Ćwiczenie 3

OpenGL - modelowanie obiektów 3-D

1. Cel ćwiczenia

Celem ćwiczenia jest wprowadzenie w zagadnienia modelowania i wizualizacji scen 3D z wykorzystaniem biblioteki OpenGL z rozszerzeniem **GLUT**. Zamieszczone w treści opisu ćwiczenia przykłady pokazują jak w układzie współrzędnych trójwymiarowych wykonuje się transformacje obiektów oraz jak na podstawie równań parametrycznych można stworzyć własny model nietrywialnego obiektu. Przy okazji pokazano także sposób sterowania wykonaniem programu przy pomocy klawiatury oraz prosty przykład z zakresu animacji.

2. Układ współrzędnych i obiekt 3D

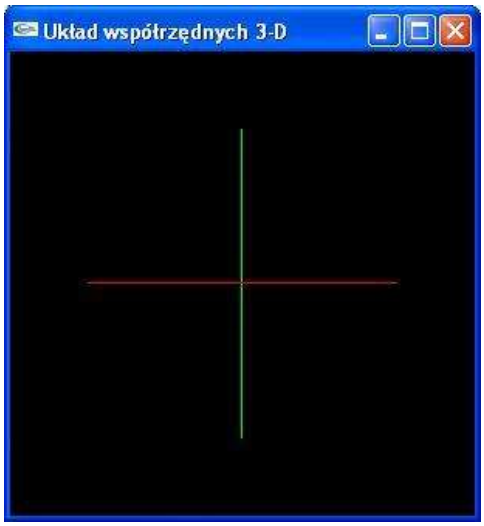
W środowisku programistycznym MSVisual Studio utworzyć (w ten sam sposób jak w poprzednim ćwiczeniu) nowy projekt i w pliku źródłowym umieścić kod zamieszczony niżej. Program tworzy okno graficzne i rysuje obrazy osi układu współrzędnych 3-D. Rysunek osi może być pomocny w wielu przypadkach tworzenia scen 3-D, zwłaszcza w początkowej fazie zaznajamiania się z biblioteką OpenGL. Obraz osi **x** rysowany jest kolorem czerwonyn, osi **y** zielonym a osi **z** niebieskim. Ze względu na fakt, że w przykładzie wykorzystano rzutowanie ortograficzne (proste rzutowania przecinają rzutnię pod kątem prostym) obraz osi **z** nie jest widoczny.

```
/*************************************************************************************/
//  Szkielet programu do tworzenia modelu sceny 3-D z wizualizacją osi
//  układu współrzednych
/*************************************************************************************/
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>

typedef float point3[3];
/*************************************************************************************/
// 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);
```

```
      // Czyszczenie okna aktualnym kolorem czyszczącym

      glLoadIdentity();
      // Czyszczenie macierzy bieżącej


      Axes();
      // Narysowanie osi przy pomocy funkcji zdefiniowanej wyżej

      glFlush();
      // Przekazanie poleceń rysujących do wykonania



      glutSwapBuffers();
      //

 }

/*************************************************************************************/
// 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(void)
{

      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |GLUT_DEPTH);

      glutInitWindowSize(300, 300);

      glutCreateWindow("Układ współrzędnych 3-D");

      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

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

Efekt działania programu pokazany został na rysunku 1.



Rys. 1 Układ współrzędnych w 3-D (rzutowanie ortograficzne)

Uzyskany obraz nie jest może zbyt ciekawy, ale można go łatwo wzbogacić wprowadzając w funkcji `RenderScene()`, po linii z wywołaniem funkcji rysującej osie `Axes()` dwie nowe linie kodu:

```
glColor3f(1.0f, 1.0f, 1.0f); // Ustawienie koloru rysowania na biały
glutWireTeapot(3.0); // Narysowanie obrazu czajnika do herbaty
```

Po uruchomieniu zmodyfikowanego programu otrzymuje się obraz taki jak na rysunku 2.

Rys. 2 Czajnik do herbaty w położeniu początkowym

Funkcja `glutWireTeapot()` jest jedną z ponad dwudziestu funkcji biblioteki *GLUT* pozwalających na łatwe rysowanie obiektów 3D. Jest to funkcja dość szczególna, bowiem czajnik jako obiekt zdecydowanie asymetryczny nadaje się bardzo dobrze do śledzenia rezultatów zastosowanych transformacji. Pokazany na rysunku 2 czajnik występuje w literaturze grafiki komputerowej bardzo często Jest to jeden z najbardziej popularnych i lubianych przez Autorów przykładów. Inne obiekty, jakie oferuje biblioteka *GLUT* to:

glutSolidCone (GLdouble base, GLdouble height, GLint slices, GLint stacks)
glutSolidCube (GLdouble width)
glutSolidCylinder (GLdouble radius, GLdouble height, GLint slices, GLint stacks)
glutSolidDodecahedron (void)
glutSolidIcosahedron (void)
glutSolidOctahedron (void)
glutSolidRhombicDodecahedron (void)
glutSolidSierpinskiSponge (int num_levels, const GLdouble offset[3], GLdouble scale)
glutSolidSphere (GLdouble radius, GLint slices, GLint stacks)
glutSolidTeapot (GLdouble size)
glutSolidTetrahedron (void)
glutSolidTorus (GLdouble dInnerRadius, GLdouble dOuterRadius, GLint nSides, GLint nRings)
glutWireCone (GLdouble base, GLdouble height, GLint slices, GLint stacks)

glutWireCone (GLdouble base, GLdouble height, GLint slices, GLint stacks)
glutWireCube (GLdouble width)
glutWireCylinder (GLdouble radius, GLdouble height, GLint slices, GLint stacks)
glutWireDodecahedron (void)
glutWireIcosahedron (void)
glutWireOctahedron (void)
glutWireRhombicDodecahedron (void)
glutWireSierpinskiSponge (int num_levels, const GLdouble offset[3], GLdouble scale)
glutWireSphere (GLdouble radius, GLint slices, GLint stacks)
glutWireTeapot (GLdouble size)
glutWireTetrahedron (void)
glutWireTorus (GLdouble dInnerRadius, GLdouble dOuterRadius, GLint nSides, GLint nRings)

3. Transformacje w przestrzeni 3-D

Przemieszczania obiektu w przestrzeni 3D realizuje się używając formalnego aparatu transformacji geometrycznych. Najwygodniejszą formą przedstawienia transformacji jest macierz, w związku z tym wszelkie operacje związane z transformacjami w bibliotece OpenGL wykonuje się stosując operacje rachunku macierzowego. Dokładniejsze wyjaśnienie zagadnień związanych z ideą zastosowania macierzy i ich implementacją w OpenGL znaleźć można w literaturze i dokumentacjach biblioteki. Dla potrzeb niniejszego ćwiczenia podanych zostanie jedynie tylko kilka funkcji pozwalających w prosty sposób manipulować położeniem obiektów. Funkcjami tymi są:

```
glMatrixMode(GLenum mode);E
glLoadIdenty(void);
glTranslated(TYP x, TYPE y, TYPE z);
glScaled(TYPE x, TYPE y, TYPE z);
glRotated(TYPE angle, TYPE x, TYPE y, TYPE z,);
```

Dwie pierwsze funkcje `glMatrixMode()` i `glLoadIdenty()` nie ralizują konkretnych tansformacji, pełnią natomiast ważną rolę organizacyjną w procesie przetwarzania danych geometrycznych sceny.

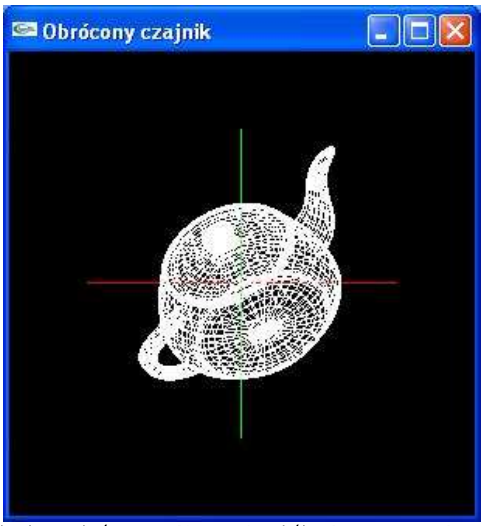
Funkcja `glMatrixMode()` ustala która z macierzy stanowiących elementy ciągu procedury wizualizacji będzie tak zwaną macierzą bieżącą, czyli macierzą, która będzie dalej modyfikowna aż do następnej ewentualnej zmiany macierzy bieżącej. Moliwe są trzy wartości argumentu funkcji: `GL_MODELVIEW`, `GL_PROJECTION` i `GL_TEXTURE`. Jeśli argumentem bedzie `GL_MODELVIEW` modyfikowana będzie macierz widoku modelu (odpowiedzialana za przeliczanie geometrii modelu). W przypadku argumentu `GL_PROJECTION` nastepne operacje macierzowe dotyczyć bedą macierzy projekcji (odpowiedzialnej za rzutowanie). Trzeci przypadek `GL_TEXTURE` dotyczy tekstury i zostanie omówiony przy okazji innego ćwiczenia.
Funkcja `glLoadIdenty()` "czyści" macierz bieżącą (ustawioną przez funkcję `glMatrixMode()`. Formalnie sprowadza się to do ustawienia macierzy bieżącej jako macierzy jednostkowej.

Kolejne dwie funkcje `glTranslated()` i `glScaled()` służą do wykonywania translacji i skalowania natoniast funkcja `glRotated(angle, x, y, z)` relaizuje obrót o kąt angle wokół osi wyznacznej przez punkt (0, 0, 0) czyli środek układu współrzędnych i punkt (x, y, z).

Dla przykładu obrót czajnika o 60 stopni wokół osi wyznaczonej przez punkty (0, 0, 0) i (1, 1, 1) realizuje następujący fragment kodu:

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

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

Efekt operacji transformacji obrotu ilustruje rysunek 3



Rys. 3 Czajnik do herbaty obrócony o 60 stopni wokół osi wyznaczonej przez wektor [1 1 1]

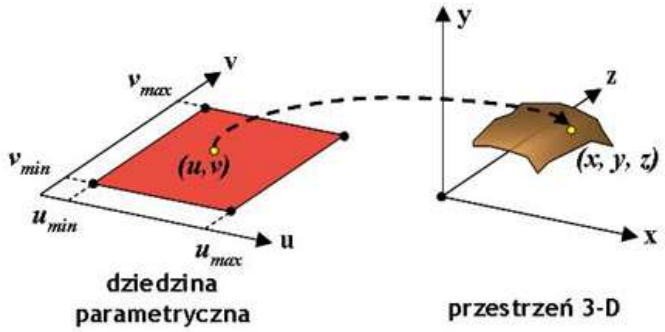
Biblioteka *OpenGL* oferuje jescze szereg innych funkcji pozwalających w szcególności na definowanie własnych macierzy, wykonywanie operacji macierzowych, składania przekształceń na stos i pobieranie ich ze stosu oraz innych. Szczegóły można znaleźć w dokumentacji.

4. Budowa własnego modelu obiektu 3-D

Modelowanym obiektem będzie jajko określone jako powierzchnia opisana równaniami parmetrycznymi. Na wykładzie pokazano jak przy pomocy obrócenia odpowiednio dobranej krzywej Beziera można uzyskać wzory opisujące powierzchnię jajka. Wzory te wygladają następująco:

$$x(u,v) = (-90u^5 + 225u^4 - 270u^3 + 180u^2 - 45u)\cos(\pi v)$$
$$y(u,v) = 160u^4 - 320u^3 + 160u^2$$
$$z(u,v) = (-90u^5 + 225u^4 - 270u^3 + 180u^2 - 45u)\sin(\pi v)$$
$$0 \le u \le 1$$
$$0 \le v \le 1$$

Rysunek 4 ilustruje ideę budowy powierzchni opisanej równaniami parametrycznymi.



Rys. 4 Przekształcenie dziedziny parametrycznej w powierzchnię w przestrzeni 3D

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. Do przeliczenia będą służyły wyżej podane równania.

Szkic algorytmu można zapisać tak:

1. Zadać liczbę **N**, która określała będzie na ile przedziałów podzielony zostanie bok kwadratu jednostkowego dziedziny parametrycznej.

2. Zadeklarować tablicę o rozmiarze **NxN**, która będzie służyła do zapisywania współrzednych punktów w przestrzeni 3-D. Każdy element tablicy zawierał będzie trzy liczby będące współrzędnymi `x`, `y`, `z` jednego punktu.

3. Nałożyć na kwadrat jednostkowy dziedziny parametrycznej równomierną siatkę **NxN** punktów.

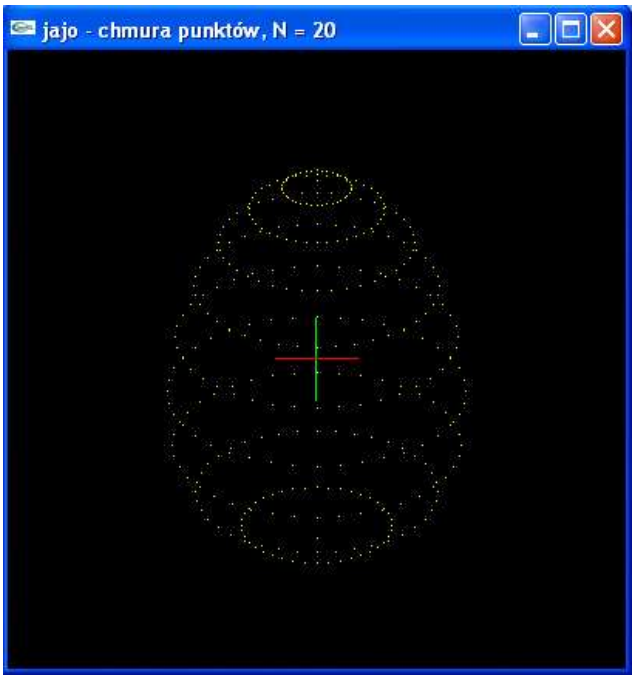
4. Dla każdego punktu `u`, `v` nałożonej w kroku poprzednim siatki, obliczyć, przy pomocy podanych wyżej równań współrzędne `x(u, v)`, `y(u, v)` i `z(u, v)` i zapisać je w zadeklarowanej w kroku 2 tablicy.

5. Wyświetlić na ekranie elementy tablicy współrzędnych punktów posługując się konstrukcją:

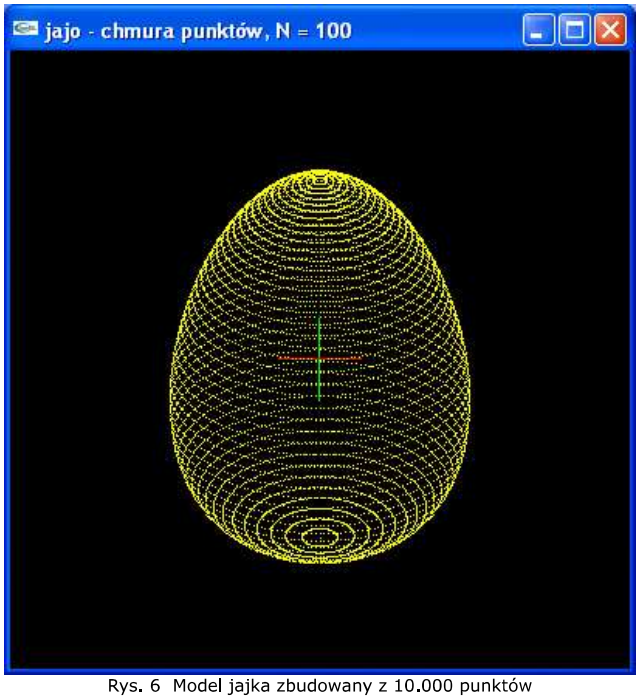
```
glBegin(GL_POINTS);
...
//wierzchołki odpowiadające punktom z tablicy współrzędnych punktów
...
glEnd();
```

Program należy napisać tak, aby rysowanie obrazu jajka następowało po wywołaniu odpowiedniej funkcji nazwanej na przykład `Egg()`. Funkcja ta powinna być wywoływana wewnątrz funkcij rysującej scenę `RenderScene()`.

Uzyskany na ekranie obraz ma wyglądać mniej więcej tak jak pokazane to zostało na rysunkach 5 i 6. Dla lepszej prezentacji, model przesunięto wzdłuż osi `y` tak, by środek jajka zanlazł się w środku układu współrzednych i lekko obrócono wokół osi `x`.



Rys. 5 Model jajka zbudowany z 400 punktów



Rys. 6 Model jajka zbudowany z 10.000 punktów

5. Modyfikacja modelu obiektu

Kolejne zadanie będzie polegało na wykorzystaniu wygenerowanej w poprzednim punkcie chmury punktów do budowy modelu w postaci siatki linii i modelu zbudowanego z trójkątów. Należy także zapewnić możliwość przełącznia pomiędzy widokami poszczególnych modeli, przy pomocy klawiszy. Naciśniecie klawisza "**p**" powinno spowodować wyświetlenie chmury punktów, klawisza "**w**" modelu w postaci siatki a "**s**", modelu złożonego z wypełnionych trójkątów.

Wprowadzenie do programu obsługi klawiatury jest pierwszym elementem interakcji z użytkownikiem. Bardziej rozbudowane sposoby z wykorzystaniem myszy będą przedmiotem kolejnego ćwiczenia.

Aby umożliwić sterowanie wykonaniem programu przy pomocy klawiatury należy poprzednio napisany program uzupełnić o następujące elementy:

1. Zadeklarować zmienną globalną służącą do przechowywania inforamcji o tym, który model ma być wyświetlany np.

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

2. Dodać do dotychczasowego kodu funkcję funkcję zwrotną (callback function)

```
void keys(unsigned char key, int x, int y)
{
    if(key == 'p') model = 1;
```

```
    if(key == 'p') model = 1;
    if(key == 'w') model = 2;
    if(key == 's') model = 3;

    RenderScene(); // przerysowanie obrazu sceny

}
```

3. W funkcj `main()` dodać funkcję

   ```
   glutKeyboardFunc(keys)
   ```

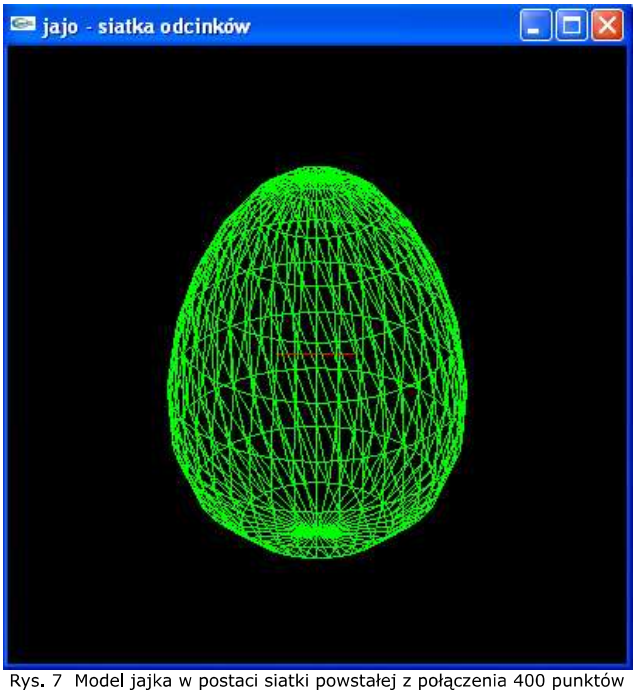
   Funkcja ta rejestruje funkcję zwrotną wywoływaną przez biblitekę *GLUT*. W analizowanym przykładzie jest to określona wyżej funkcja `keys()`. Po każdym nacisnięciu klawisza podawany jest jego jest kod ASCII (zmienna `key`) i dodatkowo pozycja kursora myszy (zmienne `x` i `y`).

4. W funkcji definiującej obiekt, uzależnić sposób rysowania od wartości zmiennej `model`

Rysowanie modelu obiektu jako siatki, polega na odpowiednim połączeniu odcinkami punktów, znajdujących sie w tablicy punktów, która została wygenerowana w poprzednim punkcie. W kodzie rysowania siatki nalezy więc zastosować konstrucję:

```
glBegin(GL_LINES);
...
//wierzchołki odpowiadające punktom z tablicy współrzędnych punktów
//pobierane z tablicy w odpowiedniej kolejności
...
glEnd();
```

Siatka powinna wyglądac mniej wiecej tak jak na pokazane to zosttało na rysunku 7.



Rys. 7 Model jajka w postaci siatki powstałej z połączenia 400 punktów

Narysowanie obiektu w postaci zbioru wypełnionych trójkątów jest trochę trudniejsze wymaga bowiem okreslenia koloru wypełniania poszczególnych trójkątów. Może być to oczywiście jeden kolor ustawiony przed rysowaniem, jedank w takim przypadku rysunek okaże się niezbyt ciekawy, bowiem na ekranie pojawi się tylko jednolicie wypełniony ustawinym wcześniej kolorem obszar. Aby uzyskać lepszy efekt można zaproponować następujące rozwiązanie:

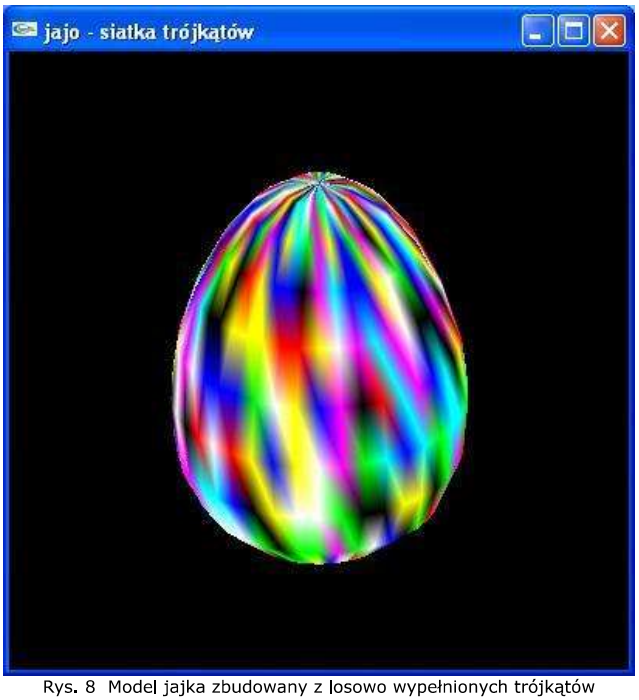
1. Podobnie jak przy tworzeniu modelu w postaci chmury punktów, zadeklarować drugą tablicę (tablicę kolorów punktów), także o rozmiarze **NxN**, która bedzie służyła do zapisywania tym razem kolorów dla poszczególnych punktów siatki. Każdy element tablicy zawierał bedzie trzy liczby z przedziału `[0, 1]` będace odpowiednio składowymi **R**, **G** i **B** koloru punktu.

2. Przed rysowaniem, najlepiej zaraz po staracie programu, wypełnić tablicę losowymi wartościami , które będą w niej dalej przechowywane i nie będą już zmieniane.

3. W funkcji rysującej obraz modelu w postaci wypełnionych trójkątów zastosować konstrukcję:

   ```
   glBegin(GL_TRIANGLES);
       ...
      //kolor następnego wierzchołka pobrany z tablicy kolorów punktów
       //wierzchołek trójkąta pobrany z tablicy współrzędnych punktów(wierzchołki należy pobierac
   w odpowiedniej kolejności)
       ...
   glEnd();
   ```

   Jeśli kolejność wierzchołków poszczególnych trójkątów zostanie prawidłowo ustalona na ekranie powinien pokazać się obraz podobny do tego jaki pokazano na rysunku 8. Płynne przejścia pomiedzy poszczególnymi trójkątami powstały dlatego, że wierzchołki przyległych trójkątów mają przypisany ten sam kolor.



Rys. 8 Model jajka zbudowany z losowo wypełnionych trójkątów

6. Wprawienie obiektu w ruch

Wrażenie ruchu można uzyskać przez odpowiednio szybkie modyfikownie położenia obiektu i wyświetlanie kolejnch obrazów pokazujących go w aktualnym położeniu. W bibliotece **GLUT** dostępny jest mechanizm pozwalający na stosunkowo łatwe programowaie ruchu obiektu na scenie. Zastosowanie go jest podobne do pokazanego już poprzednio sposobu obsługi klawiatury. Problem najlepiej zilustrować na przykładzie. Należy w nim zmodyfikować dotyczczas napisany kod programu tak, aby model jajka obracał się stale o niewielki kąt wokół każdej z osi ukladu współrzędnych. Modyfikacji można dokonać tak:

1. Zadeklarować zmienną globalną służącą do przechowywania inforamcji o aktualnym kącie obrotu modelu np.

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

   `theta[0]` będzie kątem obrotu wokół osi `x`, `theta[1]` wokół osi `y` a `theta[2]` wokół osi **z**.

2. Dodać do dotychczasowego kodu funkcję funkcję zwrotną (callback function) na przykład taką:

   ```
   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;

       glutPostRedisplay(); //odświeżenie zawartości aktualnego okna
   }
   ```

   W funkcji `spinEgg()` użyto jeszcze jednej funkcji z bibliteki *GLUT*. Funkcja `glutPostRedisplay()`, która przy każdorazowym wywołaniu spowoduje odświeżenie zawartości aktualnego okna graficznego.

3. W funkcj `main()` dodać funkcję

   ```
   glutIdleFunc(spinEgg);
   ```

   Funkcja ta podobnie jak przy obsłudze klawiatury rejestruje funkcję zwrotną wywoływaną przez biblitekę *GLUT*. Jest to wyżej określona funkcja `spinEgg()` modyfikująca kąty obrotu wokół osi.

4. W funkcji rysującej **`RenderScene()`** przed wywołaniem funkcji rysującej obiekt, czyli funkcji `Egg()`, dodać trzy liniie kodu:

   ```
   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);
   ```

   spowoduje to obrócenie modelu wokół osi `x`, `y`, `z` o kąty aktualnie wyliczone w funkcji `spinEgg()`.

W efekcie po uruchomieniu programu model jajka powinien zacząć się obracać.

Zadania domowe

Ćwiczenie opracował: Jacek Jarnicki, korekty Marek Woda (2016-10-26)

Back