUTOR:                          Weronika Luźna
//
//  DATA                            11 Stycznia 2016 (Versja 1.00).
//
//  PLATFORMA:                      System operacyjny:  Microsoft Windows 8.1.
//                                          Kompilator:         Microsoft Visual Studio 2015
//
//  UŻYTE BIBLIOTEKI Nie używano.
//  NIESTANDARDOWE
//
//*************************************************************************

#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <math.h>
#include <time.h>

typedef float point3[3];

const float PI = 3.14159265;
point3 **coordinates;
point3 **norm_coordinates;

static GLfloat       phi[2] = { 5.76f, 1.05f },  //kąty obrotu
theta[2] = { 4.68f, 4.68f };
static GLfloat       pix2angle_x = 0.0,          // przelicznik pikseli na stopnie
pix2angle_y = 0.0;
static GLint  status = 0;
static int    x_pos_old = 0,                          // poprzednia pozycja kursora myszy
y_pos_old = 0;
static int    delta_x = 0,
delta_y = 0;

int N = 50;
float verLength = 1.0;  //Dlugosc boku kwadratu
float lightsR = 10.0;       //Promien sfery swiatel

float Calculate_x(float u, float v) {
       float x, a = v*PI;

       x = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) *
cos(a);
       return x;
}
```

```
float Calculate_y(float u, float v) {
	float y;

	y = 160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2);
	return y - 5;
}

float Calculate_z(float u, float v) {
	float z, a = v*PI;

	z = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) *
sin(a);
	return z;
}

float Calculate_norm_x(float u, float v) {
	float x, a = v*PI;

	float yu = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
	float yv = 0;
	float zu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*sin(a);
	float zv = -PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) +
45 * u)*cos(a);

	x = (GLfloat)(yu*zv - zu*yv);
	return x;
}

float Calculate_norm_y(float u, float v) {
	float y, a = v*PI;

	float xu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*cos(a);
	float xv = PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45
* u)*sin(a);
	float zu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*sin(a);
	float zv = -PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) +
45 * u)*cos(a);

	y = (GLfloat)(zu*xv - xu*zv);
	return y;
}

float Calculate_norm_z(float u, float v) {
	float z, a = v*PI;

	float xu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45)*cos(a);
	float xv = PI*(90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45
* u)*sin(a);
	float yu = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
	float yv = 0;

	z = (GLfloat)(xu*yv - yu*xv);
	return z;
}

void Mesh() {
	float stepXY = verLength / N;

	for (int i = 0; i<N + 1; i++) {
		for (int j = 0; j<N + 1; j++) {
			coordinates[i][j][0] = j*stepXY;
```

```
                        coordinates[i][j][1] = i*stepXY;
                }
        }

        float u, v;
        for (int i = 0; i<N + 1; i++) {
                for (int j = 0; j<N + 1; j++) {
                        v = coordinates[i][j][0];
                        u = coordinates[i][j][1];
                        coordinates[i][j][0] = Calculate_x(u, v);
                        coordinates[i][j][1] = Calculate_y(u, v);
                        coordinates[i][j][2] = Calculate_z(u, v);

                        //Wyliczenie wspolrzednych wektorow normalnych do powierzchni jajka
                        float x = Calculate_norm_x(u, v);
                        float y = Calculate_norm_y(u, v);
                        float z = Calculate_norm_z(u, v);

                        //Normalizacja wektorow normalnych do powierzchni jajka
                        //Wektory na bokach jajka
                        if (i < N / 2) {
                                norm_coordinates[i][j][0] = x / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
                                norm_coordinates[i][j][1] = y / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
                                norm_coordinates[i][j][2] = z / (float)sqrt(pow(x, 2) + pow(y, 2) +
pow(z, 2));
                        }
                        if (i > N / 2) {
                                norm_coordinates[i][j][0] = -1.0*x / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
                                norm_coordinates[i][j][1] = -1.0*y / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
                                norm_coordinates[i][j][2] = -1.0*z / (float)sqrt(pow(x, 2) + pow(y,
2) + pow(z, 2));
                        }
                        //Wektory na "szczycie" jajka
                        if (i == N / 2) {
                                norm_coordinates[i][j][0] = 0;
                                norm_coordinates[i][j][1] = 1;
                                norm_coordinates[i][j][2] = 0;
                        }
                        //Wektory na "dnie" jajka
                        if (i == 0 || i == N)
                        {
                                norm_coordinates[i][j][0] = 0;
                                norm_coordinates[i][j][1] = -1;
                                norm_coordinates[i][j][2] = 0;
                        }
                }
        }
}

void Egg(void) {
        //Wygenerowanie siatki 3D punktow
        Mesh();

        glColor3f(1.0, 1.0, 1.0);

        for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                        glBegin(GL_TRIANGLES);
```

```
                    glNormal3fv(norm_coordinates[i][j + 1]);
                    glVertex3fv(coordinates[i][j + 1]);
                    glNormal3fv(norm_coordinates[i + 1][j]);
                    glVertex3fv(coordinates[i + 1][j]);
                    glNormal3fv(norm_coordinates[i + 1][j + 1]);
                    glVertex3fv(coordinates[i + 1][j + 1]);
                    glEnd();

                    glBegin(GL_TRIANGLES);
                    glNormal3fv(norm_coordinates[i][j]);
                    glVertex3fv(coordinates[i][j]);
                    glNormal3fv(norm_coordinates[i + 1][j]);
                    glVertex3fv(coordinates[i + 1][j]);
                    glNormal3fv(norm_coordinates[i][j + 1]);
                    glVertex3fv(coordinates[i][j + 1]);
                    glEnd();
                }
        }
}

void Axes(void)
{
        point3  x_min = { -2.0, 0.0, 0.0 };
        point3  x_max = { 2.0, 0.0, 0.0 };
        // pocz¹tek i koniec obrazu osi x

        point3  y_min = { 0.0, -2.0, 0.0 };
        point3  y_max = { 0.0,  2.0, 0.0 };
        // pocz¹tek i koniec obrazu osi y

        point3  z_min = { 0.0, 0.0, -2.0 };
        point3  z_max = { 0.0, 0.0,  2.0 };
        //  pocz¹tek 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();
}

void RenderScene(void)
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();

        if (status == 1) {   // jeœli lewy klawisz myszy wciœniêty
                theta[0] -= delta_x*pix2angle_x;
                //Ograniczenie dla azymutu
                if (theta[0] <= 0)
```

```
                        theta[0] += 2 * PI;
                if (theta[0] >= 2 * PI)
                        theta[0] -= 2 * PI;

                phi[0] -= delta_y*pix2angle_y;
                //Ograniczenie dla elewacji
                if (phi[0] <= 0)
                        phi[0] += 2 * PI;
                if (phi[0] >= 2 * PI)
                        phi[0] -= 2 * PI;
        }
        else if (status == 2) {     // jeœli prawy klawisz myszy wciœniêty
                theta[1] -= delta_x*pix2angle_x;
                //Ograniczenie dla azymutu
                if (theta[1] <= 0)
                        theta[1] += 2 * PI;
                if (theta[1] >= 2 * PI)
                        theta[1] -= 2 * PI;

                phi[1] -= delta_y*pix2angle_y;
                //Ograniczenie dla elewacji
                if (phi[1] <= 0)
                        phi[1] += 2 * PI;
                if (phi[1] >= 2 * PI)
                        phi[1] -= 2 * PI;
        }

        gluLookAt(0.0, 0.0, -10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        // Zdefiniowanie po³o¿enia obserwatora

        GLfloat lights_positions[4] = { 0 };
        lights_positions[0] = lightsR * cos(theta[0]) * cos(phi[0]);
        lights_positions[1] = lightsR * sin(phi[0]);
        lights_positions[2] = lightsR * sin(theta[0]) * cos(phi[0]);
        lights_positions[3] = 1.0;
        glLightfv(GL_LIGHT0, GL_POSITION, lights_positions);
        //Aktualizacja pozycji swiatla 0

        lights_positions[0] = lightsR * cos(theta[1]) * cos(phi[1]);
        lights_positions[1] = lightsR * sin(phi[1]);
        lights_positions[2] = lightsR * sin(theta[1]) * cos(phi[1]);
        lights_positions[3] = 1.0;
        glLightfv(GL_LIGHT1, GL_POSITION, lights_positions);
        //Aktualizacja pozycji swiatla 1

        Axes();

        Egg();

        glFlush();
        glutSwapBuffers();
}

// Funkcja ustalaj¹ca stan renderowania
void MyInit(void)
{
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
        // Kolor czyszc¹cy (wype³nienia okna) ustawiono na czarny

        // Definicja materia³u z jakiego zrobiony jest przedmiot
        //----------------------------------------------------------
        GLfloat mat_ambient[] = { 0.3f, 0.3f, 0.3f, 1.0f };
```

```
	// wspó³czynniki ka =[kar,kag,kab] dla œwiat³a otoczenia

	GLfloat mat_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
	// wspó³czynniki kd =[kdr,kdg,kdb] œwiat³a rozproszonego

	GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
	// wspó³czynniki ks =[ksr,ksg,ksb] dla œwiat³a odbitego

	GLfloat mat_shininess = { 100.0 };
	// wspó³czynnik n opisuj¹cy po³ysk powierzchni


	// Definicja Ÿród³a œwiat³a
	//-------------------------------------------------------
	GLfloat light_position[2][4] = { { -10.0, -10.0, -10.0, 1.0 },{ -10.0, -10.0, -10.0, 1.0
} };
	// po³o¿enie Ÿród³a

	GLfloat light_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a otoczenia
	// Ia = [Iar,Iag,Iab]

	GLfloat light_diffuse[2][4] = { { 1.0, 0.0, 0.0, 0.0 },{ 0.0, 0.0, 1.0, 1.0 } };
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a powoduj¹cego
	// odbicie dyfuzyjne Id = [Idr,Idg,Idb]

	GLfloat light_specular[2][4] = { { 1.0f, 1.0f, 0.0f, 1.0f },{ 0.7f, 0.7f, 1.0f,
1.0f } };
	// sk³adowe intensywnoœci œwiecenia Ÿród³a œwiat³a powoduj¹cego
	// odbicie kierunkowe Is = [Isr,Isg,Isb]

	GLfloat att_constant = { 1.0 };
	// sk³adowa sta³a ds dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	GLfloat att_linear = { 0.001f };
	// sk³adowa liniowa dl dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	GLfloat att_quadratic = { 0.001f };
	// sk³adowa kwadratowa dq dla modelu zmian oœwietlenia w funkcji
	// odleg³oœci od Ÿród³a

	// Ustawienie patrametrów materia³u
	//-------------------------------------------------------
	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 Ÿród³a œwiat³a
	//-------------------------------------------------------
	glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
	glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse[0]);
	glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular[0]);
	glLightfv(GL_LIGHT0, GL_POSITION, light_position[0]);

	glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, att_constant);
	glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, att_linear);
	glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, att_quadratic);

	glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient);
```

```
	glLightfv(GL_LIGHT1, GL_DIFFUSE, light_diffuse[1]);
	glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular[1]);
	glLightfv(GL_LIGHT1, GL_POSITION, light_position[1]);

	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 oœwietlania sceny
	//----------------------------------------------------------
	glShadeModel(GL_SMOOTH); // w³aczenie ³agodnego cieniowania
	glEnable(GL_LIGHTING);   // w³aczenie systemu oœwietlenia sceny
	glEnable(GL_LIGHT0);     // w³¹czenie Ÿród³a o numerze 0
	glEnable(GL_LIGHT1);     // w³¹czenie Ÿród³a o numerze 1
	glEnable(GL_DEPTH_TEST); // w³¹czenie mechanizmu z-bufora
}

void ChangeSize(GLsizei horizontal, GLsizei vertical) {
	pix2angle_x = 360.0*0.0125 / (float)horizontal; // przeliczenie pikseli na stopnie
	pix2angle_y = 360.0*0.0125 / (float)vertical;

	glMatrixMode(GL_PROJECTION);
	// Prze³¹czenie macierzy bie¿¹cej na macierz projekcji

	glLoadIdentity();
	// Czyszcznie macierzy bie¿¹cej

	gluPerspective(70.0, 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 wielkoœci okna okna widoku (viewport) w zale¿noœci
	// relacji pomiêdzy wysokoœci¹ i szerokoœci¹ okna

	glMatrixMode(GL_MODELVIEW);
	// Prze³¹czenie macierzy bie¿¹cej na macierz widoku modelu

	glLoadIdentity();
	// Czyszczenie macierzy bie¿¹cej
}

// Funkcja "bada" stan myszy i ustawia wartosci odpowiednich zmiennych globalnych
void Mouse(int btn, int state, int x, int y) {
	if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
		x_pos_old = x;        // przypisanie aktualnie odczytanej pozycji kursora
		y_pos_old = y;        // jako pozycji poprzedniej
		status = 1;          // wciœniêty zosta³ lewy klawisz myszy
	}
	else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
		x_pos_old = x;        // przypisanie aktualnie odczytanej pozycji kursora
		y_pos_old = y;        // jako pozycji poprzedniej
		status = 2;                //wciœniêty zosta³ prawy klawisz myszy
	}
	else
		status = 0;          // nie zosta³ wciœniêty ¿aden klawisz
}

// Funkcja "monitoruje" polozenie kursora myszy i ustawia wartosci odpowiednich
// zmiennych globalnych
```

```
void Motion(GLsizei x, GLsizei y) {
	delta_x = x - x_pos_old;	// obliczenie ró¿nicy po³o¿enia kursora myszy
	x_pos_old = x;		// podstawienie bie¿acego po³o¿enia jako poprzednie

	delta_y = y - y_pos_old;	// obliczenie ró¿nicy po³o¿enia kursora myszy
	y_pos_old = y;		// podstawienie bie¿acego po³o¿enia jako poprzednie

	glutPostRedisplay(); // przerysowanie obrazu sceny
}

void main(void)
{
	//Ziarno losowosci
	srand((unsigned)time(NULL));

	coordinates = new point3*[N + 1];
	for (int i = 0; i<N + 1; i++) {
		coordinates[i] = new point3[N + 1];
	}

	norm_coordinates = new point3*[N + 1];
	for (int i = 0; i < N + 1; i++) {
		norm_coordinates[i] = new point3[N + 1];
	}

	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(800, 600);

	glutCreateWindow("Oswietlone jajko, dwa zrodla swiatla");

	glutDisplayFunc(RenderScene);
	glutReshapeFunc(ChangeSize);
	MyInit();
	glutMouseFunc(Mouse);
	glutMotionFunc(Motion);
	glEnable(GL_DEPTH_TEST);
	glutMainLoop();

	for (int i = 0; i < N + 1; i++) {
		delete[] coordinates[i];
		delete[] norm_coordinates[i];
		coordinates[i] = 0;
		norm_coordinates[i] = 0;
	}
	delete[] coordinates;
	delete[] norm_coordinates;
	coordinates = 0;
	norm_coordinates = 0;
}
```

3.2. Demonstracja działania.

Rys.2. Oświetlone z dwóch miejsc jajko.