Sprawozdanie Lab 3

Wojciech Kosierkiewicz 272926

12 listopada 2024

1 Wstęp

Podczas tych laboratoriów uczyłem się jak tworzyć obiekty 3D. Zadania które musiałem wykonać polegały na stworzeniu jajka w 3 wymiarach i wyświetlenie go na różne sposoby. Pierwsze przez kropki, potem przez linie a następnie przez trójkąty. W ostatnim zdaniu musiałem użyć Triangle strip. Jajko musiało być nieprzerwane oraz obracać się podczas wyświetlania.

2 Zadania

2.1 zadanie 1

By wykonać zadanie pierwsze musiałem pierwsze stworzyć macierz w której będę mógł przechowywać wszystkie swoje punkty a następnie uzupełnić ją punktami w odpowiednich pozycjach na osiach x,y i z. Na szczęście nie jestem pierwszą osobą która chciała malować jajka i matematycy stworzyli wzory podwalające na wyliczenie koordynatów każdego z punktów. Te wzory zostały zaimplementowane poniżej.

```
def getx(u,v):
    return (-90 * pow(u,5)+225 * pow(u,4) -270 * pow(u,3)
     + 180 * u*u - 45*u)*math.cos(math.pi*v)

def gety(u,v):
    return (160 * pow(u,4) - 320 * pow(u,3)
     + 160 * u * u - 5)

def getz(u,v):
    return (-90 * pow(u,5) + 225 * pow(u,4) - 270 * pow(u,3)
     +180 * pow(u,2) - 45*u) * math.sin(math.pi*v)
```

Rysunek 1: Funkcje użyte do generowania punktów

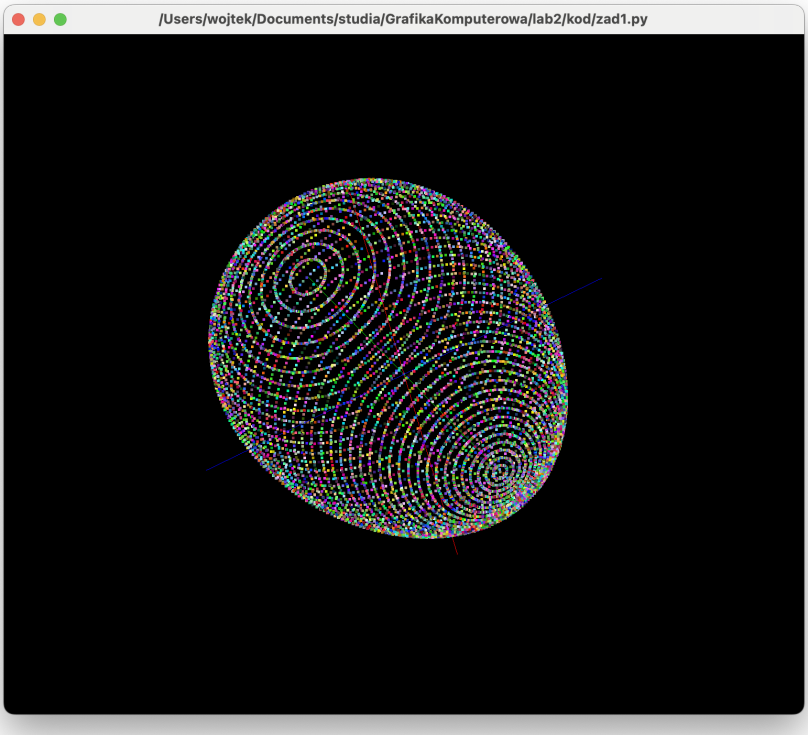
Powyżej napisane funkcje wywoływałem dla każdego punktu jajka którego chciałem wyświetlić i przypisałem do macierzy.

```
N = 100
points = [[[0] * 3 for i in range(N)] for j in range(N)];

def fillpoints(N):
    for i in range(N):
        for j in range(N):
            if i != 0 and j != 0:
                points[i][j][0]= getx(i/N,j/N)
                points[i][j][1]= gety(i/N,j/N)
                points[i][j][2]= getz(i/N,j/N)
```

Posiadając już wszystkie punkty tworzące moje jajko pozostało mi jedynie je wyświetlić oraz obracać. Zrobiłem oba w poniższym kodzie. Funkcja spin odpowiadała za obrót jajka w wszystkich osiach o podany kąt w zmiennej angle. Podczas wyświetlania używałem czasu do obliczania kątu obrotu.

```
def spin(angle):
    glRotatef(angle,1.0,0.0,0.0)
    glRotatef(angle,0.0,1.0,0.0)
    glRotatef(angle,0.0,0.0,1.0)

def render(time):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    spin(time * 180 / 3.1415)
    glPointSize(5)
    axes()
    glBegin(GL_POINTS)
    for i in range(N):
        for j in range(N):
            glColor3f(random.uniform(0,1),random.uniform(0,1),random.uniform(0,1))
            glVertex3f(points[i][j][0],points[i][j][1],points[i][j][2])
    glEnd()
    glFlush()
```

Efektem końcowym było jajko stworzone z pojedynczych kolorowych punktów.



Rysunek 2: Wynik zadania 1

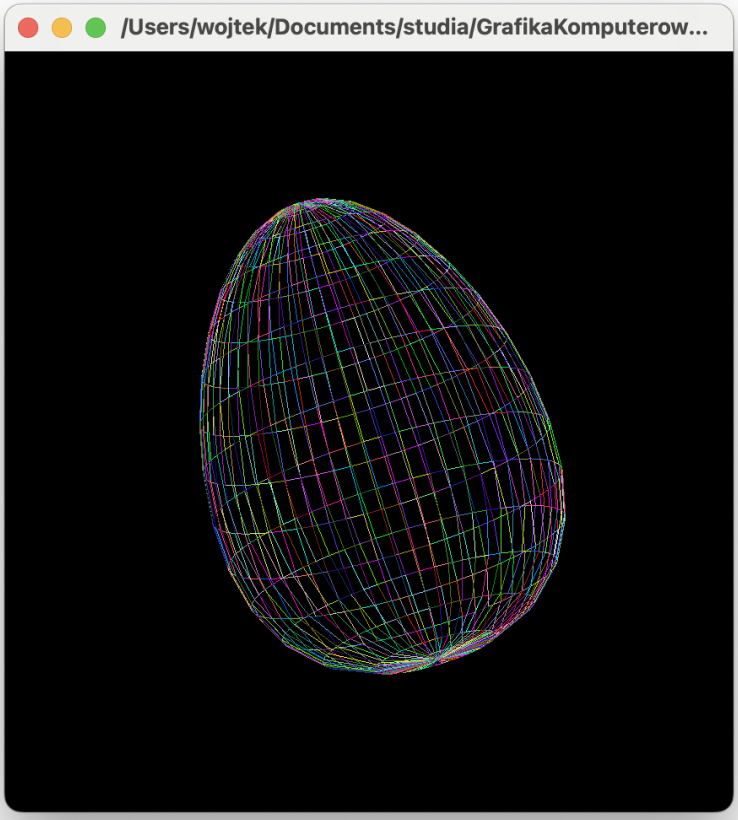
2.2 zadanie 2

W zadaniu 2 mogłem ponownie wykorzystać większość mojego poprzedniego kodu ale musiałem zmienić kolejność wyświetlania elementów by móc uzyskać połączone jajko. Dlatego dodawałem po dwa połączenia dla każdego punktu. Pierwsze połączenie $(i, j) \rightarrow (i + 1, j)$ a potem $(i, j) \rightarrow (i, j + 1)$.

```
glBegin(GL_LINES)
for i in range(N)
        for j in range(N):
                #(i,j)->(i+1,j)
                glVertex3f(points[i][j][0],points[i][j][1],points[i][j][2])
                glVertex3f(points[i+1][j][0],points[i+1][j][1],points[i+1][j][2])
                #(i,j)->(i,j+1)
                glVertex3f(points[i][j][0],points[i][j][1],points[i][j][2])
                glVertex3f(points[i][j+1][0],points[i][j+1][1],points[i][j+1][2])
```

Dzięki takiej kolejności dodawaniu punktów do wyświetlania uzyskałem jajko stworzona z połączonych lini. By uzyskać losowe kolory bez migotania stworzyłem drugą identyczną macierz do tej w której przechowywałem pozycje punktów i wypełniałem ją losowymi kolorami które potem używałem do kolorowania każdej z linii.

```
clrs = [[[0.0] * 3 for i in range(N+1)] for j in range(N+1)];
def colors(N):
    for i in range(N+1):
        for j in range(N+1):
            random.seed(i*j)
            clrs[i][j][0]= random.uniform(0.0,1.0)
            clrs[i][j][1]= random.uniform(0.0,1.0)
            clrs[i][j][2]= random.uniform(0.0,1.0)
```

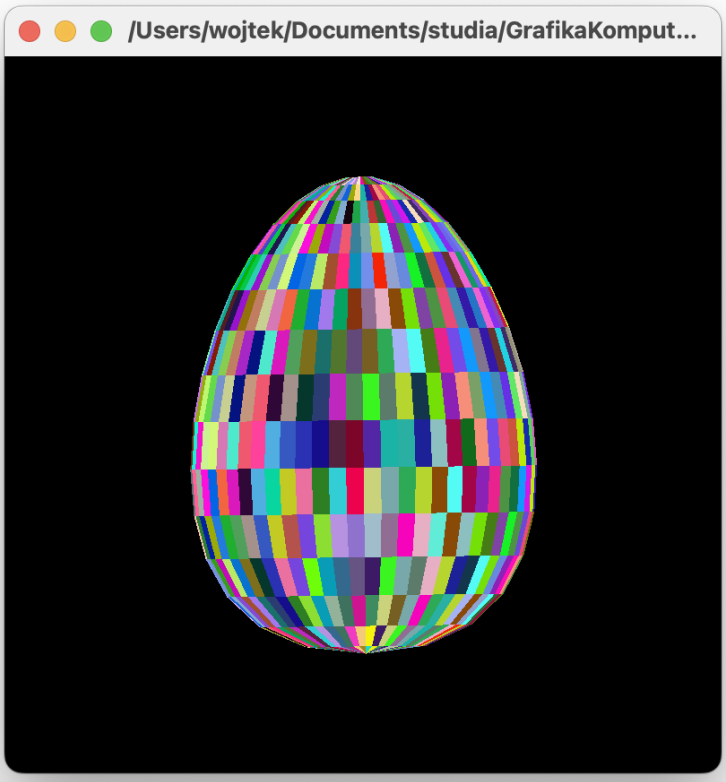


Rysunek 3: Wynik zadania 2

2.3 zadanie 3

Jedyne konieczne zmiany w zadaniu 3 były w kolejności wyświetlania. Natomiast dowiedziałem się także o możliwości użycia funkcji glVertex3fv zamiast glVertex3f która przyjmuje całą macierz współrzędnych na raz zamiast przekazywania każdego elementu. Natomiast by wyświetlić wszystkie prostokąty używając trójkątów musiałem stworzyć dwa trójkąty dzielące ze sobą wierzchołki $(i, j+1), (i+1, j)$ ale 3 wierzchołek musiał być przeciwny, tworzyłem więc dwa trójkąty o współrzednych $(i, j), (i+1, j), (i, j+1)$, $(i+1, j+1), (i+1, j), (i, j+1)$.

```
glBegin(GL_TRIANGLES)
for i in range(N):
        for j in range(N):
                glColor3fv(clrs[i+1][j+1])
                #pierwszy trójkąt
                glVertex3fv(points[i][j])
                glVertex3fv(points[i+1][j])
                glVertex3fv(points[i][j+1])
                #drugi trójkąt
                glVertex3fv(points[i+1][j+1])
                glVertex3fv(points[i][j+1])
                glVertex3fv(points[i+1][j])
```

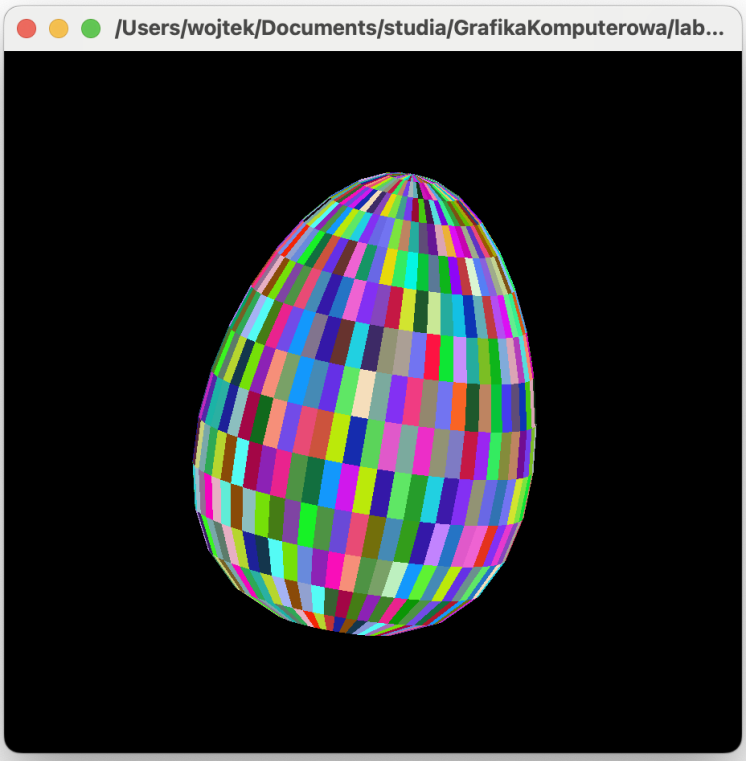


Rysunek 4: Wynik zadania 3

2.4 zadanie 4

Zadanie 4 nie było trudne i jedynie wymagało przekazywania na przemian punktów z górnego i dolnego wiersza. wykonałem to w poniższym kodzie.

```
for i in range(N):
        for j in range(N):
                glColor3fv(clrs[i+1][j+1])

                glVertex3fv(points[i][j])
                glVertex3fv(points[i+1][j])
                glVertex3fv(points[i][j+1])
                glVertex3fv(points[i+1][j+1])
```



Rysunek 5: Wynik zadania 4

3 wnioski

Obsługa obiektów 3D nie jest dużo trudniejsza od obsługi obiektów 2D. Wszystkie funkcje są zaskakująco podobne.