pwr**POLITECHNIKA WROCŁAWSKA**

**Instytut Informatyki, Automatyki i Robotyki**

**Zakład Systemów Komputerowych**

**Wprowadzenie do grafiki komputerowej**

**Kurs: INEK00012L**

**Sprawozdanie z ćwiczenia nr**

**TEMAT ĆWICZENIA :**

**OpenGL – Teksturowanie**

|  |  |
| --- | --- |
| **Wykonał:** | **Paweł Biel** |
| **Termin:** | **WT TN 13:15 – 16:15** |
| **Data wykonania ćwiczenia:** | **11.12.2017** |
| **Data oddania sprawozdania:** | **19.12.17** |
| **Ocena:** |  |

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

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

//

// PLIK ŹRÓDŁOWY: Source.cpp

//

// OPIS: Teksturowanie

//

//

// 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/gl.h>

#include <gl/glut.h>

#include <math.h>

#include <cstdlib>

#include <iostream>

using namespace std;

#define PI 3.14159265

#pragma warning (disable : 4996)

typedef float point3[3]; // Definicja typu przechowującego współrzędne X,Y,Z punktu

const int N = 50; // Poziom szczegółowości - rysowana figura składa się z N^2 punktów lub wierzchołków trójkątów

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

static GLfloat p = 1.0; // Współrzędna Y skrócenia kamery

static GLfloat thetax = 0.0, thetay = 0.0, theta\_zoom = 10.0;

static GLfloat pix2angle; // Przelicznik pikseli na stopień

//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;

static GLint status = 0; // Stan wciśnięcia przycisków myszy:

// 0) żaden przycisk nie jest wciśnięty,

// 1) wciśnięty został lewy przycisk,

// 2) wciśnięty został prawy przycisk

static int x\_pos\_old = 0; // Poprzednia pozycja X,Y kursora myszy

static int y\_pos\_old = 0;

static int delta\_x = 0; // Różnica w położeniu bieżącym i poprzednim

static int delta\_y = 0;

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

// Przeliczenie wspolrzednych dwu wymiarowych na wspolrzedne trzy wymiarowe dla x, y, z

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

float x(int i, int j, float n)

{

float u = i / (n - 1);

float v = j / (n - 1);

float xx = ((-90 \* pow(u, 5)) + (225 \* pow(u, 4)) - (270 \* pow(u, 3)) + (180 \* pow(u, 2)) - 45 \* u)\*cos(PI \* v);

return xx;

}

float y(int i, int j, float n)

{

float u = i / (n - 1);

float v = j / (n - 1);

float yy = (160 \* pow(u, 4)) - (320 \* pow(u, 3)) + (160 \* pow(u, 2));

return yy;

}

float z(int i, int j, float n)

{

float u = i / (n - 1);

float v = j / (n - 1);

float zz = ((-90 \* pow(u, 5)) + (225 \* pow(u, 4)) - (270 \* pow(u, 3)) + (180 \* pow(u, 2)) - 45 \* u)\*sin(PI \* v);

return zz;

}

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

// Wywołanie funkcji rysującej jajko

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

void rysuj\_jajko()

{

point3 points[N][N]; // Macierz przechowująca współrzędne N^2 punktów z których składa się obraz

point3 normal[N][N]; // Macierz współrzędnych wektorów normalnych do wierzchołków trójkątów

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

{

float u = (float)i / (float)(N - 1);

float v = (float)j / (float)(N - 1);

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

// Rzutowanie punktow o wsporzednych 3d na macierz

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

points[i][j][0] = x(i, j, N);

points[i][j][1] = y(i, j, N);

points[i][j][2] = z(i, j, N);

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

// Obliczenie wektorów normalnych

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

float xu = (-450 \* pow(u, 4) + 900 \* pow(u, 3) - 810 \* pow(u, 2) + 360 \* u - 45)\*cos(PI \* v);

float yu = 640 \* pow(u, 3) - 960 \* pow(u, 2) + 320 \* u;

float zu = (-450 \* pow(u, 4) + 900 \* pow(u, 3) - 810 \* pow(u, 2) + 360 \* u - 45)\*sin(PI \* v);

float xv = PI \* (90 \* pow(u, 5) - 225 \* pow(u, 4) + 270 \* pow(u, 3) - 180 \* pow(u, 2) + 45 \* u)\*sin(PI \* v);

float yv = 0;

float zv = -PI \* (90 \* pow(u, 5) - 225 \* pow(u, 4) + 270 \* pow(u, 3) - 180 \* pow(u, 2) + 45 \* u)\*cos(PI \* v);

if (i < (N / 2))

{

normal[i][j][0] = (yu \* zv - zu \* yv);

normal[i][j][1] = (zu \* xv - xu \* zv);

normal[i][j][2] = (xu \* yv - yu \* xv);

}

else

{

normal[i][j][0] = -(yu \* zv - zu \* yv);

normal[i][j][1] = -(zu \* xv - xu \* zv);

normal[i][j][2] = -(xu \* yv - yu \* xv);

}

// Zamiana na wektory jednostkowe

float len = sqrt(pow(normal[i][j][0], 2) + pow(normal[i][j][1], 2) + pow(normal[i][j][2], 2));

for (int k = 0; k<3; k++)

normal[i][j][k] /= len;

}

glColor3f(1.0f, 0.0f, 1.0f);

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

// Wyświetlenie wszystkich trójkątów w pierwszej połowie jajka

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

for (int i = N/2; i < N - 1; i++)

for (int j = 0; j < N - 1; j++)

{

glBegin(GL\_TRIANGLES);

glNormal3fv(normal[i][j]); // Naniesienie wektora normalnego do punktu

glTexCoord2f(0.0f, 0.0f);

glVertex3f(points[i + 1][j + 1][0], points[i + 1][j + 1][1] - 5, points[i + 1][j + 1][2]);

glNormal3fv(normal[i + 1][j]);

glTexCoord2f(1.0f, 0.0f);

glVertex3f(points[i + 1][j][0], points[i + 1][j][1] - 5, points[i + 1][j][2]);

glNormal3fv(normal[i + 1][j + 1]);

glTexCoord2f(0.5f, 1.0f);

glVertex3f(points[i][j][0], points[i][j][1] - 5, points[i][j][2]); // Naniesienie punktu

glEnd();

glBegin(GL\_TRIANGLES);

glNormal3fv(normal[i][j]);

glTexCoord2f(0.0f, 0.0f);

glVertex3f(points[i][j][0], points[i][j][1] - 5, points[i][j][2]);

glNormal3fv(normal[i][j + 1]);

glTexCoord2f(1.0f, 0.0f);

glVertex3f(points[i][j + 1][0], points[i][j + 1][1] - 5, points[i][j + 1][2]);

glNormal3fv(normal[i + 1][j + 1]);

glTexCoord2f(0.5f, 1.0f);

glVertex3f(points[i + 1][j + 1][0], points[i + 1][j + 1][1] - 5, points[i + 1][j + 1][2]);

glEnd();

}

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

// druga połowa jajka

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

for (int i = 0 ; i < N/2; i++)

for (int j = 0; j < N - 1; j++)

{

glBegin(GL\_TRIANGLES);

glNormal3fv(normal[i][j]); // Naniesienie wektora normalnego do punktu

glTexCoord2f(0.5f, 1.0f);

glVertex3f(points[i][j][0], points[i][j][1] - 5, points[i][j][2]); // Naniesienie punktu

glNormal3fv(normal[i + 1][j]);

glTexCoord2f(1.0f, 0.0f);

glVertex3f(points[i + 1][j][0], points[i + 1][j][1] - 5, points[i + 1][j][2]);

glNormal3fv(normal[i + 1][j + 1]);

glTexCoord2f(0.0f, 0.0f);

glVertex3f(points[i + 1][j + 1][0], points[i + 1][j + 1][1] - 5, points[i + 1][j + 1][2]);

glEnd();

glBegin(GL\_TRIANGLES);

glNormal3fv(normal[i][j]);

glTexCoord2f(0.5f, 1.0f);

glVertex3f(points[i + 1][j + 1][0], points[i + 1][j + 1][1] - 5, points[i + 1][j + 1][2]);

glNormal3fv(normal[i][j + 1]);

glTexCoord2f(1.0f, 0.0f);

glVertex3f(points[i][j + 1][0], points[i][j + 1][1] - 5, points[i][j + 1][2]);

glTexCoord2f(0.0f, 0.0f);

glVertex3f(points[i][j][0], points[i][j][1] - 5, points[i][j][2]);

glEnd();

}

}

void rysuj\_piramide(GLfloat \*a, GLfloat \*b, GLfloat \*c, GLfloat \*d, GLfloat \*e) {

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

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

// Nadanie koloru białego dla scian figury, ponieważ rezygnuję z oświetlenia

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

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

// wyznaczenie 4 trojkątów dla stworzenia ostrosłupa

glBegin(GL\_TRIANGLES);

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

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(a);

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

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(c);

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

glTexCoord2f(0.5f, 1.0f);

glVertex3fv(e);

glEnd();

glBegin(GL\_TRIANGLES);

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

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(b);

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

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(c);

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

glTexCoord2f(0.5f, 1.0f);

glVertex3fv(d);

glEnd();

glBegin(GL\_TRIANGLES);

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

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(c);

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

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(e);

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

glTexCoord2f(0.5f, 1.0f);

glVertex3fv(d);

glEnd();

glBegin(GL\_TRIANGLES);

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

glTexCoord2f(0.0f, 0.0f);

glVertex3fv(a);

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

glTexCoord2f(1.0f, 0.0f);

glVertex3fv(b);

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

glTexCoord2f(0.5f, 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);

}

}

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

// Funkcja rysująca osie układu współrzędnych

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

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();

}

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

// Funkcja określająca co ma być rysowane (zawsze wywoływana gdy trzeba

// przerysować scenę)

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

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, p, 0.0);

Axes();

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

// Jeśli wciśnięto lewy przycisk myszy

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

if (status == 1)

{

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

// Obliczenie położenia obserwatora - kąta azymutu

//i elewacji (X, Y) względem środka sfery

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

thetax += delta\_x \* pix2angle / 30.0;

thetay += delta\_y \* pix2angle / 30.0;

}

else if (status == 2) // Jeśli wciśnięto prawy przycisk myszy

// Obliczenie odległości od środka sfery, obserwatora

theta\_zoom += delta\_y / 10.0;

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

// Ograniczenie zakresu obliczanych wartości, do tych podanych w instrukcji

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

if (thetay > 3.1415) thetay -= 2 \* 3.1415;

else if (thetay <= -3.1415) thetay += 2 \* 3.1415;

if (thetay > 3.1415 / 2 || thetay < -3.1415 / 2) p = -1.0;

else p = 1.0;

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

// Obliczenie położenia we współrzędnych kartezjańskich, obserwatora

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

viewer[0] = theta\_zoom \* cos(thetax)\*cos(thetay);

viewer[1] = theta\_zoom \* sin(thetay);

viewer[2] = theta\_zoom \* sin(thetax)\*cos(thetay);

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

**// Wywołanie funkcji rysującej jajko**

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

**//rysuj\_jajko();**

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

**// Wywołanie funkcji rysującej piramide**

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

**podziel\_piramide(piramida[0], piramida[1], piramida[2], piramida[3], piramida[4], iteracje);**

glFlush();

// Przekazanie poleceń rysujących do wykonania

glutSwapBuffers();

}

// Funkcja obsługująca zdarzenie wciśnięcia przycisku myszy

void Mouse(int btn, int state, int x, int y)

{

// Jeśli wciśnięto lewy przycisk myszy

if (btn == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{

// Ustawienie obecnego położenia jako poprzedniego, na potrzeby funkcji Motion

x\_pos\_old = x;

y\_pos\_old = y;

status = 1;

}

// Jeśli wciśnięto prawy przycisk myszy

else if (btn == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)

{

// Ustawienie obecnego położenia jako poprzedniego, na potrzeby funkcji Motion

y\_pos\_old = y;

status = 2;

}

// Jeśli zwolniono przyciski myszy

else

status = 0;

}

// Funkcja obsługująca zdarzenie przesunięcia myszy

void Motion(GLsizei x, GLsizei y)

{

// Obliczenie przesunięcia względem położenia poprzedniego

delta\_x = x - x\_pos\_old;

x\_pos\_old = x;

delta\_y = y - y\_pos\_old;

y\_pos\_old = y;

glutPostRedisplay();

}

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

// Funkcja wczytuje dane obrazu zapisanego w formacie TGA w pliku o nazwie

// FileName, alokuje pamięć i zwraca wskaźnik (pBits) do bufora w którym

// umieszczone są dane.

// Ponadto udostępnia szerokość (ImWidth), wysokość (ImHeight) obrazu

// tekstury oraz dane opisujące format obrazu według specyfikacji OpenGL

// (ImComponents) i (ImFormat).

// Jest to bardzo uproszczona wersja funkcji wczytującej dane z pliku TGA.

// Działa tylko dla obrazów wykorzystujących 8, 24, or 32 bitowy kolor.

// Nie obsługuje plików w formacie TGA kodowanych z kompresją RLE.

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

GLbyte \*LoadTGAImage(const char \*FileName, GLint \*ImWidth, GLint \*ImHeight, GLint \*ImComponents, GLenum \*ImFormat)

{

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

// Struktura dla nagłówka pliku TGA

#pragma pack(1)

typedef struct

{

GLbyte idlength;

GLbyte colormaptype;

GLbyte datatypecode;

unsigned short colormapstart;

unsigned short colormaplength;

unsigned char colormapdepth;

unsigned short x\_orgin;

unsigned short y\_orgin;

unsigned short width;

unsigned short height;

GLbyte bitsperpixel;

GLbyte descriptor;

}TGAHEADER;

#pragma pack(8)

FILE \*pFile;

TGAHEADER tgaHeader;

unsigned long lImageSize;

short sDepth;

GLbyte \*pbitsperpixel = NULL;

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

// Wartości domyślne zwracane w przypadku błędu

\*ImWidth = 0;

\*ImHeight = 0;

\*ImFormat = GL\_BGR\_EXT;

\*ImComponents = GL\_RGB8;

pFile = fopen(FileName, "rb");

if (pFile == NULL)

return NULL;

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

// Przeczytanie nagłówka pliku

fread(&tgaHeader, sizeof(TGAHEADER), 1, pFile);

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

// Odczytanie szerokości, wysokości i głębi obrazu

\*ImWidth = tgaHeader.width;

\*ImHeight = tgaHeader.height;

sDepth = tgaHeader.bitsperpixel / 8;

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

// Sprawdzenie, czy głębia spełnia założone warunki (8, 24, lub 32 bity)

if (tgaHeader.bitsperpixel != 8 && tgaHeader.bitsperpixel != 24 && tgaHeader.bitsperpixel != 32)

return NULL;

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

// Obliczenie rozmiaru bufora w pamięci

lImageSize = tgaHeader.width \* tgaHeader.height \* sDepth;

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

// Alokacja pamięci dla danych obrazu

pbitsperpixel = (GLbyte\*)malloc(lImageSize \* sizeof(GLbyte));

if (pbitsperpixel == NULL)

return NULL;

if (fread(pbitsperpixel, lImageSize, 1, pFile) != 1)

{

free(pbitsperpixel);

return NULL;

}

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

// Ustawienie formatu OpenGL

switch (sDepth)

{

case 3:

\*ImFormat = GL\_BGR\_EXT;

\*ImComponents = GL\_RGB8;

break;

case 4:

\*ImFormat = GL\_BGRA\_EXT;

\*ImComponents = GL\_RGBA8;

break;

case 1:

\*ImFormat = GL\_LUMINANCE;

\*ImComponents = GL\_LUMINANCE8;

break;

};

fclose(pFile);

return pbitsperpixel;

}

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

// Funkcja ustalająca stan renderowania

void MyInit(void)

{

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

// Zmienne dla obrazu tekstury

GLbyte \*pBytes;

GLint ImWidth, ImHeight, ImComponents;

GLenum ImFormat;

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

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

// Teksturowanie będzie prowadzone tyko po jednej stronie ściany

glEnable(GL\_CULL\_FACE);

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

// Przeczytanie obrazu tekstury z pliku o nazwie tekstura.tga

pBytes = LoadTGAImage("t\_256.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat);

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

// Zdefiniowanie tekstury 2-D

glTexImage2D(GL\_TEXTURE\_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat, GL\_UNSIGNED\_BYTE, pBytes);

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

// Zwolnienie pamięci

free(pBytes);

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

// Włączenie mechanizmu teksturowania

glEnable(GL\_TEXTURE\_2D);

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

// Ustalenie trybu teksturowania

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

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

// Określenie sposobu nakładania tekstur

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

}

// Funkcja ma za zadanie utrzymanie stałych proporcji rysowanych

// w przypadku zmiany rozmiarów okna.

// Parametry vertical i horizontal (wysokość i szerokość okna) są

// przekazywane do funkcji za każdym razem gdy zmieni się rozmiar okna.

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(100.0, 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 wielkości okna okna widoku (viewport) w zale?no?ci

// relacji pomiędzy wysokością i szerokością okna

glMatrixMode(GL\_MODELVIEW);

// Przełączenie macierzy bieżącej na macierz widoku modelu

glLoadIdentity();

// Czyszczenie macierzy bieżącej

}

// Główny punkt wejścia programu. Program działa w trybie konsoli

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

{

// podanie ilosci iteracji

cout << "Podaj liczbe interacji: ";

cin >> iteracje;

if (iteracje != 0)

{

iteracje--;

}

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(600, 600);

glutCreateWindow("Grafika Komputerowa");

// Uruchomienie obsługi zdarzeń wciśnięcia klawisza, przycisku myszy

// oraz jej przesunięcia, wybranymi funkcjami

glutMouseFunc(Mouse);

glutMotionFunc(Motion);

glutDisplayFunc(RenderScene);

// Określenie, że funkcja RenderScene będzie funkcją zwrotną

// (callback function). Bedzie ona wywoływana za każdym razem

// gdy zajdzie potrzba przeryswania okna

glutReshapeFunc(ChangeSize);

// Dla aktualnego okna ustala funkcję zwrotną odpowiedzialną

// zazmiany rozmiaru okna

MyInit();

// Funkcja MyInit() (zdefiniowana powyżej) wykonuje wszelkie

// inicjalizacje konieczne przed przystąpieniem do renderowania

glEnable(GL\_DEPTH\_TEST);

// Włączenie mechanizmu usuwania powierzchni niewidocznych

glutMainLoop();

// Funkcja uruchamia szkielet biblioteki GLUT

}

