Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Grafika komputerowa i komunikacja człowiek-komputer			
Temat:	Laboratorium 5 – Oświetlanie scen		
Termin zajęć:	środa TN 14:15 – 17:15 semestr zimowy 2023/2024		
Autor:	Eryk Mika 264451	**Prowadzący:**	Dr inż. arch. Tomasz Zamojski

1. Cel laboratorium

Celem laboratorium było zapoznanie się z zagadnieniem oświetlania scen. Poznany został model oświetlenia Phonga. Implementowano obsługę źródeł światła za pomocą *OpenGL* – w tym celu wykorzystano także umiejętności zdobyte podczas wcześniejszych laboratoriów, w tym poświęconych programowej obsłudze zdarzeń myszy i klawiatury.

2. Kod wspólny dla wszystkich zadań

W zadaniach zostały w znacznej części wykorzystane funkcje i zmienne zdefiniowane w plikach dostarczonych do wcześniejszych laboratoriów. W skryptach będących rozwiązaniami zadań z tego laboratorium zostały zdefiniowane zmienne globalne, które odpowiadają parametrom z modelu Phonga i są wykorzystane do programowego manipulowania oświetleniem (Rysunek 2.1).

```
19  mat_ambient = [1.0, 1.0, 1.0, 1.0]
20  mat_diffuse = [1.0, 1.0, 1.0, 1.0]
21  mat_specular = [1.0, 1.0, 1.0, 1.0]
22  mat_shininess = 20.0
23
24  # Zrodlo swiatla 0
25  light_ambient = [0.1, 0.1, 0.0, 1.0]
26  light_diffuse = [0.8, 0.8, 0.0, 1.0]
27  light_specular = [1.0, 1.0, 1.0, 1.0]
28  light_position = [0.0, 0.0, 10.0, 1.0]
29
30  # Zrodlo swiatla 1
31  light_ambient1 = [0.02, 0.2, 0.0, 1.0]
32  light_diffuse1 = [1.0, 0.0, 1.0, 1.0]
33  light_specular1 = [1.0, 1.0, 1.0, 1.0]
34  light_position1 = [-10.0, 8.0, 0.0, 1.0]
35
36  att_constant = 1.0
37  att_linear = 0.05
38  att_quadratic = 0.001
```

Rysunek 2.1 Zmienne globalne wykorzystane do manipulowania składowymi modelu Phonga materiału oraz światła

W liniach 19-22 zdefiniowane są parametry materiałowe. Fragmenty kodu z linii 24-28 oraz 30-34 zawierają parametry poszczególnych źródeł światła. Listy *light_position* oraz *light_position1* definiują położenie źródeł światła za pomocą współrzędnych jednorodnych[1]. Jako ostatnie

[1] https://registry.khronos.org/OpenGL-Refpages/gl4/html/gl_Position.xhtml

zdefiniowane są składowe funkcji strat natężenia. W podanym listingu kodu zdefiniowane są składowe dla dwóch źródeł światła.

Należy rozróżnić 3 podstawowe składowe, które różnią się znaczeniem, których wektory wartości są także zdefiniowane na Rysunek 2.1. :

- *diffuse* – rozpraszanie – reprezentuje intensywność światła rozproszonego na powierzchni obiektu,
- *ambient* – otoczenie - reprezentuje intensywność światła otoczenia, czyli światła odbitego i rozproszonego przez otoczenie,
- *specular* – lustrzane odbicie - reprezentuje intensywność światła odbitego w kierunku widza lub kamery.

Zostały dodane wywołania funkcji, które uaktywniają model oświetlenia - *glMaterialfv()*, *glLightfv()*, *glLightf()*. Oświetlany model – sfera – jest tworzona za pomocą funkcji *gluSphere()* - odpowiedni kod został dostarczony w instrukcji przez Prowadzącego.

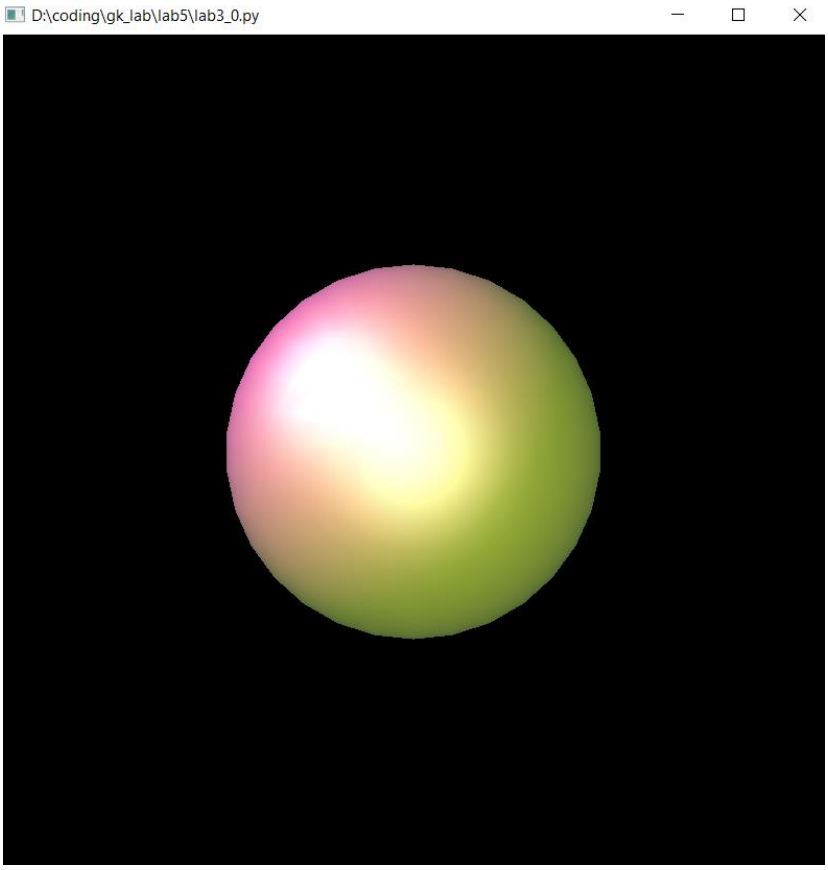
3. Zadanie na ocenę 3.0 – skrypt *lab3_0.py*

W zadaniu tym należało wprowadzić drugie źródło światła identyfikowane jako *GL_LIGHT1* – obok istniejącego już w kodzie *GL_LIGHT0*. W tym celu zostały wykorzystane wektory wartości z Rysunek 2.1. W funkcji *startup()* zostały dodane analogiczne wywołania funkcji dla nowego źródła światła (Rysunek 3.1).

```
61      # Zrodlo swiatla 1
62      glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient1)
63      glLightfv(GL_LIGHT1, GL_DIFFUSE, light_diffuse1)
64      glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular1)
65      glLightfv(GL_LIGHT1, GL_POSITION, light_position1)
66
67      glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, att_constant)
68      glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, att_linear)
69      glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, att_quadratic)
70
71      glShadeModel(GL_SMOOTH)
72      glEnable(GL_LIGHTING)
73      glEnable(GL_LIGHT0)
74      glEnable(GL_LIGHT1)
```

Rysunek 3.1 Zmodyfikowany fragment funkcji startup()

W kolejnych wywołaniach funkcji następuje ustawienie określonych składowych opisanych poprzednio, pozycji źródła światła i składowych funkcji strat natężenia. Wywołanie funkcji *glEnable()* dla odpowiedniego źródła światła powoduje jego uaktywnienie. Rezultaty wprowadzonych zmian przedstawia Rysunek 3.2.



Rysunek 3.2 Efekt działania skryptu na ocenę 3.0 – sfera oświetlana dwoma źródłami światła

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

W zadaniu tym należało wprowadzić dynamiczną zmianę składowych koloru światła w celu zaobserwowania ich wpływu. Ograniczono się do jednego źródła światła – wspominanego wcześniej *GL_LIGHT0*.

Wprowadzono szereg nowych zmiennych globalnych (Rysunek 4.1).

```
33  # Flagi - czy klawisze w gore/dol nacisniete
34  key_down = False
35  key_up = False
36  # Flagi - czy modyfikowana dana skladowa
37  specular = False
38  ambient = False
39  diffuse = False
40  # Indeks modyfikowanego elementu listy skladowej
41  index = 0
42  # Modyfikowana lista wartosci skladowej koloru swiatla (referencja)
43  current_mode = light_specular
```

Rysunek 4.1 Zmienne globalne użyte w skrypcie na ocenę 3.5

Kolejne zmienne (tzw. flagi) sygnalizują, czy klawisze w górę/dół są naciśnięte, która składowa jest obecnie modyfikowana, który indeks 4-elementowego wektora składowej jest modyfikowany (numerowany od 0). Ostatnia zmienna to właściwie **referencja** na tablicę (listę) przechowującą wartości modyfikowanej składowej.

```
142     # Obsluga klawiszy w gore/w dol
143     if key == GLFW_KEY_DOWN and action == GLFW_REPEAT:
144         key_down = True
145     else:
146         key_down = False
147     if key == GLFW_KEY_UP and action == GLFW_REPEAT:
148         key_up = True
149     else:
150         key_up = False
151     # Obsluga klawiszy zmieniajacych flagi wskazujace na modyfikowana obecnie skladowa
152     if key == GLFW_KEY_A and action == GLFW_PRESS:
153         ambient = True
154     else:
155         ambient = False
156     if key == GLFW_KEY_D and action == GLFW_PRESS:
157         diffuse = True
158     else:
159         diffuse = False
160     if key == GLFW_KEY_S and action == GLFW_PRESS:
161         specular = True
162     else:
163         specular = False
164     # Obsluga klawiszy zmieniajacych flagi wskazujace na indeks modyfikowanego elementu skladowej
165     if key == GLFW_KEY_0 and action == GLFW_PRESS:
166         index = 0
167     if key == GLFW_KEY_1 and action == GLFW_PRESS:
168         index = 1
169     if key == GLFW_KEY_2 and action == GLFW_PRESS:
170         index = 2
171     if key == GLFW_KEY_3 and action == GLFW_PRESS:
172         index = 3
```

Rysunek 4.2 Obsługa zdarzeń związanych z flagami – zmiennymi globalnymi

W celu zmiany stanów flag zmodyfikowana została funkcja *keyboard_key_callback()* (Rysunek 4.2). Przytrzymanie klawiszy w górę/w dół (akcja określona jako *GLFW_REPEAT*) powoduje odpowiednie zmiany flag związanych z tymi klawiszami. Naciśnięcie (akcja *GLFW_PRESS*) klawiszy *a*, *d*, *s* powoduje zmianę stanu flag związanych odpowiednio ze składowymi *ambient*, *diffuse* oraz *specular*. Naciśnięcie klawiszy 0, 1, 2, 3 powoduje odpowiednią zmianę zmiennej *index*.

W funkcji *render()* (Rysunek 4.3) następuje przypisanie referencji do odpowiedniej składowej na podstawie stanu flag. Pomocniczo, aktualna składowa jest wypisywana w konsoli (wywołanie funkcji *print()*). W przypadku przytrzymania klawiszy w górę/w dół następuje odpowiednio jednostkowe zwiększenie lub zmniejszenie danego elementu (wskazywanego przez *index*) danej składowej o 0,01. Użycie funkcji *min()* oraz *max()* ma na celu zapobieżenie przekroczeniu dozwolonych zakresów [-1.0, 1.0]. Pomocniczo, *index* oraz lista danej składowej są wypisywane w konsoli. Ostatecznie, wywołania funkcji *glLightfv()* powodują aktualizację stanu światła. Przykładowe efekty pracy ze skryptem przedstawia Rysunek 4.4.

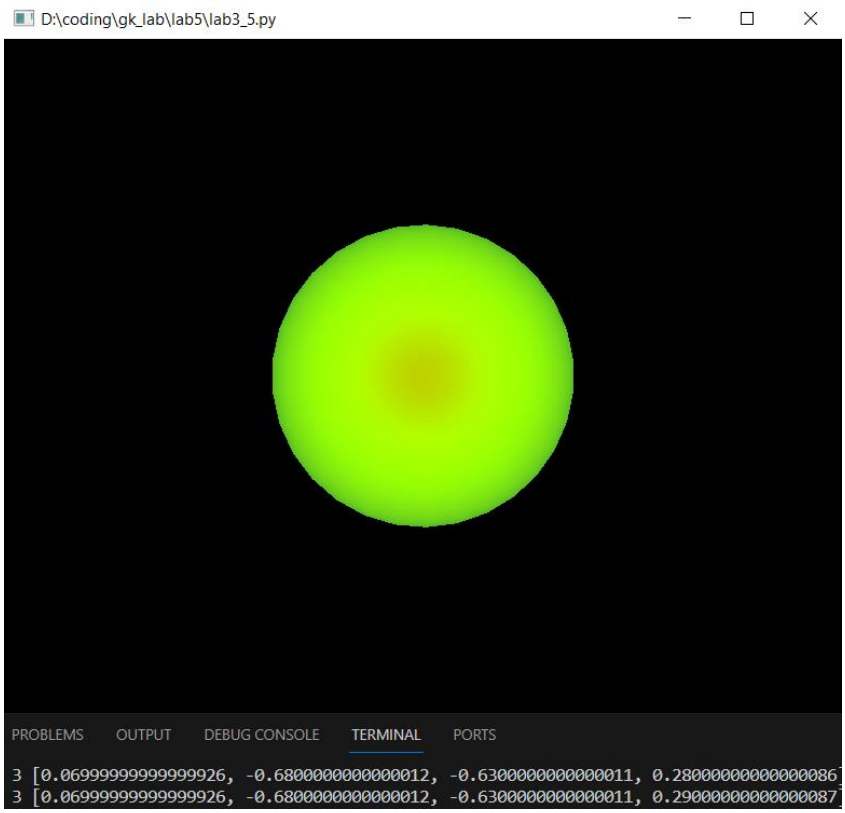
Sposób obsługi skryptu: **klawisze a**, **d**, **s** – wybór składowej ***a**mbient*, ***d**iffuse* oraz ***s**pecular*, **klawisze 0**, **1**, **2**, **3** – wybór elementu składowej, **klawisze w górę/dół** – zwiększanie/zmniejszanie.

```python
    # Wybor modyfikowanej skladowej
    if specular:
        print("Specular")
        current_mode = light_specular
    elif diffuse:
        print("Diffuse")
        current_mode = light_diffuse
    elif ambient:
        print("Ambient")
        current_mode = light_ambient

    # Zwiekszenie/zmniejszenie w zaleznosci od nacisnietego klawisza
    if key_up:
        current_mode[index] = min(current_mode[index] + 0.01, 1.0)
        print(index, current_mode, sep=" ")
    elif key_down:
        current_mode[index] = max(current_mode[index] - 0.01, 0.0)
        print(index, current_mode, sep=" ")

    # Aktualizacja skladowych koloru swiatla
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular)
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse)
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient)
```

Rysunek 4.3 Zmodyfikowany fragment funkcji render() do zadania na ocenę 3.5



Rysunek 4.4 Obserwacja dynamicznej zmiany wartości składowych – zmiany oświetlenia sfery

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

W zadaniu tym należało dodać poruszanie źródłami światła oraz ich wizualizację. Podobnie jak w przypadku zadań z laboratorium nr 4, zostały zdefiniowane zmienne globalne związane z obsługą ruchu myszy.

```
13  theta = 0.0
14  phi = 0.0
15  theta1 = 0.0
16  phi1 = 0.0
17  pix2angle = 1.0
18
19  left_mouse_button_pressed = 0
20  right_mouse_button_pressed = 0
21  mouse_x_pos_old = 0
22  mouse_y_pos_old = 0
23  delta_x = 0
24  delta_y = 0
25  LIGHT_DISTANCE = 5
```

Rysunek 5.1 Zmienne globalne użyte w skrypcie na ocenę 4.0

Zostały zdefiniowane pary kątów *theta*, *phi* oraz *theta1*, *phi1* dla obsługi ruchu myszy przy wciśniętym odpowiednio lewym lub prawym przycisku myszy. Są one użyte do określenia położenia świateł *GL_LIGHT0* oraz *GL_LIGHT1*. Wciśnięcie danego przycisku myszy jest sygnalizowane przez zmienne *left_mouse_button_pressed* oraz *right_mouse_button_pressed* a ich ustawianie odbywa się identycznie jak podczas poprzedniego laboratorium – podobnie z ustaleniem położenia myszy – zmienne z linii 21-24 (Rysunek 5.1). Ostatnia zmienna *LIGHT_DISTANCE* jest właściwie stałą w skrypcie i określa ona odległość źródeł światła od przyjętego środka.

W zmodyfikowanej funkcji render tworzony jest oświetlany model – w sposób przedstawiony w przykładowym kodzie. Następnie obliczane są położenia dwóch źródeł światła na podstawie wartości odpowiednich kątów – zgodnie ze wzorami dostarczonymi przez Prowadzącego (Rysunek 5.2).

```
101     quadric = gluNewQuadric()
102     gluQuadricDrawStyle(quadric, GLU_FILL)
103     gluSphere(quadric, 3.0, 8, 8)
104     gluDeleteQuadric(quadric)
105
106     # Obliczenie pozycji swiatla 0
107     x_s = LIGHT_DISTANCE * cos(pi*theta/180) * cos(pi*phi/180)
108     y_s = LIGHT_DISTANCE * sin(pi*phi/180)
109     z_s = LIGHT_DISTANCE * sin(pi*theta/180) * cos(pi*phi/180)
110
111     # Obliczenie pozycji swiatla 1
112     x_s1 = LIGHT_DISTANCE * cos(pi*theta1/180) * cos(pi*phi1/180)
113     y_s1 = LIGHT_DISTANCE * sin(pi*phi1/180)
114     z_s1 = LIGHT_DISTANCE * sin(pi*theta1/180) * cos(pi*phi1/180)
```

Rysunek 5.2 Tworzenie oświetlanego modelu oraz obliczenie pozycji świateł w funkcji render().

Następnie, wizualizacje dwóch źródeł światła są tworzone i przemieszczane na odpowiednie pozycje, korzystając z wartości uzyskanych z obliczeń z Rysunek 5.2. Umieszczenie źródeł w odpowiednim miejscu następuje poprzez wywołanie funkcji *glTranslatef()*. Po stworzeniu wizualizacji, transformacje są odwracane poprzez wykorzystanie identycznych, ale zanegowanych parametrów. Umożliwia to poprawną kontynuację działania całego algorytmu.

```
116     # Wizualizacja zrodla swiatla 0
117     glTranslatef(x_s, y_s, z_s)
118     quadric = gluNewQuadric()
119     gluQuadricDrawStyle(quadric, GLU_LINE)
120     gluSphere(quadric, 0.5, 6, 5)
121     gluDeleteQuadric(quadric)
122     glTranslatef(-x_s, -y_s, -z_s)
123
124     # Wizualizacja zrodla swiatla 1
125     glTranslatef(x_s1, y_s1, z_s1)
126     quadric = gluNewQuadric()
127     gluQuadricDrawStyle(quadric, GLU_LINE)
128     gluSphere(quadric, 0.5, 6, 5)
129     gluDeleteQuadric(quadric)
130     glTranslatef(-x_s1, -y_s1, -z_s1)
```

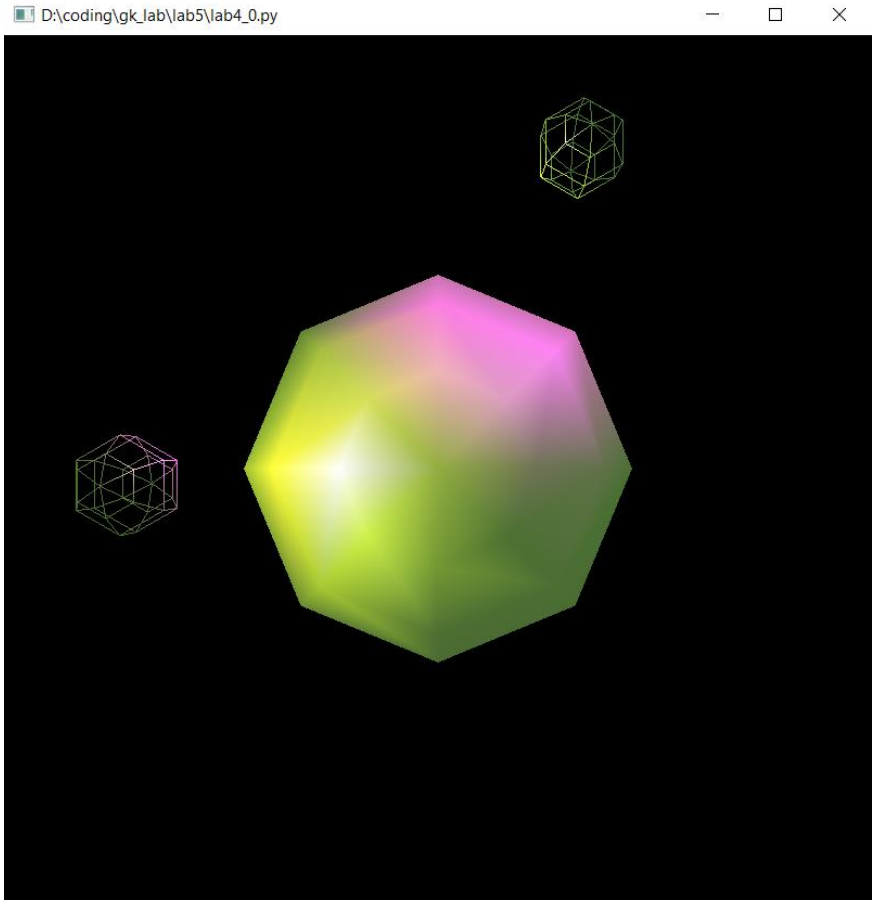
Rysunek 5.3 Tworzenie wizualizacji źródeł światła

Ostatecznie, listy zawierające wartości położenia dwóch świateł – *light_position* oraz *light_position1* – są aktualizowane na podstawie wartości obliczonych w etapie z Rysunek 5.2. Ustalane są nowe zmiany kątów położenia świateł na podstawie ruchu myszą – wybór światła w zależności od tego, czy wciśnięty jest **lewy lub prawy przycisk myszy** (Rysunek 5.4).

```
132     # Pozycja swiatla 0
133     light_position[0] = x_s
134     light_position[1] = y_s
135     light_position[2] = z_s
136
137     # Pozycja swiatla 1
138     light_position1[0] = x_s1
139     light_position1[1] = y_s1
140     light_position1[2] = z_s1
141
142     # Aktualizacja swiatla
143     glLightfv(GL_LIGHT0, GL_POSITION, light_position)
144     glLightfv(GL_LIGHT1, GL_POSITION, light_position1)
145
146     if left_mouse_button_pressed:
147         theta -= delta_x * pix2angle
148         phi -= delta_y * pix2angle
149     elif right_mouse_button_pressed:
150         theta1 -= delta_x * pix2angle
151         phi1 -= delta_y * pix2angle
152
```

Rysunek 5.4 Aktualizacja pozycji światła

Efekt działania skryptu w postaci przemieszczonych i zwizualizowanych źródeł światła przedstawia Rysunek 5.5.



Rysunek 5.5 Efekt działania skryptu na ocenę 4.0

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

W zadaniu tym należało dodać wektory normalne do modelu jajka. Został wykorzystany kod tworzący model jajka z użyciem prymitywu *GL_TRIANGLE_STRIP* przedstawiony w laboratorium 4.

Obok listy *tab* przechowującej wartości *x*, *y*, *z* dla danych par wartości *u* i *v*, została zdefiniowana tablica *normal*, która zawiera współrzędne wektorów normalnych dla tych par (Rysunek 6.1).

```
13    tab = [[[0] * 3 for i in range(N)] for j in range(N)]
14    normal = [[[0] * 3 for i in range(N)] for j in range(N)]
```

Rysunek 6.1 Listy globalne

Następnie, w tej samej pętli, w której wypełniana jest lista *tab*, obliczane są pochodne cząstkowe dla danej pary (*u*, *v*) – zgodnie ze wzorami dostarczonymi przez Prowadzącego. Są one następnie użyte do obliczenia składowych wektora normalnego, który jest normalizowany poprzez podzielenie wszystkich jego składowych przez długość wektora (*vector_length*). Jednoliniowa instrukcja warunkowa *if* ma na celu zabezpieczenie przed dzieleniem przez 0 (Rysunek 6.2).

```
# Obliczamy wartosci x, y, z
for i in range(N):
    for j in range(N):
        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])
        tab[i][j][1] = 160 * u[i] ** 4 - 320 * u[i] ** 3 + 160 * u[i] * u[i] - 5
        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])
        # Pochodne czastkowe obliczone wg wzorow
        xu = (-450 * u[i]**4 + 900 * u[i]**3 - 810 * u[i]**2 + 360 * u[i] - 45) * cos(pi * v[j])
        xv = pi * (90 * u[i]**5 - 225 * u[i]**4 + 270 * u[i]**3 - 180 * u[i]**2 + 45 * u[i]) * sin(pi * v[j])
        yu = 640 * u[i]**3 - 960 * u[i]**2 + 320 * u[i]
        yv = 0
        zu = (-450 * u[i]**4 + 900 * u[i]**3 - 810 * u[i]**2 + 360 * u[i] - 45) * sin(pi * v[j])
        zv = -pi * (90 * u[i]**5 - 225 * u[i]**4 + 270 * u[i]**3 - 180 * u[i]**2 + 45 * u[i]) * cos(pi * v[j])
        # Wartosci skladowych wektora normalnego
        a = yu * zv - zu * yv
        b = zu * xv - xu * zv
        c = xu * yv - yu * xv
        # Dlugosc wektora normalnego
        vector_length = sqrt(a*a+b*b+c*c)
        # Normalizacja
        normal[i][j][0] = a / vector_length if vector_length!=0 else 0
        normal[i][j][1] = b / vector_length if vector_length!=0 else 0
        normal[i][j][2] = c / vector_length if vector_length!=0 else 0
```

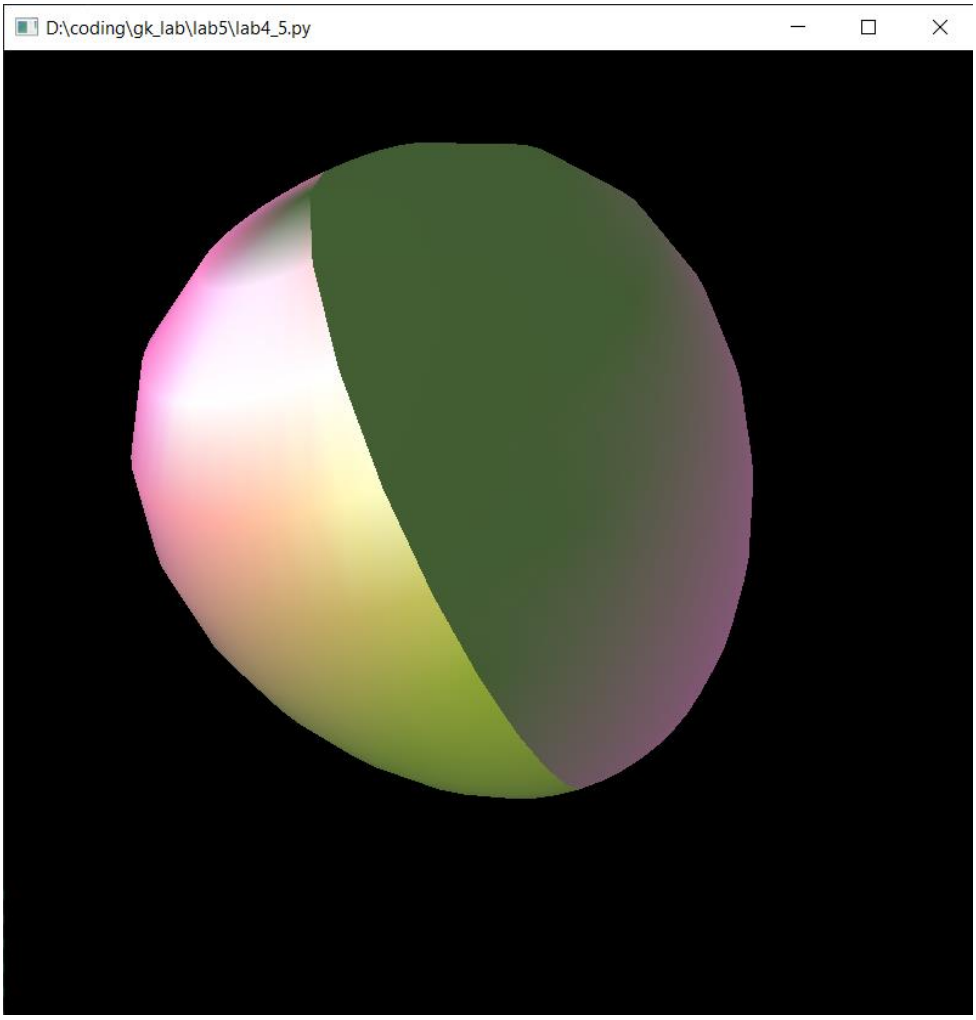
Rysunek 6.2 Wyznaczanie i normalizacja wektorów normalnych

W funkcji *render()* następuje skojarzenie wektorów normalnych z poszczególnymi wierzchołkami jaja – dokonane jest to poprzez wywołanie funkcji *glNormal()* tuż przed danym wywołaniem funkcji *glVertex3f()*. Wywołanie funkcji *spin()* – także pochodzącej z laboratorium 4 – powoduje obracanie modelu jajka i umożliwia wygodniejszą obserwację zachodzących zjawisk (Rysunek 6.3).

```
    spin(time * 180 / 3.1415)
    colorIndex = 0
    for j in range(N-1):
        glBegin(GL_TRIANGLE_STRIP)
        for i in range(N-1):
            glNormal(normal[i][j][0], normal[i][j][1], normal[i][j][2])
            glVertex3f(tab[i][j][0], tab[i][j][1], tab[i][j][2])
            glNormal(normal[i+1][j][0], normal[i+1][j][1], normal[i+1][j][2])
            glVertex3f(tab[i+1][j][0], tab[i+1][j][1], tab[i+1][j][2])
            glNormal(normal[i][j+1][0], normal[i][j+1][1], normal[i][j+1][2])
            glVertex3f(tab[i][j+1][0], tab[i][j+1][1], tab[i][j+1][2])
            glNormal(normal[i+1][j+1][0], normal[i+1][j+1][1], normal[i+1][j+1][2])
            glVertex3f(tab[i+1][j+1][0], tab[i+1][j+1][1], tab[i+1][j+1][2])
            colorIndex+=1
        glEnd()
    glFlush()
```

Rysunek 6.3 Zmodyfikowany fragment funkcji render()

Efekt działania skryptu jest zgodny ze wskazówkami zawartymi w instrukcji do laboratorium – oświetlenie zachowuje się **poprawnie na połowie modelu** (Rysunek 6.4).



Rysunek 6.4 Efekt działania skryptu na ocenę 4.5

7. Bibliografia

[1] Składowe i ich znaczenie - https://www.khronos.org/opengl/wiki/How_lighting_works