Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Grafika komputerowa i komunikacja człowiek-komputer			
Temat:	Laboratorium 3 – Modelowanie obiektów 3D		
Termin zajęć:	środa TN 14:15 – 17:15 semestr zimowy 2023/2024		
Autor:	Eryk Mika 264451	**Prowadzący:**	Dr inż. arch. Tomasz Zamojski

1. Kod wspólny dla wszystkich zadań

Zadania realizowane w trakcie tego laboratorium różniły się od zadań z poprzedniego laboratorium tym, że wprowadzony został trzeci wymiar – tworzone są modele 3D.

Pierwszym krokiem jest wygenerowanie tablicy współrzędnych punktów w przestrzeni 3D, które są przeniesione z powierzchni parametrycznej o dziedzinach *u* i *v*. Rzutowanie to zostało wykonane według wzorów dostarczonych w instrukcji od Prowadzącego.

```
11  N = 50
12
13  tab = [[[0] * 3 for i in range(N)] for j in range(N)]
14
15  # Tablice wartosci parametrow u i v
16  u, v = [], []
17
18  # Wyznaczamy n-elementowe tablice wartosci dla parametrow u i v
19  for i in range(N):
20      u.append(i/(N-1))
21      v.append(i/(N-1))
22
23  # Obliczamy wartosci x, y, z
24  for i in range(N):
25      for j in range(N):
26          tab[i][j][0] =  (-90 * u[i]**5 + 225 * u[i]**4 - 270 * u[i]**3 + 180 * u[i]*u[i] - 45*u[i]) * cos(pi * v[j])
27          tab[i][j][1] = 160 * u[i] ** 4 - 320 * u[i] ** 3 + 160 * u[i] * u[i] - 5
28          tab[i][j][2] =  (-90 * u[i]**5 + 225 * u[i]**4 - 270 * u[i]**3 + 180 * u[i]*u[i] - 45*u[i]) * sin(pi * v[j])
29
```

Rysunek 1.1

Algorytm tworzenia punktów 3D jest określony w następujący sposób. Przyjmujemy liczbę punktów w przestrzeni 3D – *N*. Następnie tworzona jest tablica (dokładniej w języku Python – lista) o wymiarach *N x N x 3* oraz dwie tablice *u* i *v*, które przechowują kolejne N wartości z dziedziny *u* i *v*, z których pierwsze są równe 0, a ostatnie 1. Ostatecznie przechodzimy po wszystkich parach wartości *u* i *v* w zagnieżdżonej pętli *for*, generując dla każdej z par współrzędne x, y, z odpowiadających punktów w przestrzeni 3D – odpowiednio elementy tablicy *[i][j][0]*, *[i][j][1]*, *[i][j][2]*.

Najistotniejsze fragmenty kodu odpowiedzialne za realizację rozwiązań kolejnych zadań są zawarte w funkcji *render()*.

```
89  def render(time):
90      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
91      glLoadIdentity()
92
93      spin(time * 180 / 3.1415)
94
95      axes()
```

Rysunek 1.2

Kolejne linie tej funkcji są odpowiedzialne za obsługę buforów oraz przywrócenie stanu macierzy do macierzy jednostkowej[1]. Następnie wywoływana jest funkcja *spin*() (pochodząca z materiałów Prowadzącego), która powoduje obracanie się generowanego modelu i umożliwia jego lepszą obserwację. Kolejno wywoływana jest funkcja *axes()*, która generuje osie.

2. Zadanie na ocenę 3.0 – skrypt *lab3_0.py*

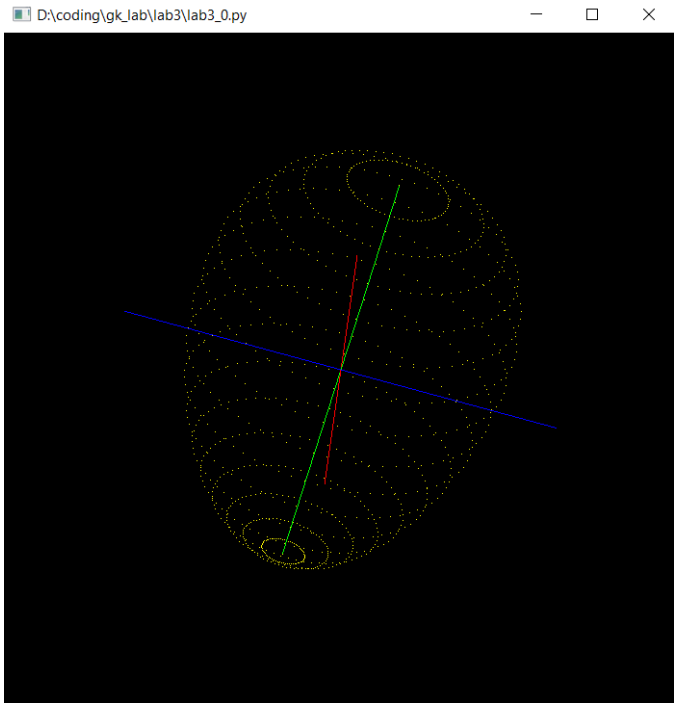
W zadaniu tym należało narysować model jajka przy pomocy punktów opisanych w poprzednim punkcie sprawozdania.

Właściwy algorytm (linia 73) zaczyna się ustawieniem koloru rysowania na żółty. Następnie w zagnieżdżonej pętli for przechodzimy po wszystkich elementach tablicy punktów 3D (każdej parze wartości *u* i *v*) i rysujemy je – najpierw poprzez wywołanie funkcji *glBegin(GL_POINTS)*, podanie współrzędnych do funkcji *glVertex3f()* oraz zakończenie wywołaniem *glEnd()*. Funkcja *glFlush()* powoduje wyświetlenie modelu. Poniżej przedstawiono efekt działania skryptu dla N = 35 (Rysunek 2.2).

```
73      glColor3f(1, 0.984, 0)
74
75      for i in range(N):
76          for j in range(N):
77              glBegin(GL_POINTS)
78              glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2])
79              glEnd()
80      glFlush()
81
```

Rysunek 2.1

[1] https://docs.gl/gl3/glLoadIdentity



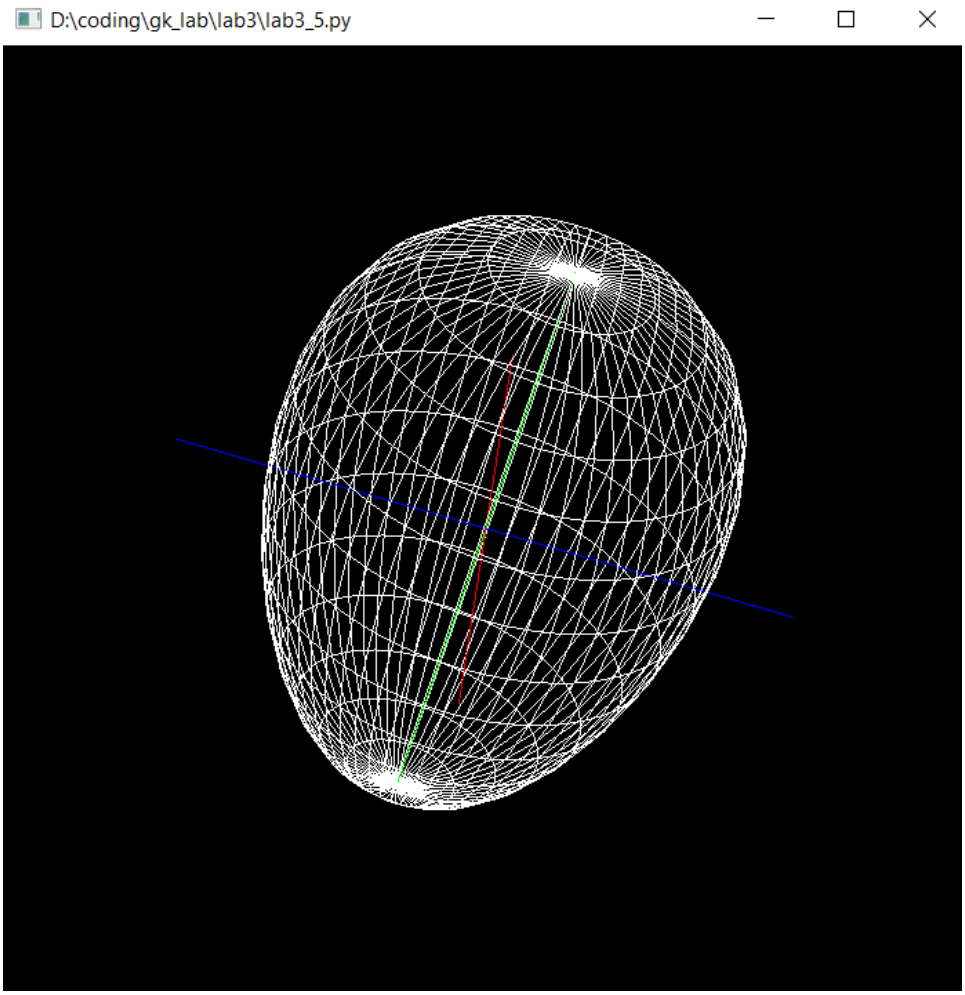
Rysunek 2.2

3. Zadanie na ocenę 3.5 – skrypt *lab3_5.py*

```
75      glColor3f(1, 1, 1)
76
77      # Rysujemy linie
78      for i in range(N-1):
79          for j in range(N-1):
80              glBegin(GL_LINES)
81              glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2])
82              glVertex3f(tab[i+1][j][0], tab[i+1][j][1], tab[i+1][j][2])
83              glEnd()
84
85              glBegin(GL_LINES)
86              glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2])
87              glVertex3f(tab[i][j+1][0], tab[i][j+1][1], tab[i][j+1][2])
88              glEnd()
89
90      glFlush()
```

Rysunek 3.1

W zadaniu tym należało zbudować model jajka przy użyciu linii łączących wcześniej opisywanie punkty. Zostało to wykonane przy użyciu prymitywu *GL_LINES*. Dla każdej pary *(i, j)* łączymy ten punkt z punktami *(i + 1, j)* oraz *(i, j + 1)*. Każde połączenie kończymy wywołaniem funkcji *glEnd()*. Efekt działania skryptu dla N = 30 przedstawia Rysunek 3.2.



Rysunek 3.2

4. Zadanie na ocenę 4.0 – skrypt *lab4_0.py*

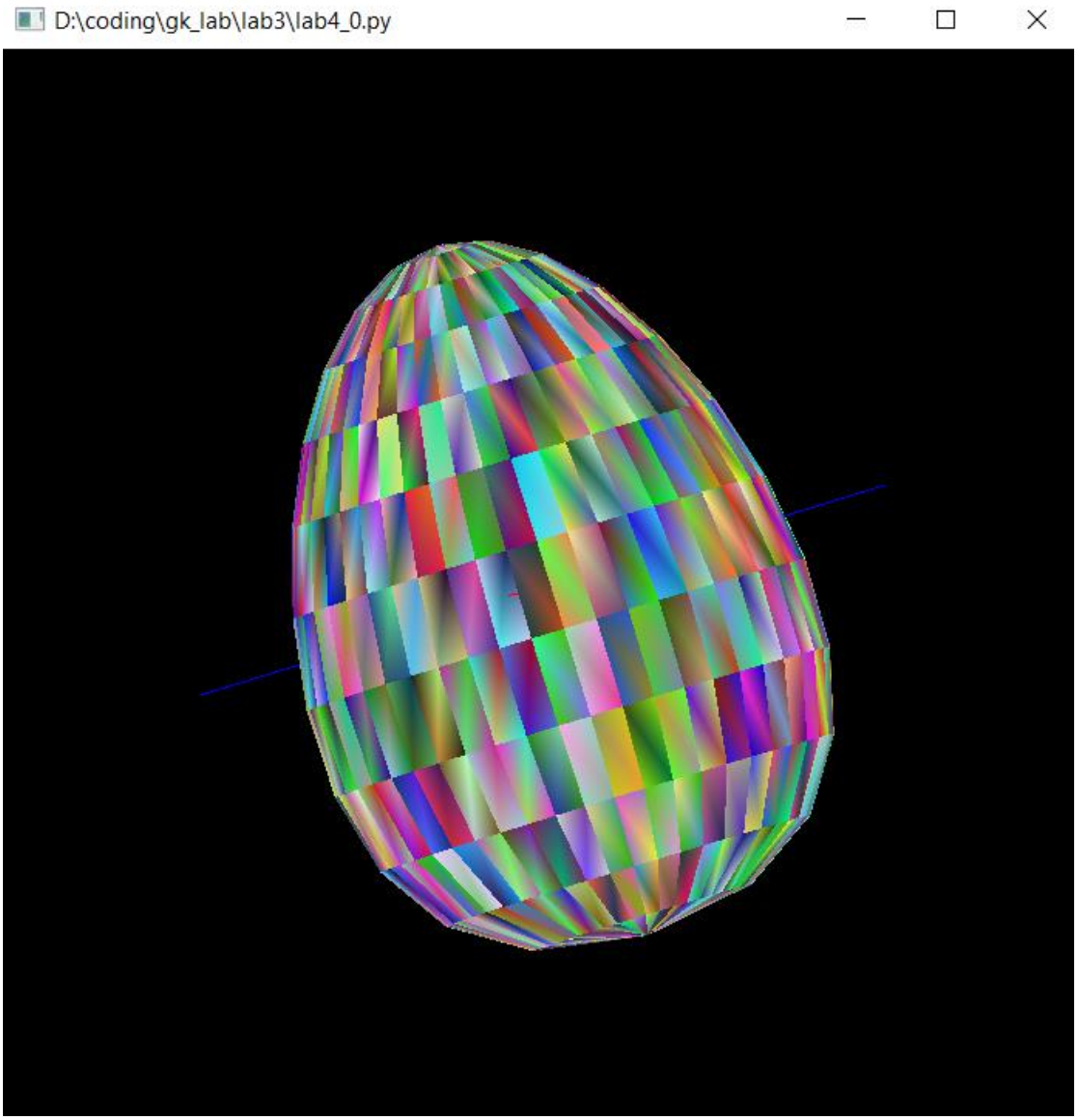
W zadaniu tym należało stworzyć model jaja przy pomocy trójkątów, wykorzystując prymityw *GL_TRIANGLES*. Konieczne było także pokolorowanie każdego wierzchołka trójkąta w losowy sposób tak, aby uniknąć efektu migotania. Osiągnięte zostało to poprzez stworzenie list kolorów dla wierzchołków, które nie zmieniają się w trakcie renderowania modelu. Najpierw inicjalizowany jest generator liczb pseudolosowych, następnie listy wypełniane są wartościami pochodzącymi z tego generatora. Podzielenie liczby z zakresu 0-255 przez 255 gwarantuje otrzymanie wartości z zakresu 0-1, która jest następnie użyta jako argument dla funkcji *glColor3f()* (Rysunek 4.1). W funkcji *render()* inicjalizowany jest licznik *colorIndex*, który jest użyty do iterowania po listach kolorów. W *(N-1)x(N-1)* wykonaniach wewnętrznej pętli tworzone są dwa trójkąty, które łączą wierzchołki *(i, j), (i + 1, j),(i, j + 1) oraz (i + 1, j),(i, j + 1), (i + 1, j + 1).* Przed umieszczeniem każdego wierzchołka w pamięci zmieniany jest kolor rysowania z wykorzystaniem wcześniej opisanych list kolorów (Rysunek 4.2). Efekt działania skryptu *lab4_0.py* dla *N* = 25 przedstawia Rysunek 4.3.

```
33  # Wyznaczamy tablice kolorow wierzcholkow trojkatow - dzieki temu nie bedzie migotania
34  # (brak zmian kolorow przy wielokrotnym wywolywaniu funkcji render)
35  a1, b1, c1, a2, b2, c2 = [], [], [], [], [], []
36  d1, e1, f1 = [], [], []
37
38  random.seed(None)
39
40  for i in range((N-1)*(N-1)):
41      a1.append(random.randint(0,255)/255)
42      b1.append(random.randint(0,255)/255)
43      c1.append(random.randint(0,255)/255)
44      a2.append(random.randint(0,255)/255)
45      b2.append(random.randint(0,255)/255)
46      c2.append(random.randint(0,255)/255)
47      d1.append(random.randint(0,255)/255)
48      e1.append(random.randint(0,255)/255)
49      f1.append(random.randint(0,255)/255)
50
```

Rysunek 4.1

```
95      colorIndex = 0
96      # Rysujemy trojkaty
97      for i in range(N-1):
98          for j in range(N-1):
99              glBegin(GL_TRIANGLES)
100
101             glColor3f(d1[colorIndex], e1[colorIndex], f1[colorIndex])
102             glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2])
103
104             glColor3f(a1[colorIndex], b1[colorIndex], c1[colorIndex])
105             glVertex3f(tab[i+1][j][0], tab[i+1][j][1], tab[i+1][j][2])
106
107             glColor3f(a2[colorIndex], b2[colorIndex], c2[colorIndex])
108             glVertex3f(tab[i][j+1][0], tab[i][j+1][1], tab[i][j+1][2])
109
110             glEnd()
111
112             glBegin(GL_TRIANGLES)
113
114             glColor3f(a2[colorIndex], b2[colorIndex], c2[colorIndex])
115             glVertex3f(tab[i][j+1][0], tab[i][j+1][1], tab[i][j+1][2])
116
117             glColor3f(a1[colorIndex], b1[colorIndex], c1[colorIndex])
118             glVertex3f(tab[i+1][j][0], tab[i+1][j][1], tab[i+1][j][2])
119
120             glColor3f(d1[colorIndex], e1[colorIndex], f1[colorIndex])
121             glVertex3f(tab[i+1][j+1][0], tab[i+1][j+1][1], tab[i+1][j+1][2])
122
123             glEnd()
124             colorIndex += 1
125     glFlush()
126
```

Rysunek 4.2



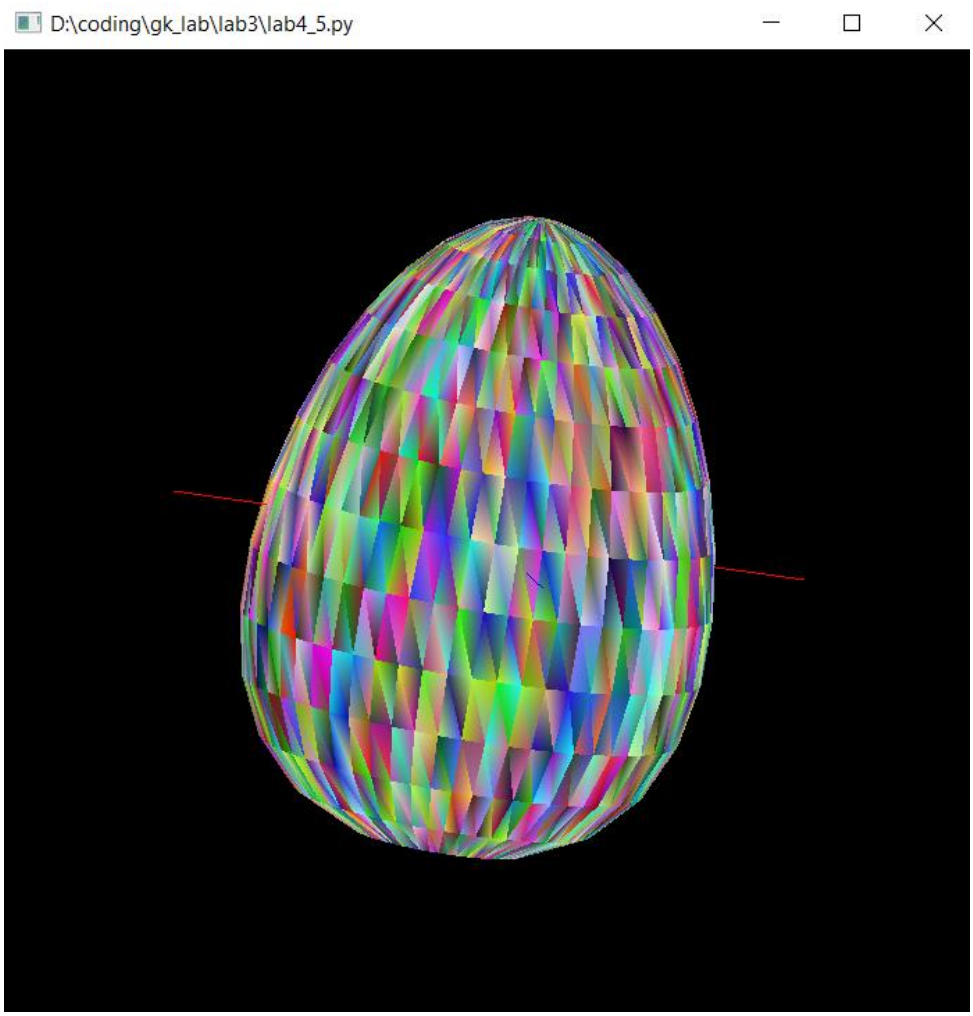
Rysunek 4.3

5. Zadanie na ocenę 4.5 – skrypt *lab4_5.py*

W zadaniu tym należało uzyskać taki sam efekt jak poprzednio, ale z wykorzystaniem prymitywu *GL_TRIANGLE_STRIP*. Do kolorowania wierzchołków trójkątów został wykorzystany ten sam kod, co poprzednio (Rysunek 4.1). W zewnętrznej pętli for inicjalizujemy tworzenie prymitywu za pomocą funkcji *glBegin()*, a następnie w każdej iteracji wewnętrznej pętli zapisujemy wierzchołki w pamięci – o współrzędnych *(i, j), (i + 1, j),(i, j + 1) oraz (i + 1, j + 1*) (Rysunek 5.1). Powoduje to poprawne tworzenie pasów, które tworzą trójkąt - efekt działania skryptu dla *N* = 30 przedstawia Rysunek 5.2.

```
    colorIndex = 0
    for j in range(N-1):
        glBegin(GL_TRIANGLE_STRIP)
        for i in range(N-1):
            glColor3f(d1[colorIndex], e1[colorIndex], f1[colorIndex])
            glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2])
            glColor3f(a1[colorIndex], b1[colorIndex], c1[colorIndex])
            glVertex3f(tab[i+1][j][0], tab[i+1][j][1], tab[i+1][j][2])
            glColor3f(a2[colorIndex], b2[colorIndex], c2[colorIndex])
            glVertex3f(tab[i][j+1][0], tab[i][j+1][1], tab[i][j+1][2])
            glColor3f(d1[colorIndex], e1[colorIndex], f1[colorIndex])
            glVertex3f(tab[i+1][j+1][0], tab[i+1][j+1][1], tab[i+1][j+1][2])
            colorIndex+=1
        glEnd()
    glFlush()
```

Rysunek 5.1



Rysunek 5.2