Sprawozdanie - Grafika komputerowa i komunikacja człowiek-komputer

Maksim Birszel

14 stycznia 2020

1 Wstęp

Na zajęciach laboratoryjnych poznajemy podstawy tworzenia grafiki komputerowej przy użyciu biblioteki OpenGL z rozszerzeniem GLUT. Przy realizacji ćwiczeń wykorzystane zostało środowisko programistyczne Microsoft Visual Studio.

2 Podstawy OpenGL

Celem zajęć jest poznanie podstawowych pojęć związanych z przedmiotem kursu oraz wstępne zapoznanie się z pracą w wykorzystywanym środowisku.

OpenGL - (ang. Open Graphics Library) – specyfikacja otwartego i uniwersalnego API do tworzenia grafiki. Zestaw funkcji składa się z 250 podstawowych wywołań, umożliwiających budowanie złożonych trójwymiarowych scen z podstawowych figur geometrycznych.

2.1 Podstawowe pojęcia

- Wierzchołek - punkt przestrzeni
- Prymityw - podstawowa jednostka renderingu w OpenGL (np. punkt, linia, trójkąt)
- Renderowanie - proces budowania obrazu w komputerze
- Rasteryzacja - przekształcenie prymitywów w zbiór pikseli
- Piksel - najmniejsza jednostka wizualna na wyświetlaczu
- Ramka - pojedynczy obraz (zbiór pikseli)

2.2 Struktura każdego programu

```
1  #include <gl/gl.h>
2  #include <gl/glut.h>
3
4  // Funkcja okreslajaca co ma byc rysowane (zawsze wywolywana gdy trzeba
5  // przerysowac scene)
6  void RenderScene(void)
7  {
8          // Czyszczenie okna aktualnym kolorem czyszczacym
9          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
10
11         // Przekazanie polecen rysujacych do wykonania
12         glFlush();
13 }
14
15 // Funkcja ustalajaca stan renderowania
16 void MyInit(void)
17 {
18         // Kolor czyszcacy (wypelnienia okna) ustawiono na czarny
19         glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
20 }
21
22 // Glowny punkt wejscia programu. Program dziala w trybie konsoli
23 void main(int argc, char** argv)
24 {
25         glutInit(&argc, argv);
26
27         glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
28
29         glutInitWindowSize(300, 300);
30
31         glutCreateWindow("Nazwa okna");
32
33         // Okreslenie ze funkcja RenderScene bedzie funkcja zwrotna
34         // (callback function). Bedzie ona wywolywana za kazdym razem
35         // gdy zajdzie potrzeb przerysowania okna
36         glutDisplayFunc(RenderScene);
37
38         // Dla aktualnego okna ustala funkcje zwrotna odpowiedzialna
39         // za zmiany rozmiaru okna
40         glutReshapeFunc(ChangeSize);
41
42         // Funkcja MyInit() (zdefiniowana powyzej) wykonuje wszelkie
43         // inicjalizacje konieczne przed przystapieniem do renderowania
44         MyInit();
```

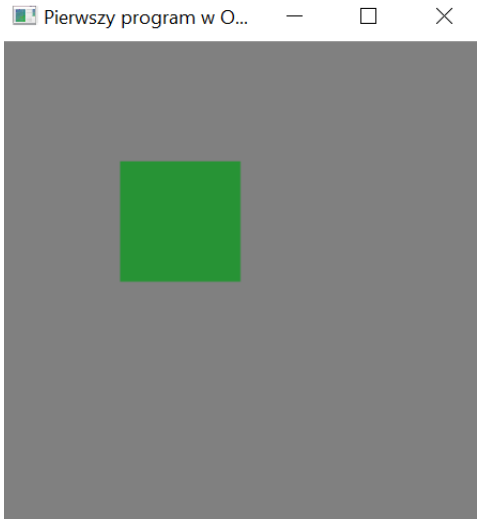
```
45
46          // Wlaczenie mechanizmu usuwania powierzchni niewidocznych
47          glEnable(GL_DEPTH_TEST);
48
49          // Funkcja uruchamia szkielet biblioteki GLUT
50          glutMainLoop();
51  }
```

2.3 Wykonane zadania

2.3.1 Narysowanie kwadratu i funkcji losującej kolor

Zadanie polegało na narysowaniu kwadratu. Dodatkowo została utworzona funkcja losująca kolor i przypisująca go figurze.

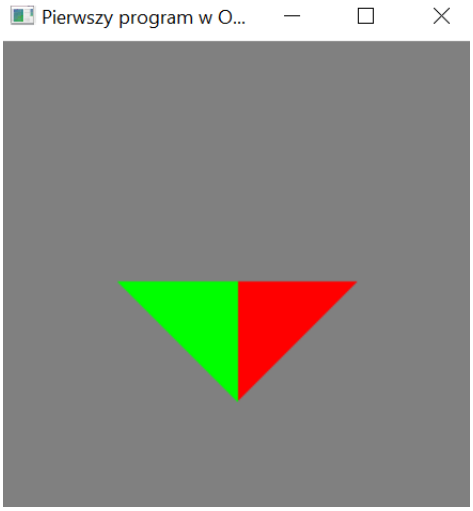
```
1   // Funkcja losujaca kolor
2   byte RandColors()
3   {
4           srand(time(NULL));
5
6           return rand() % 127;
7   }
8
9   // Funkcja rysujaca kwadrat
10  void DrawQuarter()
11  {
12          // Ustawienie losowego koloru
13          srand(time(NULL));
14          glColor3ub(rand() % 255, rand() % 255, rand() % 255);
15
16          Utworzenie polowki kwadratu podajac odpowiednie wspolrzedne
17          glBegin(GL_TRIANGLES);
18                  glVertex2f(-50.0f, 0.0f);
19                  glVertex2f(-50.0f, 50.0f);
20                  glVertex2f(0.0f, 0.0f);
21          glEnd();
22
23          // Utworzenie drugiej polowki kwadratu
24          glBegin(GL_TRIANGLES);
25                  glVertex2f(0.0f, 0.0f);
26                  glVertex2f(-50.0f, 50.0f);
27                  glVertex2f(0.0f, 50.0f);
28          glEnd();
29  }
```



2.3.2 Narysowanie trójkąta w dwóch kolorach

Zadanie polegało na narysowaniu trójkąta w dwóch kolorach.

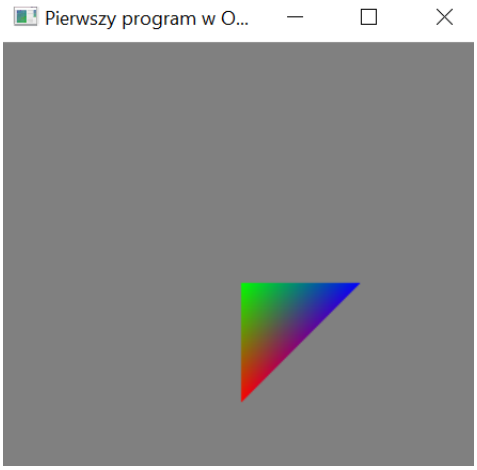
```
1  void DrawTriangle()
2  {
3          // Ustawienie koloru pierwszej polowy trojkata
4          glColor3f(1.0f, 0.0f, 0.0f);
5
6          // Narysowanie pierwszej polowy trojkata
7          glBegin(GL_TRIANGLES);
8                  glVertex2f(0.0f, -50.0f);
9                  glVertex2f(0.0f, 0.0f);
10                 glVertex2f(50.0f, 0.0f);
11         glEnd();
12
13         // Ustawienie koloru drugiej polowy trojkata
14         glColor3f(0.0f, 1.0f, 0.0f);
15
16         // Narysowanie drugiej polowy trojkata
17         glBegin(GL_TRIANGLES);
18                 glVertex2f(-50.0f, 0.0f);
19                 glVertex2f(0.0f, -50.0f);
20                 glVertex2f(0.0f, 0.0f);
21         glEnd();
22 }
```



2.3.3 Narysowanie trójkąta w trzech kolorach

Zadanie polegało na narysowaniu trójkąta w trzech kolorach (gradient).

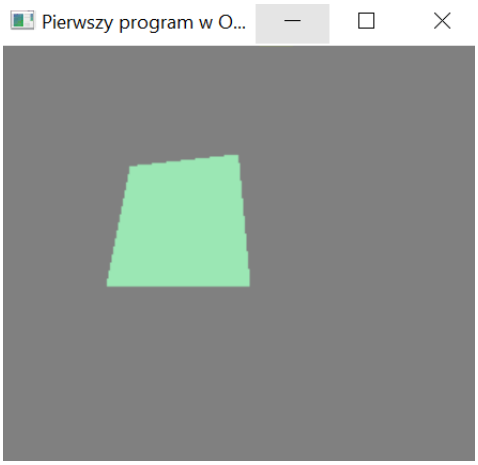
```
1  void DrawTriangle3Colors()
2  {
3          // Rysowanie wierzcholkow i ustawianie im roznych kolorow
4          glBegin(GL_TRIANGLES);
5                  glColor3f(1.0f, 0.0f, 0.0f);
6                  glVertex2f(0.0f, -50.0f);
7                  glColor3f(0.0f, 1.0f, 0.0f);
8                  glVertex2f(0.0f, 0.0f);
9                  glColor3f(0.0f, 0.0f, 1.0f);
10                 glVertex2f(50.0f, 0.0f);
11         glEnd();
12 }
```



2.3.4 Narysowanie zniekształconego kwadratu

Zadanie polegało na narysowaniu kwadratu, który będzie losowo zniekształcony.

```
1  void DistortedQuarter()
2  {
3          // Wylosowanie wartosci add (dodane zniekształcenie)
4          srand(time(NULL));
5          GLint add = rand() % 30;
6
7          // Ustawienie losowego koloru
8          glColor3ub(rand() % 255, rand() % 255, rand() % 255);
9
10         // Narysowanie pierwszej polowy kwadratu
11         // z naniesionym zniekształceniem add
12         glBegin(GL_TRIANGLES);
13         glVertex2i(-50 -add, 0);
14         glVertex2i(-50 + add, 50);
15         glVertex2i(0 + add, 0);
16         glEnd();
17
18         // Narysowanie drugiej polowy kwadratu
19         // z naniesionym zniekształceniem add
20         glBegin(GL_TRIANGLES);
21         glVertex2i(0 + add, 0);
22         glVertex2i(-50 + add, 50);
23         glVertex2i(0, 50 + add);
24         glEnd();
25 }
```



3 Modelowanie 3D

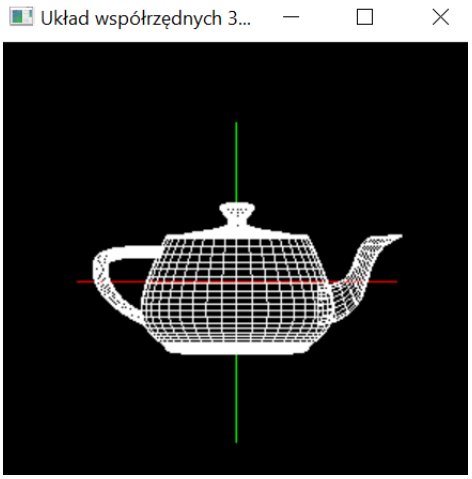
Celem laboratorium jest wprowadzenie w zagadnienia modelowania i wizualizacji scen 3D. W tym zadaniu uczymy się wykorzystywać równania parametryczne, aby utworzyć nietrywialne obiekty.

3.1 Układ współrzędnych i obiekt 3D

Pierwszym zadaniem było narysowanie układu współrzędnych oraz obrazu czajnika.

```
1  // Funkcja rysujaca osie ukladu wspolrzednych
2  void Axes(void)
3  {
4          // poczatek i koniec obrazu osi x
5          point3  x_min = { -5.0, 0.0, 0.0 };
6          point3  x_max = { 5.0, 0.0, 0.0 };
7
8          // poczatek i koniec obrazu osi y
9          point3  y_min = { 0.0, -5.0, 0.0 };
10         point3  y_max = { 0.0,  5.0, 0.0 };
11
12         //  poczatek i koniec obrazu osi z
13         point3  z_min = { 0.0, 0.0, -5.0 };
14         point3  z_max = { 0.0, 0.0,  5.0 };
15
16         // kolor rysowania osi - czerwony
17         glColor3f(1.0f, 0.0f, 0.0f);
18         // rysowanie osi x
19         glBegin(GL_LINES);
20         glVertex3fv(x_min);
21         glVertex3fv(x_max);
22         glEnd();
23
24         // kolor rysowania - zielony
25         glColor3f(0.0f, 1.0f, 0.0f);
26         // rysowanie osi y
27         glBegin(GL_LINES);
28
29         glVertex3fv(y_min);
30         glVertex3fv(y_max);
31         glEnd();
32
33         // kolor rysowania - niebieski
34         glColor3f(0.0f, 0.0f, 1.0f);
35         // rysowanie osi z
36         glBegin(GL_LINES);
```

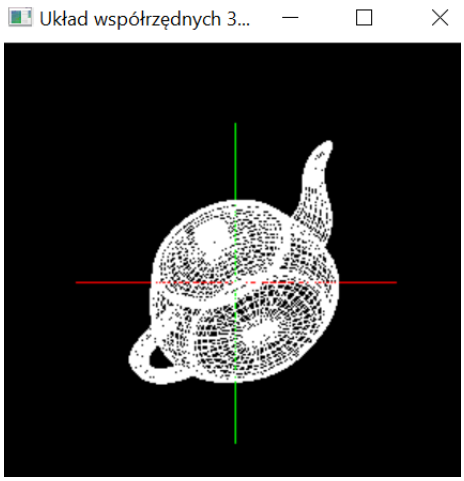
```
37
38          glVertex3fv(z_min);
39          glVertex3fv(z_max);
40          glEnd();
41
42          // Ustawienie koloru rysowania na bialy
43          glColor3f(1.0f, 1.0f, 1.0f);
44
45          // Narysowanie obrazu czajnika do herbaty
46          glutWireTeapot(3.0);
47  }
```



3.2 Transformacja obiektu

Kolejnym zadaniem było zrobienie transformacji danego obiektu.

```
1  // Transformacja o 60 stopni wyswietlanej grafiki
2  void Transformation()
3  {
4           // Mozliwe parametry: GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE
5           glMatrixMode(GL_MODELVIEW);
6
7           // Wykonanie translacji
8           //glTranslated(TYP x, TYPE y, TYPE z);
9
10          // Wykonanie skalowania
11          //glScaled(TYPE x, TYPE y, TYPE z);
12
13          // Wykonuje obrot o kat angle wokol osi oznaczonej przez
14          // punkt (0,0,0) i punkt (1, 1, 1) - OBROT O 60 STOPNI
15          glRotated(60.0, 1.0, 1.0, 1.0);
16  }
```



3.3 Budowa własnego modelu obiektu 3D - jajko

Kolejnym krokiem było narysowanie własnego modelu obiektu 3D, w tym wypadku jajka. Zadanie to zostało wykonane na dwa sposoby: kropkami oraz paskami. Pierwszym zadaniem jest wygenerowanie chmury punktów leżących na powierzchni obiektu. Algorytm generacji zbioru punktów polega na pokryciu dziedziny parametrycznej (kwadratu jednostkowego w przestrzeni u, v) równomierną siatką punktów a następnie przeliczeniu współrzędnych poszczególnych punktów siatki z dziedziny parametrycznej, na punkty w przestrzeni trójwymiarowej. Poszczególny kroki:

- Zadanie liczby N=100 która określa na ile przedziałów podzielony jest bok kwadratu jednostkowego dziedziny parametrycznej.
- Zadeklarowanie tablicy o rozmiarze 100x100, która służy do zapisywania współrzednych punktów w przestrzeni 3-D. Każdy element tablicy zawiera trzy liczby będące współrzędnymi x, y, z jednego punktu.
- Nałożenie na kwadrat jednostkowy dziedziny parametrycznej równoległą siatkę 100x100 punktów.
- Dla każdego punktu u, v nałożonej w kroku poprzednim siatki, obliczenie, przy pomocy podanych równań współrzędne x(u, v), y(u, v) i z(u, v).
- Wyświetlenie na ekranie elementy tablicy współrzędnych punktów.

Powyższe kroki przedstawia poniższy kod:

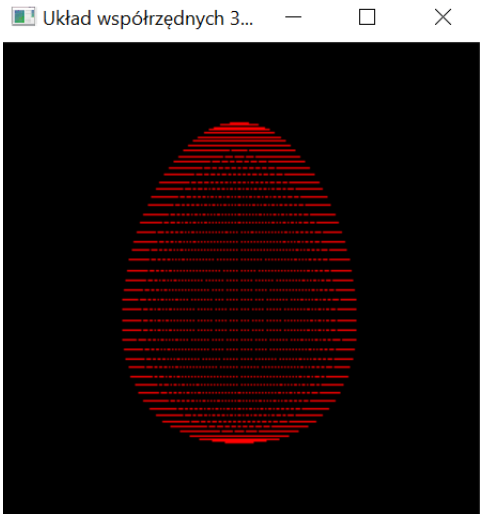
```
1 // Zadanie liczby N
2         int N = 100;
3 // Utworzenie tablicy punktow oraz trzy wspolrzedne x y z
4         float tab[100][100][3];
5         float u, v;
6         float x, y, z;
7         float a = 0.0;
8
```

```
9  // Funkcja obliczajaca wartosci wspolrzednych przy pomocy rownan
10 // parametrycznych i zapisujaca je w zadeklarowanej tablicy
11         for (int i = 0; i < 100; i++)
12         {
13                 for (int j = 0; j < 100; j++)
14                 {
15                 u = (float) i / 99.0;
16                 v = (float) j / 99.0;
17                 // x
18                 tab[i][j][0] = ((-90 * pow(u, 5) + 225 * pow(u, 4)
19                 - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u)
20                  * cos(v * M_PI));
21                 // y
22                 tab[i][j][1] = (160 * pow(u, 4) - 320 * pow(u, 3)
23                 + 160 * pow(u, 2)) - 5;
24                 // z
25                 tab[i][j][2] = ((-90 * pow(u, 5) + 225 * pow(u, 4)
26                 - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u)
27                 * sin(v * M_PI));
28                 }
29         }
```

3.3.1 Rysowanie jajka kropkami

Następnym krokiem było narysowanie jajka rysując punkty na utworzonej przestrzeni.

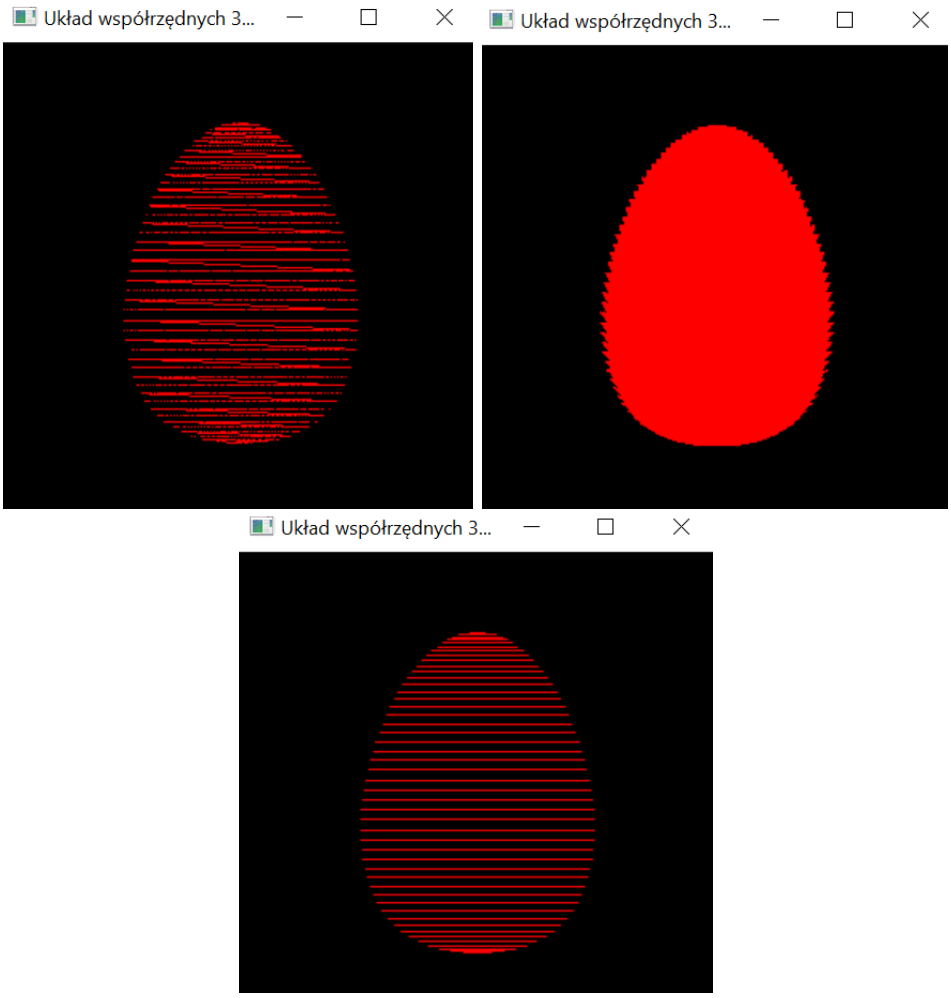
```
1  // Rysowanie jajka kropkami
2          glBegin(GL_POINTS);
3          for (int i = 0; i < 100; i++)
4          {
5                  for (int j = 0; j < 100; j++)
6                  {
7                  glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2]);
8                  }
9          }
10         glEnd();
```



3.3.2 Rysowanie jajka paskami

Następnie należało narysować jajko łącząc poszczególne punkty dzięki czemu stosując różne kombinacje połączeń możemy osiągnąć następujące efekty (poprzez odkomentowanie odpowiednich linii kodu i zakomentowanie innych).

```
1  glBegin(GL_LINES);
2    for (int i = 0; i < 100; i++)
3    {
4            for (int j = 0; j < 100; j++)
5            {
6                    if (j != 99)
7                    {
8                    // Kilka roznych kombinacji polaczen
9                    glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2]);
10                   glVertex3f(tab[i][j + 1][0], tab[i][j + 1][1], tab[i][j + 1]
11                   glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2]);
12                   //glVertex3f(tab[j][i][0], tab[j][i][1], tab[j][i][2]);
13                   }
14           }
15   }
16 glEnd();
```



4 Interakcja z użytkownikiem

Celem zajęć laboratoryjnych było utworzenie prostej interakcji z użytkownikiem polegającej na sterowaniu ruchem obiektu i obserwatora za pomocą myszki. Zostały również utworzone obiekty z rzucie perspektywicznym.

4.1 Rzutowanie perspektywiczne

W poprzednim zadaniu stosowaliśmy rzutowanie równoległe (rzut ortograficzny). W rzucie ortograficznym rzutnia, czyli płaszczyzna na której powstawał obraz, była równoległa do płaszczyzny tworzonej przez osie x i y, a proste rzutowania biegły równoległe do osi z. Ze względu na to, że proste rzutowania są równoległe do osi z, przesuwanie obiektu wzdłuż tej osi nie spowoduje żadnego efektu na obrazie. Aby umożliwić pokazanie efektów przemieszczeń obiektu we wszystkich osiach należy zastosować rzutowanie perspektywiczne. Rzut perspektywiczny jest lepszy od równoległego nie tylko ze względu na możliwość prezentacji przemieszczeń, pozwala także lepiej pokazać na płaszczyźnie geometrie trójwymiarowego obiektu.
Czajnik w tym wypadku wygląda tak:

4.2 Realizacja prostej interakcji

Do sterowania rzutowanym obiektem potrzebne są dwie funkcje, zdefiniowane następująco:

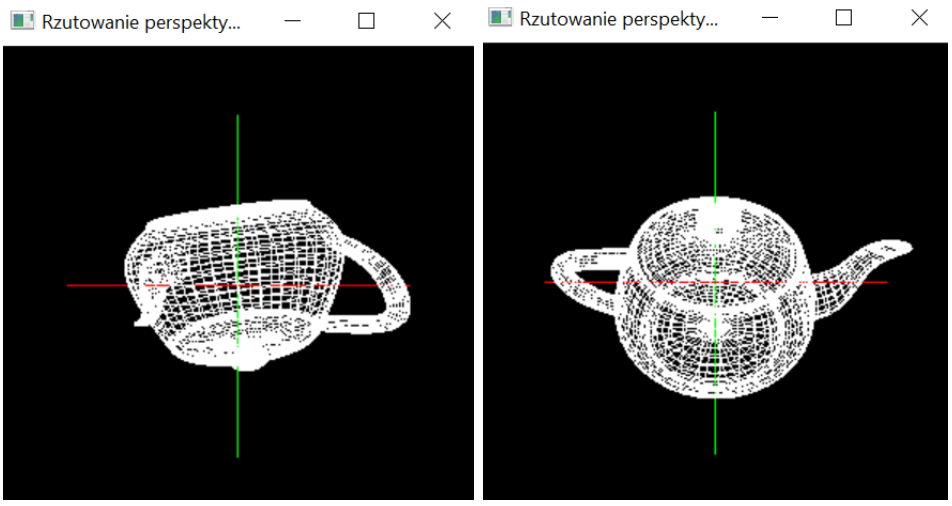
```
1  // Funkcja "bada" stan myszy i ustawia wartosci
2  // odpowiednich zmiennych globalnych
3  void Mouse(int btn, int state, int x, int y)
4  {
5          if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
6          {
7  // przypisanie aktualnie odczytanej pozycji kursora jako pozycji poprzedniej
8                  x_pos_old = x;
9                  y_pos_old = y;
10
11                 status = 1; // wcieniety zostal lewy klawisz myszy
12         }
13         else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
14         {
15                 status = 2;
16         }
17         else
18                 status = 0; // nie zostal wcisniety zaden klawisz
19 }
20
21 // Funkcja "monitoruje" polozenie kursora myszy i ustawia
22 // wartosci odpowiednich zmiennych globalnych
23 void Motion(GLsizei x, GLsizei y)
24 {
25         // obliczenie roznicy polozenia kursora myszy
26         delta_x = x - x_pos_old;
27         delta_y = y - y_pos_old;
```

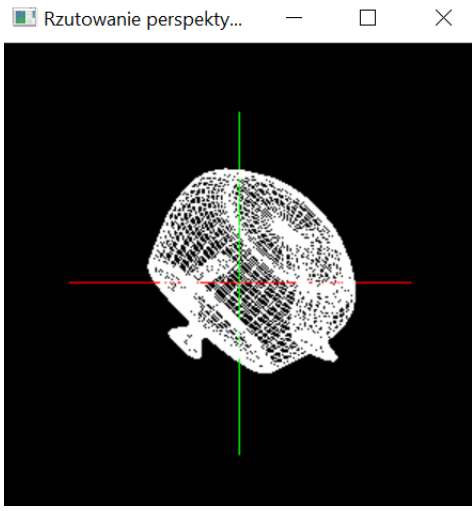
```
28
29          x_pos_old = x; // podstawienie biezacego polozenia jako poprzednie
30          y_pos_old = y;
31
32          glutPostRedisplay(); // przerysowanie obrazu sceny
33  }
```

Następnie musimy powyższe funkcje wywołać w funkcji RenderScene():

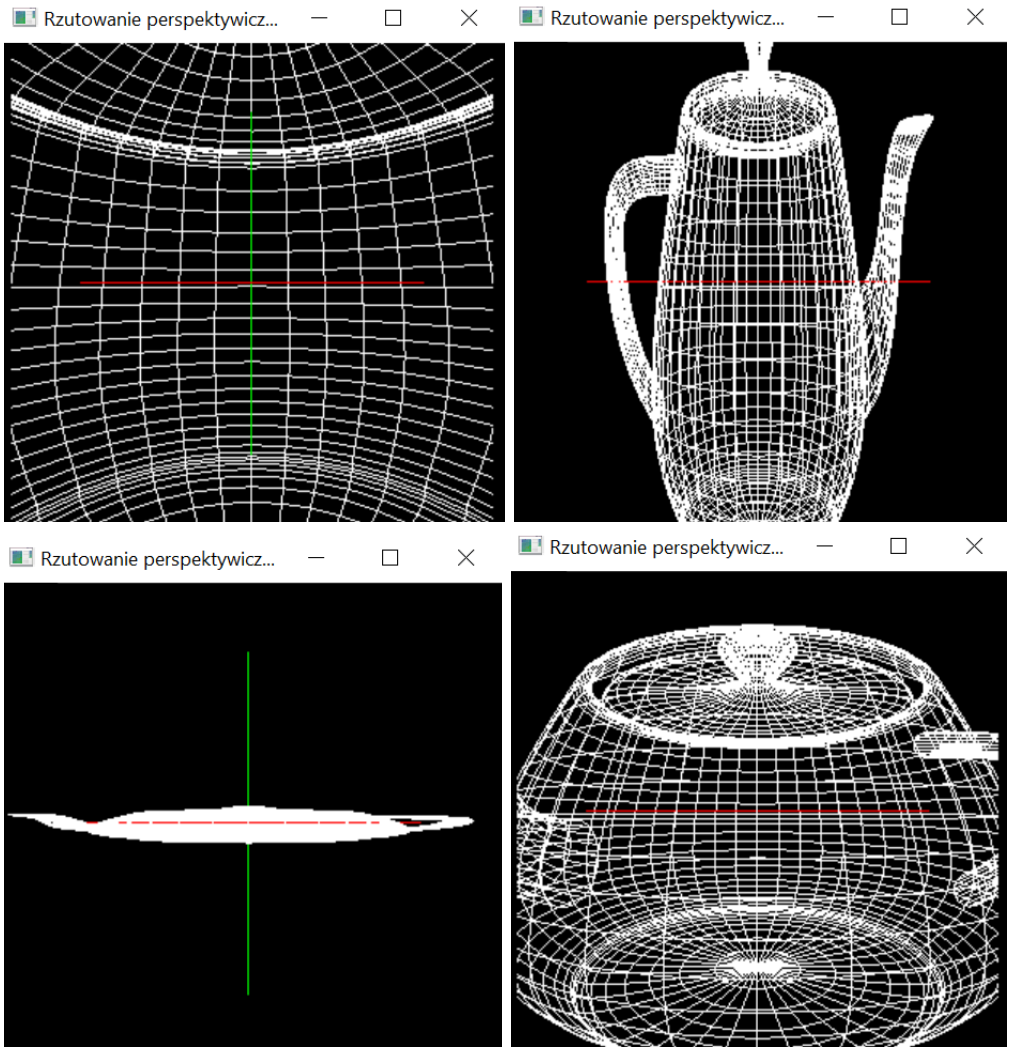
```
1  if (status == 1) // jesli lewy klawisz myszy wcieniety
2          {
3          // modyfikacja kata obrotu o kat proporcjonalny
4                  theta += delta_x * pix2angle;
5          // do roznicy polozen kursora myszy
6                  theta2 += delta_y * pix2angle;
7          //obrot obiektu o nowy kat
8                  glRotatef(theta, 0.0, 1.0, 0.0);
9                  glRotatef(theta2, 1.0, 0.0, 0.0);
10                 // drugi sposob skalowania obiektu
11                 //glScalef(theta * 0.05, theta2 * 0.05, theta * 0.05);
12         }
13         else if (status == 2) // jesli prawy klawisz wcisniety
14         {
15         // modyfikacja kata obrotu o kat proporcjonalny
16                 theta += delta_x * pix2angle;
17         // do roznicy polozen kursora myszy
18                 theta2 += delta_y * pix2angle;
19                 glScalef(theta * 0.05, theta2 * 0.05, theta * 0.05);
20         }
```

Dzięki powyższej implementacji klikając lewy przycisk myszy mamy możliwość obracania i oglądania czajnika z każdej możliwej strony. Przykładowe screeny obrotów:





Po kliknięciu prawego przycisku myszy, mamy możliwość przybliżania, oddalania oraz spłaszczania i rozciągania czajnika:



5 Oświetlenie scen

Celem zajęć laboratoryjnych było pokazanie możliwości oświetlenia obiektów na scenach 3D, a także zastosowanie ich w programie. Opisujmy własności materiału oświetlanego obiektu, definiujemy światło, dobieramy jego parametry oraz wyznaczamy równania wektorów normalnych do punktów powierzchni opisanej za pomocą równań parametrycznych.

5.1 Wprowadzenie na scenę 3D źródła światła

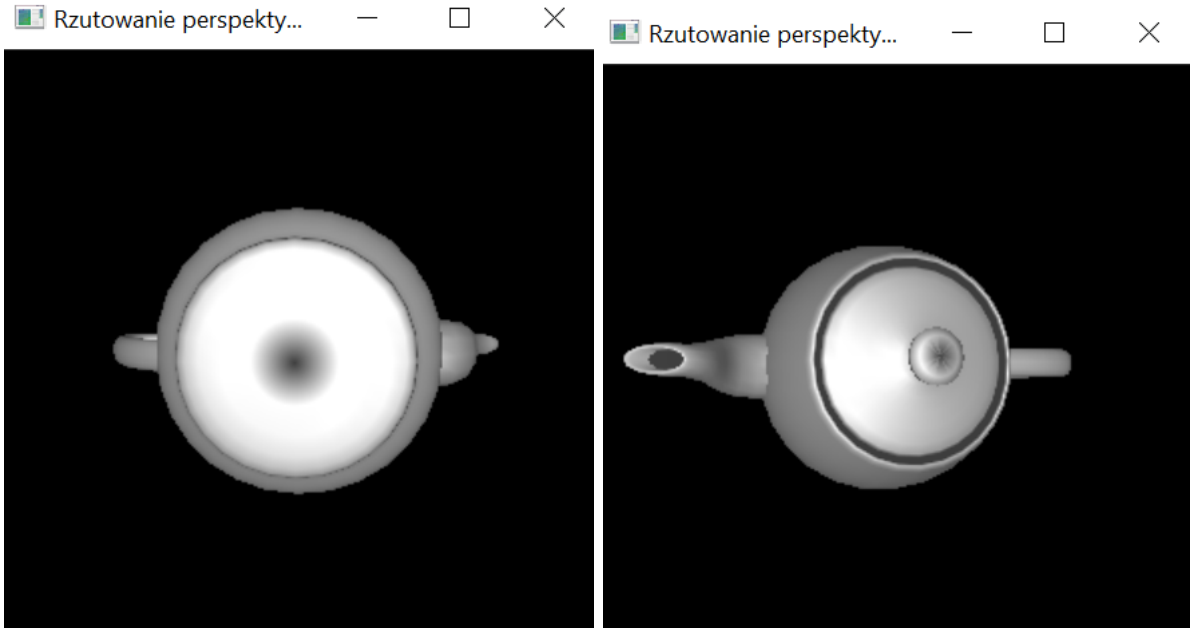
W tym miejscu ćwiczenia zmieniamy czajnik z poprzednich zadań na model złożony z wypełnionych wieloboków zamiast modelu szkieletowego funkcją glutSolidTeapot(). Następnym krokiem będzie dodanie źródła światła oświetlającego nasz czajnik. W tym celu używamy następującego kodu (dodanego w funkcji MyInit):

```
1  // Definicja materialu z jakiego zrobiony jest czajnik
2
3       GLfloat mat_ambient[]   = {1.0, 1.0, 1.0, 1.0};
4       // wspolczynniki ka =[kar,kag,kab] dla swiatla otoczenia
5
6       GLfloat mat_diffuse[]   = {1.0, 1.0, 1.0, 1.0};
7       // wspolczynniki kd =[kdr,kdg,kdb] swiatla rozproszonego
8
9       GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
10      // wspolczynniki ks =[ksr,ksg,ksb] dla swiatla odbitego
11
12      GLfloat mat_shininess   = {20.0};
13      // wspolczynnik n opisujacy polysk powierzchni
14
15 // Definicja zrodla swiatla
16
17       GLfloat light_position[] = {0.0, 0.0, 10.0, 1.0};
18      // polozenie zrodla
19
20      GLfloat light_ambient[] = {0.1, 0.1, 0.1, 1.0};
21      // skladowe intensywnosci swiecenia zrodla swiatla otoczenia
22      // Ia = [Iar,Iag,Iab]
23
24      GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
25      // skladowe intensywnosci swiecenia zrodla swiatla powodujacego
26      // odbicie dyfuzyjne Id = [Idr,Idg,Idb]
27
28      GLfloat light_specular[]= {1.0, 1.0, 1.0, 1.0};
29      // skladowe intensywnosci swiecenia zrodla swiatla powodujacego
30      // odbicie kierunkowe Is = [Isr,Isg,Isb]
31
```

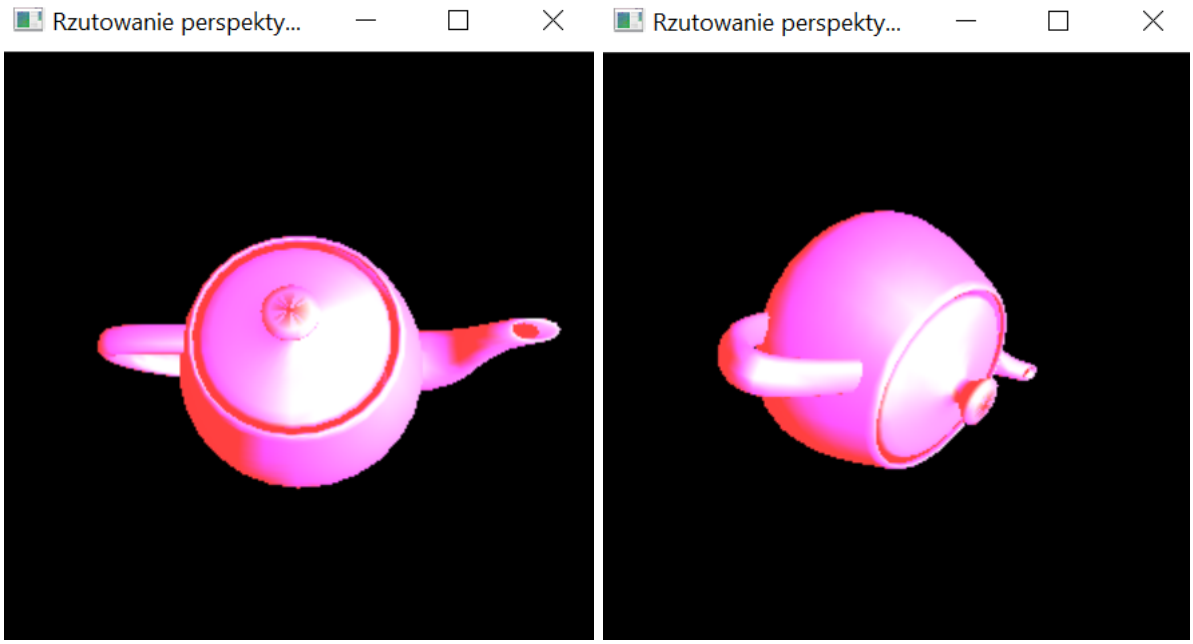
```
32      GLfloat att_constant  = {1.0};
33      // skladowa stala ds dla modelu zmian oswietlenia w funkcji
34      // odleglosci od zrodla
35
36      GLfloat att_linear    = {0.05};
37      // skladowa liniowa dl dla modelu zmian oswietlenia w funkcji
38      // odleglosci od zrodla
39
40      GLfloat att_quadratic  = {0.001};
41      // skladowa kwadratowa dq dla modelu zmian oswietlenia w funkcji
42      // odleglosci od zrodla
43
44  // Ustawienie patrametrow materialu
45
46      glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
47      glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
48      glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
49      glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
50
51  // Ustawienie parametrow zrodla
52
53      glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
54      glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
55      glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
56      glLightfv(GL_LIGHT0, GL_POSITION, light_position);
57
58      glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, att_constant);
59      glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, att_linear);
60      glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, att_quadratic);
61
62  // Ustawienie opcji systemu oswietlania sceny
63
64      glShadeModel(GL_SMOOTH); // wlaczenie lagodnego cieniowania
65      glEnable(GL_LIGHTING);   // wlaczenie systemu oswietlenia sceny
66      glEnable(GL_LIGHT0);     // wlaczenie zrodla o numerze 0
67      glEnable(GL_DEPTH_TEST); // wlaczenie mechanizmu z-bufora
```

Po dodaniu powyższych linii kodu model naszego czajnika wygląda następująco:

Obracając model czajnika możemy zauważyć, że jest on oświetlany cały czas w tym samym punkcie (w środku układu współrzędnych), ponieważ położenie naszego źródła światła się nie zmienia:



Zmieniając wartości z pierwszego listingu takie jak: światło otoczenia, światło odbite, światło rozproszone oraz współczynnik opisujący połysk powierzchni możemy zmieniać wygląd czajniczka:



5.2 Dodanie drugiego źródła światła

Dodanie drugiego źródła światła wygląda w taki sam sposób jak dodanie pierwszego z nich. Jedyną różnicą jest liczba "1źamiast "0"w definicji zmiennej określającej nowe światło. Świateł możemy dodawać wiele, Visual Studio podpowiada że może być ich aż 8. Poniższy kod definiuje drugie źródło światła:

```
1         GLfloat light_position2[] = { 0.0, 0.0, 10.0, 0.0 };
2         // polozenie drugiego zrodla swiatla
3
4     GLfloat light_ambient2[] = { 1.0, 1.0, 0.0, 1.0 };
5         // skladowe intensywnosci swiecenia drugiego zrodla swiatla otoczeni
6         // Ia = [Iar, Iag, Iab]
7
8         GLfloat light_diffuse2[] = { 1.0, 1.0, 0.0, 1.0 };
9         // skladowe intensywnosci swiecenia drugiego zrodla swiatla powoduja
10        // odbicie dyfuzyjne Id = [Idr, Idg, Idb]
11
12        GLfloat light_specular2[] = { 0.0, 0.0, 1.0, 1.0 };
13        // skladowe intensywnosci swiecenia drugiego zrodla swiatla powoduja
14        // odbicie kierunkowe Is = [Isr, Isg, Isb]
15
16        // ustawienie parametrow drugiego zrodla swiatla
17        // (tutaj mamy wczesniej wspomniane "1" zamiast "0")
18        glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient2);
19        glLightfv(GL_LIGHT1, GL_DIFFUSE, light_diffuse2);
20        glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular2);
```

```
21          glLightfv(GL_LIGHT1, GL_POSITION, light_position2);
22
23          glEnable(GL_LIGHT1);      // wlaczenie zrodla o numerze 1
```

6 Teksturowanie obiektów

Celem laboratoriów jest nauczenie się podstawowych technik teksturowania obiektów.

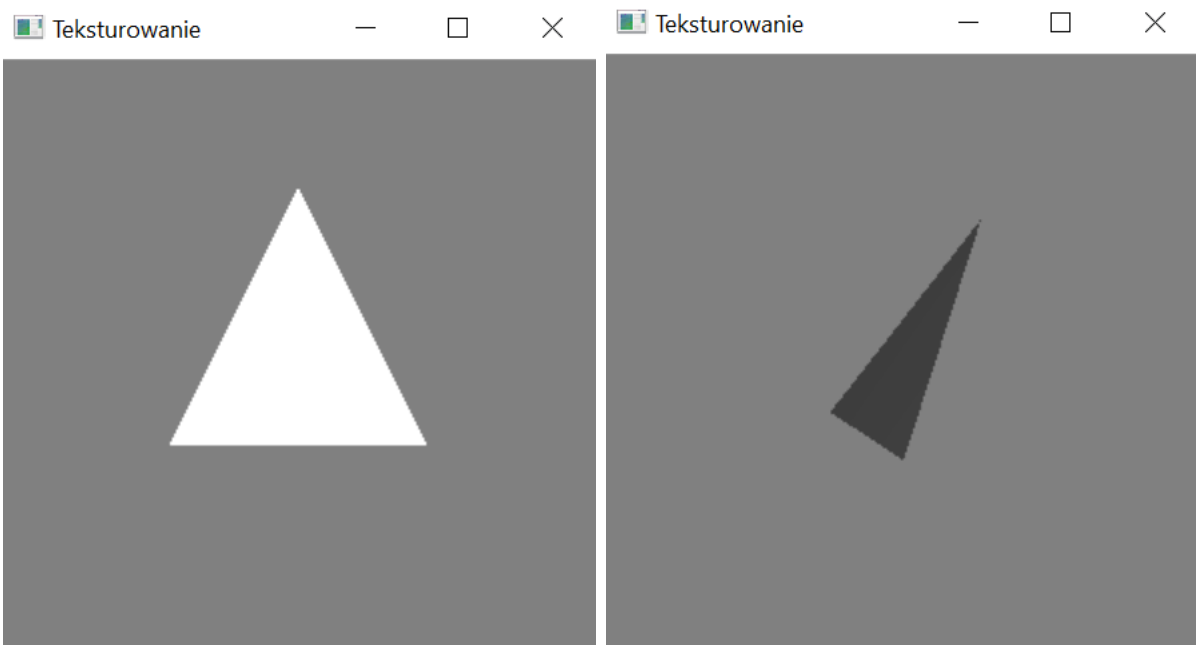
6.1 Teksturowanie trójkąta

W tym etapie ćwiczenia rysujemy (bazując na wcześniejszych laboratoriach) oświetlony trójkąt z możliwością obracania.

Kod funkcji rysującej powyżej opisany trójkąt:

```
1  glBegin(GL_TRIANGLES);
2    glNormal3f(0.0f, 0.0f, 3.0f); // Wektory normalne
3    glTexCoord2f(0.0f, 0.0f);     // Funkcja okreslajaca, ktore wierzcholki
4    glVertex3f(0.0f, 4.0f, 3.0f); // trojkata naniesionego na wzorzec tekstury
5    glNormal3f(0.0f, 0.0f, 3.0f); // odpowiadaja ktorym wierzcholkom
6    glTexCoord2f(3.0f, 0.0f);
7    glVertex3f(-4.0f, 0.0f, 0.0f);
8    glNormal3f(0.0f, 0.0f, 3.0f);
9    glTexCoord2f(1.5f, 2.0f);
10   glVertex3f(4.0f, 0.0f, 0.0f);
11 glEnd();
```

Po dodaniu powyższych linii kodu obraz wygląda następująco:



Następnym krokiem było dodanie do programu funkcji służącej do odczytywania tekstury z pliku graficznego w formacie TGA (targa). Kod tej funkcji:

```
1  // Funkcja wczytuje dane obrazu zapisanego w formacie TGA w pliku o nazwie
2  // FileName, alokuje pamiec i zwraca wskaznik (pBits) do bufora w ktorym
3  // umieszczone sa dane.
4  // Ponadto udostepnia szerokosc (ImWidth), wysokosc (ImHeight) obrazu
5  // tekstury oraz dane opisujace format obrazu wedlug specyfikacji OpenGL
6  // (ImComponents) i (ImFormat).
7  // Dziala tylko dla obrazow wykorzystujacych 8, 24, or 32 bitowy kolor.
8  GLbyte *LoadTGAImage(const char *FileName, GLint *ImWidth, GLint *ImHeight,
9  {
10 // Struktura dla naglowka pliku  TGA
11     #pragma pack(1)
12     typedef struct
13     {
14         GLbyte    idlength;
15         GLbyte    colormaptype;
16         GLbyte    datatypecode;
17         unsigned short    colormapstart;
18         unsigned short    colormaplength;
19         unsigned char     colormapdepth;
20         unsigned short    x_orgin;
21         unsigned short    y_orgin;
22         unsigned short    width;
```

```
23          unsigned short     height;
24          GLbyte     bitsperpixel;
25          GLbyte     descriptor;
26      }TGAHEADER;
27      #pragma pack(8)
28      FILE *pFile;
29      TGAHEADER tgaHeader;
30      unsigned long lImageSize;
31      short sDepth;
32      GLbyte     *pbitsperpixel = NULL;
33
34  // Wartosci domyslne zwracane w przypadku bledu
35      *ImWidth = 0;
36      *ImHeight = 0;
37      *ImFormat = GL_BGR_EXT;
38      *ImComponents = GL_RGB8;
39
40      pFile = fopen(FileName, "rb");
41      if(pFile == NULL)
42        return NULL;
43
44  // Przeczytanie naglowka pliku
45      fread(&tgaHeader, sizeof(TGAHEADER), 1, pFile);
46
47  // Odczytanie szerokosci, wysokosci i glebi obrazu
48      *ImWidth = tgaHeader.width;
49      *ImHeight = tgaHeader.height;
50      sDepth = tgaHeader.bitsperpixel / 8;
51
52  // Sprawdzenie, czy glebia spelnia zalozone warunki (8, 24, lub 32 bity)
53      if(tgaHeader.bitsperpixel != 8 && tgaHeader.bitsperpixel != 24 && tgaHea
54          return NULL;
55
56  // Obliczenie rozmiaru bufora w pamieci
57      lImageSize = tgaHeader.width * tgaHeader.height * sDepth;
58
59  // Alokacja pamieci dla danych obrazu
60        pbitsperpixel = (GLbyte*)malloc(lImageSize * sizeof(GLbyte));
61
62        if(pbitsperpixel == NULL)
63          return NULL;
64
65      if(fread(pbitsperpixel, lImageSize, 1, pFile) != 1)
66          {
67          free(pbitsperpixel);
```

```
68          return NULL;
69          }
70
71  // Ustawienie formatu OpenGL
72      switch(sDepth)
73          {
74          case 3:
75              *ImFormat = GL_BGR_EXT;
76              *ImComponents = GL_RGB8;
77              break;
78
79          case 4:
80              *ImFormat = GL_BGRA_EXT;
81              *ImComponents = GL_RGBA8;
82              break;
83
84          case 1:
85              *ImFormat = GL_LUMINANCE;
86              *ImComponents = GL_LUMINANCE8;
87              break;
88          };
89      fclose(pFile);
90      return pbitsperpixel;
91  }
```

Następnie w funkcji MyInit() zostały dopisane funkcji definiujące właściwości procesu teksturowania istotne dla mechanizmu renderowania OpenGL.

```
1  // Zmienne dla obrazu tekstury
2   GLbyte *pBytes;
3   GLint ImWidth, ImHeight, ImComponents;
4   GLenum ImFormat;
5
6  // Teksturowanie bedzie prowadzone tylko po jednej stronie sciany
7   glEnable(GL_CULL_FACE);
8
9  //  Przeczytanie obrazu tekstury z pliku o nazwie tekstura.tga
10  pBytes = LoadTGAImage("tekstura.tga", &ImWidth, &ImHeight, &ImComponents, &
11
12 // Zdefiniowanie tekstury 2-D
13  glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat
14
15 // Zwolnienie pamieci
16  free(pBytes);
17
18 // Wlaczenie mechanizmu teksturowania
```

```
19   glEnable(GL_TEXTURE_2D);
20
21  // Ustalenie trybu teksturowania
22   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
23
24  // Okreslenie sposobu nakladania tekstur
25   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
26   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

Efektem dodania powyższych linii kodu jest narysowany wcześniej trójkąt z nałożoną teksturą (pobraną z pliku dostępnego w instrukcji).