pwr**POLITECHNIKA WROCŁAWSKA**

**Instytut Informatyki, Automatyki i Robotyki**

**Zakład Systemów Komputerowych**

**Wprowadzenie do grafiki komputerowej**

**Kurs: INE4234L**

**Sprawozdanie z ćwiczenia nr 2**

**TEMAT ĆWICZENIA :**

**Dywan i trójkąt Sierpińskiego**

|  |  |
| --- | --- |
| **Wykonał:** | **Paweł Biel** |
| **Termin:** | **WT TN 13:15 – 16:15** |
| **Data wykonania ćwiczenia:** | **10.10.2017** |
| **Data oddania sprawozdania:** | **24.10.17** |
| **Ocena:** |  |

|  |
| --- |
| **Uwagi prowadzącego:** |

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// PLIK ŹRÓDŁOWY: Source.cpp

//

// OPIS: Program służy do rysowania dywanu i trójkąta  
// Sierpińskiego

//

// AUTOR: Paweł Biel

//

// DATA 23.10.2017

// 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 <ctime>

#include <gl/gl.h>

#include <iostream>

#include <gl/glut.h>

#include <cmath>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja okreœlaj¹ca, co ma byæ rysowane

// (zawsze wywo³ywana, gdy trzeba przerysowaæ scenê)

typedef float point2[2];

using namespace std;

//----------------------------------------------------------------------

// Funkcja rysujaca trojkat sierpinskiego 1 sposob

// ---------------------------------------------------------------------

int SierpinskiTriangle1(point2 a, point2 b, int level)

{

float side = b[0] - a[0];

float small\_side = side / 2;

//----------------------------------------------------------------------

// Jeślii licznik = 1 wypelnij boki i biały trójkąt w srodku

// ---------------------------------------------------------------------

if (level == 1)

{

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

point2 new\_a = { a[0] + (small\_side\*j) / 2 + small\_side\*i, a[1] + (small\_side\*((sqrt(3)) / 2)\*j) };

point2 new\_b = { a[0] + small\_side + (small\_side\*i) + (small\_side\*j) / 2, a[1] + (small\_side\*((sqrt(3)) / 2) \* j) };

point2 new\_g = { a[0] + small\_side / 2 + (small\_side\*i) + ((small\_side / 2)\*j),a[1] + (small\_side\*((sqrt(3)) / 2)) + small\_side\*j\*((sqrt(3)) / 2) };

point2 b\_d = { a[0] + small\_side, a[1] };

point2 b\_l = { a[0] + small\_side / 2,a[1] + (small\_side\*((sqrt(3)) / 2)) };

point2 b\_p = { a[0] + small\_side / 2 + small\_side,a[1] + (small\_side\*((sqrt(3)) / 2)) };

//----------------------------------------------------------------------

// Wypełnienie trókąta odpowiednio kolorami

// ---------------------------------------------------------------------

if ((i == 1) && (j == 1)) {

glBegin(GL\_TRIANGLES);

glColor3f(1.0f, 1.0f, 1.0f); // kolor biały

glVertex2fv(b\_l);

glVertex2fv(b\_d);

glVertex2fv(b\_p);

glEnd();

}

else {

// trojkat wypelniony

glBegin(GL\_TRIANGLES);

glColor3f(1.0f, 0.0f, 0.0f); // kolor czerwony

glVertex2fv(new\_a);

glVertex2fv(new\_g);

glVertex2fv(new\_b);

glEnd();

}

}

}

return 0;

}

// jesli licznik > 1

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

point2 new\_ld = { a[0] + (small\_side\*j) / 2 + small\_side\*i, a[1] + (small\_side\*j\*((sqrt(3)) / 2)) };

point2 new\_pd = { a[0] + small\_side + (small\_side\*i) + (small\_side\*j) / 2, a[1] + (small\_side \* j\*((sqrt(3)) / 2)) };

point2 new\_g = { a[0] + small\_side / 2 + (small\_side\*i) + ((small\_side / 2)\*j),a[1] + (small\_side\*((sqrt(3)) / 2)) + (small\_side\*j\*((sqrt(3)) / 2)) };

point2 b\_d = { a[0] + small\_side, a[1] };

point2 b\_l = { a[0] + small\_side / 2,a[1] + (small\_side\*((sqrt(3)) / 2)) };

point2 b\_p = { a[0] + small\_side / 2 + small\_side,a[1] + (small\_side\*((sqrt(3)) / 2)) };

//----------------------------------------------------------------------

// wypełnienie trójkąta lub rekurencja

// ---------------------------------------------------------------------

if ((i == 1) && (j == 1))

{

glBegin(GL\_TRIANGLES);

glColor3f(1.0f, 1.0f, 1.0f); // kolor buały

glVertex2fv(b\_l);

glVertex2fv(b\_d);

glVertex2fv(b\_p);

glEnd();

}

else {

SierpinskiTriangle1(new\_ld, new\_pd, level - 1);

}

}

}

return 0;

}

// ------------------------------------------------------------------

// drugi sposob na trojkat sierpinskiego

// -----------------------------------------------------------------

void SierpinskiTriangle2()

{

// ------------------------------------------------------------------

// Inicjalizacja wspołrzędnych trójkata

// -----------------------------------------------------------------

GLfloat vertices[3][2] = { { 0.0, 0.0 },{ 50.0, 100.0 },{ 100.0, 0.0 } };

int pointsNum, midPoint;

GLfloat p[2] = { 15.0 , 10.0 };

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POINTS);

// ------------------------------------------------------------------

// Wygenerowanie trójkąta sierpinskiego z 40000 punktów

// -----------------------------------------------------------------

for (pointsNum = 0; pointsNum < 40000; pointsNum++)

{

midPoint = rand() % 3;

// middle

p[0] = (p[0] + vertices[midPoint][0]) / 2.0;

p[1] = (p[1] + vertices[midPoint][1]) / 2.0;

glVertex2fv(p);

}

glEnd();

}

// ------------------------------------------------------------------

// argumenty funkcji to lewy dolny i prawy dolny bok kwadratu oraz stopieñ deformacji kwadratu

// -----------------------------------------------------------------

int SierpinskiCarpet(point2 ld, point2 pd, int level) {

float bok = pd[0] - ld[0];

float bok\_maly = bok / 3;

// ------------------------------------------------------------------

// licznik = 1 wypelnij boki i czarny kwardat w srodku

// -----------------------------------------------------------------

if (level == 1) {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

point2 new\_ld = { ld[0] + (bok\_maly \* i), ld[1] + (bok\_maly \* j) };

point2 new\_lg = { ld[0] + (bok\_maly \* i), ld[1] + bok\_maly + (bok\_maly \* j) };

point2 new\_pg = { ld[0] + bok\_maly + (bok\_maly \* i), ld[1] + bok\_maly + (bok\_maly \* j) };

point2 new\_pd = { ld[0] + bok\_maly + (bok\_maly \* i), ld[1] + (bok\_maly \* j) };

if ((i != 1) || (j != 1)) {

// ------------------------------------------------------------------

// kwadrat wypełnony

// -----------------------------------------------------------------

glBegin(GL\_POLYGON); // ustalenie kolorów paleta rgb dla kwadratu

glColor3f(float(rand() % 300) / 1000, float(rand() % 1000) / 1000, float(rand() % 1000) / 1000);

glVertex2fv(new\_ld);

glColor3f(float(rand() % 500) / 1000, float(rand() % 1000) / 1000, float(rand() % 1000) / 1000);

glVertex2fv(new\_lg);

glColor3f(float(rand() % 800) / 1000, float(rand() % 1000) / 1000, float(rand() % 1000) / 1000);

glVertex2fv(new\_pg);

glColor3f(float(rand() % 1000) / 1000, float(rand() % 1000) / 1000, float(rand() % 1000) / 1000);

glVertex2fv(new\_pd);

glEnd();

}

else {

// ------------------------------------------------------------------

// kwadrat niewypełnony

// -----------------------------------------------------------------

glBegin(GL\_POLYGON);

glColor3f(0.5f, 0.5f, 0.5f); // kolor szary

glVertex2fv(new\_ld);

glVertex2fv(new\_lg);

glVertex2fv(new\_pg);

glVertex2fv(new\_pd);

glEnd();

}

}

}

return 0;

}

// ------------------------------------------------------------------

// jeśli level > 1

// -----------------------------------------------------------------

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

point2 new\_ld = { ld[0] + (bok\_maly \* i), ld[1] + (bok\_maly \* j) }; //punkt lewy dolny

point2 new\_lg = { ld[0] + (bok\_maly \* i), ld[1] + bok\_maly + (bok\_maly \* j) }; //punkt lewy górny

point2 new\_pg = { ld[0] + bok\_maly + (bok\_maly \* i), ld[1] + bok\_maly + (bok\_maly \* j) }; // punkt prawy górny

point2 new\_pd = { ld[0] + bok\_maly + (bok\_maly \* i), ld[1] + (bok\_maly \* j) }; // punkt prawy dolny

if ((i == 1) && (j == 1)) {

// nie rób nic

}

else {

//rysuj szary kwadrat

glBegin(GL\_POLYGON);

glColor3f(0.5f, 0.5f, 0.5f); // kolor szary

glVertex2fv(new\_ld);

glVertex2fv(new\_lg);

glVertex2fv(new\_pg);

glVertex2fv(new\_pd);

glEnd();

SierpinskiCarpet(new\_ld, new\_pd, level - 1);

}

}

}

return 0;

}

void RenderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

point2 a = { -50, -50 };

point2 b = { 50, -50 };

int level = 3;

SierpinskiTriangle1(a, b, level);

//sierpinski\_triangle\_2();

glFlush();

// Przekazanie poleceñ rysuj¹cych do wykonania

}

void RenderScene2(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

point2 a = { -50, -50 };

point2 b = { 50, -50 };

//int level;

//cout << "Podaj liczbę powtorzen: ";

//cin >> level;

//sierpinski\_triangle\_1(a, b, level);

SierpinskiTriangle2();

glFlush();

// Przekazanie poleceñ rysuj¹cych do wykonania

}

void RenderScene3(void) {

glClear(GL\_COLOR\_BUFFER\_BIT);

// Czyszczenie okna aktualnym kolorem czyszcz¹cym

point2 ld = { -50, -50 };

point2 pd = { 50, -50 };

int level = 3;

SierpinskiCarpet(ld, pd, level);

glFlush();

// Przekazanie poleceñ rysuj¹cych do wykonania

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja ustalaj¹ca stan renderowania

void MyInit(void)

{

glClearColor(0.5f, 0.5f, 0.5f, 1.0f);

// Kolor okna wnêtrza okna - ustawiono na szary

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Funkcja s³u¿¹ca do kontroli zachowania proporcji

void ChangeSize(GLsizei horizontal, GLsizei vertical) {

// Parametry horizontal i vertical (szerokoœæ i wysokoœæ okna) s¹

// przekazywane do funkcji za ka¿dym razem, gdy zmieni siê rozmiar okna

GLfloat AspectRatio;

// Deklaracja zmiennej AspectRatio okreœlaj¹cej proporcjê wymiarów okna

if (vertical == 0)

// Zabezpieczenie pzred dzieleniem przez 0

vertical = 1;

glViewport(0, 0, horizontal, vertical);

// Ustawienie wielkoœciokna okna urz¹dzenia (Viewport)

// W tym przypadku od (0,0) do (horizontal, vertical)

glMatrixMode(GL\_PROJECTION);

// Okreœlenie uk³adu wspó³rzêdnych obserwatora

glLoadIdentity();

// Okreœlenie przestrzeni ograniczaj¹cej

AspectRatio = (GLfloat)horizontal / (GLfloat)vertical;

// Wyznaczenie wspó³czynnika proporcji okna

// Gdy okno na ekranie nie jest kwadratem wymagane jest

// okreœlenie okna obserwatora.

// Pozwala to zachowaæ w³aœciwe proporcje rysowanego obiektu

// Do okreœlenia okna obserwatora s³u¿y funkcja glOrtho(...)

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);

// Okreœlenie uk³adu wspó³rzêdnych

glLoadIdentity();

}

int main(int argc, char \*\*argv) {

//srand(time(NULL));

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGBA);

// Ustawienie trybu wyœwietlania

// GLUT\_SINGLE - pojedynczy bufor wyœwietlania

// GLUT\_RGBA - model kolorów RGB

glutCreateWindow("Drugi program w OpenGL");

// Utworzenie okna i okreœlenie treœci napisu w nag³ówku okna

int k = 1;

cout << "Wybierz sposob\n1 -> Sposob z instrukcji\n2 -> sposob punktowy\n3 -> Dywan Sierpinskiego";

cin >> k;

if (k == 1)

{

glutDisplayFunc(RenderScene);

// Okreœlenie, ¿e funkcja RenderScene bêdzie funkcj¹ zwrotn¹ (callback)

// Biblioteka GLUT bêdzie wywo³ywa³a t¹ funkcjê za ka¿dym razem, gdy

// trzeba bêdzie przerysowaæ okno

}

else if (k == 2)

{

glutDisplayFunc(RenderScene2);

}

else if (k == 3)

{

glutDisplayFunc(RenderScene3);

}

glutReshapeFunc(ChangeSize);

// Dla aktualnego okna ustala funkcjê zwrotn¹ odpowiedzialn¹ za

// zmiany rozmiaru okna

MyInit();

// Funkcja MyInit (zdefiniowana powy¿ej) wykonuje wszelkie

// inicjalizacje konieczneprzed przyst¹pieniem do renderowania

glutMainLoop();

// Funkcja uruchamia szkielet biblioteki GLUT

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/