POLITECHNIKA WROCŁAWSKA
Instytut Informatyki, Automatyki i Robotyki
Zakład Systemów Komputerowych

Grafika komputerowa

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 4

OPEN GL – interakcja z użytkownikiem

Wykonała:	**Weronika Luźna**
Termin:	**PN/N 16.15-19.15**
Data wykonania ćwiczenia:	**07.12.2015**
Data oddania sprawozdania:	**11.01.2016**
Ocena:	

Uwagi prowadzącego:

1. Cel projektu.

Celem projektu było zaimplementowanie 3 programów umożliwiających interakcję z użytkownikiem.

2. Obracanie i przybliżanie obiektu – czajnik.

Zaimplementowano na podstawie wytycznych zawartych w instrukcji do laboratorium.

2.1. Kod źródłowy.

```
//*************************************************************************
//
//  PLIK ?RÓD?OWY:          interakcja_czajnik.cpp
//
//  OPIS:                           Program s?u?y do rysowania sceny 3-D
//                                         z mo?liwo?ci? interakcji u?ytkownika.
//
//  AUTOR:                          Weronika Lu?na
//
//  DATA                            7 Grudnia 2015 (Versja 1.00).
//
//  PLATFORMA:                      System operacyjny:  Microsoft Windows 8.1.
//                                         Kompilator:         Microsoft Visual Studio 2015
//
//  MATERIA£Y               http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_4_dz/
//     ?RÓD?OWE:
//
//  U?YTE BIBLIOTEKI Nie u?ywano.
//  NIESTANDARDOWE
//
//*************************************************************************

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

typedef float point3[3];

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

static GLfloat       theta_y = 0.0,
theta_x = 0.0;
static GLfloat       pix2angle_x,  // przelicznik pikseli na stopnie
pix2angle_y;
static GLint  status = 0;          // stan klawiszy myszy

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

void Axes(void) {
       point3  x_min = { -5.0, 0.0, 0.0 };
       point3  x_max = { 5.0, 0.0, 0.0 };

       point3  y_min = { 0.0, -5.0, 0.0 };
       point3  y_max = { 0.0,  5.0, 0.0 };
```

```
	point3  z_min = { 0.0, 0.0, -5.0 };
	point3  z_max = { 0.0, 0.0,  5.0 };

	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 == 1) {   // jeœli lewy klawisz myszy wciœniêty
		theta_x += delta_x*pix2angle_x;          // modyfikacja k¹ta obrotu o k¹t
proporcjonalny
		theta_y += delta_y*pix2angle_y;          // do ró¿nicy po³o¿eñ kursora myszy
	}
	else if (status == 2) {
		viewer[2] += delta_y;
		if (viewer[2] <= 4.0)       // ograniczenie zblizenia
			viewer[2] = 4.0;
		if (viewer[2] >= 30) // ograniczenie oddalenia
			viewer[2] = 30;
	}

	gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

	Axes();

	glRotatef(theta_x, 0.0, 1.0, 0.0);
	glRotatef(theta_y, 1.0, 0.0, 0.0);

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

	glutWireTeapot(3.0);

	glFlush();
	glutSwapBuffers();
}

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

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

```
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	gluPerspective(80, 1.0, 1.0, 30.0);

	if (horizontal <= vertical)
		glViewport(0, (vertical - horizontal) / 2, horizontal, horizontal);
	else
		glViewport((horizontal - vertical) / 2, 0, vertical, vertical);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

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) {
		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) {
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(600, 600);

	glutCreateWindow("Rzutowanie perspektywiczne, prosta interakcja");

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

3. Zmiana położenia obserwatora – jajko.

Stworzono w oparciu o algorytm podany w instrukcji do zadania.

3.1. Kod źródłowy:

```
//*************************************************************************
//
//  PLIK ŹRÓDŁOWY:          interakcja.cpp
//
//  OPIS:                           Program służy do rysowania sceny 3-D
//                                          z możliwością interakcji użytkownika.
//
//  AUTOR:                          Weronika Luźna
//
//  DATA                            7 Grudnia 2015 (Versja 1.00).
//
//  PLATFORMA:                      System operacyjny:  Microsoft Windows 8.1.
//                                          Kompilator:         Microsoft Visual Studio 2015
//
//  MATERIA£Y               http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/gk/lab/cw_4_dz/
//     ŹRÓDŁOWE:
//
//  UŻYTE BIBLIOTEKI Nie używano.
//  NIESTANDARDOWE
//
//*************************************************************************

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


typedef float point3[3];

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

static GLfloat theta = 0.0;   // kąt obrotu obiektu
static GLfloat phi = 0.0;
static GLfloat pix2angle;     // przelicznik pikseli na stopnie

static GLint status = 0;       // stan klawiszy myszy

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


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

static int zoom = 10;

const int n = 20;
const int R = 3;
const int r = 1;
```

```
const float PI = 3.1415;
point3 coordinates[n][n];
point3 colors[n][n];
int model = 3; // 1- punkty, 2- siatka, 3 - wypełnienie


//----------------------------------------------------------------------
// Funkcja badająca stan myszy.
//----------------------------------------------------------------------
void Mouse(int btn, int state, int x, int y)
{
	if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		x_pos_old = x;
		y_pos_old = y;				// przypisanie aktualnie odczytanej
pozycji kursora jako pozycji poprzedniej
		status = 1;				//lewy klawisz myszy
	}
	else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
	{
		y_pos_old = y;
		status = 2;				//prawy klawisz myszy
	}
	else
		status = 0;		// żaden klawisz
}

//----------------------------------------------------------------------
// Funkcja badająca położenie myszy.
//----------------------------------------------------------------------
void Motion(GLsizei x, GLsizei y)
{
	delta_x = x - x_pos_old;	// obliczenie róznicy położenia kursora myszy
	x_pos_old = x;

	delta_y = y - y_pos_old;
	y_pos_old = y;

	delta_zoom = y - zoom_pos_old;
	zoom_pos_old = y;

	glutPostRedisplay();	// przerysowanie obrazu sceny
}

//----------------------------------------------------------------------
// Obliczanie współrzędnych punktów siatki.
//----------------------------------------------------------------------
void CalculateCoordinates()
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			float u = i / (n - 1.0);
			float v = j / (n - 1.0);
			coordinates[i][j][0] = (-90 * u*u*u*u*u + 225 * u*u*u*u - 270 * u*u*u +
180 * u*u - 45 * u)*cos(PI*v);
			coordinates[i][j][1] = (160 * u*u*u*u - 320 * u*u*u + 160 * u*u) - 5;
			coordinates[i][j][2] = (-90 * u*u*u*u*u + 225 * u*u*u*u - 270 * u*u*u +
180 * u*u - 45 * u)*sin(PI*v);

			colors[i][j][0] = ((float)rand()) / RAND_MAX;
```

```
                    colors[i][j][1] = ((float)rand()) / RAND_MAX;
                    colors[i][j][2] = ((float)rand()) / RAND_MAX;

                }
        }
}

//-----------------------------------------------------------------------
// Rysowanie modelu.
//-----------------------------------------------------------------------
void Egg()
{
        switch (model) {
        case 1:
                glColor3f(1.0f, 1.0f, 0.0f);
                glBegin(GL_POINTS);
                for (int i = 0; i < n; i++)
                        for (int j = 0; j < n; j++)
                                glVertex3fv(coordinates[i][j]);
                glEnd();
                break;
        case 2:
                glColor3f(1.0f, 1.0f, 0.0f);
                glBegin(GL_LINES);
                for (int i = 0; i < n - 1; i++)
                        for (int j = 0; j < n - 1; j++)
                        {
                                glVertex3fv(coordinates[i][j]);
                                glVertex3fv(coordinates[(i)+1][j]);
                                glVertex3fv(coordinates[i][j]);
                                glVertex3fv(coordinates[i][(j + 1)]);
                                glVertex3fv(coordinates[i][j]);
                                glVertex3fv(coordinates[(i + 1)][(j + 1)]);
                        }
                glEnd();
                break;
        case 3:
                glBegin(GL_TRIANGLES);
                for (int i = 0; i < n - 1; i++)
                        for (int j = 0; j < n - 1; j++)
                        {
                                glColor3fv(colors[i][j]);
                                glVertex3fv(coordinates[i][j]);

                                glColor3fv(colors[i + 1][j]);;
                                glVertex3fv(coordinates[(i)+1][j]);

                                glColor3fv(colors[i][j + 1]);
                                glVertex3fv(coordinates[i][(j + 1)]);

                                glColor3fv(colors[i][j + 1]);
                                glVertex3fv(coordinates[i][(j + 1)]);

                                glColor3fv(colors[i + 1][j]);;
                                glVertex3fv(coordinates[(i)+1][j]);

                                glColor3fv(colors[i + 1][j + 1]);
                                glVertex3fv(coordinates[(i + 1)][(j + 1)]);
                        }
                glEnd();
        }
```

```
}

void Axes(void)
{
	point3  x_min = { -5.0, 0.0, 0.0 };
	point3  x_max = { 5.0, 0.0, 0.0 };

	point3  y_min = { 0.0, -5.0, 0.0 };
	point3  y_max = { 0.0, 5.0, 0.0 };

	point3  z_min = { 0.0, 0.0, -5.0 };
	point3  z_max = { 0.0, 0.0, 5.0 };

	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();

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

	Axes();

	if (status == 1)                    // jeśli lewy klawisz myszy wciśnięty
	{
		theta += 0.01*delta_x*pix2angle;
		phi += 0.01*delta_y*pix2angle;           // modyfikacja kąta obrotu o kąt
proporcjonalny do różnicy położeń kursora myszy
	}
	else if(status == 2)
	{
		zoom += delta_zoom;
	}

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

```
	glColor3f(1.0f, 1.0f, 1.0f);

	//glutWireTeapot(3.0);
	// Narysowanie czajnika

	Egg();
	glFlush();
	glutSwapBuffers();
}

void keys(unsigned char key, int x, int y)
{
	if (key == 'p') model = 1;
	if (key == 'w') model = 2;
	if (key == 's') model = 3;

	RenderScene(); // przerysowanie obrazu sceny
}

void MyInit(void)
{

	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);	// Kolor czyszczący ustawiono na czarny

}

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(70.0, 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);

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

	glLoadIdentity();	// Czyszczenie macierzy bieżącej
}

void main(void)
{

	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(300, 300);

	glutCreateWindow("Rzutowanie perspektywiczne");
	CalculateCoordinates();
```

```
	glutKeyboardFunc(keys);
	glutDisplayFunc(RenderScene);

	glutMouseFunc(Mouse);
	glutMotionFunc(Motion);

	glutReshapeFunc(ChangeSize);

	MyInit();

	glEnable(GL_DEPTH_TEST);

	glutMainLoop();
}
```

4. Zad.dom – łańcuch z torusów – interakcja.

4.1. Kod źródłowy:

```
//*************************************************************************
//
//  PLIK ?RÓD?OWY:          Source.cpp
//
//  OPIS:                           Program służy do interakcji użytkownika z łańcuchem
//                utworzonym z torusów.
//
//  AUTOR:                          Weronika Lu?na
//
//  DATA                            7 Grudnia 2015 (Versja 1.00).
//
//  PLATFORMA:                      System operacyjny:  Microsoft Windows 8.1.
//                                         Kompilator:         Microsoft Visual Studio 2015
//
//
//  U?YTE BIBLIOTEKI Nie u?ywano.
//  NIESTANDARDOWE
//
//*************************************************************************

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


typedef float point3[3];

static GLfloat viewer[] = { 0.0, 0.0, 30.0 };
// inicjalizacja po³o¿enia obserwatora

static GLfloat theta = 0.0;   // k¹t obrotu obiektu
static GLfloat phi = 0.0;
static GLfloat pix2angle;     // przelicznik pikseli na stopnie

static GLint status = 0;       // stan klawiszy myszy
                                                // 0 - nie naciœniêto ¿adnego klawisza
                                                // 1 - naciœniêty zostaæ lewy klawisz

static int y_pos_old = 0;
```

```
static int x_pos_old = 0;
static int zoom_pos_old = 0;
// poprzednia pozycja kursora myszy

static int delta_x = 0;
static int delta_y = 0;
static int delta_zoom = 0;
// ró¿nica pomiêdzy pozycj¹ bie¿¹c¹
// i poprzedni¹ kursora myszy

static int zoom = 25;

float p = 0;
float h = 0;
float q = 0;
const int n = 20;
const int R = 3;
const int r = 1;
const float PI = 3.1415;
point3 coordinates[n][n];
point3 colors[n][n];
int model = 3; // 1- punkty, 2- siatka, 3 - wype?nione trójk?ty



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

                        // Funkcja rysuj¹ca osie uk³adu wspó?rz?dnych

                        // Funkcja "bada" stan myszy i ustawia wartoœci 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;
              y_pos_old = y;                          // przypisanie aktualnie odczytanej
pozycji kursora
                                                             // jako pozycji poprzedniej
              status = 1;          // wciêniêty zosta³ lewy klawisz myszy
       }
       else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
       {

              y_pos_old = y;
              status = 2;
       }
       else
              status = 0;          // nie zosta³ wciêniêty ¿aden klawisz
}


void SpecialInput(int key, int x, int y)
{
       switch (key)
       {
       case GLUT_KEY_UP:
              if (h < 0.5) {
                     h += 0.1;
```

```
                        q += 1;
                }
                break;
        case GLUT_KEY_DOWN:
                if (h > -0.5) {
                        h -= 0.1;
                        q -= 1;
                }
                break;
        case GLUT_KEY_LEFT:
                if (p > -0.5) {
                        p -= 0.1;
                }
                break;
        case GLUT_KEY_RIGHT:
                if (p < 0.5) {
                        p += 0.1;
                }
                break;
        }

        glutPostRedisplay();
}


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

// Funkcja "monitoruje" po³o¿enie kursora myszy i ustawia wartoœci 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¿¹cego po³o¿enia jako poprzednie

        delta_y = y - y_pos_old;
        y_pos_old = y;

        delta_zoom = y - zoom_pos_old;
        zoom_pos_old = y;

        glutPostRedisplay();     // przerysowanie obrazu sceny
}

void CalculateCoordinates()
{
        for (int i = 0; i < n; i++)
        {
                for (int j = 0; j < n; j++)
                {
                        float u = i / (n - 1.0);
                        float v = j / (n - 1.0);
                        coordinates[i][j][0] = ((R + r*cos(2 * PI*v))*cos(2 * PI*u));
                        coordinates[i][j][1] = ((R + r*cos(2 * PI*v))*sin(2 * PI*u));
                        coordinates[i][j][2] = r*sin(2 * PI*v);

                        colors[i][j][0] = ((float)rand()) / RAND_MAX;
                        colors[i][j][1] = ((float)rand()) / RAND_MAX;
                        colors[i][j][2] = ((float)rand()) / RAND_MAX;
                }
        }
```

```
}

void Torus()
{
	switch (model) {
	case 1:
		glColor3f(1.0f, 1.0f, 0.0f);
		glBegin(GL_POINTS);
		for (int i = 0; i < n; i++)
			for (int j = 0; j < n; j++)
				glVertex3fv(coordinates[i][j]);
		glEnd();
		break;
	case 2:
		glColor3f(1.0f, 1.0f, 0.0f);
		glBegin(GL_LINES);
		for (int i = 0; i < n - 1; i++)
			for (int j = 0; j < n - 1; j++)
			{
				glVertex3fv(coordinates[i][j]);
				glVertex3fv(coordinates[(i)+1][j]);
				glVertex3fv(coordinates[i][j]);
				glVertex3fv(coordinates[i][(j + 1)]);
				glVertex3fv(coordinates[i][j]);
				glVertex3fv(coordinates[(i + 1)][(j + 1)]);
			}
		glEnd();
		break;
	case 3:
		glBegin(GL_TRIANGLES);
		for (int i = 0; i < n - 1; i++)
			for (int j = 0; j < n - 1; j++)
			{
				glColor3fv(colors[i][j]);
				glVertex3fv(coordinates[i][j]);

				glColor3fv(colors[i + 1][j]);;
				glVertex3fv(coordinates[(i)+1][j]);

				glColor3fv(colors[i][j + 1]);
				glVertex3fv(coordinates[i][(j + 1)]);

				glColor3fv(colors[i][j + 1]);
				glVertex3fv(coordinates[i][(j + 1)]);

				glColor3fv(colors[i + 1][j]);;
				glVertex3fv(coordinates[(i)+1][j]);

				glColor3fv(colors[i + 1][j + 1]);
				glVertex3fv(coordinates[(i + 1)][(j + 1)]);
			}
		glEnd();
	}

}

void Chain() {

	Torus();
	glTranslatef((3 * R / 2)+p, 0+h, 0);
	glRotated(90.0, 1.0, 0.0, 0.0);
	Torus();
```

```
        glTranslatef((3 * R / 2)+p, 0+h, 0);
        glRotated(90.0, 1.0, 0.0, 0.0);
        Torus();
        glTranslatef((3 * R / 2)+p, (R / 2)+h, 0);
        glRotated(90.0, 1.0, 0.0, 0.0);
        glRotated(30.0+q, 0.0, 1.0, 0.0);
        Torus();
        glTranslatef((3 * R / 2)+p, 0+h, 0);
        glRotated(90.0, 1.0, 0.0, 0.0);
        Torus();
        glTranslatef(R+p, R+h, 0);
        glRotated(90.0, 1.0, 0.0, 0.0);
        glRotated(60.0+q, 0.0, 1.0, 0.0);
        Torus();
        glTranslatef((3 * R / 2)+p, (R / 2)+h, 0);
        glRotated(90.0, 1.0, 0.0, 0.0);
        glRotated(30.0+q, 0.0, 1.0, 0.0);
        Torus();
        glTranslatef((3 * R / 2)+p, (R / 2)+h, 0);
        glRotated(90.0, 1.0, 0.0, 0.0);
        glRotated(30.0+q, 0.0, 1.0, 0.0);
        Torus();

}

void Axes(void)
{

        point3  x_min = { -5.0, 0.0, 0.0 };
        point3  x_max = { 5.0, 0.0, 0.0 };
        // pocz?tek i koniec obrazu osi x

        point3  y_min = { 0.0, -5.0, 0.0 };
        point3  y_max = { 0.0, 5.0, 0.0 };
        // pocz?tek i koniec obrazu osi y

        point3  z_min = { 0.0, 0.0, -5.0 };
        point3  z_max = { 0.0, 0.0, 5.0 };
        //  pocz?tek 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 okreœlaj¹ca co ma byæ rysowane (zawsze wywo³ywana, gdy trzeba
// przerysowaæ scenê)



void RenderScene(void)
{

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	// Czyszczenie okna aktualnym kolorem czyszcz¹cym

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

	gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, cos(phi), 0.0);
	// Zdefiniowanie po³o¿enia obserwatora

	Axes();
	// Narysowanie osi przy pomocy funkcji zdefiniowanej powy¿ej

	if (status == 1)                    // jeœli lewy klawisz myszy wciêniêty
	{
		theta += 0.01*delta_x*pix2angle;
		phi += 0.01*delta_y*pix2angle;
		// modyfikacja k¹ta obrotu o kat proporcjonalny
	}                                                                    // do ró¿nicy
po³o¿eñ kursora myszy

	else if (status == 2)
	{
		zoom += delta_zoom;
	}

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


	glColor3f(1.0f, 1.0f, 1.0f);
	// Ustawienie koloru rysowania na bia³y

	//glutWireTeapot(3.0);
	// Narysowanie czajnika
	glTranslatef(-5 * R, 0, 0);

	Chain();

	glFlush();
	// Przekazanie poleceñ rysuj¹cych do wykonania

	glutSwapBuffers();



}
/*************************************************************************************/
```

```
// Funkcja ustalaj¹ca stan renderowania

void keys(unsigned char key, int x, int y)
{
	if (key == 'p') model = 1;
	if (key == 'w') model = 2;
	if (key == 's') model = 3;

	RenderScene(); // przerysowanie obrazu sceny
}

void MyInit(void)
{

	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
	// Kolor czyszcz¹cy (wype³nienia okna) ustawiono na czarny

}

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

// Funkcja ma za zadanie utrzymanie sta³ych proporcji rysowanych
// w przypadku zmiany rozmiarów okna.
// Parametry vertical i horizontal (wysokoœæ i szerokoœæ okna) s¹
// przekazywane do funkcji za ka¿dym razem gdy zmieni siê rozmiar okna.



void ChangeSize(GLsizei horizontal, GLsizei vertical)
{

	pix2angle = 360.0 / (float)horizontal;  // przeliczenie pikseli na stopnie

	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

}

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

```
// G³ówny punkt wejœcia programu. Program dzia³a w trybie konsoli


void main(void)
{
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(300, 300);

	glutCreateWindow("Rzutowanie perspektywiczne");
	CalculateCoordinates();
	glutKeyboardFunc(keys);
	glutSpecialFunc(SpecialInput);

	glutDisplayFunc(RenderScene);
	// Okreœlenie, ¿e funkcja RenderScene bêdzie funkcj¹ zwrotn¹
	// (callback function).  Bêdzie ona wywo³ywana za ka¿dym razem
	// gdy zajdzie potrzeba przerysowania okna

	glutMouseFunc(Mouse);
	// Ustala funkcjê zwrotn¹ odpowiedzialn¹ za badanie stanu myszy

	glutMotionFunc(Motion);
	// Ustala funkcjê zwrotn¹ odpowiedzialn¹ za badanie ruchu myszy

	glutReshapeFunc(ChangeSize);
	// Dla aktualnego okna ustala funkcjê zwrotn¹ odpowiedzialn¹
	// za zmiany rozmiaru okna



	MyInit();
	// Funkcja MyInit() (zdefiniowana powy¿ej) wykonuje wszelkie
	// inicjalizacje konieczne  przed przyst¹pieniem do renderowania

	glEnable(GL_DEPTH_TEST);
	// W³¹czenie mechanizmu usuwania niewidocznych elementów sceny

	glutMainLoop();
	// Funkcja uruchamia szkielet biblioteki GLUT

}



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