Grafika Komputerowa

Kierunek	Termin
Informatyka	*Poniedziałek TP 11:00*
Autor	**Rodzaj dokumentu**
241281 Karol Kulawiec	*Sprawozdanie*
Prowadzący	**data**
Mgr inż. Szymon Datko	*9 kwietnia 2020*

1 Wstęp

Podczas zajęć wykonywaliśmy różne przekształcenia obrazu. W moim przypadku korzystałem z PyOpenGl, ponieważ pisałem w języku Python. Wszystkie zadania będą omówiona w kolejnych podpunktach.

2 Laboratoria

2.1 Ćwiczenie 2: OpenGL - podstawy, 28.10.2019

2.1.1 Cel ćwiczenia

Celem ćwiczenia było zaprezentowanie elementarnych możliwości biblioteki graficznej OpenGL wraz z rozszerzeniem GL Utility Toolkit (GLUT). Ćwiczenie obejmowało inicjalizację i zamykanie trybu OpenGL oraz rysowanie tworów pierwotnych (prymitywów) w przestrzeni 2D.
Głównym zadaniem było wykonanie Dywanu Sierpińskiego wraz z modyfikacją kolorów oraz kształtów kwadratów, z których Dywan się składał.

2.1.2 Wykonane zadania

Podczas zajęć wykonałem funkcje rysującą kwadrat oraz funkcję generującą losowy kolor. W domu wykonałem dodatkowo rekurencyjne wywołanie funkcji rysującej fraktal, losowe zniekształcenie rysowanych kwadratów oraz iteracyjne wywołanie funkcji rysującej fraktal.

2.1.3 Prezentacja i omówienie poszczególnych funkcjonalności

1. Generowanie losowego koloru dla poszczególnego wierzchołka:

Listing 1: Funkcja GetColor()

```
1 def GetColor():
2     color = list(np.random.random(size=3))
3     return glColor3f(color[0],color[1],color[2])
```

Korzystając z funkcji random, z biblioteki Numpy, w funkcji generowana jest lista 3 losowych liczb z przedziału $[0.0, 1.0)$, które są zwracane jako kolor.

2. Generowanie losowego zniekształcenia kwadratu:

Listing 2: Funkcja NewCurvature()

```
1 def NewCurvature():
2     return np.random.random()/10
```

Podobnie jak w powyższym przypadku, zwracana jest liczba z przedziału $[0.0, 0.1)$, która wyznacza stopień wykrzywienia odpowiedniej części fraktala.

3. Rysowanie kolorowego czworokąta:

Listing 3: Funkcja DrawColorPolygon(width center curvature)

```
1 def DrawColorPolygon(width, center, curvature):
2     glBegin(GL_POLYGON)
3     GetColor()
4     glVertex2f(center[0]-0.5*width, center[1]+0.5*width)
```

```
5       GetColor()
6       glVertex2f(center[0]+0.5*width, center[1]+0.5*width+curvature*width
            )
7       GetColor()
8       glVertex2f(center[0]+0.5*width, center[1]-0.5*width+curvature*width
            )
9       GetColor()
10      glVertex2f(center[0]-0.5*width, center[1]-0.5*width)
11      glEnd()
```

W funkcji, wykorzystując "GL_POLYGON", rysowany jest czworokąt. Miejsce położenia czterech wierzchołków wyznaczane jest na podstawie środka(center) położenia danego czworokąta oraz jego domyślnej szerokości(width). Aby zniekształcić czworokąt, jego prawa krawędź, jest przesuwana do góry o losową wartość, z przedziału $[0.0, 0.1 * width)$.

4. Renderowanie Sceny:

Listing 4: Funkcja RenderScene()

```
1   def RenderScene():
2       glClear(GL_COLOR_BUFFER_BIT)
3       Recursion(step=1, width=200.0, center=np.array([0.0,0.0]))
4       glFlush()
```

Funkcja renderująca scenę o centrum w (0.0, 0.0).

5. Rekurencyjne rysowanie fraktalu:

Listing 5: Funkcja Recursion(step width center)

```
1   def Recursion(step, width, center):
2       if step == 1:
3           curvature = NewCurvature()
4           #DrawColorPolygon(width, center, curvature)
5           new_center = center + [0,width/2*curvature]
6           Recursion(step+1, width/3.0, new_center)
7
8       elif step <= MAX_STEP:
9           #Lewy Gorny
10          curvature = NewCurvature() * (-1)
11          new_center = center + [-width, width] + [0,width/2*curvature]
12          IfMaxStepDraw(width, new_center, curvature, step)
13          new_center += [0,width/2*curvature]
14          Recursion(step+1, width/3.0, new_center)
15
16          #Srodkowy Gorny
17          curvature = NewCurvature() * (-1)
18          new_center = center + [0.0, width]
19          IfMaxStepDraw(width, new_center, curvature, step)
20          new_center += [0,width/2*curvature]
21          Recursion(step+1, width/3.0, new_center)
22
23          #Prawy Gorny
```

```
24          curvature = NewCurvature() * (-1)
25          new_center = center + [+width, width]
26          IfMaxStepDraw(width, new_center, curvature, step)
27          new_center += [0,width/2*curvature]
28          Recursion(step+1, width/3.0, new_center)
29
30          #Lewy Srodkowy
31          curvature = NewCurvature()
32          new_center = center + [-width, 0.0]
33          IfMaxStepDraw(width, new_center, curvature, step)
34          new_center += [0,width/2*curvature]
35          Recursion(step+1, width/3.0, new_center)
36
37          #Prawy Srodkowy
38          curvature = NewCurvature()
39          new_center = center + [+width, 0.0]
40          IfMaxStepDraw(width, new_center, curvature, step)
41          new_center += [0,width/2*curvature]
42          Recursion(step+1, width/3.0, new_center)
43
44          #Lewy Dolny
45          curvature = NewCurvature()
46          new_center = center + [-width, -width]
47          IfMaxStepDraw(width, new_center, curvature, step)
48          new_center += [0,width/2*curvature]
49          Recursion(step+1, width/3.0, new_center)
50
51          #Srodkowy Dolny
52          curvature = NewCurvature()
53          new_center = center + [0.0, -width]
54          IfMaxStepDraw(width, new_center, curvature, step)
55          new_center += [0,width/2*curvature]
56          Recursion(step+1, width/3.0, new_center)
57
58          #Prawy Dolny
59          curvature = NewCurvature()
60          new_center = center + [+width, -width]
61          IfMaxStepDraw(width, new_center, curvature, step)
62          new_center += [0,width/2*curvature]
63          Recursion(step+1, width/3.0, new_center)
```

Funkcja ta dzieli obecny kwadrat na 8 mniejszych kwadratów (pomijam środek, ponieważ pozostawiam go białego). Dla każdego z kwadratu oblicza stopień zniekształcenia, wyznacza jego środek, który jest potrzebny do jego narysowania, sprawdza czy to już maksymalny stopień rekurencyjny (jeżeli tak, to go rysuje przy pomocy funkcji IfMaxStepDraw), wyznacza centrum, potrzebne do kolejnego wywołania rekurencyjnego oraz wywołuje rekurencyjnie funkcję Recursion. Jeżeli jesteśmy w funkcji pierwszy raz (czyli step == 1), nie wyznaczamy centrum potrzebnego do jego narysowania oraz nie wywołujemy funkcji IfMaxStepDraw.

6. Iteracyjne rysowanie fraktalu:

(a) Szukanie nowego centrum i rysowanie:

Listing 6: Funkcja New_Center_and_Draw(i width step center)

```
1  def New_Center_and_Draw(i, width, step, center):
2      if i==0: #Lewy Gorny
3          curvature = NewCurvature() * (-1)
4          new_center = center + [-width, width] + [0,width/2*curvature
               ]
5          IfMaxStepDraw(width, new_center, curvature, step)
6          new_center += [0,width/2*curvature]
7          return new_center
8
9      if i==1: #Srodkowy Gorny
10         curvature = NewCurvature() * (-1)
11         new_center = center + [0.0, width]
12         IfMaxStepDraw(width, new_center, curvature, step)
13         new_center += [0,width/2*curvature]
14         return new_center
15
16     if i==2: #Prawy Gorny
17         curvature = NewCurvature() * (-1)
18         new_center = center + [+width, width]
19         IfMaxStepDraw(width, new_center, curvature, step)
20         new_center += [0,width/2*curvature]
21         return new_center
22
23     if i==3: #Lewy Srodkowy
24         curvature = NewCurvature()
25         new_center = center + [-width, 0.0]
26         IfMaxStepDraw(width, new_center, curvature, step)
27         new_center += [0,width/2*curvature]
28         return new_center
29
30     if i==5: #Prawy Srodkowy
31         curvature = NewCurvature()
32         new_center = center + [+width, 0.0]
33         IfMaxStepDraw(width, new_center, curvature, step)
34         new_center += [0,width/2*curvature]
35         return new_center
36
37     if i==6: #Lewy Dolny
38         curvature = NewCurvature()
39         new_center = center + [-width, -width]
40         IfMaxStepDraw(width, new_center, curvature, step)
41         new_center += [0,width/2*curvature]
42         return new_center
43
44     if i==7: #Srodkowy Dolny
45         curvature = NewCurvature()
46         new_center = center + [0.0, -width]
47         IfMaxStepDraw(width, new_center, curvature, step)
```

```
48          new_center += [0,width/2*curvature]
49          return new_center
50
51      if i==8: #Prawy Dolny
52          curvature = NewCurvature()
53          new_center = center + [+width, -width]
54          IfMaxStepDraw(width, new_center, curvature, step)
55          new_center += [0,width/2*curvature]
56          return new_center
```

W przypadku iteracyjnego rysowania fraktalu, funkcja rysująca, różni się od funkcji Recursion tym, że nie wywołuje funkcji Recursion, a zwraca nowy środek. Funkcja rozpoznaje, z którym kwadratem ma do czynienia, poprzez parametr i.

(b) Iteracyjne wywołanie rysowania fraktalu:

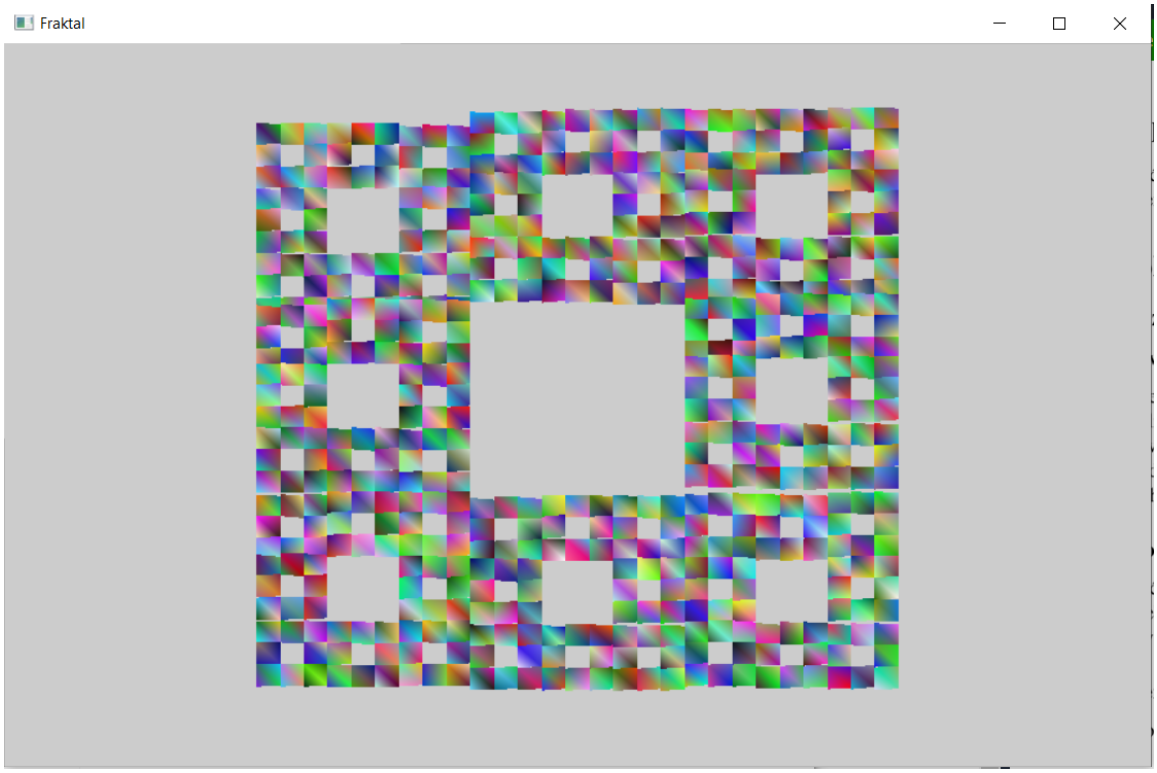
Listing 7: Funkcja Iteratively(step width center)

```
1  def Iteratively(step, width, center):
2      step_1 = step
3      width_1 = width
4      center_1 = center + [0,width/2*NewCurvature()]
5      for i_2 in range(9):
6          if i_2 == 4:
7              continue
8          step_2 = step_1 +1
9          width_2 = width_1/3
10         center_2 = New_Center_and_Draw(i_2, width_2, step_2,
               center_1)
11         for i_3 in range(9):
12             if i_3 == 4:
13                 continue
14             step_3 = step_2 +1
15             width_3 = width_2/3
16             center_3 = New_Center_and_Draw(i_3, width_3, step_3,
                   center_2)
17             for i_4 in range(9):
18                 if i_4 == 4:
19                     continue
20                 step_4 = step_3 +1
21                 width_4 = width_3/3
22                 center_4 = New_Center_and_Draw(i_4, width_4, step_4,
                       center_3)
```

Aby iteracyjne rysowanie fraktalu działało poprawnie, potrzebna jet funkcja, która będzie posiadała tyle pętli w pętlach, ile kroków chcemy zrobić. Za każdym razem wyznacza z którym krokiem mamy do czynienia, szerokość bieżącego kwadratu, oraz przy pomocy funkcji New_Center_and_Draw, centrum danego kwadratu.

2.1.4 Efekt wykonanej pracy

Efekty przy czterokrotnym wywołaniu rekurencyjnym ($MAX\_STEP = 4$):



Rysunek 1: Dywan Sierpińskiego

2.2 Ćwiczenie 3: OpenGL - modelowanie obiektów 3-D, 13.11.2019

2.2.1 Cel ćwiczenia

Celem ćwiczenia było wykonanie w trójwymiarowym układzie współrzędnych transformacji obiektów oraz stworzenie własnego modelu nietrywialnego obiektu za pomocą równań parametrycznych. Dodatkowo podczas wykonania programu, do jego sterowania używano klawiatury.

2.2.2 Wykonane zadania

Podczas zajęć napisałem rysowanie układu współrzędnych 3-D, transformacje w przestrzeni 3-D, zbudowałem jajko z chmury punktów, z siatki odcinków, z siatki trójkątów oraz z siatki GL_TRIANGLE_STRIP. Jajko zostało pokolorowane, nie posiadało żadnych wybrzuszeń, ani zaburzeń, poza pionową linią dookoła jajka, powstałą z niezgrania się kolorów. Dodatkowo wprowadziłem obiekt w ruch. Zmiana pomiędzy poszczególnymi modelami, wykonuje się przy pomocy klawiatury.

2.2.3 Prezentacja i omówienie poszczególnych funkcjonalności

1. Renderowanie Sceny:

Listing 8: Funkcja RenderScene()

```
def RenderScene():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    Axes()
    glColor3f(1.0, 1.0, 1.0)
    glRotated(30.0, 2.0, 0.0, 0.0)
    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)
    Egg()
    glFlush()
    glutSwapBuffers()
```

Funkcja renderująca scenę przed wyświetleniem jajka (przed Egg()), obraca model wokół osi x, y, z o kąty wyliczone w funkcji spinEgg(), co spowoduje obracanie się modelu jajka.

2. Odczyt klawisza z klawiatury:

Listing 9: Funkcja keys(key x y)

```
def keys(key, x, y):
    global model
    global N
    if key == b'p' :
        model = 1
    if key == b'w' :
        model = 2
    if key == b's' :
        model = 3
    if key == b'r' :
        model = 4
    if key == b'q' :
```

```
13              if N>2:
14                  N-=1
15          if key == b'e' :
16              if N<50:
17                  N+=1
```

Funkcja rozpoznaje, który klawisz został wciśnięty i na jego podstawie nadaje wartość zmiennej globalnej model oraz N. Model określa czy jajko będzie zbudowane z chmury punktów, linii, trójkątów czy GL_TRIANGLE_STRIP, natomiast N zmienia ilość tych elementów.

3. Obrót jaja:

Listing 10: Funkcja spinEgg()

```
1  def spinEgg():
2      theta[0] -= .5
3      if theta[0] > 360.0: theta[0] -= 360.0
4      theta[1] -= .5
5      if theta[1] > 360.0: theta[1] -= 360.0
6      theta[2] -= .5
7      if theta[2] > 360.0: theta[2] -= 360.0
8      glutPostRedisplay()
```

Funkcja zmienia parametr powodujący obrót jajka.

4. Rysowanie i wybór modelu jajka:

Listing 11: Funkcja Egg()

```
1  def Egg():
2      tab_xyz = np.zeros((N,N,3))
3      for i in range(N):
4          u = i/(N-1)
5          for j in range(N):
6              v = j/(N-1)
7              tmp = (-90*u**5 + 225*u**4 - 270*u**3 + 180*u**2-45*u)
8              tab_xyz[i,j,0]=tmp*np.cos(np.pi*v)
9              tab_xyz[i,j,1]=160*u**4 - 320*u**3 + 160*u**2
10             tab_xyz[i,j,2]=tmp*np.sin(np.pi*v)
11
12     np.random.seed(1)
13     tab_colours = np.random.rand(N,N,3)
14
15     if model == 1:
16         EggPoints(tab_xyz)
17     if model == 2:
18         EggNet(tab_xyz)
19     if model == 3:
20         EggTriangles(tab_xyz,tab_colours)
21     if model == 4:
22         EggTriangleStrip(tab_xyz,tab_colours)
```

Funkcja tworzy tabelę na współrzędne x, y, z, które są obliczane przy pomocy obrócenia odpowiednio dobranej krzywej Bezier'a, oraz tworzy tabelę takich samych rozmiarów, przechowującą losowo wybrany kolor punktu. Następnie, zależnie od sprawdzonego modelu, rysuje jajko.

5. Jajo wykonane z chmury punktów:

Listing 12: Funkcja EggPoints(tab_xyz)

```
def EggPoints(tab_xyz):
    glBegin(GL_POINTS)
    for i in range(N):
        for j in range(N):
            x=tab_xyz[i,j,0]
            y=tab_xyz[i,j,1]-5
            z=tab_xyz[i,j,2]
            glVertex3f(x,y,z)
    glEnd()
```

Funkcja dla każdej współrzędnej rysuje punkt.

6. Jajo wykonane z siatki odcinków:

Listing 13: Funkcja EggNet(tab_xyz)

```
def EggNet(tab_xyz):
    glBegin(GL_LINES)
    for i in range(N):
        for j in range(N-1):
            x_1=tab_xyz[i,j,0]
            y_1=tab_xyz[i,j,1]-5
            z_1=tab_xyz[i,j,2]
            glVertex3f(x_1,y_1,z_1)
            x_2=tab_xyz[i,j+1,0]
            y_2=tab_xyz[i,j+1,1]-5
            z_2=tab_xyz[i,j+1,2]
            glVertex3f(x_2,y_2,z_2)

    for i in range(N-1):
        for j in range(N):
            x_1=tab_xyz[i,j,0]
            y_1=tab_xyz[i,j,1]-5
            z_1=tab_xyz[i,j,2]
            glVertex3f(x_1,y_1,z_1)
            x_2=tab_xyz[i+1,j,0]
            y_2=tab_xyz[i+1,j,1]-5
            z_2=tab_xyz[i+1,j,2]
            glVertex3f(x_2,y_2,z_2)

    glEnd()
```

Funkcja pobiera dwa kolejne punkty, wg pionu oraz poziomu, i rysuje linię.

7. Jajo wykonane z siatki trójkątów:

Listing 14: Funkcja EggTriangles(tab_xyz tab_colours)

```
1  def EggTriangles(tab_xyz, tab_colours):
2      glBegin(GL_TRIANGLES)
3
4      for i in range(N-1):
5          for j in range(N-1):
6              glColor3f(tab_colours[i,j,0], tab_colours[i,j,1],
                   tab_colours[i,j,2])
7              x_1=tab_xyz[i,j,0]
8              y_1=tab_xyz[i,j,1]-5
9              z_1=tab_xyz[i,j,2]
10             glVertex3f(x_1,y_1,z_1)
11
12             glColor3f(tab_colours[i,j+1,0], tab_colours[i,j+1,1],
                   tab_colours[i,j+1,2])
13             x_2=tab_xyz[i,j+1,0]
14             y_2=tab_xyz[i,j+1,1]-5
15             z_2=tab_xyz[i,j+1,2]
16             glVertex3f(x_2,y_2,z_2)
17
18             glColor3f(tab_colours[i+1,j,0], tab_colours[i+1,j,1],
                   tab_colours[i+1,j,2])
19             x_3=tab_xyz[i+1,j,0]
20             y_3=tab_xyz[i+1,j,1]-5
21             z_3=tab_xyz[i+1,j,2]
22             glVertex3f(x_3,y_3,z_3)
23
24             if j == N-2:
25                 glColor3f(tab_colours[i,0,0], tab_colours[i,0,1],
                       tab_colours[i,0,2])
26             else:
27                 glColor3f(tab_colours[i,j+1,0], tab_colours[i,j+1,1],
                       tab_colours[i,j+1,2])
28             x_1=tab_xyz[i,j+1,0]
29             y_1=tab_xyz[i,j+1,1]-5
30             z_1=tab_xyz[i,j+1,2]
31             glVertex3f(x_1,y_1,z_1)
32
33             glColor3f(tab_colours[i+1,j,0], tab_colours[i+1,j,1],
                   tab_colours[i+1,j,2])
34             x_2=tab_xyz[i+1,j,0]
35             y_2=tab_xyz[i+1,j,1]-5
36             z_2=tab_xyz[i+1,j,2]
37             glVertex3f(x_2,y_2,z_2)
38
39             if j == N-2:
40                 glColor3f(tab_colours[i+1,0,0], tab_colours[i+1,0,1],
                       tab_colours[i+1,0,2])
41             else:
```

```
42                  glColor3f(tab_colours[i+1,j+1,0], tab_colours[i+1,j
                        +1,1], tab_colours[i+1,j+1,2])
43              x_3=tab_xyz[i+1,j+1,0]
44              y_3=tab_xyz[i+1,j+1,1]-5
45              z_3=tab_xyz[i+1,j+1,2]
46              glVertex3f(x_3,y_3,z_3)
47
48      glEnd()
```

Funkcja dla 3 sąsiadujących punktów rysuje je, tworząc w ten sposób trójkąt oraz koloruje te wierzchołki. W tej samej pętli rysuje kolejny trójkąt, wykorzystując 2 wierzchołki, użyte w poprzednim trójkącie, które koloruje na ten sam kolor.

8. Jajo wykonane z siatki GL_TRIANGLE_STRIP:

Listing 15: Funkcja EggTriangleStrip(tab_xyz tab_colours)

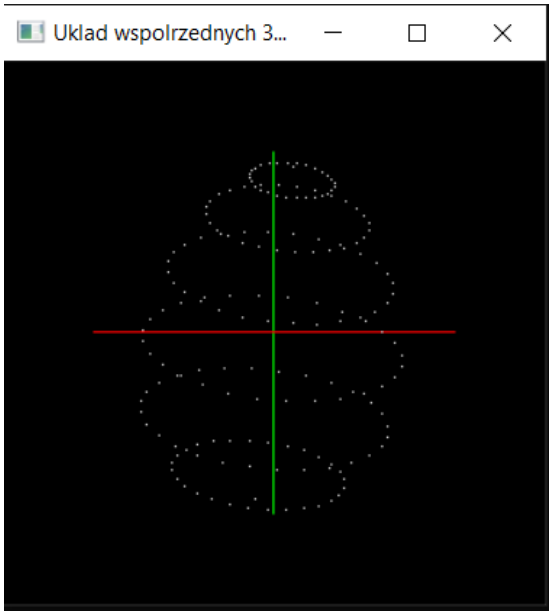
```
1   def EggTriangleStrip(tab_xyz,tab_colours):
2       glBegin(GL_TRIANGLE_STRIP)
3
4       for i in range(N-1):
5           glColor3f(tab_colours[i,0,0], tab_colours[i,0,1], tab_colours[i
                ,0,2])
6           x_1=tab_xyz[i,0,0]
7           y_1=tab_xyz[i,0,1]-5
8           z_1=tab_xyz[i,0,2]
9           glVertex3f(x_1,y_1,z_1)
10
11          glColor3f(tab_colours[i,1,0], tab_colours[i,1,1], tab_colours[i
                ,1,2])
12          x_2=tab_xyz[i,1,0]
13          y_2=tab_xyz[i,1,1]-5
14          z_2=tab_xyz[i,1,2]
15          glVertex3f(x_2,y_2,z_2)
16          for j in range(0,N-2,2):
17              glColor3f(tab_colours[i+1,j,0], tab_colours[i+1,j,1],
                    tab_colours[i+1,j,2])
18              x_3=tab_xyz[i+1,j,0]
19              y_3=tab_xyz[i+1,j,1]-5
20              z_3=tab_xyz[i+1,j,2]
21              glVertex3f(x_3,y_3,z_3)
22
23              glColor3f(tab_colours[i+1,j+1,0], tab_colours[i+1,j+1,1],
                    tab_colours[i+1,j+1,2])
24              x_3=tab_xyz[i+1,j+1,0]
25              y_3=tab_xyz[i+1,j+1,1]-5
26              z_3=tab_xyz[i+1,j+1,2]
27              glVertex3f(x_3,y_3,z_3)
28
29              glColor3f(tab_colours[i+1,j+2,0], tab_colours[i+1,j+2,1],
                    tab_colours[i+1,j+2,2])
30              x_3=tab_xyz[i+1,j+2,0]
```

```
31                  y_3=tab_xyz[i+1,j+2,1]-5
32                  z_3=tab_xyz[i+1,j+2,2]
33                  glVertex3f(x_3,y_3,z_3)
34
35                  glColor3f(tab_colours[i,j+1,0], tab_colours[i,j+1,1],
                        tab_colours[i,j+1,2])
36                  x_2=tab_xyz[i,j+1,0]
37                  y_2=tab_xyz[i,j+1,1]-5
38                  z_2=tab_xyz[i,j+1,2]
39                  glVertex3f(x_2,y_2,z_2)
40
41                  glColor3f(tab_colours[i,j+2,0], tab_colours[i,j+2,1],
                        tab_colours[i,j+2,2])
42                  x_3=tab_xyz[i,j+2,0]
43                  y_3=tab_xyz[i,j+2,1]-5
44                  z_3=tab_xyz[i,j+2,2]
45                  glVertex3f(x_3,y_3,z_3)
46
47                  if j != N-3:
48                      if j == N-4:
49                          glColor3f(tab_colours[i,0,0], tab_colours[i,0,1],
                                tab_colours[i,0,2])
50                      else:
51                          glColor3f(tab_colours[i,j+3,0], tab_colours[i,j
                                +3,1], tab_colours[i,j+3,2])
52                      x_3=tab_xyz[i,j+3,0]
53                      y_3=tab_xyz[i,j+3,1]-5
54                      z_3=tab_xyz[i,j+3,2]
55                      glVertex3f(x_3,y_3,z_3)
56
57                  if j == N-4 :
58                      glColor3f(tab_colours[i+1,j+2,0], tab_colours[i+1,j
                            +2,1], tab_colours[i+1,j+2,2])
59                      x_3=tab_xyz[i+1,j+2,0]
60                      y_3=tab_xyz[i+1,j+2,1]-5
61                      z_3=tab_xyz[i+1,j+2,2]
62                      glVertex3f(x_3,y_3,z_3)
63
64                      glColor3f(tab_colours[i+1,0,0], tab_colours[i+1,0,1],
                            tab_colours[i+1,0,2])
65                      x_3=tab_xyz[i+1,j+3,0]
66                      y_3=tab_xyz[i+1,j+3,1]-5
67                      z_3=tab_xyz[i+1,j+3,2]
68                      glVertex3f(x_3,y_3,z_3)
69
70      glEnd()
```

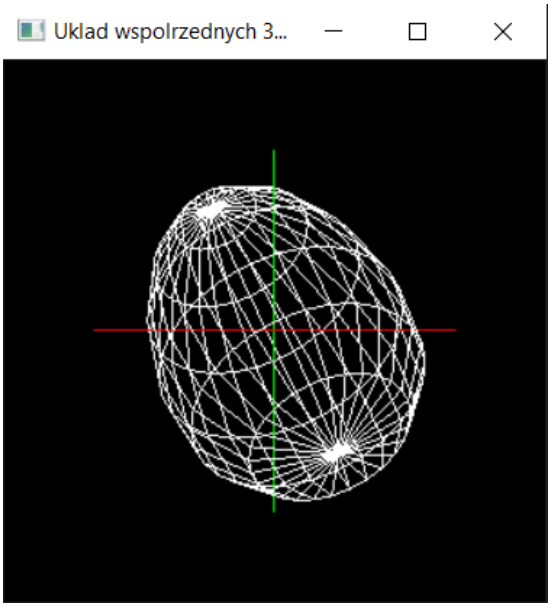
Funkcja wykonuje to samo, co poprzednia funkcja, ale przy użyciu GL_TRIANGLE_STRIP. Funkcja ma w zamiarze wywoływania wierzchołków w takiej kolejności, aby powstał zygzak pokrywający całą planszę.

Funkcja nie jest dokładna, niektóre trójkąty mają wierzchołki współliniowe, czyli nie widać ich, ale działa tak jak widać w kolejnym podpunkcie.

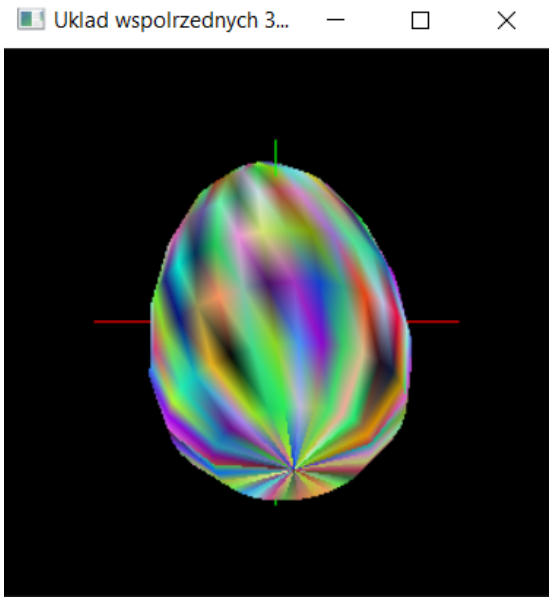
2.2.4 Efekty wykonanej pracy



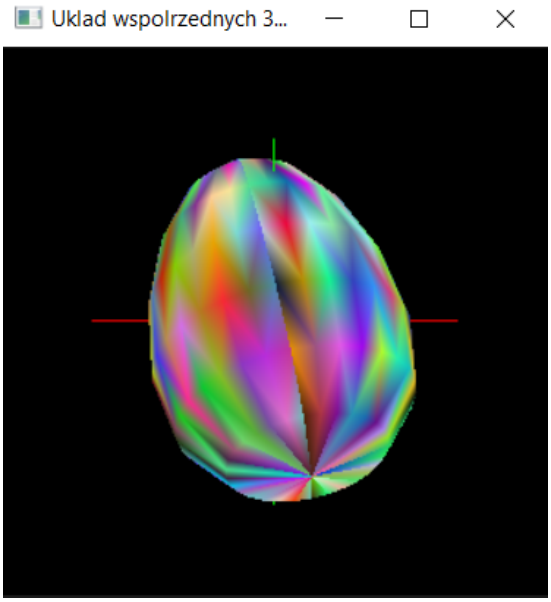
Rysunek 2: Chmura punktów



Rysunek 3: Siatka odcinków



Rysunek 4: Siatka trójkątów



Rysunek 5: Siatka GL_TRIANGLE_STRIP

2.3 Ćwiczenie 4: OpenGL - interakcja z użytkownikiem, 25.11.2019

2.3.1 Cel ćwiczenia

Celem ćwiczenia było wykonanie programu wyświetlającego czajnik, którym użytkownik mógł obracać, przybliżać i oddalać oraz zmieniać położenie kamery patrzącej na niego przy użyciu myszki.

2.3.2 Wykonane zadania

Podczas zajęć wykonałem obracanie czajnika, ńiby-zoom", poruszanie kamerą dookoła modelu, ograniczenie zakresu przybliżania/oddalenia kamery, połączenie pomiędzy trybem obracania obiektu i poruszania kamerą.

2.3.3 Prezentacja i omówienie poszczególnych funkcjonalności

1. Odczytywanie ruchu myszką:

Listing 16: Funkcja Mouse(btn state x y)

```
def Mouse(btn, state, x, y):
    global status, x_pos_old, y_pos_old, zoom_pos_old
    if btn==GLUT_LEFT_BUTTON and state == GLUT_DOWN:
        x_pos_old = x
        y_pos_old = y
        status = 1
    elif btn==GLUT_RIGHT_BUTTON and state == GLUT_DOWN:
        zoom_pos_old = x
        status = 2
    else:
        status = 0
```

Funkcja ta wywołana jest w main'ie poprzez funkcję glutMouseFunc(Mouse). Bada stan myszy i ustawia wartości odpowiednich zmiennych globalnych.

2. Funkcja Motion:

Listing 17: Funkcja Motion(x y)

```
def Motion(x, y):
    global delta_x, x_pos_old, delta_y, y_pos_old, delta_zoom,
        zoom_pos_old
    delta_x = x - x_pos_old
    x_pos_old = x
    delta_y = y - y_pos_old
    y_pos_old = y
    delta_zoom = x - zoom_pos_old
    zoom_pos_old = x
    glutPostRedisplay()
```

Funkcja wywołana jest w main'ie poprzez funkcję glutMotionFunc(Motion), zaraz po glutMouseFunc(Mouse). Monitoruje położenie kursora myszy i ustawia wartości odpowiednich zmiennych globalnych.

3. Odczyt danych z klawiatury:

Listing 18: Funkcja keys(key x y)

```
def keys(key, x, y):
    global function
    if key == b'q':
        function += 1
        function = function%2
```

Funkcja sprawdza czy został naciśnięty klawisz 'q', jeżeli został, to zmienna globalna function, odpowiedzialna za ustalenie czy należy obracać czajnikiem, czy kamerą, zostanie zmieniona.

4. Funkcja rysująca osie układu współrzędnych:

Listing 19: Funkcja Axes()

```
def Axes():
    x_min = [-5.0, 0.0, 0.0]
    x_max = [5.0, 0.0, 0.0]
    y_min = [0.0, -5.0, 0.0]
    y_max = [0.0, 5.0, 0.0]
    z_min = [0.0, 0.0, -5.0]
    z_max = [0.0, 0.0, 5.0]

    glColor3f(1.0, 0.0, 0.0)
    glBegin(GL_LINES)
    glVertex3fv(x_min)
    glVertex3fv(x_max)
    glEnd()

    glColor3f(0.0, 1.0, 0.0)
    glBegin(GL_LINES)

    glVertex3fv(y_min)
    glVertex3fv(y_max)
    glEnd()

    glColor3f(0.0, 0.0, 1.0)
    glBegin(GL_LINES)

    glVertex3fv(z_min)
    glVertex3fv(z_max)
    glEnd()
```

Funkcja na podstawie zmiennych x/y/z_min/max określa położenie osi współrzędnych, każdej z nich nadaje inny kolor i rysuje te osie, które odwzorowują układ współrzędnych.

5. Główna Funkcja rysująca RenderScene:

Listing 20: Funkcja RenderScene()

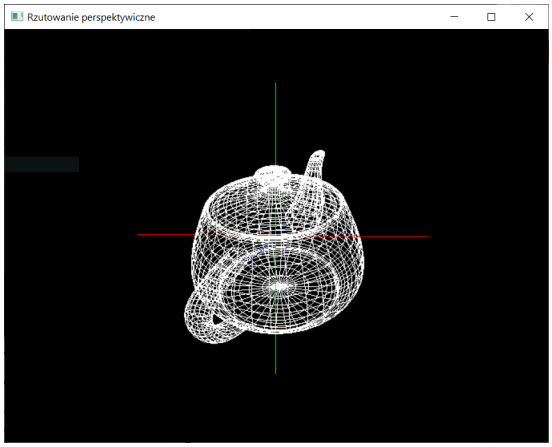
```
def RenderScene():
    global theta, viewer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
```

```
5       gluLookAt(viewer[0], viewer[1], viewer[2], center[0], center[1],
            center[2], .0, 1.0, 0.0)
6       Axes()
7
8       if(status == 1 and function == 0):
9           theta[0] += delta_x * pix2angle
10          theta[1] += delta_y * pix2angle
11      elif(status == 2):
12          theta[2] += (delta_zoom * pix2angle)/300
13          max_r = 65
14          R = np.sqrt((viewer[0]-center[0])**2 + (viewer[1]-center[1])**2
                + (viewer[2]-center[2])**2 )
15          if R * theta[2] > 65:
16              theta[2] = max_r/R
17          elif R*theta[2] <1:
18              theta[2] = 1/R
19      elif(status == 1 and function == 1):
20          R = np.sqrt((viewer[0]-center[0])**2 + (viewer[1]-center[1])**2
                + (viewer[2]-center[2])**2 )
21          elewacja = np.arcsin(viewer[1]/R)
22          azymut = np.arccos(viewer[0]/(R*np.cos(elewacja)))
23
24          elewacja += (delta_x * pix2angle)/150
25          azymut += (delta_y * pix2angle)/150
26
27
28          viewer[1] = R*np.cos(azymut)*np.cos(elewacja)
29          viewer[0] = R*np.sin(elewacja)
30          viewer[2] = R*np.sin(azymut)*np.cos(elewacja)
31
32
33      glRotatef(theta[1], 1.0, 0.0, 0.0)
34      glRotatef(theta[0], 0.0, 1.0, 0.0)
35      glScale(theta[2],theta[2],theta[2])
36      glColor3f(1.0, 1.0, 1.0)
37      glutWireTeapot(3.0)
38      glFlush()
39      glutSwapBuffers()
```

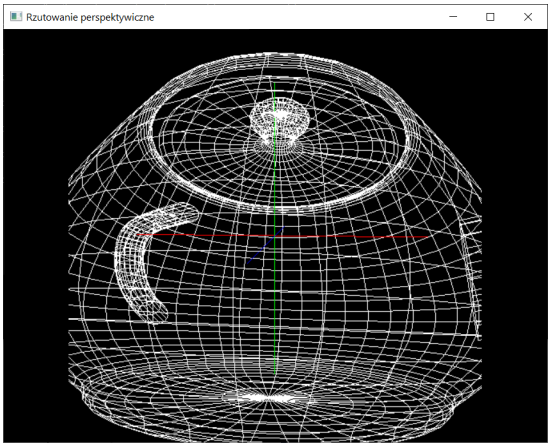
Funkcja na początku, przy pomocy funkcji gluLookAt(), definiuje położenie obserwatora oraz przy pomocy Axes() rysuje osie x, y, z. Następnie, sprawdza co użytkownik robi i wg tego wprowadza zmiany w zmiennych. Do wyboru ma zmianę parametrów odpowiedzialnych za rotację przedmiotu, zmianę parametrów i sprawdzenie czy mieszczą się w ograniczeniach, odpowiedzialnych za zoom kamery bądź zmianę parametrów odpowiedzialnych za ustawienie pozycji obserwatora.

2.3.4 Efekt wykonanej pracy

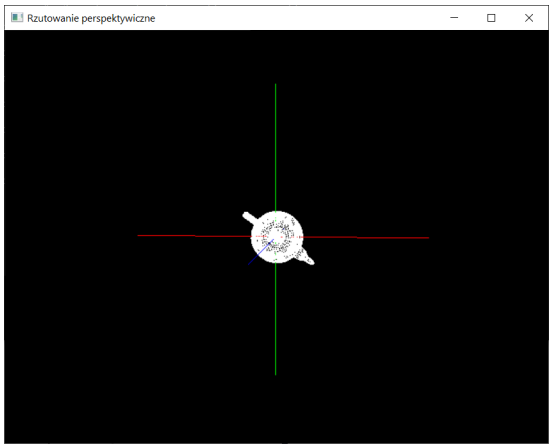
Poniżej przedstawione są 4 różne położenia czajnika oraz kamery:



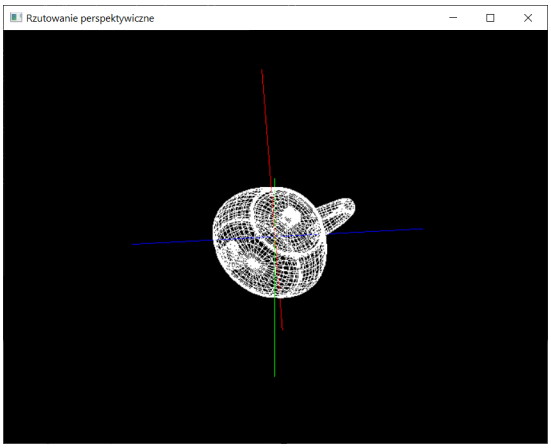
Rysunek 6: Normalny czajnik



Rysunek 7: Czajnik przybliżony



Rysunek 8: Czajnik oddalony



Rysunek 9: Czajnik obrócony

2.4 Ćwiczenie 5: OpenGL - oświetlanie scen 3-D, 9.12.2019

2.4.1 Cel ćwiczenia

Celem ćwiczenia było zaimplementowanie źródła światła, pokazanie w jaki sposób będzie wyglądał obiekt, w zależności od położenia źródła światła.

2.4.2 Wykonane zadania

Podczas zajęć wykonałem przełączanie położenia kamery, oświetlone jajko, dodanie drugiego źródła światła, poruszanie światłami wokół obiektu, wizualizację położenia źródeł światła oraz dynamiczną zmianę składowych koloru źródeł światła.

2.4.3 Prezentacja i omówienie poszczególnych funkcjonalności

1. Odczytywanie ruchu myszką:

Listing 21: Funkcja Mouse(btn state x y) oraz Motion(x y)

```
def Mouse(btn, state, x, y):
    global status, x_pos_old, y_pos_old
    if btn==GLUT_LEFT_BUTTON and state == GLUT_DOWN:
        x_pos_old[0] = x
        y_pos_old[0] = y
        status = 1
    elif btn==GLUT_RIGHT_BUTTON and state == GLUT_DOWN:
        x_pos_old[1] = x
        y_pos_old[1] = y
        status = 2
    else:
        status = 0

def Motion(x, y):
    global delta_x, x_pos_old, delta_y, y_pos_old
    for i in [0,1]:
        delta_x[i] = x -x_pos_old[i]
        x_pos_old[i] = x
        delta_y[i] = y - y_pos_old[i]
        y_pos_old[i] = y

    glutPostRedisplay()
```

Funkcje sprawdzające stan myszy oraz na podstawie ich położenia zmienia parametry określające położenie źródeł światła.

2. Zmiana położenia obserwatora:

Listing 22: Funkcja keys(key x y)

```
def keys(key, x, y):
    global function
    if key == b'q':
        function += 1
        function = function%3
```

Zmiana położenia obserwatora na jednego z statycznie ustawionych trzech.

3. Definicja scenerii:

Listing 23: Funkcja MyInit()

```
def MyInit():
    glClearColor(0.0, 0.0, 0.0, 1.0)
    mat_ambient   = [1.0, 1.0, 1.0, 1.0]
    mat_diffuse   = [fabs(position[0][0])/23, fabs(position[0][1])/23,
        fabs(position[0][2])/23, 1.0]
    mat_diffuse_1   = [fabs(position[1][0])/23, fabs(position[1][1])/23,
         fabs(position[1][2])/23, 1.0]
    mat_specular = [1.0, 1.0, 1.0, 1.0]
    mat_shininess   = 20.0
    light_position = [position[0][0], position[0][1], position[0][2],
        1.0]
    light_position_1 = [position[1][0], position[1][1], position[1][2],
         1.0]
    light_ambient = [0.1, 0.1, 0.1, 1.0]
    light_diffuse = [1.0, 1.0, 1.0, 1.0]
    light_specular= [1.0, 1.0, 0.0, 1.0]
    att_constant  = 1.0
    att_linear    = 0.05
    att_quadratic  = 0.001

    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular)
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient)
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse)
    glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess)

    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient)
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse)
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular)
    glLightfv(GL_LIGHT0, GL_POSITION, light_position)
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, att_constant)
    glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, att_linear)
    glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, att_quadratic)

    glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient)
    glLightfv(GL_LIGHT1, GL_DIFFUSE, light_diffuse)
    glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular)
    glLightfv(GL_LIGHT1, GL_POSITION, light_position_1)
    glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, att_constant)
    glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, att_linear)
    glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, att_quadratic)

    glShadeModel(GL_SMOOTH)
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glEnable(GL_LIGHT1)
    glEnable(GL_DEPTH_TEST)
```

W funkcji zgodnie z instrukcją definiuję materiał z jakiego zrobiony jest czajnik, źródło światła, ustawiam parametry materiału, źródła oraz ustawiam opcję systemu oświetlenia sceny. Ponieważ posiadam 2 źródła światła, ustawienia dla światłą są podwajane. Barwa, z jaką świeci światło, jest uzależniona od zmiennej position, która zmienia się wraz z położeniem źródła światła, dlatego podczas poruszania się światła, zmienia się również jego barwa.

4. Renderowanie Sceny:

Listing 24: Funckje RenderScene()

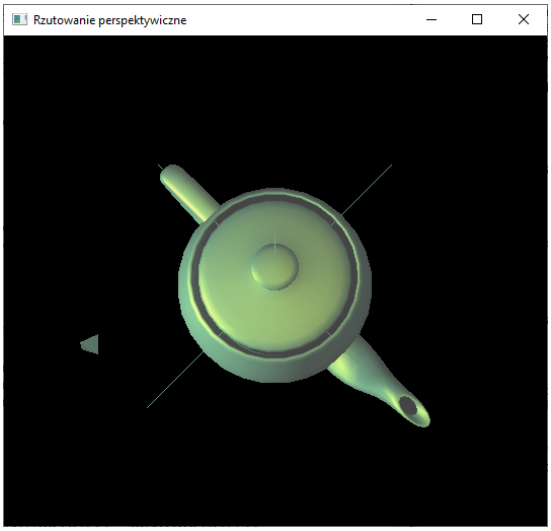
```
1  def RenderScene():
2      global theta, viewer, position
3      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
4      glLoadIdentity()
5      gluLookAt(viewer[function][0], viewer[function][1], viewer[function
           ][2], center[function][0], center[function][1], center[function
           ][2], .0, 1.0, 0.0)
6      Axes()
7
8      if(status > 0):
9          R = np.sqrt((position[status-1][0]-center[function][0])**2 + (
               position[status-1][1]-center[function][1])**2 + (position[
               status-1][2]-center[function][2])**2 )
10         elewacja = np.arcsin(position[status-1][1]/R)
11         azymut = np.arccos(position[status-1][0]/(R*np.cos(elewacja)))
12
13         elewacja += (delta_x[status-1] * pix2angle)/50
14         azymut += (delta_y[status-1] * pix2angle)/50
15
16         position[status-1][0] = R*np.cos(azymut)*np.cos(elewacja)
17         position[status-1][1] = R*np.sin(elewacja)
18         position[status-1][2] = R*np.sin(azymut)*np.cos(elewacja)
19
20     glRotatef(0.0, 1.0, 0.0, 0.0)
21     glRotatef(0.0, 0.0, 1.0, 0.0)
22     glScale(1.0, 1.0, 1.0)
23     glColor3f(1.0, 1.0, 1.0)
24     glutSolidTeapot(3.0)
25     glTranslated(position[0][0], position[0][1], position[0][2])
26     glutSolidTetrahedron()
27     glTranslated(position[1][0]-position[0][0], position[1][1]-position
           [0][1], position[1][2]-position[0][2])
28     glutWireOctahedron()
29     MyInit()
30     glFlush()
31     glutSwapBuffers()
```

Funkcja na początku ustawia położenie kamery wg wybranej wcześniej lokalizacji. Następnie, jeżeli wykonujemy jakiś ruch myszką, to na podstawie zmiany parametrów, zmienia wartości position, która przechowuje współrzędne źródła światła oraz symboli, które je obrazują. Każdy z symboli jest rysowany przy pomocy funkcji glTranslated.

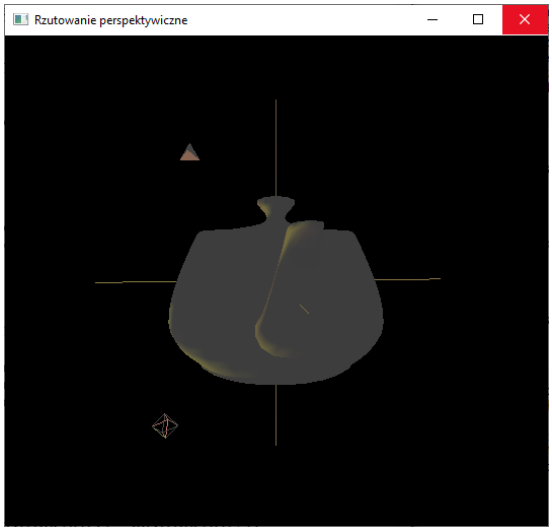
2.4.4 Efekt wykonanej pracy

Poniżej przedstawione są 4 różne efekty pracy:

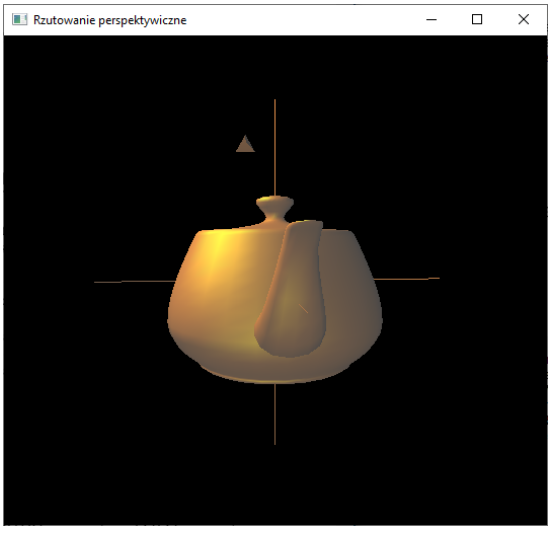
Rysunek 10: Dwa punkty światła



Rysunek 11: Czajnik z górnej perspektywy



Rysunek 12: Czajnik z nieoświetlonej strony wraz z źródłami światła za nim



Rysunek 13: Czajnik z 3 perspektywy

2.5 Ćwiczenie 6: OpenGL - teksturowanie powierzchni obiektów, 10.01.2020

2.5.1 Cel ćwiczenia

Celem ćwiczenia było pokazanie podstawowych technik teksturowania powierzchni obiektów.

2.5.2 Wykonane zadania

Podczas zajęć wykonałem nałożenie tekstury na trójkąt i jego obrót, oteksturowanie sześcianu, zastosowanie opcji Cull Face, przygotowanie własnej tekstury, zastosowanie dwóch tekstur w programie oraz nałożenie tekstury na jajko.

2.5.3 Prezentacja i omówienie poszczególnych funkcjonalności

Z powodu trudności przepisania gotowej funkcji od prowadzącego na język Python, w programie użyłem innej, znalezionej na StackOverflow, korzystającej z PyGame. Użycie PyGame i korzystanie z jego metod spowodowało, że program różni się od pozostałych, dlatego zostanie tutaj całościowo omówiony.

1. Główna funkcja main():

Listing 25: Funkcja main()

```
1  def main():
2      pygame.init()
3      display = (800, 600)
4      pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL |
           pygame.OPENGLBLIT)
5      gluPerspective(45, display[0] / display[1], 0.1, 50.0)
6      glTranslatef(0.0, 0.0, -5)
7
8      while True:
9          loadTexture()
10         keys()
11         RenderScene()
12
13         time.sleep(0.01)
14         pygame.display.flip()
```

Tworzenie okienka wymaga jedynie zainicjowania PyGame (2), przekazanie jako tuple wielkości okienka (3), jego stworzenie (4) oraz odświeżanie (14). Dodatkowo zastosowano gluPerspective() oraz glTranslatef(), w celu lepszego zobrazowania rezultatów pracy. Pętla nieskończona, ładuje teksturę, sprawdza zdarzenia na klawiszach oraz renderuje scenę, wszystko to odbywa się pomijając funkcję GLUT'a. Dodatkowo zastosowano uśpienie programu na 0.01 sekundy, w celu ograniczenie szybkości obrotu obiektów.

2. Renderowanie Sceny:

Listing 26: Funkcja renderScene()

```
1  def RenderScene():
2      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
3      glRotated(1.0, 1.0, 1.0, 0.0)
4      glRotatef(0.1, 1.0, 0.0, 0.0)
5      glRotatef(0.1, 0.0, 1.0, 0.0)
6      glRotatef(0.1, 0.0, 0.0, 1.0)
```

```
7       if model == 1:
8           draw()
9       elif model == 2:
10          draw_cube()
11      else :
12          draw_jajo()
```

W funkcji najpierw obiekt wprowadzany jest w obrót, a następnie, zależnie od wybranego modelu, rysowany jest ten model.

3. Odczyt z klawiatury:

Listing 27: Funkcja keys()

```
1  def keys():
2      global model, nazwa
3      for event in pygame.event.get():
4          if event.type == pygame.QUIT:
5              pygame.quit()
6              sys.exit()
7          if event.type == pygame.KEYDOWN:
8              if event.key == pygame.K_1:
9                  nazwa = 'kot.png'
10             if event.key == pygame.K_2:
11                 nazwa = 'pies.png'
12             if event.key == pygame.K_3:
13                 nazwa = 'kotopies.png'
14             if event.key == pygame.K_q:
15                 model += 1
16                 model = model%3
```

Funkcja, do odczytu klawiatury, korzysta z event z pygame. Dla wszystkich zdarzeń, jeżeli typ zdarzenia to naciśnięcie klawisza w dół, to zależnie od tego klawisza, wybierana jest tekstura, lub model.

4. Załadowanie tekstury:

Listing 28: Funkcja loadTexture()

```
1  def loadTexture():
2      textureSurface = pygame.image.load(nazwa)
3      textureData = pygame.image.tostring(textureSurface, "RGBA", 1)
4      width = textureSurface.get_width()
5      height = textureSurface.get_height()
6
7      glEnable(GL_CULL_FACE)
8      glEnable(GL_TEXTURE_2D)
9      glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL)
10
11     glBindTexture(GL_TEXTURE_2D, glGenTextures(1))
12     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGBA,
           GL_UNSIGNED_BYTE, textureData)
13
14     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
```

```
15      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
16      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
17      glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
```

Funkcja z SlackOverflow do wczytania tekstury. Przy pomocy PyGame wczytuje obrazek, a następnie podobnie do kodu z zajęć, ustawia go. Zastosowano Cull Face, co powoduje, że tylko jedna strona płaszczyzny jest renderowana i przez to może dla nas być albo widoczna, albo niewidoczna.

5. Narysowanie trójkąta z teksturą:

Listing 29: Funkcja draw()

```
1  def draw():
2      glBegin(GL_TRIANGLES)
3      glTexCoord2f(0.0, 0.0)
4      glVertex3f(-1.0, -1.0, 1.0)
5      glTexCoord2f(1.0, 0.0)
6      glVertex3f(1.0, -1.0, 1.0)
7      glTexCoord2f(0.5, 1.0)
8      glVertex3f(0.0, 1.0, 1.0)
9      glEnd()
```

Funkcja standardowo rysuje trójkąt przy użyciu GL_TRIANGLES, a dodatkowo, przed ustawieniem każdego wierzchołka, wywołuje glTexCoord2f(x,y) z koordynatami, które wskazują na część obrazka, która ma zostać przypisana do tej krawędzi. Działa podobnie jak ustawienie koloru, zmienne współrzędne x,y podobnie przechodzą płynnie z jednego glTexCoord2f do drugiego, co powoduje wyświetlenie części obrazka, znajdującej się pomiędzy tymi koordynatami.

6. Narysowanie sześcianu z teksturą:

Listing 30: Funkcja draw_cube()

```
1  def draw_cube():
2      glBegin(GL_QUADS)
3      glTexCoord2f(0.0, 0.0)
4      glVertex3f(-1.0, -1.0, 1.0)
5      glTexCoord2f(1.0, 0.0)
6      glVertex3f(1.0, -1.0, 1.0)
7      glTexCoord2f(1.0, 1.0)
8      glVertex3f(1.0, 1.0, 1.0)
9      glTexCoord2f(0.0, 1.0)
10     glVertex3f(-1.0, 1.0, 1.0)
11     glTexCoord2f(1.0, 0.0)
12     glVertex3f(-1.0, -1.0, -1.0)
13     glTexCoord2f(1.0, 1.0)
14     glVertex3f(-1.0, 1.0, -1.0)
15     glTexCoord2f(0.0, 1.0)
16     glVertex3f(1.0, 1.0, -1.0)
17     glTexCoord2f(0.0, 0.0)
18     glVertex3f(1.0, -1.0, -1.0)
19     glTexCoord2f(0.0, 1.0)
20     glVertex3f(-1.0, 1.0, -1.0)
21     glTexCoord2f(0.0, 0.0)
```

```
22      glVertex3f(-1.0,  1.0,  1.0)
23      glTexCoord2f(1.0, 0.0)
24      glVertex3f(1.0,  1.0,  1.0)
25      glTexCoord2f(1.0, 1.0)
26      glVertex3f(1.0,  1.0, -1.0)
27      glTexCoord2f(1.0, 1.0)
28      glVertex3f(-1.0, -1.0, -1.0)
29      glTexCoord2f(0.0, 1.0)
30      glVertex3f(1.0, -1.0, -1.0)
31      glTexCoord2f(0.0, 0.0)
32      glVertex3f(1.0, -1.0,  1.0)
33      glTexCoord2f(1.0, 0.0)
34      glVertex3f(-1.0, -1.0,  1.0)
35      glTexCoord2f(1.0, 0.0)
36      glVertex3f(1.0, -1.0, -1.0)
37      glTexCoord2f(1.0, 1.0)
38      glVertex3f(1.0,  1.0, -1.0)
39      glTexCoord2f(0.0, 1.0)
40      glVertex3f(1.0,  1.0,  1.0)
41      glTexCoord2f(0.0, 0.0)
42      glVertex3f(1.0, -1.0,  1.0)
43      glTexCoord2f(0.0, 0.0)
44      glVertex3f(-1.0, -1.0, -1.0)
45      glTexCoord2f(1.0, 0.0)
46      glVertex3f(-1.0, -1.0,  1.0)
47      glTexCoord2f(1.0, 1.0)
48      glVertex3f(-1.0,  1.0,  1.0)
49      glTexCoord2f(0.0, 1.0)
50      glVertex3f(-1.0,  1.0, -1.0)
51      glEnd()
```

Przy pomocy GL_QUADS rysowane jest 6 czworokątów(tutaj kwadratów). Wierzchołki podawane są w taki sposób, aby były widoczne z zewnętrznej strony sześcianu, dzięki czemu, sześcian jest w całości widoczny podczas obrotu.

7. Narysowanie jajka:

Listing 31: Funkcja draw_jajo()

```
1  def EggTriangles(tab_xyz, tab_colours):
2      glBegin(GL_TRIANGLES)
3      for i in range(N-1):
4          for j in range(N-1):
5              glColor3f(tab_colours[i,j,0], tab_colours[i,j,1],
                   tab_colours[i,j,2])
6              x_1=tab_xyz[i,j,0]
7              y_1=tab_xyz[i,j,1]-1
8              z_1=tab_xyz[i,j,2]
9
10             glColor3f(tab_colours[i,j+1,0], tab_colours[i,j+1,1],
                   tab_colours[i,j+1,2])
11             x_2=tab_xyz[i,j+1,0]
```

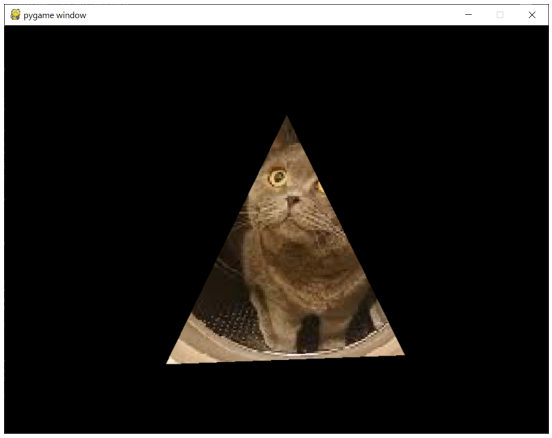
```
12              y_2=tab_xyz[i,j+1,1]-1
13              z_2=tab_xyz[i,j+1,2]
14
15              glColor3f(tab_colours[i+1,j,0], tab_colours[i+1,j,1],
                    tab_colours[i+1,j,2])
16              x_3=tab_xyz[i+1,j,0]
17              y_3=tab_xyz[i+1,j,1]-1
18              z_3=tab_xyz[i+1,j,2]
19              if i<N/2:
20                  glTexCoord2f(x_1,y_1/2+0.5)
21                  glVertex3f(x_1,y_1,z_1)
22
23                  glTexCoord2f(x_2,y_2/2+0.5)
24                  glVertex3f(x_2,y_2,z_2)
25
26                  glTexCoord2f(x_3,y_3/2+0.5)
27                  glVertex3f(x_3,y_3,z_3)
28
29              else:
30                  glTexCoord2f(x_1,y_1/2+0.5)
31                  glVertex3f(x_1,y_1,z_1)
32
33                  glTexCoord2f(x_3,y_3/2+0.5)
34                  glVertex3f(x_3,y_3,z_3)
35
36                  glTexCoord2f(x_2,y_2/2+0.5)
37                  glVertex3f(x_2,y_2,z_2)
38
39
40              if j == N-2:
41                  glColor3f(tab_colours[i,0,0], tab_colours[i,0,1],
                        tab_colours[i,0,2])
42              else:
43                  glColor3f(tab_colours[i,j+1,0], tab_colours[i,j+1,1],
                        tab_colours[i,j+1,2])
44              x_1=tab_xyz[i,j+1,0]
45              y_1=tab_xyz[i,j+1,1]-1
46              z_1=tab_xyz[i,j+1,2]
47
48              glColor3f(tab_colours[i+1,j,0], tab_colours[i+1,j,1],
                    tab_colours[i+1,j,2])
49              x_2=tab_xyz[i+1,j,0]
50              y_2=tab_xyz[i+1,j,1]-1
51              z_2=tab_xyz[i+1,j,2]
52
53              if j == N-2:
54                  glColor3f(tab_colours[i+1,0,0], tab_colours[i+1,0,1],
                        tab_colours[i+1,0,2])
55              else:
```

```
56                  glColor3f(tab_colours[i+1,j+1,0], tab_colours[i+1,j
                        +1,1], tab_colours[i+1,j+1,2])
57              x_3=tab_xyz[i+1,j+1,0]
58              y_3=tab_xyz[i+1,j+1,1]-1
59              z_3=tab_xyz[i+1,j+1,2]
60
61              if i<N/2:
62                  glTexCoord2f(x_1,y_1/2+0.5)
63                  glVertex3f(x_1,y_1,z_1)
64
65                  glTexCoord2f(x_3,y_3/2+0.5)
66                  glVertex3f(x_3,y_3,z_3)
67
68                  glTexCoord2f(x_2,y_2/2+0.5)
69                  glVertex3f(x_2,y_2,z_2)
70
71              else:
72                  glTexCoord2f(x_1,y_1/2+0.5)
73                  glVertex3f(x_1,y_1,z_1)
74
75                  glTexCoord2f(x_2,y_2/2+0.5)
76                  glVertex3f(x_2,y_2,z_2)
77
78                  glTexCoord2f(x_3,y_3/2+0.5)
79                  glVertex3f(x_3,y_3,z_3)
80
81      glEnd()
82
83  def draw_jajo():
84      tab_xyz = np.zeros((N,N,3))
85      for i in range(N):
86          u = i/(N-1)
87          for j in range(N):
88              v = j/(N-1)
89              tmp = (-90*u**5 + 225*u**4 - 270*u**3 + 180*u**2-45*u)
90              tab_xyz[i,j,0]=(tmp*np.cos(np.pi*v))/5
91              tab_xyz[i,j,1]=(160*u**4 - 320*u**3 + 160*u**2)/5
92              tab_xyz[i,j,2]=(tmp*np.sin(np.pi*v))/5
93
94      np.random.seed(1)
```

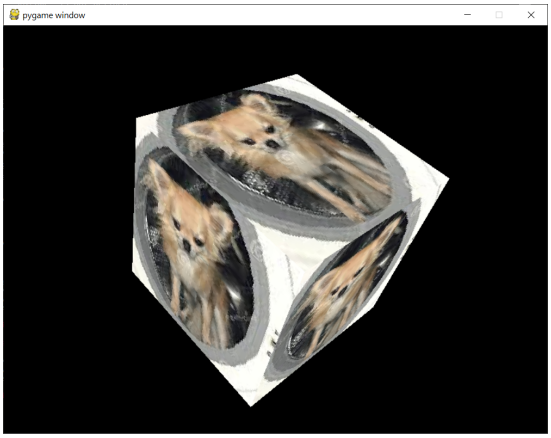
Funkcja ta, korzysta z kodu napisanego na zajęciach z ćwiczenia 3, wzbogaconego o teksturowanie. Jajko tworzone jest z trójkątów, współrzędne są tak samo generowane. Główną różnicą jest zmiana kolejności przekazywanych wierzchołków, ponieważ w starym rozwiązaniu, co drugi trójkąt był niewidoczny. Dodatkowo, aby tekstura zajmowała całą wysokość jajka, współrzędne tekstury, które są zależne od współrzędnych punktów jajka, są odpowiednio modyfikowane.

2.5.4 Efekt wykonanej pracy

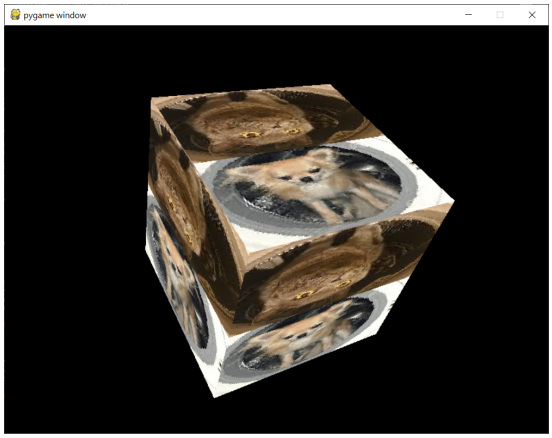
Poniżej przedstawione są 4 różne efekty pracy:



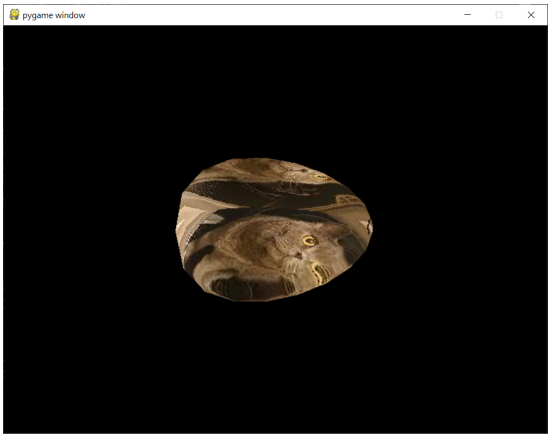
Rysunek 14: Kot w pralce w trójkącie



Rysunek 15: Pies w pralce na sześcianie



Rysunek 16: Kot i pies w pralkach na sześcianie



Rysunek 17: Kot w pralce na jajku