POLITECHNIKA WROCŁAWSKA
Instytut Informatyki, Automatyki i Robotyki
Zakład Systemów Komputerowych

Wprowadzenie do grafiki komputerowej

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr

TEMAT ĆWICZENIA :
OpenGL – Interakcja z użytkownikiem

Wykonał:	**Paweł Biel**
Termin:	**WT TN 13:15 – 16:15**
Data wykonania ćwiczenia:	**7.11.2017**
Data oddania sprawozdania:	**21.11.17**
Ocena:	

Uwagi prowadzącego:

```
//*************************************************************************
//
// PLIK ŹRÓDŁOWY:               Source.cpp
//
// OPIS:                        Program służy Obracania i przybliżania piramidy Sierpinskiego
//
//
// AUTOR:                               Paweł Biel
//
// DATA                         5.11.2017
// MODYFIKACJI:
//
// PLATFORMA:                   System operacyjny:  Microsoft Windows 10.
//                                      Kompilator:     Microsoft Visual C++ v2017.
//
// MATERIAŁY                    Nie wykorzystano.
// ŹRÓDŁOWE:
//
// UŻYTE BIBLIOTEKI     Nie używano.
// NIESTANDARDOWE
//
//*************************************************************************
/*****************************************************************************************/

#include <windows.h>
#include <GL/glut.h>
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;


// Prototypy funkcji
void rysuj_piramide(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, GLfloat *e);
void podziel_piramide(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, GLfloat *e, int
iteraciones);
void RenderScene();
void MyInint();

//punkty dla piramidy
GLfloat piramida[5][3] =
{ { 1.0f, -1.0f, 1.0f },
{ -1.0f,-1.0f, 1.0f },
{ 0.0f, 1.0f, 0.0f },
{ -1.0f,-1.0f,-1.0f },
{ 1.0f, -1.0f, -1.0f }};

int iteracje = 0;



typedef float point3[3];
static GLfloat viewer[] = { 0.0, 0.0, 10.0 };
static GLfloat theta = 0.0;   // kąt obrotu obiektu
static GLfloat theta_y = 0.0;
static GLfloat beta = 0.0;
static GLfloat pix2angle;     // przelicznik pikseli na stopnie

static GLint status = 0;       // stan klawiszy myszy
```

```
                                                // 0 - nie naciśnięto żadnego
klawisza
                                                // 1 - naciśnięty zostać lewy
klawisz

static int x_pos_old = 0;       // poprzednia pozycja kursora myszy
static int y_pos_old = 0;
static int delta_y = 0;
static int delta_x = 0;        // różnica pomiędzy pozycją bieżącą
                                                // inicjalizacja położenia
obserwatora

/*************************************************************************
/
                                                // Funkcja rysująca osie układu
wspó?rz?dnych


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;          // wcięnięty został lewy klawisz myszy
       }
       else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
       {
              x_pos_old = x;         // przypisanie aktualnie odczytanej pozycji
kursora

              status = 2;
       }
       else
              status = 0;          // nie został wcięnięty 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
       delta_y = y - y_pos_old;
       x_pos_old = x;            // podstawienie bieżącego położenia jako poprzednie
       y_pos_old = y;

       glutPostRedisplay();     // przerysowanie obrazu sceny
}

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 };
```

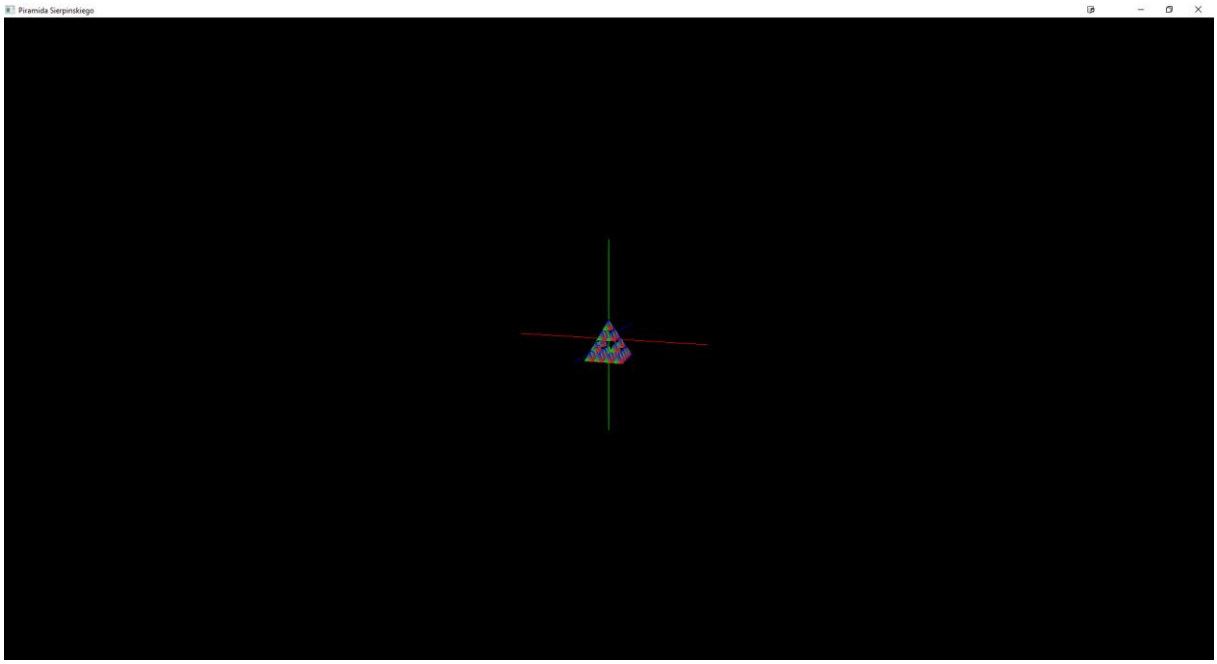
```
	point3  y_max = { 0.0,  5.0, 0.0 };
	// pocz?tek i koniec obrazu osi y

	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();
}

void rysuj_piramide(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, GLfloat *e) {
	glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);


	// wyznaczenie 4 trojkątów dla stworzenia ostrosłupa
	glBegin(GL_TRIANGLES);
	glColor3f(1.0f, 0.0f, 0.0f);
	glVertex3fv(a);
	glColor3f(0.0f, 1.0f, 0.0f);
	glVertex3fv(c);
	glColor3f(0.0f, 0.0f, 1.0f);
	glVertex3fv(e);
	glEnd();



	glBegin(GL_TRIANGLES);
	glColor3f(1.0f, 0.0f, 0.0f);
	glVertex3fv(b);
	glColor3f(0.0f, 1.0f, 0.0f);
	glVertex3fv(c);
	glColor3f(0.0f, 0.0f, 1.0f);
	glVertex3fv(d);
	glEnd();


	glBegin(GL_TRIANGLES);
	glColor3f(1.0f, 0.0f, 0.0f);
	glVertex3fv(c);
	glColor3f(0.0f, 1.0f, 0.0f);
	glVertex3fv(e);
	glColor3f(0.0f, 0.0f, 1.0f);
	glVertex3fv(d);
	glEnd();

	glBegin(GL_TRIANGLES);
	glColor3f(1.0f, 0.0f, 0.0f);
```

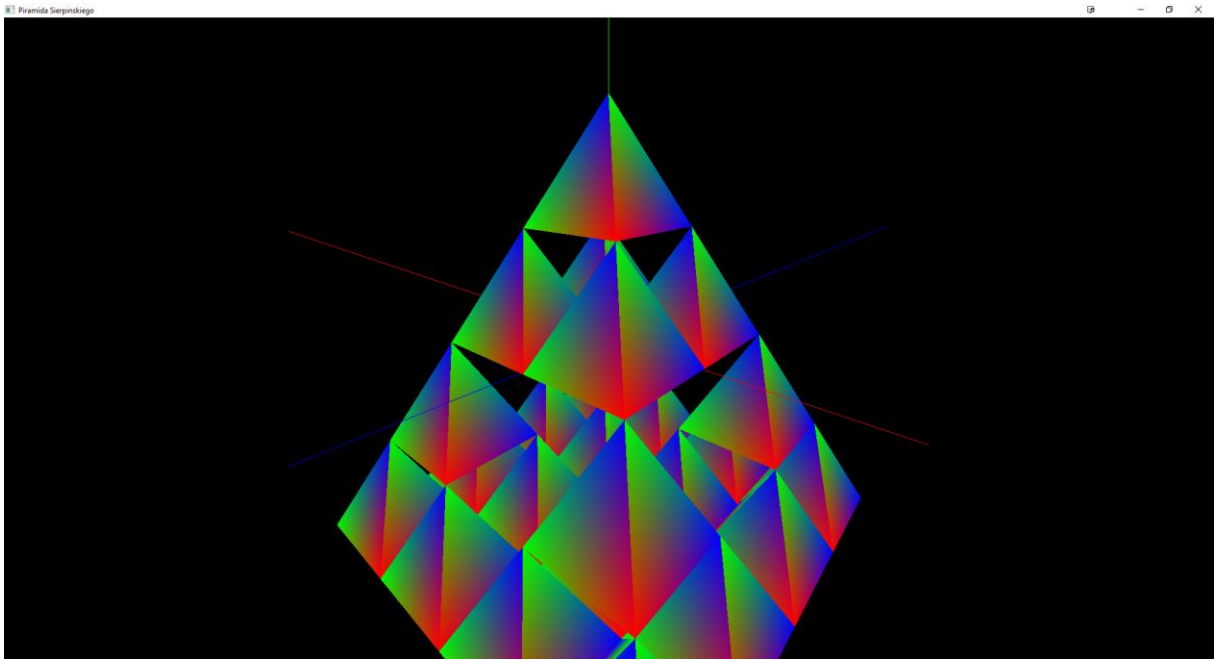
```
	glVertex3fv(a);
	glColor3f(0.0f, 1.0f, 0.0f);
	glVertex3fv(b);
	glColor3f(0.0f, 0.0f, 1.0f);
	glVertex3fv(c);
	glEnd();
}

void podziel_piramide(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, GLfloat *e, int
iteracja) {
	GLfloat wierzcholek[9][3];
	int j;
	if (iteracja > 0) {
		//znajdz punkty środkowe każdej krawędzi
		//podział krawędzi wokół podstawy figury
		for (j = 0; j < 3; j++) {
			wierzcholek[0][j] = (a[j] + b[j]) / 2;
		}
		for (j = 0; j < 3; j++) {
			wierzcholek[1][j] = (b[j] + d[j]) / 2;
		}
		for (j = 0; j < 3; j++) {
			wierzcholek[2][j] = (d[j] + e[j]) / 2;
		}
		for (j = 0; j < 3; j++) {
			wierzcholek[3][j] = (e[j] + a[j]) / 2;
		}

		// podział krawędzi bocznych
		for (j = 0; j < 3; j++) {
			wierzcholek[4][j] = (c[j] + a[j]) / 2;
		}
		for (j = 0; j < 3; j++) {
			wierzcholek[5][j] = (c[j] + b[j]) / 2;
		}
		for (j = 0; j < 3; j++) {
			wierzcholek[6][j] = (c[j] + d[j]) / 2;
		}
		for (j = 0; j < 3; j++) {
			wierzcholek[7][j] = (c[j] + e[j]) / 2;
		}
		for (j = 0; j < 3; j++) {
			wierzcholek[8][j] = (wierzcholek[3][j] + wierzcholek[1][j]) / 2;
		}

		//dla każdego trójkąta, który wchodzi, tworzone są 5 mniejsze trójkąty i
rekurencyjnie są one podzielone po kolei
		// od wierzchołka lewego dolnego w kierunku odwrotnym do wskazówek
zegara, a na samym koncu górny trójkąt
		podziel_piramide(a, wierzcholek[0], wierzcholek[4],
wierzcholek[8],wierzcholek[3], iteracja - 1);
		podziel_piramide(wierzcholek[0], b, wierzcholek[5], wierzcholek[1],
wierzcholek[8], iteracja - 1);
		podziel_piramide(wierzcholek[8], wierzcholek[1], wierzcholek[6], d,
wierzcholek[2], iteracja - 1);
		podziel_piramide(wierzcholek[3], wierzcholek[8], wierzcholek[7],
wierzcholek[2], e, iteracja - 1);
		podziel_piramide(wierzcholek[4], wierzcholek[5], c, wierzcholek[6],
wierzcholek[7], iteracja - 1);


	}
```

```
	else {
		//narysuj piramide gdy iteracja 0
		rysuj_piramide(a, b, c, d, e);
	}
}

//void RenderScene() {
//	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
//	glPushMatrix();
//	glRotated(15, 0.0, 1.0, 0.0);  // Obrót o 15 stopni
//	//random(true);
//	//podczas wywoływania funkcji divideTriangle, czwarty parametr to liczba
potrzebnych iteracji podpodziałów
//	podziel_piramide(piramida[0], piramida[1], piramida[2], piramida[3],piramida[4],
iteracje);
//
//	glPopMatrix();
//	glFlush();
//
//
//}


void RenderScene() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();
	// Czyszczenie macierzy bie??cej
	GLfloat div = 100;
	if (status == 1)                    // jeśli lewy klawisz myszy wcięnięty
	{
		theta += delta_x*pix2angle / div;    // modyfikacja kąta obrotu o kat
proporcjonalny
		theta_y += delta_y*pix2angle / div;
	}                                  // do różnicy położeń kursora myszy
	if (status == 2)
	{
		beta += delta_x*pix2angle;
	}
	GLfloat x1, x2, x3;
	x1 = 10 * cos(theta)*cos(theta_y);
	x2 = 10 * sin(theta_y);
	x3 = 10 * sin(theta)*cos(theta_y);

	gluLookAt(x1, x2, x3 + beta, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);//w odpowidnim
momencie przestawić 1.0 na -1.0 aby nie było przeskoku

			/*else

			gluLookAt(x1, x2, x3 + beta, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0);*/

			// Zdefiniowanie położenia obserwatora
	Axes();
	// Narysowanie osi przy pomocy funkcji zdefiniowanej powyżej

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


	glRotatef(theta, 0.0, 1.0, 0.0);  //obrót obiektu o nowy kąt
	glRotatef(theta_y, 1.0, 0.0, 0.0);
	podziel_piramide(piramida[0], piramida[1], piramida[2], piramida[3],
piramida[4], iteracje);
```

```
	// Narysowanie czajnika
	glFlush();
	// Przekazanie poleceń rysujących do wykonania
	glutSwapBuffers();



}

void ChangeSize(GLsizei horizontal, GLsizei vertical)
{
	pix2angle = 360.0 / (float)horizontal;

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

	glLoadIdentity();
	// Czyszcznie macierzy bieżącej

	gluPerspective(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

}

void MyInint(void) {
	glClearColor(0.0, 0.0, 0.0, 1.0);		// czyszczenie coloru
	glColor3f(0.0, 0.0, 0.0);		//ustawienie koloru
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(-2.0, 2.0, -2.0, 2.0, -20.0, 20.0);
	glEnable(GL_DEPTH_TEST);
}



int main(int argc, char** argv) {
	srand(time(NULL));

	// podanie ilosci iteracji
	cout << "Podaj liczbe interacji: ";
	cin >> iteracje;
	if (iteracje != 0)
		{
		iteracje--;
		}
```

```
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    // Ustawienie trybu wyœwietlania
    // GLUT_SINGLE - pojedynczy bufor wyświetlania
    // GLUT_RGBA - model kolorów RGB

    glutInitWindowSize(600, 600);
    glutCreateWindow("Piramida Sierpinskiego");
    // Utworzenie okna i określenie treści napisu w nagłówku okna
    glutDisplayFunc(RenderScene);
    glutMouseFunc(Mouse);


    glutMotionFunc(Motion);
    glutReshapeFunc(ChangeSize);
    MyInint();
    // Funkcja MyInit (zdefiniowana powyżej) wykonuje wszelkie
    // inicjalizacje konieczneprzed przystąpieniem do renderowania
    //glutDisplayFunc(RenderScene);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    // Funkcja uruchamia szkielet biblioteki GLUT
}
```




Piramida Sierpinskiego