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

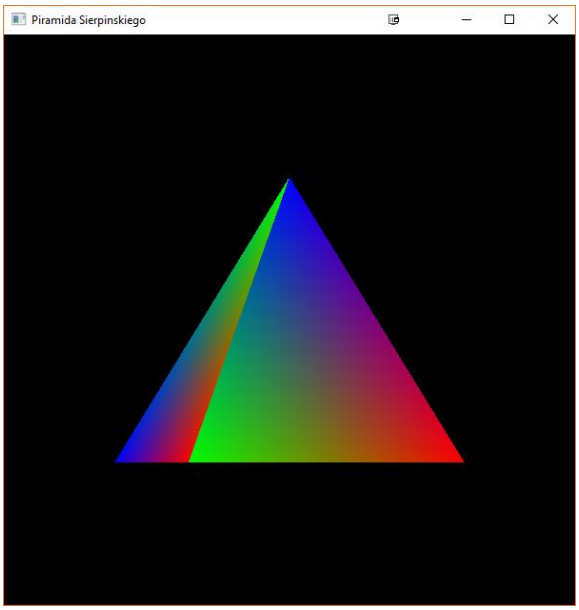
```
1.  #include <windows.h>
2.  #include <GL/glut.h>
3.  #include <cstdlib>
4.  #include <ctime>
5.  #include <iostream>
6.
7.  using namespace std;
8.
9.
10.
11. // ************************************************************
12. // Prototypy funkcji
13. // ************************************************************
14. void rysuj_piramide(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, GLfloat
    *e);
15. void podziel_piramide(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, GLfloat
    *e, int iteraciones);
16. void RenderScene();
17. void MyInint();
18.
19. // ************************************************************
20. // Punkty "startowe" dla rysowania piramidy
21. // ************************************************************
22. GLfloat piramida[5][3] =
23. { { 1.0f, -1.0f, 1.0f },
24. { -1.0f,-1.0f, 1.0f },
25. { 0.0f, 1.0f, 0.0f },
26. { -1.0f,-1.0f,-1.0f },
27. { 1.0f, -1.0f, -1.0f }};
28.
29. int iteracje = 0;
30.
31. // ************************************************************
```

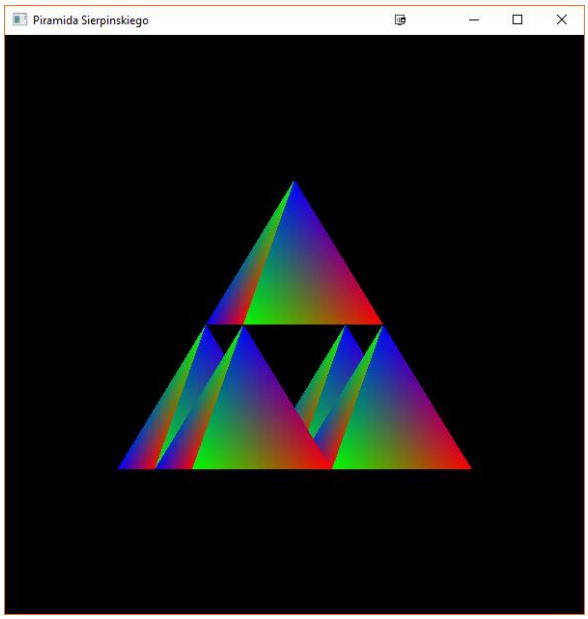
```
32. // Funkcja rysująca piramide
33. // ****************************************************************
34. void rysuj_piramide(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, GLfloat *e)
    {
35.         glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
36.
37.         // ****************************************************************
38.         // wyznaczenie 4 trojkątów dla stworzenia ostrosłupa
39.         // ****************************************************************
40.
41.         glBegin(GL_TRIANGLES);
42.         glColor3f(1.0f, 0.0f, 0.0f);
43.         glVertex3fv(a);
44.         glColor3f(0.0f, 1.0f, 0.0f);
45.         glVertex3fv(c);
46.         glColor3f(0.0f, 0.0f, 1.0f);
47.         glVertex3fv(e);
48.         glEnd();
49.
50.
51.
52.         glBegin(GL_TRIANGLES);
53.         glColor3f(1.0f, 0.0f, 0.0f);
54.         glVertex3fv(b);
55.         glColor3f(0.0f, 1.0f, 0.0f);
56.         glVertex3fv(c);
57.         glColor3f(0.0f, 0.0f, 1.0f);
58.         glVertex3fv(d);
59.         glEnd();
60.
61.
62.         glBegin(GL_TRIANGLES);
63.         glColor3f(1.0f, 0.0f, 0.0f);
64.         glVertex3fv(c);
65.         glColor3f(0.0f, 1.0f, 0.0f);
66.         glVertex3fv(e);
67.         glColor3f(0.0f, 0.0f, 1.0f);
68.         glVertex3fv(d);
69.         glEnd();
70.
71.         glBegin(GL_TRIANGLES);
72.         glColor3f(1.0f, 0.0f, 0.0f);
73.         glVertex3fv(a);
74.         glColor3f(0.0f, 1.0f, 0.0f);
75.         glVertex3fv(b);
76.         glColor3f(0.0f, 0.0f, 1.0f);
77.         glVertex3fv(c);
78.         glEnd();
79. }
80.
81. // ****************************************************************
82. // Funkcja dzieląca piramide zaleznie od ilosci iteracji
83. // ****************************************************************
84. void podziel_piramide(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, GLfloat
    *e, int iteracja) {
85.         GLfloat wierzcholek[9][3];
86.         int j;
87.
88.         // ****************************************************************
89.         // znajdz punkty środkowe dla  każdej krawędzi
90.         // ****************************************************************
91.
```

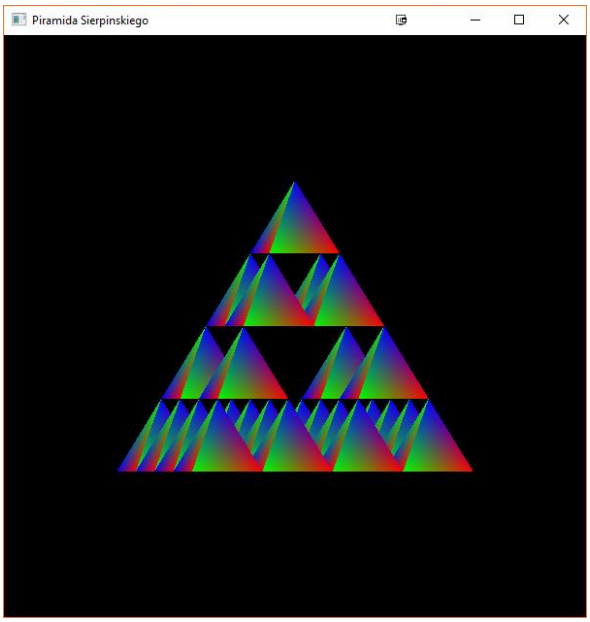
```
92.       if (iteracja > 0) {
93.
94.             //
    ************************************************************
95.             // podział krawędzi wokół podstawy figury
96.             //
    ************************************************************
97.
98.             for (j = 0; j < 3; j++) {
99.                   wierzcholek[0][j] = (a[j] + b[j]) / 2;
100.                  }
101.                  for (j = 0; j < 3; j++) {
102.                        wierzcholek[1][j] = (b[j] + d[j]) / 2;
103.                  }
104.                  for (j = 0; j < 3; j++) {
105.                        wierzcholek[2][j] = (d[j] + e[j]) / 2;
106.                  }
107.                  for (j = 0; j < 3; j++) {
108.                        wierzcholek[3][j] = (e[j] + a[j]) / 2;
109.                  }
110.
111.
112.                  //
    ************************************************************
113.                  // podział krawędzi bocznych
114.                  //
    ************************************************************
115.                  for (j = 0; j < 3; j++) {
116.                        wierzcholek[4][j] = (c[j] + a[j]) / 2;
117.                  }
118.                  for (j = 0; j < 3; j++) {
119.                        wierzcholek[5][j] = (c[j] + b[j]) / 2;
120.                  }
121.                  for (j = 0; j < 3; j++) {
122.                        wierzcholek[6][j] = (c[j] + d[j]) / 2;
123.                  }
124.                  for (j = 0; j < 3; j++) {
125.                        wierzcholek[7][j] = (c[j] + e[j]) / 2;
126.                  }
127.                  for (j = 0; j < 3; j++) {
128.                        wierzcholek[8][j] = (wierzcholek[3][j] +
    wierzcholek[1][j]) / 2;
129.                  }
130.
131.
132.                  //
    ************************************************************
133.                  //dla każdego trójkąta, który wchodzi, tworzone są 5
    mniejsze trójkąty i rekurencyjnie są one podzielone po kolei
134.                  // od wierzchołka lewego dolnego w kierunku odwrotnym do
    wskazówek zegara, a na samym koncu górny trójkąt
135.                  //
    ************************************************************
136.
137.                  podziel_piramide(a, wierzcholek[0], wierzcholek[4],
    wierzcholek[8],wierzcholek[3], iteracja - 1);
138.                  podziel_piramide(wierzcholek[0], b, wierzcholek[5],
    wierzcholek[1], wierzcholek[8], iteracja - 1);
139.                  podziel_piramide(wierzcholek[8], wierzcholek[1],
    wierzcholek[6], d, wierzcholek[2], iteracja - 1);
140.                  podziel_piramide(wierzcholek[3], wierzcholek[8],
    wierzcholek[7], wierzcholek[2], e, iteracja - 1);
```

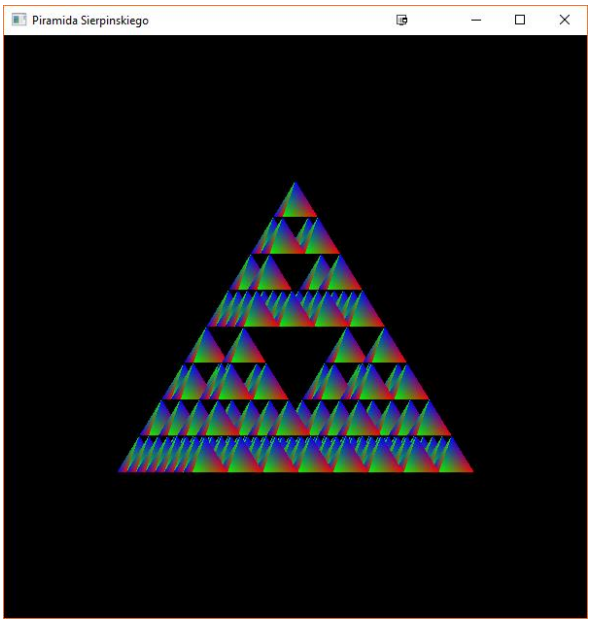
```
141.                    podziel_piramide(wierzcholek[4], wierzcholek[5], c,
    wierzcholek[6], wierzcholek[7], iteracja - 1);
142.
143.
144.            }
145.            else {
146.                    //
    ************************************************************
147.                    // narysuj piramide gdy iteracja = 0
148.                    //
    ************************************************************
149.
150.                    rysuj_piramide(a, b, c, d, e);
151.            }
152.      }
153.
154.      void RenderScene() {
155.            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
156.            glPushMatrix();
157.            glRotated(15, 0.0, 1.0, 0.0);  // Obrót o 15 stopni
158.
159.
160.            podziel_piramide(piramida[0], piramida[1], piramida[2],
    piramida[3],piramida[4], iteracje);
161.
162.            glPopMatrix();
163.            glFlush();
164.
165.
166.      }
167.
168.
169.      void MyInint(void) {
170.            glClearColor(0.0, 0.0, 0.0, 1.0);        // czyszczenie coloru
171.            glColor3f(0.0, 0.0, 0.0);         //ustawienie koloru
172.            glMatrixMode(GL_PROJECTION);
173.            glLoadIdentity();
174.            glOrtho(-2.0, 2.0, -2.0, 2.0, -20.0, 20.0);
175.            glEnable(GL_DEPTH_TEST);
176.      }
177.
178.
179.
180.      int main(int argc, char** argv) {
181.            srand(time(NULL));
182.
183.            // podanie ilosci iteracji
184.            cout << "Podaj liczbe interacji: ";
185.            cin >> iteracje;
186.            if (iteracje != 0)
187.                  {
188.                  iteracje--;
189.                  }
190.
191.
192.            glutInit(&argc, argv);
193.
194.            glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
195.            // Ustawienie trybu wyœwietlania
196.            // GLUT_SINGLE - pojedynczy bufor wyświetlania
197.            // GLUT_RGBA - model kolorów RGB
198.
```

```
199.            glutInitWindowSize(600, 600);
200.            glutCreateWindow("Piramida Sierpinskiego");
201.            // Utworzenie okna i określenie treści napisu w nagłówku okna
202.            glutDisplayFunc(RenderScene);
203.
204.            MyInint();
205.            // Funkcja MyInit (zdefiniowana powyżej) wykonuje wszelkie
206.            // inicjalizacje konieczneprzed przystąpieniem do renderowania
207.            //glutDisplayFunc(RenderScene);
208.
209.            glutMainLoop();
210.            // Funkcja uruchamia szkielet biblioteki GLUT
211.        }
```









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 współrzędnych 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
}
/*************************************************************************************
```