POLITECHNIKA WROCŁAWSKA

Grafika komputerowa i komunikacja człowiek-komputer

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 4

OpenGL - oświetlanie scen 3-D

Wykonał:	**Janusz Pelc 252799**
Termin:	**Np. PN/TP 7:30-10:30**
Data wykonania ćwiczenia:	**06 Grudnia 2021**
Data oddania sprawozdania:	**13 Grudnia 2021**
Ocena:	

Uwagi prowadzącego:

1. **Wstęp**

Ćwiczenie polegało na zaprogramowaniu kontroli oświetlenia obiektu 3D. Należało zaprogramować źródła światła oświetlające rysowany obiekt 3D. Następnie należało dodać możliwość zmiany pozycji świateł na pomocą myszy. W instrukcji zadania zostały przedstawione modyfikacje kodu z poprzedniego zadania pozwalające na wyświetlenie czajnika z funkcji glutSolidTeapot() oświetlonego jednym światłem z możliwością obrotu kamerą.

Rys1. glutSolidTeapot() oświetlony jednym światłem

Do oświetlenia obiektu zastosowano model Phonga. Służy on do obliczenia oświetlenia punktu leżącego na powierzchni obiektu 3D. Pozwala na połączenie własności światła otoczenia, rozproszonego i odbitego. Model składa się z trzech składowych R, G, B określających intensywność światła o danym kolorze padającego na punkt.

$$I_R = k_{aR} \cdot I_{aR} + \frac{1}{(a + bd_l + cd_l^2)}\left(k_{dR} \cdot I_{dR} \cdot (\overline{N} \cdot \overline{L}) + k_{sR} \cdot I_{sR}(\overline{R} \cdot \overline{V})^n\right)$$

$$I_G = k_{aG} \cdot I_{aG} + \frac{1}{(a + bd_l + cd_l^2)}\left(k_{dG} \cdot I_{dG} \cdot (\overline{N} \cdot \overline{L}) + k_{sG} \cdot I_{sG}(\overline{R} \cdot \overline{V})^n\right)$$

$$I_B = k_{aB} \cdot I_{aB} + \frac{1}{(a + bd_l + cd_l^2)}\left(k_{dB} \cdot I_{dB} \cdot (\overline{N} \cdot \overline{L}) + k_{sB} \cdot I_{sB}(\overline{R} \cdot \overline{V})^n\right)$$

- k – współczynnik światła danego typu rozróżnianie przez indeksy:
  - a – otoczenia
  - d – rozproszonego
  - s – odbitego
  - R – czerwony
  - G – zielony
  - B – niebieski
- I – składowa intensywności źródła światła danego typu rozróżnianie przez indeksy
- n – współczynnik połysku

- N – wektor normalny jest wektorem prostopadłym do powierzchni obiektu i jest wykorzystywaniu przy kalkulacji padającego na wierzchołek światła
- L – kierunek padania światła
- R – kierunek odbicia światła
- V – kierunek obserwacji punktu

2. Program

2.1. Zadanie 1

Pierwsze zadanie polegało modyfikacji kodu z poprzedniego zadania poprzez dodanie źródła światła. Do funkcji MyInit () dodani następujący kod z instrukcji:

```
/*************************************************************************************/
//  Definicja materiału z jakiego zrobiony jest czajnik
//  i definicja źródła światła
/*************************************************************************************/

/*************************************************************************************/
// Definicja materiału z jakiego zrobiony jest czajnik

    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_position[] = {0.0, 0.0, 10.0, 1.0};
    // położenie źródła

    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, 1.0, 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 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_DEPTH_TEST); // włączenie mechanizmu z-bufora

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

glMaterialfv() – określa parametry materiału z jakiego będzie stworzony obiekt 3D. Jako paramtry przyjmuje:

- Face – strona powierzchni obiektu, którą światło oświetlana, w tym przypadku oświetlamy przód, czyli GL_FRONT
- Parametr, który chcemy zmienić, w tym przypadku zmieniamy:
  - GL_SPECULAR – współczynnik dla światła odbitego
  - GL_AMBIENT – współczynnik dla światła otoczenia
  - GL_DIFFUSE – współczynnik dla światła rozproszonego
  - GL_SHININESS – współczynnik połysku powierzchni

  Współczynniki dla światła są tablicami określającymi kolor powierzchni, w formacie RGBA, dla danego typu światła
- Docelowa wartość
-

glLightfv() – określa parametry źródła świaitła. Jako parametry przyjmuje:

- Źródło światła – światło dla którego chcemy zmienić parametry
- Parametr, który chcemy zmienić, w tym przypadku zmieniamy:
  - GL_SPECULAR – współczynnik dla światła odbitego
  - GL_AMBIENT – współczynnik dla światła otoczenia
  - GL_DIFFUSE – współczynnik dla światła rozproszonego
  - GL_POSITION – współrzędne położenia światła
  - GL_CONSTANT_ATTENUATION – składowa stała dla odległości światła, niezależnie od odległości. Obiekty oświetlane są tak samo
  - GL_LINEAR_ATTENUATION składowa linowa dla odległości światła. Zanik światła jest liniowy
  - GL_QUADRATIC_ATTENUATION składowa kwadratowa dla odległości światła. Zanik światła jest kwadratowy

  Współczynniki dla światła są tablicami określającymi kolor światła, w formacie RGBA, dla danego typu światła. W zadaniu użyto stały zanik światła
- Docelowa wartość

Następnie zależało obliczyć wektory normalne dla każdego punktu wierzchołka obiektu.

$$x_u = \frac{\partial x(u,v)}{\partial u} = \left(-450u^4 + 900u^3 - 810u^2 + 360u - 45\right) \cdot \cos(\pi v)$$

$$x_v = \frac{\partial x(u,v)}{\partial v} = \pi \cdot \left(90u^5 - 225u^4 + 270u^3 - 180u^2 + 45u\right) \cdot \sin(\pi v)$$

$$y_u = \frac{\partial y(u,v)}{\partial u} = 640u^3 - 960u^2 + 320u$$

$$y_v = \frac{\partial y(u,v)}{\partial v} = 0$$

$$z_u = \frac{\partial z(u,v)}{\partial u} = \left(-450u^4 + 900u^3 - 810u^2 + 360u - 45\right) \cdot \sin(\pi v)$$

$$z_v = \frac{\partial z(u,v)}{\partial v} = -\pi \cdot \left(90u^5 - 225u^4 + 270u^3 - 180u^2 + 45u\right) \cdot \cos(\pi v)$$

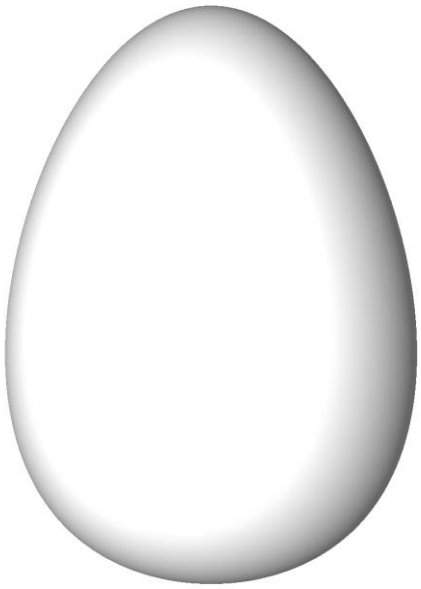
$$N(u,v) = \left[\begin{vmatrix} y_u & z_u \\ y_v & z_v \end{vmatrix}, \begin{vmatrix} z_u & x_u \\ z_v & x_v \end{vmatrix}, \begin{vmatrix} x_u & y_u \\ x_v & y_v \end{vmatrix} \right] =$$

$$= \left[y_u \cdot z_v - z_u y_v, \quad z_u \cdot x_v - x_u z_v, \quad x_u \cdot y_v - y_u x_v \right] \neq 0$$

Wektory normalne należało znormalizować

Następnie przy określaniu współrzędnych wierzchołków obiektu należało dodać funkcje glNormal3f(Nx, Ny, Nz) określająca wektor normalny dla danego wierchołka.

Ostatecznie otrzymano następujący rysunek



Rys1. Oświetlone jajko

2.2. Zadanie 2

Zadanie polegało na dodaniu drugiego źródła światła oraz możliwość zmiany pozycji światła za pomocą myszy, w ten sam sposób co obrót kamerą w poprzednim ćwiczeniu.

Dodano drugie źródło światła

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

	GLfloat light_specular1[] = { 1.0, 1.0, 1.0, 0.5 };
	// 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

/*****************************************************************************
/

	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);
```

Oraz włączono drugie źródło

```
glEnable(GL_LIGHT1);	// włączenie źródła o numerze 1
```

Dodano możliwość zmiany modelu kontroli myszą za pomocą wciśnięcia klawiszy:

- r – kontrola kamery
- l – kontrola położenia źródeł światła

Do funkcji key() dodano nowe klawisze

```
//fynkcja zwortna wyznaczają model rysowania jajka
void keys(unsigned char key, int x, int y)
{
	if (key == 'p') model = 1; //punkty
	if (key == 's') model = 2; //siatka
	if (key == 'w') model = 3; //wypełnione trójkąty
	if (key == 'r') status3 = 0; //rotacja kamery
	if (key == 'l') status3 = 1; //rotacja światła

	RenderScene(); // przerysowanie obrazu sceny
}
```

Funkcje obrotu kamery pozostawiono bez zmian, natomiast dodano funkcja obrotu światłami

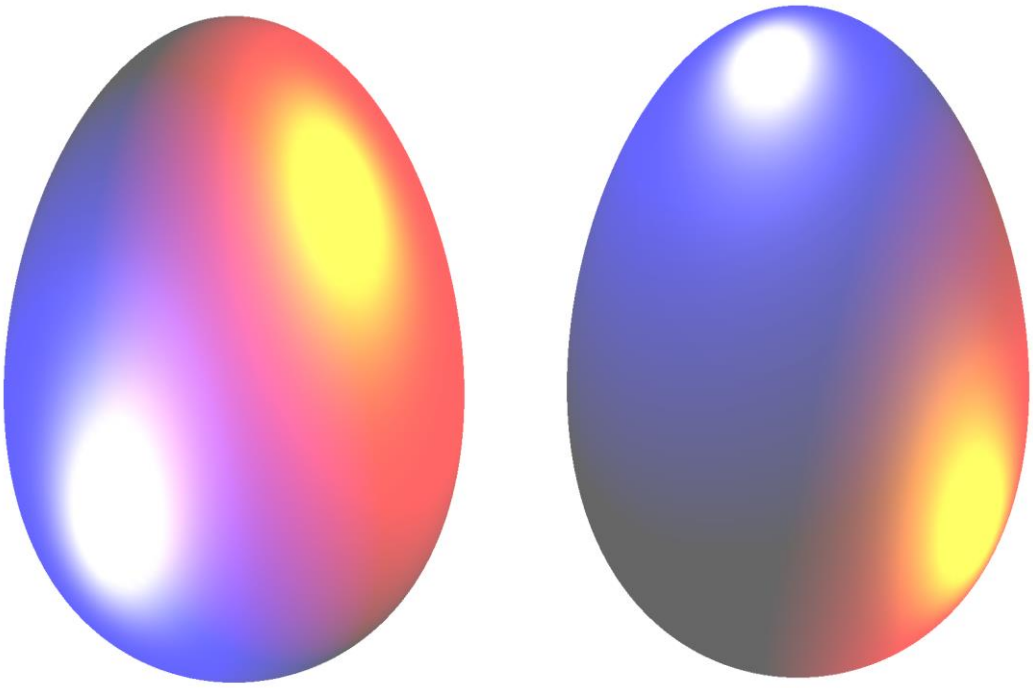
```
else if (status3 == 1) {		//jeśli model obrotu światłem (klawisz ‘l’)
		if (status1 == 1) {		// jeśli lewy klawisz myszy wciśnięty
```

```
                    thetax1 += delta_x * pix2angle / 20;     // modyfikacja kąta
obrotu o kat proporcjonalny do różnicy położeń kursora myszy
                    thetay1 += delta_y * pix2angle / 20;
                    if (thetay1 > M_PI / 2 - 0.000001) {
                           thetay1 = M_PI / 2 - 0.000001;
                    }
                    else if (thetay1 < -M_PI / 2 + 0.000001) {
                           thetay1 = -M_PI / 2 + 0.000001;
                    }
                    light_position[0] = R1 * cos(-thetax1) * cos(-thetay1);
                    light_position[1] = R1 * sin(-thetay1);
                    light_position[2] = R1 * sin(-thetax1) * cos(-thetay1);
             }
             else if (status2 == 1) {           // jeśli prawy klawisz myszy wciśnięty
                    thetax2 += delta_x * pix2angle / 20;     // modyfikacja kąta
obrotu o kat proporcjonalny do różnicy położeń kursora myszy
                    thetay2 += delta_y * pix2angle / 20;
                    if (thetay2 > M_PI / 2 - 0.000001) {
                           thetay2 = M_PI / 2 - 0.000001;
                    }
                    else if (thetay2 < -M_PI / 2 + 0.000001) {
                           thetay2 = -M_PI / 2 + 0.000001;
                    }
                    light_position1[0] = R1 * cos(-thetax2) * cos(-thetay2);
                    light_position1[1] = R1 * sin(-thetay2);
                    light_position1[2] = R1 * sin(-thetax2) * cos(-thetay2);
             }

      }
```

Kolory źródeł światła ustawiono na niebieski i czerwony

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

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

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

      GLfloat light_specular1[] = { 1.0, 1.0, 1.0, 0.5 };
      // składowe intensywności świecenia źródła światła powodującego
      // odbicie kierunkowe Is = [Isr,Isg,Isb]
```

Ostatecznie otrzymano następujący rysunek



3. Kod źródłowy

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

//  Szkielet programu do tworzenia modelu sceny 3-D z wizualizacją osi
//  układu współrzędnych dla rzutowania perspektywicznego

/*************************************************************************************
/
#define _USE_MATH_DEFINES

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

using namespace std;

typedef float point3[3];

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

static GLfloat thetax = 0.0;   // kąt obrotu obiektu
static GLfloat thetay = 0.0;   // kąt obrotu obiektu
static GLfloat thetax1 = 0.0;   // kąt obrotu obiektu
static GLfloat thetay1 = 0.0;   // kąt obrotu obiektu

static GLfloat thetax2 = 0.0;   // kąt obrotu obiektu
static GLfloat thetay2 = 0.0;   // kąt obrotu obiektu
```

```
static GLfloat pix2angle;     // przelicznik pikseli na stopnie
static GLfloat cameraz = 0.0;

static GLint status1 = 0;       // stan klawiszy myszy
static GLint status2 = 0;         // 0 - nie naciśnięto żadnego klawisza
                                                    // 1 - naciśnięty zostać lewy
klawisz
static GLint status3 = 1;         //model obrotu (1 - obrót światłami, 0 - obrót
kamerą)

static int x_pos_old = 0;       // poprzednia pozycja kursora myszy
static int y_pos_old = 0;

static int delta_x = 0;        // różnica pomiędzy pozycją bieżącą
static int delta_y = 0;                  // i poprzednią kursora myszy

static int R1 = 10;

const int N = 200; //rozmiar tablicy wierzchołków NxN
static GLfloat theta[] = { 0.0, 0.0, 0.0 }; // trzy kąty obrotu
int model = 2;  // 1- punkty, 2- siatka, 3 - wypełnione trójkąty

//tablice współrzędnych wierzchołków
float x[N][N];
float y[N][N];
float z[N][N];

float Nx[N][N];
float Ny[N][N];
float Nz[N][N];



//tablice wartości RGB wierzchołków
int R[N][N];
int G[N][N];
int B[N][N];


GLfloat light_position[] = { 10.0, 0.0, 10.0, 1.0 };
// położenie źródła
GLfloat light_position1[] = { -10.0, 0.0, -10.0, 1.0 };
// położenie źródła
/*************************************************************************
/

/*************************************************************************
/
// Funkcja "bada" stan myszy i ustawia wartości 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
              status1 = 1;          // wciśnięty został 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
		status2 = 1;		// wciśnięty został lewy klawisz myszy


	}

	else	status1 = status2 = 0;		// nie został wciśnięty żaden klawisz
}

/*************************************************************************
/
// Funkcja "monitoruje" położenie kursora myszy i ustawia wartości odpowiednich
// zmiennych globalnych

void Motion(GLsizei x, GLsizei y) {

	delta_x = x - x_pos_old;	// obliczenie różnicy położenia kursora myszy

	x_pos_old = x;			// podstawienie bieżącego położenia jako poprzednie

	delta_y = y - y_pos_old;	// obliczenie różnicy położenia kursora myszy

	y_pos_old = y;			// podstawienie bieżącego położenia jako poprzednie

	glutPostRedisplay();	// przerysowanie obrazu sceny
}

//Funkcja losująca kolor dla każdego wierzchołka
void Colors() {
	for (int i = 0; i < N; i++) {
		for (int k = 0; k < N; k++) {
			//dla wartości RGB losuje 0 lub 1
			if (x[i][k] == 0 && z[i][k] == 0) {
				R[i][k] = 1;
				G[i][k] = 1;
				B[i][k] = 1;
			}
			else {
				R[i][k] = rand() % 2;
				G[i][k] = rand() % 2;
				B[i][k] = rand() % 2;
			}
		}
	}
}

void initEgg() {
	float u = 0, v = 0; //zmienne u i v wykorzystywane w funkcjach określających
wierzchołki jajka;
	float fN = N; //wartość stałej N jako float
	float l=0;
	float xu;
	float yu;
	float zu;

	float xv;
	float yv;
	float zv;
```

```
	for (int i = 0; i < N; i++) {
		u = float(i / (fN)); //zmienna zmniejszana proporcjonalnie do stałej N, aby znajdowała się w zakresie użytej funkcji [0,1]
		for (int k = 0; k < N; k++) {
			v = float(k / (fN)); //zmienna jest zmniejszana proporcjonalnie do stałej N, aby znajdowała się w zakresie użytej funkcji [0,1]
			//na podstawie użytej funkcji wyznaczane są wierzchołki
			x[i][k] = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) * cos(M_PI * v);
			y[i][k] = 160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2) - 5;
			z[i][k] = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) * sin(M_PI * v);

			xu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45) * cos(M_PI * v);
			yu = 640 * pow(u, 3) - 960 * pow(u, 2) + 320 * u;
			zu = (-450 * pow(u, 4) + 900 * pow(u, 3) - 810 * pow(u, 2) + 360 * u - 45) * sin(M_PI * v);

			xv = M_PI * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45 * u) * sin(M_PI * v);
			yv = 0;
			zv = -M_PI * (90 * pow(u, 5) - 225 * pow(u, 4) + 270 * pow(u, 3) - 180 * pow(u, 2) + 45 * u) * cos(M_PI * v);

			Nx[i][k] = yu * zv - zu * yv;
			Ny[i][k] = zu * xv - xu * zv;
			Nz[i][k] = xu * yv - yu * xv;

			l = sqrt(Nx[i][k] * Nx[i][k] + Ny[i][k] * Ny[i][k] + Nz[i][k] * Nz[i][k]);
			Nx[i][k] /= l;
			Ny[i][k] /= l;
			Nz[i][k] /= l;

			if (i > N / 2) {
				Nx[i][k] *= -1;
				Ny[i][k] *= -1;
				Nz[i][k] *= -1;
			}
		}
	}
	Colors();
	//wyznaczenie kolorów wierzchołków
}


//Funkcja rysująca jajko na 3 różne sposoby
void Egg(void) {

	if (model == 1) { //punkty
		glColor3f(1, 1, 0); //kolor rysowania jest żółty
		for (int i = 0; i < N; i++) {
			for (int k = 0; k < N; k++) {
				glBegin(GL_POINTS);
				glVertex3f(x[i][k], y[i][k], z[i][k]);
				glEnd();
			}
		}
	}

	else if (model == 2) { //siatka
```

```
		glColor3f(1, 1, 0); //kolor rysowania jest żółty

		//rysowane są pionowe linie siatki jajka
		for (int i = 0; i < N; i++) {
			for (int k = 0; k < N; k++) {
				glBegin(GL_LINES); //funkcja rysuje linie
				glVertex3f(x[i][k], y[i][k], z[i][k]);
				glVertex3f(x[(i + 1) % N][k], y[(i + 1) % N][k], z[(i + 1)
% N][k]);
				glEnd();

			}
		}

		//rysowane są ukośne linie siatki jajka
		for (int k = 0; k < N - 1; k++) {
			int i;
			for (i = 0; i < (N - 1) / 2; i++) {
				glBegin(GL_LINES);
				glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);
				glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
				glEnd();
			}
			for (; i < N - 1; i++) {
				glBegin(GL_LINES);
				glVertex3f(x[i][k], y[i][k], z[i][k]);
				glVertex3f(x[i + 1][k + 1], y[i + 1][k + 1], z[i + 1][k +
1]);
				glEnd();
			}
		}

		//rysowane są ukośne linie łączące krawędzie siatki v=0 i v=N-1
		//kierunek skośnych linii jest zamieniany w połowie współrzędnej "u"
tablicy wierzchołków,
		//aby kierunek wszystkich ukośnych linii na jajku był taki sam
		int i;
		for (i = 0; i < N / 2; i++) {
			glBegin(GL_LINES);
			glVertex3f(x[i + 2][N - 1], y[i + 2][N - 1], z[i + 2][N - 1]);
			glVertex3f(x[N - 1 - i][0], y[N - 1 - i][0], z[N - 1 - i][0]);
			glEnd();
		}
		for (i; i < N - 1; i++) {
			glBegin(GL_LINES);
			glVertex3f(x[i + 1][N - 1], y[i + 1][N - 1], z[i + 1][N - 1]);
			glVertex3f(x[N - 2 - i][0], y[N - 2 - i][0], z[N - 2 - i][0]);
			glEnd();
		}

		//rysowane są poziome linie siatki jajka
		for (int i = 0; i < N; i++) {
			for (int k = 0; k < N - 1; k++) {
				glBegin(GL_LINES);
				glVertex3f(x[i][k], y[i][k], z[i][k]);
				glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
				glEnd();
			}
		}

		//rysowane są poziome linie łączące krawędzie siatki v=0 i v=N-1
		for (int i = 0; i < N - 1; i++) {
```

```
                        glBegin(GL_LINES);
                        glVertex3f(x[i + 1][N - 1], y[i + 1][N - 1], z[i + 1][N - 1]);
                        glVertex3f(x[N - 1 - i][0], y[N - 1 - i][0], z[N - 1 - i][0]);
                        glEnd();
                }

        }

        else if (model == 3) { //wypełnione trójkąty

                for (int i = 0; i < N - 1; i++) {
                        for (int k = 0; k < N - 1; k++) {
                                //rysowane są 2 trójkąty o wspólnym boku pomiędzy 4
sąsiadującymi w tablicy wierzchołkami
                                glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
                                glColor3f(R[i][k], G[i][k], B[i][k]);
                                glNormal3f(Nx[i][k], Ny[i][k], Nz[i][k]);
                                glVertex3f(x[i][k], y[i][k], z[i][k]);

                                glColor3f(R[i + 1][k], G[i + 1][k], B[i + 1][k]);
                                glNormal3f(Nx[i + 1][k], Ny[i + 1][k], Nz[i + 1][k]);
                                glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);

                                glColor3f(R[i][k + 1], G[i][k + 1], B[i][k + 1]);
                                glNormal3f(Nx[i][k + 1], Ny[i][k + 1], Nz[i][k + 1]);
                                glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
                                glEnd();

                                glBegin(GL_POLYGON);
                                glColor3f(R[i + 1][k + 1], G[i + 1][k + 1], B[i + 1][k +
1]);
                                glNormal3f(Nx[i + 1][k + 1], Ny[i + 1][k + 1], Nz[i + 1][k
+ 1]);
                                glVertex3f(x[i + 1][k + 1], y[i + 1][k + 1], z[i + 1][k +
1]);

                                glColor3f(R[i + 1][k], G[i + 1][k], B[i + 1][k]);
                                glNormal3f(Nx[i + 1][k], Ny[i + 1][k], Nz[i + 1][k]);
                                glVertex3f(x[i + 1][k], y[i + 1][k], z[i + 1][k]);

                                glColor3f(R[i][k + 1], G[i][k + 1], B[i][k + 1]);
                                glNormal3f(Nx[i][k + 1], Ny[i][k + 1], Nz[i][k + 1]);
                                glVertex3f(x[i][k + 1], y[i][k + 1], z[i][k + 1]);
                                glEnd();
                        }
                }

                //Rysowane są trójkąty pomiędzy krawędziami u=0 i u=N-1
                for (int i = 0; i < N - 1; i++) {
                        glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
                        glColor3f(R[0][i], G[0][i], B[0][i]);
                        glNormal3f(Nx[0][i], Ny[0][i], Nz[0][i]);
                        glVertex3f(x[0][i], y[0][i], z[0][i]);

                        glColor3f(R[N - 1][i + 1], G[N - 1][i + 1], B[N - 1][i + 1]);
                        glNormal3f(Nx[N - 1][i + 1], Ny[N - 1][i + 1], Nz[N - 1][i + 1]);
                        glVertex3f(x[N - 1][i + 1], y[N - 1][i + 1], z[N - 1][i + 1]);

                        glColor3f(R[N - 1][i], G[N - 1][i], B[N - 1][i]);
                        glNormal3f(Nx[N - 1][i], Ny[N - 1][i], Nz[N - 1][i]);
                        glVertex3f(x[N - 1][i], y[N - 1][i], z[N - 1][i]);
                        glEnd();
```

```
		}

		// rysowane są trójkąty między krawędziami v=0 i v=N-1
		for (int i = 0; i < N - 1; i++) {
			glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
			glColor3f(R[i][N - 1], G[i][N - 1], B[i][N - 1]);
			glNormal3f(Nx[i][N - 1], Ny[i][N - 1], Nz[i][N - 1]);
			glVertex3f(x[i][N - 1], y[i][N - 1], z[i][N - 1]);

			glColor3f(R[i + 1][N - 1], G[i + 1][N - 1], B[i + 1][N - 1]);
			glNormal3f(Nx[i + 1][N - 1], Ny[i + 1][N - 1], Nz[i + 1][N - 1]);
			glVertex3f(x[i + 1][N - 1], y[i + 1][N - 1], z[i + 1][N - 1]);

			glColor3f(R[N - 1 - i][0], G[N - 1 - i][0], B[N - 1 - i][0]);
			glNormal3f(Nx[N - 1 - i][0], Ny[N - 1 - i][0], Nz[N - 1 - i][0]);
			glVertex3f(x[N - 1 - i][0], y[N - 1 - i][0], z[N - 1 - i][0]);
			glEnd();
		}
		for (int i = 0; i < N - 1; i++) {
			glBegin(GL_POLYGON);  //funkcja rysuje wielokąt
			glColor3f(R[i + 1][N - 1], G[i + 1][N - 1], B[i + 1][N - 1]);
			glNormal3f(Nx[i + 1][N - 1], Ny[i + 1][N - 1], Nz[i + 1][N - 1]);
			glVertex3f(x[i + 1][N - 1], y[i + 1][N - 1], z[i + 1][N - 1]);

			glColor3f(R[N - 1 - i][0], G[N - 1 - i][0], B[N - 1 - i][0]);
			glNormal3f(Nx[N - 1 - i][0], Ny[N - 1 - i][0], Nz[N - 1 - i][0]);
			glVertex3f(x[N - 1 - i][0], y[N - 1 - i][0], z[N - 1 - i][0]);

			glColor3f(R[N - 2 - i][0], G[N - 2 - i][0], B[N - 2 - i][0]);
			glNormal3f(Nx[N - 2 - i][0], Ny[N - 2 - i][0], Nz[N - 2 - i][0]);
			glVertex3f(x[N - 2 - i][0], y[N - 2 - i][0], z[N - 2 - i][0]);
			glEnd();
		}
	}
}

void RenderScene(void)
{

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	// Czyszczenie okna aktualnym kolorem czyszczącym

	glLoadIdentity();
	// Czyszczenie macierzy bie??cej

	gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
	// Zdefiniowanie położenia obserwatora
	//Axes();
	// Narysowanie osi przy pomocy funkcji zdefiniowanej powyżej

	if (status3 == 0) {                    // jeśli model obrotu kamerą
		if (status1 == 1) {                // jeśli lewy klawisz myszy wciśnięty
			thetax += delta_x * pix2angle / 20;      // modyfikacja kąta
obrotu o kat proporcjonalny do różnicy położeń kursora myszy
			thetay += delta_y * pix2angle / 20;
			if (thetay > M_PI / 2 - 0.000001) {
				thetay = M_PI / 2 - 0.000001;
			}
			else if (thetay < -M_PI / 2 + 0.000001) {
				thetay = -M_PI / 2 + 0.000001;
			}
		}
```

```
		else if (status2 == 1) {
			R1 += delta_y;
			if (R1 < 1) R1 = 1;
			else if (R1 > 25) R1 = 25;
		}


		viewer[0] = R1 * cos(thetax) * cos(thetay);
		viewer[1] = R1 * sin(thetay);
		viewer[2] = R1 * sin(thetax) * cos(thetay);
	}
	else if (status3 == 1) {                    //jeśli model obrotu światłem
		if (status1 == 1) {               // jeśli lewy klawisz myszy wciśnięty
			thetax1 += delta_x * pix2angle / 20;     // modyfikacja kąta
obrotu o kat proporcjonalny do różnicy położeń kursora myszy
			thetay1 += delta_y * pix2angle / 20;
			if (thetay1 > M_PI / 2 - 0.000001) {
				thetay1 = M_PI / 2 - 0.000001;
			}
			else if (thetay1 < -M_PI / 2 + 0.000001) {
				thetay1 = -M_PI / 2 + 0.000001;
			}
			light_position[0] = R1 * cos(-thetax1) * cos(-thetay1);
			light_position[1] = R1 * sin(-thetay1);
			light_position[2] = R1 * sin(-thetax1) * cos(-thetay1);
		}
		else if (status2 == 1) {
			thetax2 += delta_x * pix2angle / 20;     // modyfikacja kąta
obrotu o kat proporcjonalny do różnicy położeń kursora myszy
			thetay2 += delta_y * pix2angle / 20;
			if (thetay2 > M_PI / 2 - 0.000001) {
				thetay2 = M_PI / 2 - 0.000001;
			}
			else if (thetay2 < -M_PI / 2 + 0.000001) {
				thetay2 = -M_PI / 2 + 0.000001;
			}
			light_position1[0] = R1 * cos(-thetax2) * cos(-thetay2);
			light_position1[1] = R1 * sin(-thetay2);
			light_position1[2] = R1 * sin(-thetax2) * cos(-thetay2);
		}

	}
	glRotatef(90, 0.0, 1.0, 0.0);  //obrót obiektu o nowy kąt
	//glRotatef(thetay, 1.0, 0.0, 0.0);  //obrót obiektu o nowy kąt

	glLightfv(GL_LIGHT0, GL_POSITION, light_position);
	glLightfv(GL_LIGHT1, GL_POSITION, light_position1);

	glColor3f(1.0f, 1.0f, 1.0f);
	// Ustawienie koloru rysowania na biały
	Egg();
	//glutSolidTeapot(3.0);
	// Narysowanie czajnika
	glFlush();
	// Przekazanie poleceń rysujących do wykonania
	glutSwapBuffers();
}
/*************************************************************************************
/

// Funkcja ustalająca stan renderowania
```

```
void MyInit(void)
{
/*************************************************************************************
/
// Definicja materiału z jakiego zrobiony jest czajnik

	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_position[] = { -10.0, 0.0, -10.0, 1.0 };
	// położenie źródła


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

	GLfloat light_specular[] = { 1.0, 1.0, 0.0, 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);

	/*************************************************************************
******/
// Definicja źródła światła




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

	GLfloat light_specular1[] = { 1.0, 1.0, 1.0, 0.5 };
	// 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

/*************************************************************************
/

	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 1

	glEnable(GL_DEPTH_TEST); // włączenie mechanizmu z-bufora

/*************************************************************************
/

	glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
	// Kolor czyszczący (wypełnienia okna) ustawiono na czarny
	initEgg();
}

/*************************************************************************
/


// 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;  // przeliczenie pikseli na stopnie

	glMatrixMode(GL_PROJECTION);
	// Przełączenie macierzy bieżącej na macierz projekcji

	glLoadIdentity();
	// Czyszcznie macierzy bieżącej

	gluPerspective(70, 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 zależności
	// relacji pomiędzy wysokością i szerokością okna

	glMatrixMode(GL_MODELVIEW);
	// Przełączenie macierzy bieżącej na macierz widoku modelu

	glLoadIdentity();
	// Czyszczenie macierzy bieżącej

}

//fynkcja zwortna wyznaczają model rysowania jajka
void keys(unsigned char key, int x, int y)
```

```
{
	if (key == 'p') model = 1; //punkty
	if (key == 's') model = 2; //siatka
	if (key == 'w') model = 3; //wypełnione trójkąty
	if (key == 'r') status3 = 0; //rotacja kamery
	if (key == 'l') status3 = 1; //rotacja światła

	RenderScene(); // przerysowanie obrazu sceny
}

/*************************************************************************************
/

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

void main(void)
{

	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(300, 300);

	glutCreateWindow("Rzutowanie perspektywiczne");

	glutDisplayFunc(RenderScene);
	// Określenie, że funkcja RenderScene będzie funkcją zwrotną
	// (callback function).  Będzie ona wywoływana za każdym razem
	// gdy zajdzie potrzeba przerysowania okna


	glutReshapeFunc(ChangeSize);
	// Dla aktualnego okna ustala funkcję zwrotną odpowiedzialną
	// za zmiany rozmiaru okna


	MyInit();
	// Funkcja MyInit() (zdefiniowana powyżej) wykonuje wszelkie
	// inicjalizacje konieczne  przed przystąpieniem do renderowania
	glEnable(GL_DEPTH_TEST);
	// Włączenie mechanizmu usuwania niewidocznych elementów sceny
	glutKeyboardFunc(keys);
	glutMouseFunc(Mouse);
	// Ustala funkcję zwrotną odpowiedzialną za badanie ruchu myszy
	glutMotionFunc(Motion);

	glutMainLoop();
	// Funkcja uruchamia szkielet biblioteki GLUT
}

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