pwr**POLITECHNIKA WROCŁAWSKA**

**Instytut Informatyki, Automatyki i Robotyki**

**Zakład Systemów Komputerowych**

**Wprowadzenie do grafiki komputerowej**

**Kurs: INEK00012L**

**Sprawozdanie z ćwiczenia nr**

**TEMAT ĆWICZENIA :**

**OpenGL – modelowanie obiektów 3D (jajko, piramida Sierpińskiego)**

|  |  |
| --- | --- |
| **Wykonał:** | **Paweł Biel** |
| **Termin:** | **WT TN 13:15 – 16:15** |
| **Data wykonania ćwiczenia:** | **5.11.2017** |
| **Data oddania sprawozdania:** | **7.11.17** |
| **Ocena:** |  |

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

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

//

// PLIK ŹRÓDŁOWY: Source.cpp

//

// OPIS: Program służy do rysowania jajka oraz piramidy Sierpińskiego

//

//

// 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

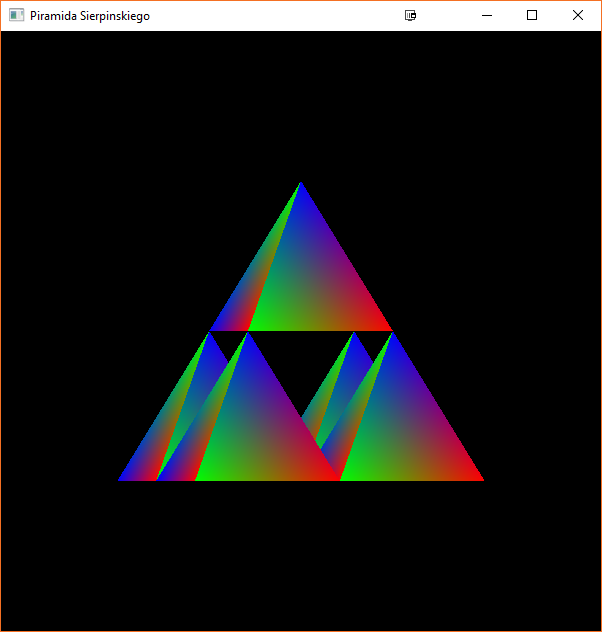
//

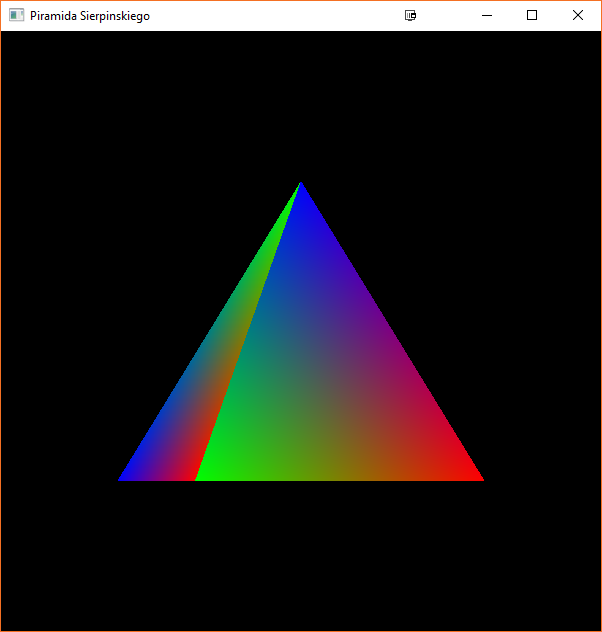
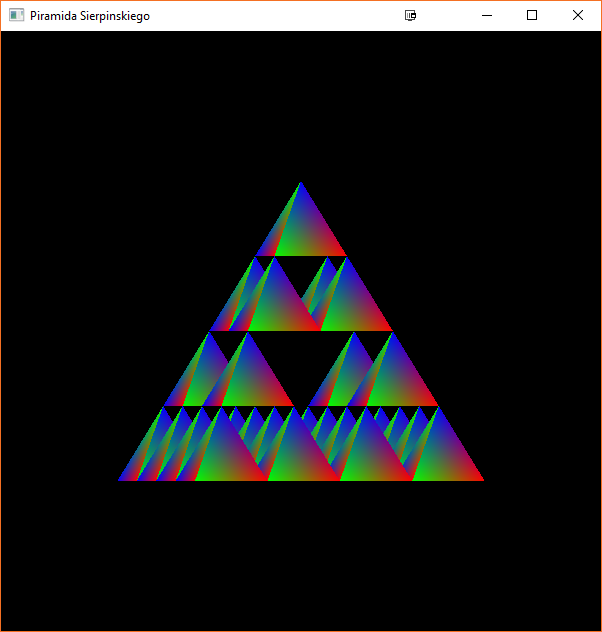
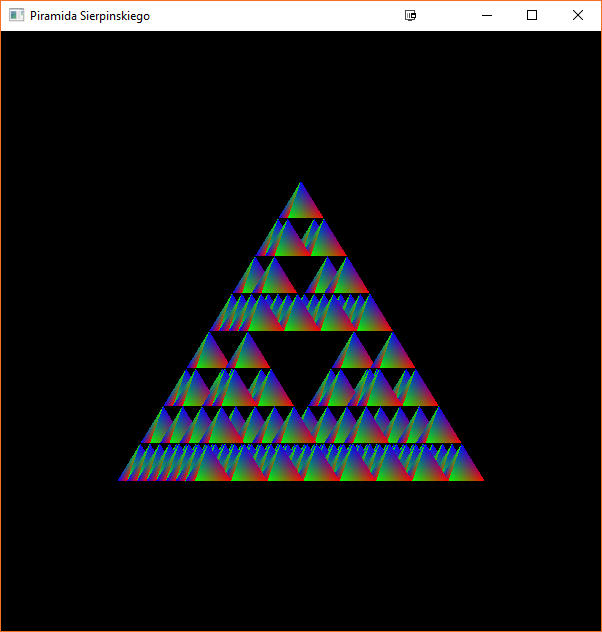
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

**1. Piramida Sierpińskiego**

1. #include <windows.h>
2. #include <GL/glut.h>
3. #include <cstdlib>
4. #include <ctime>
5. #include <iostream>
6. using namespace std;
7. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
8. // Prototypy funkcji
9. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
10. void rysuj\_piramide(GLfloat \*a, GLfloat \*b, GLfloat \*c, GLfloat \*d, GLfloat \*e);
11. void podziel\_piramide(GLfloat \*a, GLfloat \*b, GLfloat \*c, GLfloat \*d, GLfloat \*e, int iteraciones);
12. void RenderScene();
13. void MyInint();
14. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
15. // Punkty "startowe" dla rysowania piramidy
16. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
17. GLfloat piramida[5][3] =
18. { { 1.0f, -1.0f, 1.0f },
19. { -1.0f,-1.0f, 1.0f },
20. { 0.0f, 1.0f, 0.0f },
21. { -1.0f,-1.0f,-1.0f },
22. { 1.0f, -1.0f, -1.0f }};
23. int iteracje = 0;
24. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
25. // Funkcja rysująca piramide
26. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
27. void rysuj\_piramide(GLfloat \*a, GLfloat \*b, GLfloat \*c, GLfloat \*d, GLfloat \*e) {
28. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
30. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
31. // wyznaczenie 4 trojkątów dla stworzenia ostrosłupa
32. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
33. glBegin(GL\_TRIANGLES);
34. glColor3f(1.0f, 0.0f, 0.0f);
35. glVertex3fv(a);
36. glColor3f(0.0f, 1.0f, 0.0f);
37. glVertex3fv(c);
38. glColor3f(0.0f, 0.0f, 1.0f);
39. glVertex3fv(e);
40. glEnd();
42. glBegin(GL\_TRIANGLES);
43. glColor3f(1.0f, 0.0f, 0.0f);
44. glVertex3fv(b);
45. glColor3f(0.0f, 1.0f, 0.0f);
46. glVertex3fv(c);
47. glColor3f(0.0f, 0.0f, 1.0f);
48. glVertex3fv(d);
49. glEnd();
51. glBegin(GL\_TRIANGLES);
52. glColor3f(1.0f, 0.0f, 0.0f);
53. glVertex3fv(c);
54. glColor3f(0.0f, 1.0f, 0.0f);
55. glVertex3fv(e);
56. glColor3f(0.0f, 0.0f, 1.0f);
57. glVertex3fv(d);
58. glEnd();
59. glBegin(GL\_TRIANGLES);
60. glColor3f(1.0f, 0.0f, 0.0f);
61. glVertex3fv(a);
62. glColor3f(0.0f, 1.0f, 0.0f);
63. glVertex3fv(b);
64. glColor3f(0.0f, 0.0f, 1.0f);
65. glVertex3fv(c);
66. glEnd();
67. }
68. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
69. // Funkcja dzieląca piramide zaleznie od ilosci iteracji
70. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
71. void podziel\_piramide(GLfloat \*a, GLfloat \*b, GLfloat \*c, GLfloat \*d, GLfloat \*e, int iteracja) {
72. GLfloat wierzcholek[9][3];
73. int j;
75. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
76. // znajdz punkty środkowe dla każdej krawędzi
77. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
79. if (iteracja > 0) {
81. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
82. // podział krawędzi wokół podstawy figury
83. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
84. for (j = 0; j < 3; j++) {
85. wierzcholek[0][j] = (a[j] + b[j]) / 2;
86. }
87. for (j = 0; j < 3; j++) {
88. wierzcholek[1][j] = (b[j] + d[j]) / 2;
89. }
90. for (j = 0; j < 3; j++) {
91. wierzcholek[2][j] = (d[j] + e[j]) / 2;
92. }
93. for (j = 0; j < 3; j++) {
94. wierzcholek[3][j] = (e[j] + a[j]) / 2;
95. }
97. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
98. // podział krawędzi bocznych
99. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
100. for (j = 0; j < 3; j++) {
101. wierzcholek[4][j] = (c[j] + a[j]) / 2;
102. }
103. for (j = 0; j < 3; j++) {
104. wierzcholek[5][j] = (c[j] + b[j]) / 2;
105. }
106. for (j = 0; j < 3; j++) {
107. wierzcholek[6][j] = (c[j] + d[j]) / 2;
108. }
109. for (j = 0; j < 3; j++) {
110. wierzcholek[7][j] = (c[j] + e[j]) / 2;
111. }
112. for (j = 0; j < 3; j++) {
113. wierzcholek[8][j] = (wierzcholek[3][j] + wierzcholek[1][j]) / 2;
114. }
116. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
117. //dla każdego trójkąta, który wchodzi, tworzone są 5 mniejsze trójkąty i rekurencyjnie są one podzielone po kolei
118. // od wierzchołka lewego dolnego w kierunku odwrotnym do wskazówek zegara, a na samym koncu górny trójkąt
119. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
121. podziel\_piramide(a, wierzcholek[0], wierzcholek[4], wierzcholek[8],wierzcholek[3], iteracja - 1);
122. podziel\_piramide(wierzcholek[0], b, wierzcholek[5], wierzcholek[1], wierzcholek[8], iteracja - 1);
123. podziel\_piramide(wierzcholek[8], wierzcholek[1], wierzcholek[6], d, wierzcholek[2], iteracja - 1);
124. podziel\_piramide(wierzcholek[3], wierzcholek[8], wierzcholek[7], wierzcholek[2], e, iteracja - 1);
125. podziel\_piramide(wierzcholek[4], wierzcholek[5], c, wierzcholek[6], wierzcholek[7], iteracja - 1);
126. }
127. else {
128. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
129. // narysuj piramide gdy iteracja = 0
130. // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
131. rysuj\_piramide(a, b, c, d, e);
132. }
133. }
134. void RenderScene() {
135. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
136. glPushMatrix();
137. glRotated(15, 0.0, 1.0, 0.0); // Obrót o 15 stopni

140. podziel\_piramide(piramida[0], piramida[1], piramida[2], piramida[3],piramida[4], iteracje);
141. glPopMatrix();
142. glFlush();
144. }
145. void MyInint(void) {
146. glClearColor(0.0, 0.0, 0.0, 1.0); // czyszczenie coloru
147. glColor3f(0.0, 0.0, 0.0); //ustawienie koloru
148. glMatrixMode(GL\_PROJECTION);
149. glLoadIdentity();
150. glOrtho(-2.0, 2.0, -2.0, 2.0, -20.0, 20.0);
151. glEnable(GL\_DEPTH\_TEST);
152. }
153. int main(int argc, char\*\* argv) {
154. srand(time(NULL));
155. // podanie ilosci iteracji
156. cout << "Podaj liczbe interacji: ";
157. cin >> iteracje;
158. if (iteracje != 0)
159. {
160. iteracje--;
161. }
162. glutInit(&argc, argv);
163. glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGBA);
164. // Ustawienie trybu wyœwietlania
165. // GLUT\_SINGLE - pojedynczy bufor wyświetlania
166. // GLUT\_RGBA - model kolorów RGB
167. glutInitWindowSize(600, 600);
168. glutCreateWindow("Piramida Sierpinskiego");
169. // Utworzenie okna i określenie treści napisu w nagłówku okna
170. glutDisplayFunc(RenderScene);
171. MyInint();
172. // Funkcja MyInit (zdefiniowana powyżej) wykonuje wszelkie
173. // inicjalizacje konieczneprzed przystąpieniem do renderowania
174. //glutDisplayFunc(RenderScene);
176. glutMainLoop();
177. // Funkcja uruchamia szkielet biblioteki GLUT
178. }



**2. Jajko 3D**

#include <windows.h>

#include <gl/gl.h>

#include <gl/glut.h>

#include <iostream>

#include <cmath>

#define M\_PI 3.14159265358979323846

using namespace std;

typedef float point3[3];

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

static GLfloat theta[] = { 0.0, 0.0, 0.0 }; // trzy kąty obrotu

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

// 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)

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

// Przeliczenie wspołrzędnych 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(M\_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)) - 5 ;

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(M\_PI \* v);

return zz;

}

void Jajko()

{

int N = 100;

point3 \*\* tab = new point3 \*[N];

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

// Rzutowanie punktów o współrzednych 3d na macierz

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

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

{

tab[i] = new point3[N];

}

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

{

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

{

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

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

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

}

}

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

// Wygenerowanie tablicy kolorów

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

point3 \*\*colors;

colors = new point3\*[N];

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

colors[i] = new point3[N];

}

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

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

colors[i][j][0] = float(rand() % 1000) / 1000;

colors[i][j][1] = float(rand() % 1000) / 1000;

colors[i][j][2] = float(rand() % 1000) / 1000;

}

}

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

// Wyswietlenie jajka w postaci punktow

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

if (model == 1)

{

glBegin(GL\_POINTS);

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

{

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

{

glVertex3fv(tab[i][j]);

}

}

glEnd();

}

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

// Wyswietlenie jajka w postaci siatki

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

else if (model == 2)

{

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

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

// linie pionowe

glBegin(GL\_LINES);

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

glVertex3fv(tab[i][j]);

glVertex3fv(tab[(i + 1) % N][j]);

glEnd();

//linie ukośne

glBegin(GL\_LINES);

if (((j + 1) != N) && ((i + 1) != N)) {

glVertex3fv(tab[i + 1][j]);

glVertex3fv(tab[i][j + 1]);

}

else {

if (i > 0) {

glVertex3fv(tab[i][j]);

glVertex3fv(tab[N - i - 1][0]);

}

}

glEnd();

//linie poziome

glBegin(GL\_LINES);

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

if ((j + 1) == N) {

if (i > 0) {

glVertex3fv(tab[N - i][0]);

glVertex3fv(tab[i][j]);

}

}

else {

glVertex3fv(tab[i][j + 1]);

glVertex3fv(tab[i][j]);

}

glEnd();

}

}

}

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

// Wyswietlenie pokolorwanego jajaka

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

else if (model == 3)

{

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

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

if ((j + 1) != N) {

glBegin(GL\_TRIANGLES);

glColor3fv(colors[i][j]);

glVertex3fv(tab[i][j]);

glColor3fv(colors[(i + 1) % N][j]);

glVertex3fv(tab[(i + 1) % N][j]);

glColor3fv(colors[i][j + 1]);

glVertex3fv(tab[i][j + 1]);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3fv(colors[(i + 1) % N][j]);

glVertex3fv(tab[(i + 1) % N][j]);

glColor3fv(colors[(i + 1) % N][j + 1]);

glVertex3fv(tab[(i + 1) % N][j + 1]);

glColor3fv(colors[i][j + 1]);

glVertex3fv(tab[i][j + 1]);

glEnd();

}

else {

if (i > 0) {

glBegin(GL\_TRIANGLES);

glColor3fv(colors[i][j]);

glVertex3fv(tab[i][j]);

glColor3fv(colors[(i + 1) % N][j]);

glVertex3fv(tab[(i + 1) % N][j]);

glColor3fv(colors[N - i][0]);

glVertex3fv(tab[N - i][0]);

glEnd();

}

glBegin(GL\_TRIANGLES);

glColor3fv(colors[(i + 1) % N][j]);

glVertex3fv(tab[(i + 1) % N][j]);

glColor3fv(colors[(N - i) % N][0]);

glVertex3fv(tab[(N - i) % N][0]);

glColor3fv(colors[N - i - 1][0]);

glVertex3fv(tab[N - i - 1][0]);

glEnd();

}

}

}

}

}

void spinEgg()

{

theta[0] -= 0.5;

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

theta[1] -= 0.5;

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

theta[2] -= 0.5;

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

Sleep(50);

glutPostRedisplay(); //odświeżenie zawartości aktualnego okna

}

void RenderScene(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Czyszczenie okna aktualnym kolorem czyszczącym

glLoadIdentity();

// Czyszczenie macierzy bieżącej

Axes();

// Narysowanie osi przy pomocy funkcji zdefiniowanej wyżej

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);

Jajko();

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

//glutWireTeapot(3.0); // Narysowanie obrazu czajnika do herbaty

//glRotated(60.0, 1.0, 1.0, 1.0); // Obrót o 60 stopni

//glutWireTeapot(3.0); // Narysowanie obrazu czajnika do herbaty

glFlush();

// Przekazanie poleceń rysujących do wykonania

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

}

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

// Funkcja ustalająca stan renderowania

void MyInit(void)

{

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

// Kolor czyszcący (wypełnienia okna) ustawiono na czarny

}

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

// 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)

{

GLfloat AspectRatio;

// Deklaracja zmiennej AspectRatio określającej proporcję

// wymiarów okna

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

vertical = 1;

glViewport(0, 0, horizontal, vertical);

// Ustawienie wielkościokna okna widoku (viewport)

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

glMatrixMode(GL\_PROJECTION);

// Przełączenie macierzy bieżącej na macierz projekcji

glLoadIdentity();

// Czyszcznie macierzy bieżącej

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

// Wyznaczenie współczynnika proporcji okna

// Gdy okno nie jest kwadratem wymagane jest określenie tak zwanej

// przestrzeni ograniczającej pozwalającej zachować właściwe

// proporcje rysowanego obiektu.

// Do okreslenia przestrzeni ograniczjącej służy funkcja

// glOrtho(...)

if (horizontal <= vertical)

glOrtho(-7.5, 7.5, -7.5 / AspectRatio, 7.5 / AspectRatio, 10.0, -10.0);

else

glOrtho(-7.5\*AspectRatio, 7.5\*AspectRatio, -7.5, 7.5, 10.0, -10.0);

glMatrixMode(GL\_MODELVIEW);

// Przełączenie macierzy bieżącej na macierz widoku modelu

glLoadIdentity();

// Czyszcenie macierzy bieżącej

}

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

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

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

{

glutInit(&argc, argv);

cout << sin(M\_PI);

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

glutInitWindowSize(300, 300);

glutCreateWindow("Układ współrzędnych 3-D");

glutKeyboardFunc(keys);

glutIdleFunc(spinEgg);

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

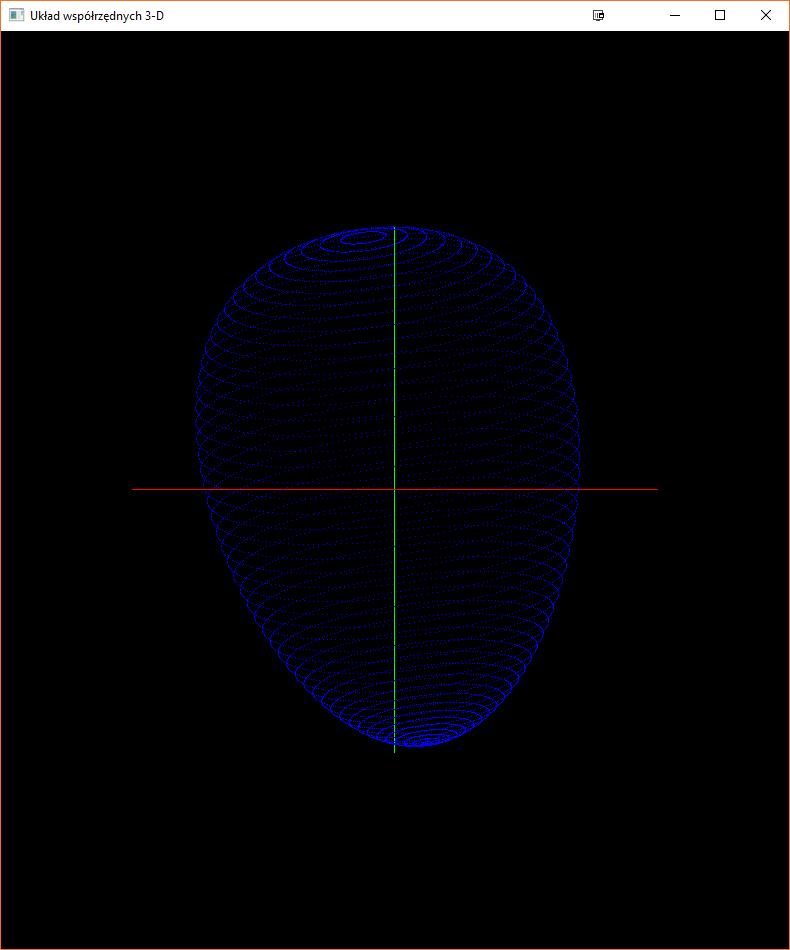
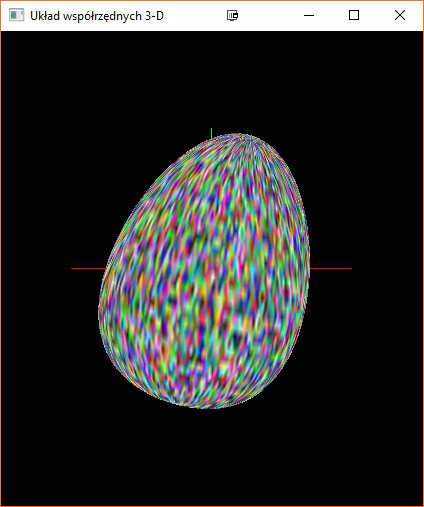
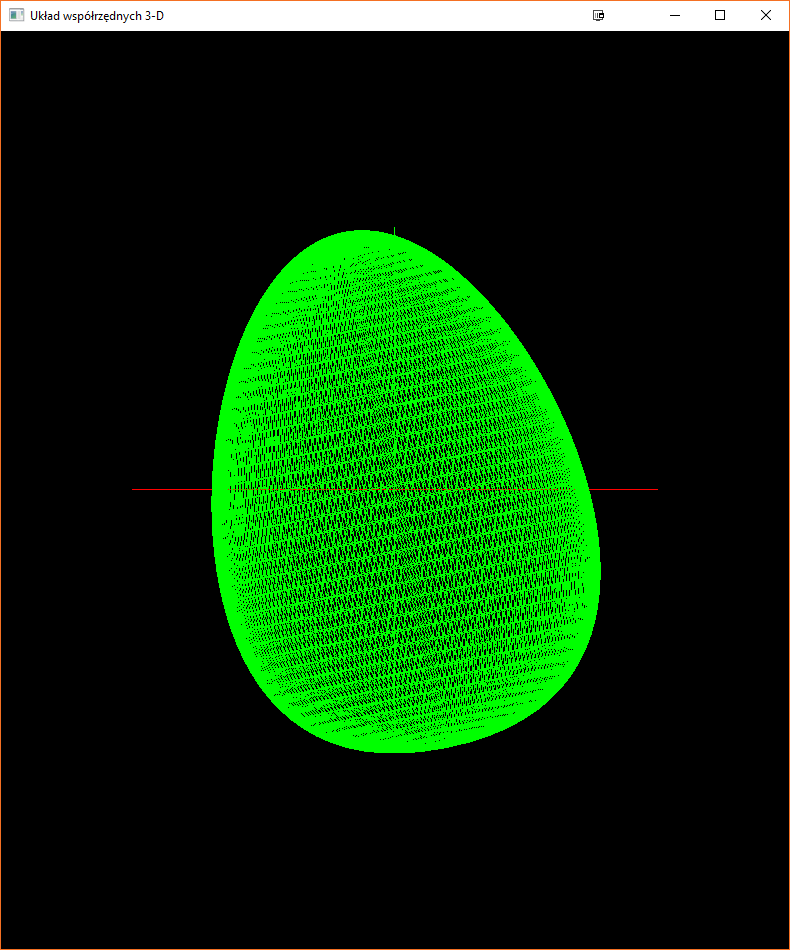
glEnable(GL\_DEPTH\_TEST);

// Włączenie mechanizmu usuwania powierzchni niewidocznych

glutMainLoop();

// Funkcja uruchamia szkielet biblioteki GLUT

}

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