POLITECHNIKA WROCŁAWSKA
Instytut Informatyki, Automatyki i Robotyki
Zakład Systemów Komputerowych

Grafika komputerowa

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 3

OPEN GL – modelowanie obiektów 3-D

Wykonała:	**Weronika Luźna**
Termin:	**PN/N 16.15-19.15**
Data wykonania ćwiczenia:	**30.11.2015**
Data oddania sprawozdania:	**7.12.2015**
Ocena:	

Uwagi prowadzącego:

1. Cel projektu.

Celem projektu było zaimplementowanie 2 programów. Pierwszy rysuje model jajka w 3-D. Drugi rysuje łańcuch, zbudowany z torusów.

2. Jajko.

Zaimplementowano na podstawie wytycznych zawartych w instrukcji do laboratorium.

2.1. Kod źródłowy.

```
//*************************************************************************
//
//  PLIK ŹRÓDŁOWY:          Jajko.cpp
//
//  OPIS:                           Program służy do rysowania układu odwzorowań iterowanych

//                                      (zbioru punktów na płąszczyźnie).
//
//  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/repository/dydaktyka/gk/zadania_domowe/zadania_3.pdf
//     ŹRÓDŁOWE:
//
//  UŻYTE BIBLIOTEKI Nie używano.
//  NIESTANDARDOWE
//
//*************************************************************************

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

typedef float point3[3];

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

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 = 1; // 1- punkty, 2- siatka, 3 - wypełnione trójkąty

static GLfloat theta[] = { 0.0, 0.0, 0.0 };

//----------------------------------------------------------------------
// Obliczanie współrzędnych siatki jajka.
//----------------------------------------------------------------------
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;

		}
	}
}

void Axes(void)
{
	point3  x_min = { -5.0, 0.0, 0.0 };
	point3  x_max = { 5.0, 0.0, 0.0 };
	// początek i koniec obrazu osi x

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

	point3  z_min = { 0.0, 0.0, -5.0 };
	point3  z_max = { 0.0, 0.0, 5.0 };
	//  początek i koniec obrazu osi y

	glColor3f(1.0f, 0.0f, 0.0f);  // kolor rysowania osi - czerwony
	glBegin(GL_LINES); // rysowanie osi x
	glVertex3fv(x_min);
	glVertex3fv(x_max);
	glEnd();

	glColor3f(0.0f, 1.0f, 0.0f);  // kolor rysowania - zielony
	glBegin(GL_LINES);  // rysowanie osi y
	glVertex3fv(y_min);
	glVertex3fv(y_max);
	glEnd();

	glColor3f(0.0f, 0.0f, 1.0f);  // kolor rysowania - niebieski
	glBegin(GL_LINES); // rysowanie osi z
	glVertex3fv(z_min);
	glVertex3fv(z_max);
	glEnd();
}

//-----------------------------------------------------------------------
// Rysowanie jajka.
//-----------------------------------------------------------------------
void Egg()
{
	switch (model) {
	case 1:
		glColor3f(1.0f, 1.0f, 0.0f);
```

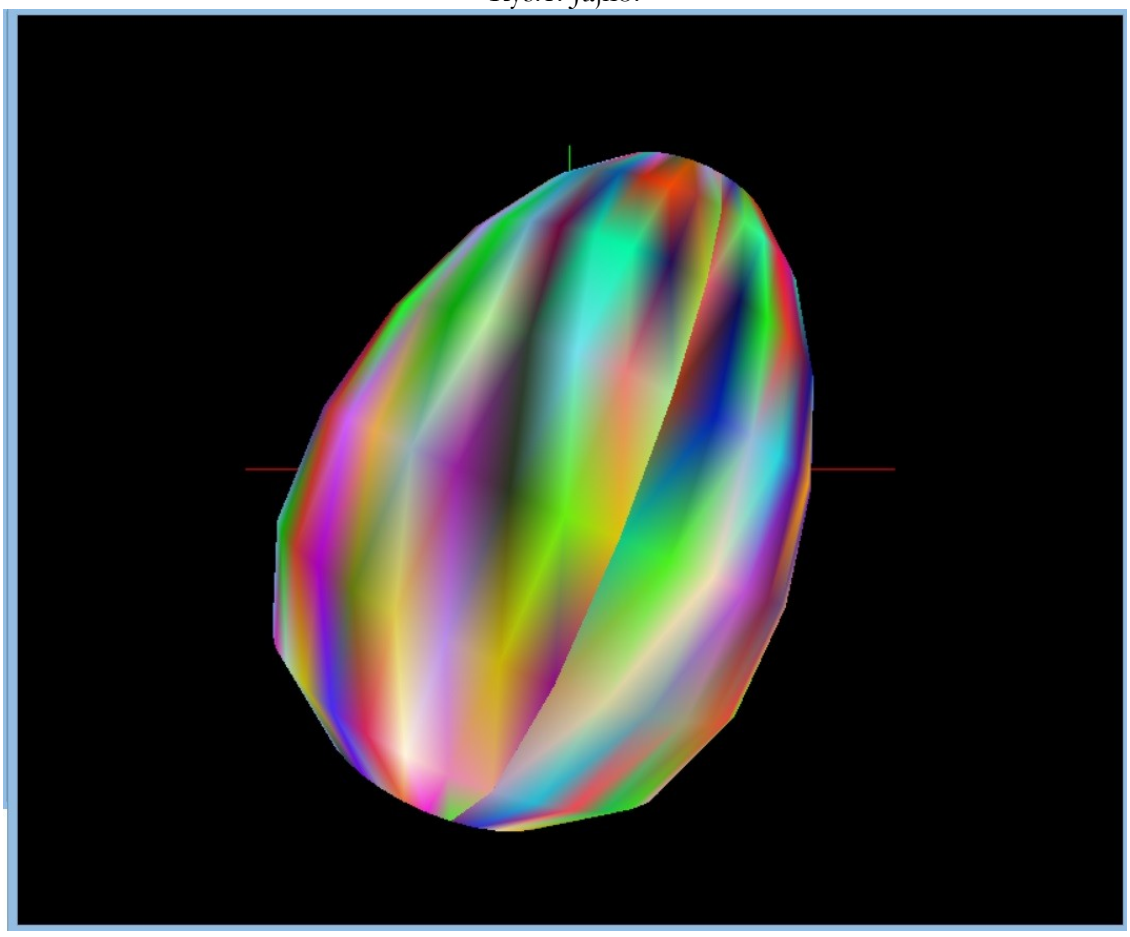
```
		glBegin(GL_POINTS);
		for (int i = 0; i < n; i++)
			for (int j = 0; j < n; j++)
				glVertex3fv(coordinates[i][j]);
		glEnd();
		break;
	case 2:
		glColor3f(1.0f, 1.0f, 0.0f);
		glBegin(GL_LINES);
		for (int i = 0; i < n - 1; i++)
			for (int j = 0; j < n - 1; j++)
			{
				glVertex3fv(coordinates[i][j]);
				glVertex3fv(coordinates[(i)+1][j]); //łączy punkty wzdłuż
				glVertex3fv(coordinates[i][j]);
				glVertex3fv(coordinates[i][(j + 1)]); //łączy punkty znajdujące sie
obok
				glVertex3fv(coordinates[i][j]);
				glVertex3fv(coordinates[(i + 1)][(j + 1)]); //łączy punkty po skosie
			}
		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]); //łączy punkty wzdłuż

				glColor3fv(colors[i][j + 1]);
				glVertex3fv(coordinates[i][(j + 1)]); //łączy punkty znajdujące sie
obok


				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)]); //łączy punkty po skosie


			}
		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, 1.0, 0.0);

	Axes();
	glRotatef(theta[0], 1.0, 0.0, 0.0);
	glRotatef(theta[1], 0.0, 1.0, 0.0);
```

```
	glRotatef(theta[2], 0.0, 0.0, 1.0);

	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 spinEgg()
{
	theta[0] -= 0.05;
	if (theta[0] > 360.0) theta[0] -= 360.0;

	theta[1] -= 0.05;
	if (theta[1] > 360.0) theta[1] -= 360.0;

	theta[2] -= 0.05;
	if (theta[2] > 360.0) theta[2] -= 360.0;

	glutPostRedisplay(); //odświeżenie zawartości aktualnego okna
}

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

void ChangeSize(GLsizei horizontal, GLsizei vertical)
{
	GLfloat AspectRatio;

	if (vertical == 0)  // Zabezpieczenie przed dzieleniem przez 0

		vertical = 1;
	glViewport(0, 0, horizontal, vertical);
	// Ustawienie wielkociokna okna widoku (viewport)
	// W tym przypadku od (0,0) do (horizontal, vertical)

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

	glLoadIdentity();
	// Czyszcznie macierzy bie¿¹cej
	AspectRatio = (GLfloat)horizontal / (GLfloat)vertical;

	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 wielkociokna okna widoku (viewport) w zale¿nosci
        // relacji pomiêdzy wysokoci¹ i szerokoci¹ okna

        glMatrixMode(GL_MODELVIEW);
        // Przes³oniêczenie macierzy bie¿¹cej na macierz widoku modelu

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

int main(void)

{
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(300, 300);
        glutCreateWindow("Jajko");
        CalculateCoordinates();
        glutIdleFunc(spinEgg);
        glutKeyboardFunc(keys);
        glutDisplayFunc(RenderScene);
        glutReshapeFunc(ChangeSize);
        MyInit();
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
        return 0;
}
```

2.2. Demonstracja działania aplikacji.

Rys.1. Jajko.



3. Łańcuch z torusów.

Stworzono w oparciu o algorytm podany w instrukcji do zadania.

3.1. Kod źródłowy:

```
//*************************************************************
//
//  PLIK ŹRÓDŁOWY:          Source.cpp
//
//  OPIS:                           Program służy do rysowania łańcucha utworzonego z
torusów.
//
//  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/repository/dydaktyka/gk/zadania_domowe/zadania_3.pdf
//      ŹRÓDŁOWE:
//
//  UŻYTE BIBLIOTEKI Nie używano.
//  NIESTANDARDOWE
//
//*************************************************************

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

typedef float point3[3];

// inicjalizacja polozenia obserwatora
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };
static GLfloat theta[] = { 0.0, 0.0, 0.0 }; // trzy kąty obrotu

const int n = 20;
const GLfloat R = 1;
const GLfloat r = R/4;
const float PI = 3.1415;
point3 coordinates[n][n];
point3 colors[n][n];
int model = 1; // 1- punkty, 2- siatka, 3 - wype?nione trójk?ty

//-----------------------------------------------------------------
// Obliczanie współrzędnych siatki torusa i generowanie siatki kolorów.
//-----------------------------------------------------------------
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;
            }
        }
}

//-----------------------------------------------------------------------
// Osie współrzędnych.
//-----------------------------------------------------------------------
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();
}

//-----------------------------------------------------------------------
// Rysowanie torusa w trzech wybieralnych modelach.
//-----------------------------------------------------------------------
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();
	}

}

//-----------------------------------------------------------------------
// Rysowanie łańcucha.
//-----------------------------------------------------------------------
void Chain() {

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

```
	Torus();
	glTranslatef(3 * R / 2, R / 2, 0);
	glRotated(90.0, 1.0, 0.0, 0.0);
	glRotated(30.0, 0.0, 1.0, 0.0);
	Torus();
	glTranslatef(3 * R / 2, R/2, 0);
	glRotated(90.0, 1.0, 0.0, 0.0);
	glRotated(30.0, 0.0, 1.0, 0.0);
	Torus();

}

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

	glTranslatef(-5* R, 0, 0);
	Chain();
	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 spin()
{

theta[0] -= 0.05;
if (theta[0] > 360.0) theta[0] -= 360.0;

theta[1] -= 0.05;
if (theta[1] > 360.0) theta[1] -= 360.0;

theta[2] -= 0.05;
if (theta[2] > 360.0) theta[2] -= 360.0;

glutPostRedisplay(); //od?wie?enie zawarto?ci aktualnego okna
}

void MyInit(void)
{
	glClearColor(0.0f, 0.0f, 0.0f, 1.0f);   // Kolor czyszc¹cy (wype³nienia okna)
ustawiono na czarny
}

void ChangeSize(GLsizei horizontal, GLsizei vertical)
{
	GLfloat AspectRatio;
	if (vertical == 0)  // Zabezpieczenie przed dzieleniem przez 0
```

```
		vertical = 1;
	glViewport(0, 0, horizontal, vertical);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	AspectRatio = (GLfloat)horizontal / (GLfloat)vertical;
	gluPerspective(70, 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();
}

int main(void)

{
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

	glutInitWindowSize(900, 600);


	glutCreateWindow("Torus chain");

	CalculateCoordinates();
	glutIdleFunc(spin);
	glutKeyboardFunc(keys);

	glutDisplayFunc(RenderScene);
	glutReshapeFunc(ChangeSize);
	MyInit();
	glEnable(GL_DEPTH_TEST);
	glutMainLoop();

	return 0;
}
```

3.2. Demonstracja działania aplikacji.

Rys. 2. Łańcuch z torusów.