POLITECHNIKA WROCŁAWSKA
Instytut Informatyki, Automatyki i Robotyki
Zakład Systemów Komputerowych

Grafika komputerowa

Kurs: INEK00012L

Sprawozdanie z ćwiczenia nr 2

OPEN GL - podstawy

Wykonała:	Weronika Luźna
Termin:	**PN / N 16.15-19.15**
Data wykonania ćwiczenia:	**19.10.2015**
Data oddania sprawozdania:	**2.11.2015**
Ocena:	

Uwagi prowadzącego:

1. Cel projektu.

Celem projektu było zaimplementowanie 2 programów. Pierwszy rysuje dywan Sierpińskiego. Drugi rysuje określony algorytmem zbiór punktów.

2. Dywan Sierpińskiego.

Zaimplementowano na podstawie wytycznych zawartych w instrukcji do laboratorium.

2.1. Kod źródłowy.

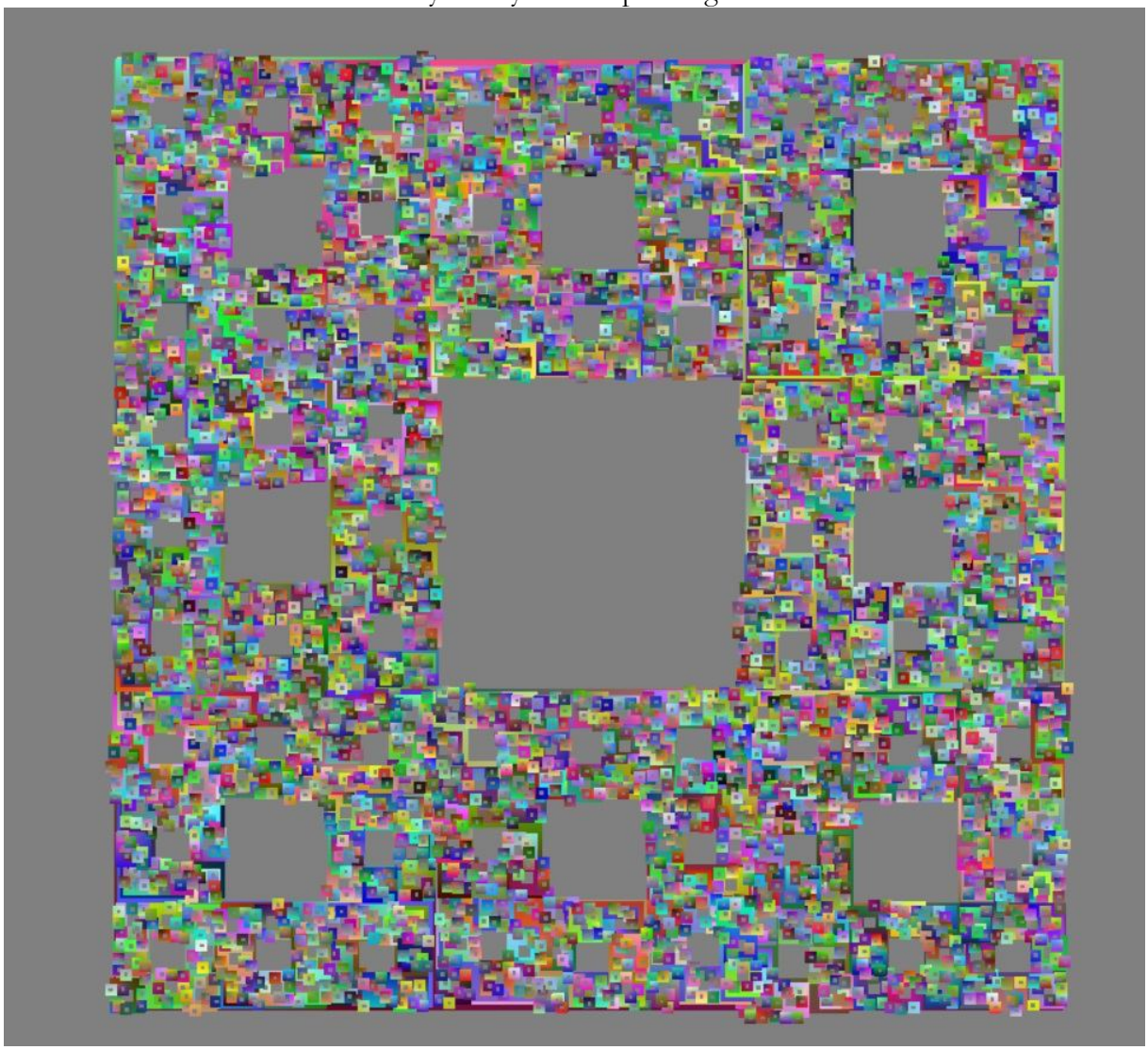
```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <cstdlib>
#include <ctime>

//-----------------------------------------------------------------------
// Rysowanie kwadratu z "dziurą" w oparciu o współrzędne środka i długość boku.
// Kolorowanie losowymi kolorami.
//-----------------------------------------------------------------------
void DrawRectangle(GLfloat x, GLfloat y, GLfloat a) {

	GLfloat R = (std::rand() % 100) / 100.0f;
	GLfloat G = (std::rand() % 100) / 100.0f;
	GLfloat B = (std::rand() % 100) / 100.0f;

	glBegin(GL_POLYGON);
	glColor3f(R, G, B);
	glVertex2f(x - a / 2.0f, y + a / 2.0f);
	R = (std::rand() % 100) / 100.0f;
	glColor3f(R, G, B);
	glVertex2f(x + a / 2.0f, y + a / 2.0f);
	G = (std::rand() % 100) / 100.0f;
	glColor3f(R, G, B);
	glVertex2f(x + a / 2.0f, y - a / 2.0f);
	B = (std::rand() % 100) / 100.0f;
	glColor3f(R, G, B);
	glVertex2f(x - a / 2.0f, y - a / 2.0f);
	glEnd();

	glBegin(GL_POLYGON);
	glColor3f(0.5f, 0.5f, 0.5f);
	glVertex2f(x - a / 6.0f, y + a / 6.0f);
	glVertex2f(x + a / 6.0f, y + a / 6.0f);
	glVertex2f(x + a / 6.0f, y - a / 6.0f);
	glVertex2f(x - a / 6.0f, y - a / 6.0f);
	glEnd();
}

//-----------------------------------------------------------------------
// Rysowanie kwadratów wokół środka poprzedniego kwadratu.
// Generowanie perturbacji (przez losowanie wartości zmiennej alpha).
//-----------------------------------------------------------------------
void CalculateCoordinates(GLfloat X, GLfloat Y, GLfloat A) {
	GLfloat a = A / 3.0f;
	GLfloat alpha;
	GLfloat coordinates[8][2] = { { X - a, Y - a },
	{ X - a, Y  },
```

```
	{ X - a , Y + a  },
	{ X    , Y + a  },
	{ X + a , Y + a  },
	{ X + a , Y  },
	{ X + a , Y - a  },
	{ X    , Y - a  }
	};

	DrawRectangle(X, Y, A);

	for (int i = 0; i < 8; i++) {
		if (a > 2.0f) {
			alpha = ((float)(rand() % 20 - 10)) / 10;
			CalculateCoordinates(coordinates[i][0]+alpha, coordinates[i][1]+alpha, a);
		}
	}
}

//-----------------------------------------------------------------------
// Generowanie obrazu.
//-----------------------------------------------------------------------
void RenderScene(void) {
	glClear(GL_COLOR_BUFFER_BIT);

	CalculateCoordinates(0.0f, 0.0f, 168.0f);

	glFlush();
}

//-----------------------------------------------------------------------
// Kolor t?a.
//-----------------------------------------------------------------------
void MyInit(void) {
	glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
}

//-----------------------------------------------------------------------
// Skalowanie obrazu z zachowaniem proporcji obiektu.
//-----------------------------------------------------------------------
void ChangeSize(GLsizei horizontal, GLsizei vertical) {
	GLfloat AspectRatio;

	if (vertical == 0)
		vertical = 1;

	glViewport(0, 0, horizontal, vertical);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	AspectRatio = (GLfloat)horizontal / (GLfloat)vertical;

	if (horizontal <= vertical)
		glOrtho(-100.0, 100.0, -100.0 / AspectRatio, 100.0 / AspectRatio, 1.0, -1.0);
	else
		glOrtho(-100.0*AspectRatio, 100.0*AspectRatio, -100.0, 100.0, 1.0, -1.0);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

void main(void) {
	srand(time(NULL));
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
```

```
        glutCreateWindow("Dywan sierpinskiego");

        glutDisplayFunc(RenderScene);
        glutReshapeFunc(ChangeSize);
        MyInit();
        glutMainLoop();
}
```

2.2. Demonstracja działania aplikacji.

Rys.1. Dywan Sierpińskiego.



3. Układ odwzorowań iterowanych.

Stworzono w oparciu o algorytm podany w instrukcji do zadania.

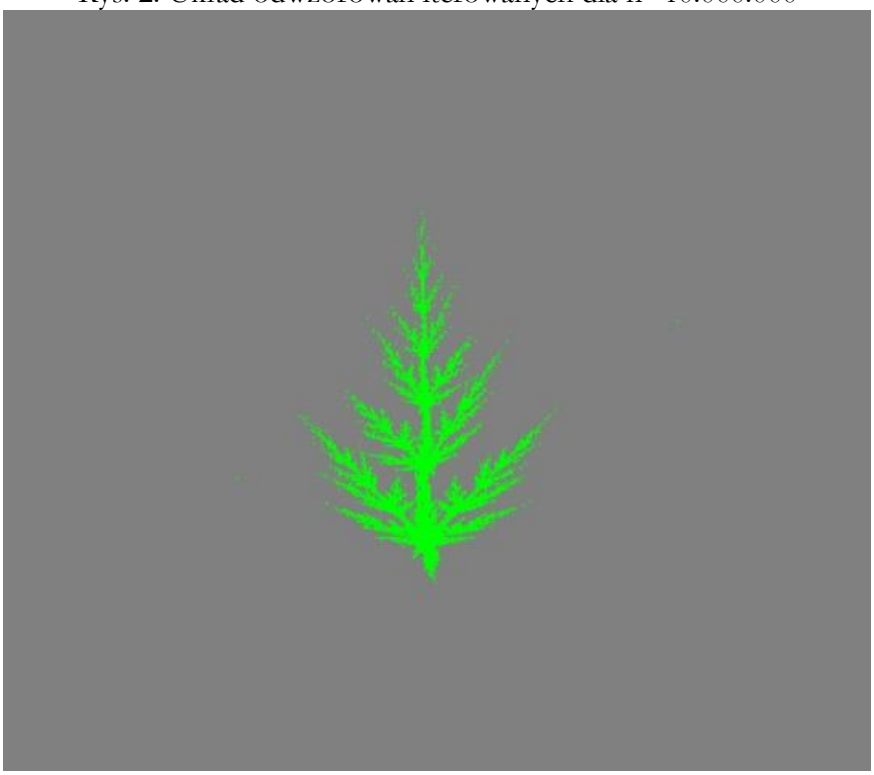
3.1. Kod źródłowy:

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <cstdlib>
#include <ctime>
#include<iostream>
using namespace std;

struct position {
	GLfloat x;
	GLfloat y;
};

//-----------------------------------------------------------------------
// Tabela współczynników odwzorowań.
//-----------------------------------------------------------------------
GLfloat ratio[4][6] = {
	{ -0.67f, -0.02f, 0.00f, -0.18f, 0.81f, 10.00f },
	{ 0.40f, 0.40f, 0.00f, -0.10f, 0.40f, 0.00f },
	{ -0.40f, -0.40f, 0.00f, -0.10f, 0.40f, 0.00f },
	{ -0.10f, 0.00f, 0.00f, 0.44f, 0.44f, -2.00f },
};

//-----------------------------------------------------------------------
// Obliczanie współrzędnych punktu dla wylosowanego odwzorowania.
//-----------------------------------------------------------------------
position CalculateCoordinates(GLfloat x, GLfloat y) {

	int phi = (rand() % 4);

	position pos;

	pos.x = ratio[phi][0] * x + ratio[phi][1] * y + ratio[phi][2];
	pos.y = ratio[phi][3] * x + ratio[phi][4] * y + ratio[phi][5];

	return pos;
}

//-----------------------------------------------------------------------
// Generowanie obrazu.
//-----------------------------------------------------------------------
void RenderScene(void) {
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0.0f, 1.0f, 0.0f);

	GLfloat x = rand();
	GLfloat	y = rand();

	glBegin(GL_POINTS);
	for (int i = 0; i < 100000; i++) {
		glVertex2f(x, y);
```

```
                position pos = CalculateCoordinates(x, y);
                x = pos.x;
                y = pos.y;
        }
        glEnd();

        glFlush();
}

void MyInit(void) {
        glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
}

//------------------------------------------------------------------
// Skalowanie obrazu z zachowaniem proporcji obiektu.
//------------------------------------------------------------------
void ChangeSize(GLsizei horizontal, GLsizei vertical) {
        GLfloat AspectRatio;

        if (vertical == 0)
                vertical = 1;

        glViewport(0, 0, horizontal, vertical);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        AspectRatio = (GLfloat)horizontal / (Glfloat)vertical;
if (horizontal <= vertical)
                glOrtho(-100.0, 100.0, -100.0 / AspectRatio, 100.0 / AspectRatio, 1.0, -1.0);
        else
                glOrtho(-100.0*AspectRatio, 100.0*AspectRatio, -100.0, 100.0, 1.0, -1.0);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}

void main(void) {
        srand(time(NULL));
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
        glutCreateWindow("Iterated function system");

        glutDisplayFunc(RenderScene);

        glutReshapeFunc(ChangeSize);
        MyInit();
        glutMainLoop();
}
```

3.2. Demonstracja działania aplikacji.

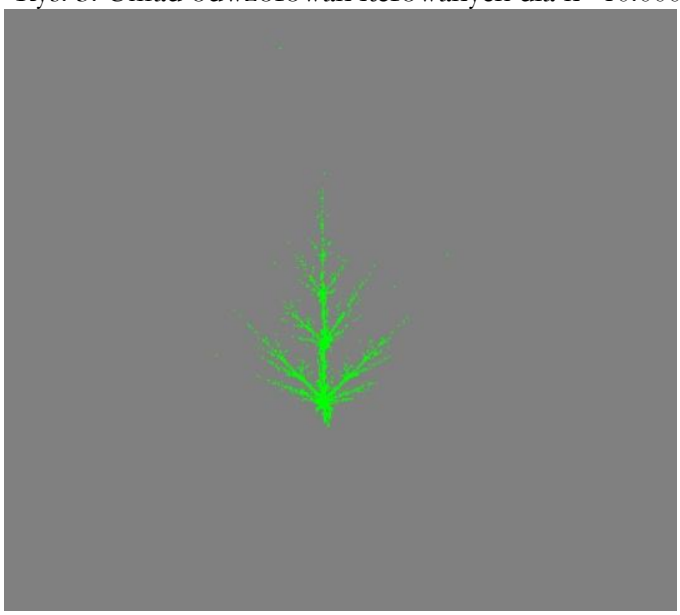
Rys. 2. Układ odwzorowań iterowanych dla n=10.000.000



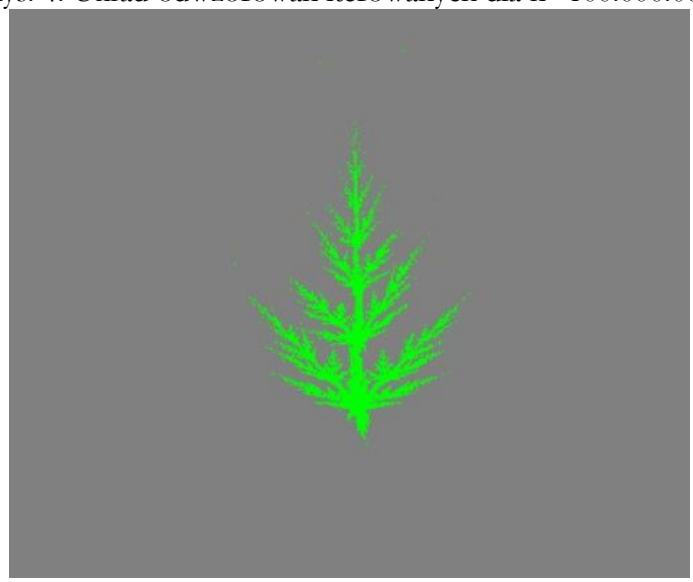
3.3. Badanie wpływu liczby iteracji i losowości na proces generacji zbioru.

Wygenerowano układy dla różnych liczb iteracji.

Rys. 3. Układ odwzorowań iterowanych dla n=10.000



Rys. 4. Układ odwzorowań iterowanych dla n=100.000.000



Zauważono, że im większa liczba iteracji, tym większe zagęszczenie punktów. Obiekt zostaje „pogrubiony”, natomiast jego rozmiar nie powiększa się i proporcje pozostają bez zmian.