Sprawozdanie
Grafika komputerowa i komunikacja człowiek-komputer
Laboratorium 2

Katsiaryna Kolyshko 276708

Dr. Inż. Jan Nikodem

Wstęp teoretyczny

Zadanie laboratoryjne polega na narysowaniu jajka na 3 różne sposoby: za pomocą punktów, za pomocą linii, za pomocą trójkątów.

Podstawy modelowania 3D w grafice komputerowej:

Modelowanie 3D w grafice komputerowej polega na tworzeniu iluzji głębi na dwuwymiarowym ekranie. Klasycznym przykładem jest „Utah Teapot", model stworzony w 1975 roku przez Martina Newella, badacza grafiki komputerowej na University of Utah. Ten prosty model teapotu pomógł wczesnym programistom graficznym eksperymentować z oświetleniem, cieniowaniem i technikami renderowania. Podobnie jak w przypadku modelowania jajka, obiekty te są reprezentowane przez zestawy wierzchołków, krawędzi i powierzchni obliczonych w przestrzeni 3D i rzutowanych na ekran 2D.

W naszym przypadku utworzenie jajka wymaga zdefiniowania jego struktury geometrycznej za pomocą serii równań w celu obliczenia położenia każdego punktu. Te punkty lub wierzchołki są rozmieszczone za pomocą funkcji matematycznych, które definiują krzywiznę obiektu i są zwykle połączone trójkątnymi ścianami, ponieważ trójkąty są najprostszymi jednostkami do definiowania powierzchni 3D w OpenGL.

Matematyka za modelami 3D:

Aby kształty 3D poprawnie wyświetlały się na ekranie, używamy transformacji do pozycjonowania, skalowania i obracania naszego modelu. Macierze są podstawą tych transformacji w grafice komputerowej, ponieważ pozwalają nam przekształcać każdy punkt obiektu za pomocą łatwych do opanowania obliczeń.

- „Translation" przesuwa obiekty wzdłuż osi X, Y i Z.
- Obrót zmienia orientację obiektów wokół osi.
- Skalowanie dostosowuje rozmiar obiektów we wszystkich lub niektórych osiach.

W OpenGL transformacje te są łączone w celu manipulowania sposobem wyświetlania obiektu na ekranie. Macierze umożliwiają stosowanie transformacji do każdego wierzchołka obiektu, definiując, w jaki sposób punkty odnoszą się do siebie w przestrzeni 3D.

Znając, czym jest Obrót i Skalowanie, miałam pytanie: czym jest to „Translation" i jak ono przesuwa obiekty?

W grafice 3D translacja to rodzaj transformacji, która przesuwa obiekt z jednej pozycji do drugiej wzdłuż osi X, Y lub Z. Nie zmienia to kształtu, rozmiaru ani orientacji obiektu — tylko jego położenie w przestrzeni 3D.

Jak działa translacja w modelowaniu 3D:

Wektor translacji:

Translacja jest definiowana przez wektor, który określa, o ile obiekt powinien się przesunąć wzdłuż każdej osi. Na przykład wektor translacji (3, -2, 5) oznaczałby przesunięcie obiektu o 3 jednostki wzdłuż osi X, -2 jednostki wzdłuż osi Y i 5 jednostek wzdłuż osi Z.

Wpływ na współrzędne:

Gdy obiekt jest translowany, wektor translacji jest dodawany do współrzędnych każdego punktu w obiekcie. Tak więc, jeśli punkt początkowo znajduje się w (x, y, z), translacja o (tx, ty, tz) przesunie go do nowej pozycji (x + tx, y + ty, z + tz).

Reprezentacja macierzowa:

W grafice komputerowej translacje są często stosowane za pomocą macierzy. Macierz translacji umożliwia łatwe stosowanie transformacji sekwencyjnych, wraz ze skalowaniem, obrotem i innymi transformacjami. W jednorodnych współrzędnych macierz translacji dla wektora translacji (tx, ty, tz) jest następująca:

$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tworzenie obiektów za pomocą macierzy:

Każdy punkt (wierzchołek) na powierzchni obiektu można zapisać w macierzy, przy czym każdy wpis reprezentuje współrzędną X, Y lub Z. Na przykład utworzenie kształtu jajka wymaga obliczenia współrzędnych na podstawie równań definiujących jego krzywiznę. Współrzędne te są przechowywane w macierzy, a różne transformacje (takie jak obrót) są stosowane w celu nadania obiektowi dynamicznego, trójwymiarowego wyglądu.

Renderowanie modeli 3D: projekcja z 3D do 2D:

Po umieszczeniu modelu w przestrzeni 3D, OpenGL wyświetla go na ekranie 2D. Ten proces projekcji obejmuje tłumaczenie współrzędnych 3D na współrzędne 2D poprzez symulację perspektywy, w której obiekty znajdujące się dalej wydają się mniejsze, tworząc głębię. Potok OpenGL przetwarza te transformacje i projekcje wydajnie, dzięki czemu obiekty wyglądają tak, jakby istniały w środowisku 3D.

OpenGL wykorzystuje GPU do przetwarzania złożonych obliczeń dla grafiki 3D w czasie rzeczywistym. Jego proces renderowania zaczyna się od przetwarzania wierzchołków, które oblicza pozycję każdego wierzchołka po zastosowaniu transformacji. Następnie przycinanie usuwa wszystkie wierzchołki poza obszarem wyświetlania w celu optymalizacji wydajności. Następnie kształty są konwertowane na piksele w procesie zwanym rasteryzacją, po którym następuje przetwarzanie fragmentów, które określa kolor i głębię każdego piksela, włączając oświetlenie i cieniowanie dla realizmu. Razem te etapy umożliwiają OpenGL renderowanie interaktywnych modeli 3D, takich jak model jajka lub kula, na płaskim ekranie klatka po klatce.

Proces rysowania jajka:

Proces modelowania jajka obejmuje konstruowanie go z coraz bardziej złożonych kształtów — punktów, linii i trójkątów — co pozwala nam osiągnąć różne poziomy szczegółowości i realizmu w renderowaniu. To stopniowe podejście jest zgodne z zasadami grafiki komputerowej, w której coraz bardziej wyrafinowane prymitywy geometryczne są używane do tworzenia iluzji form trójwymiarowych.

Budowanie modelu jajka za pomocą punktów:

Zaczynając od punktów, konstruujemy podstawowy zarys 3D. Każdy punkt jest przechowywany w macierzy 3D [N][N][3], gdzie każdy wpis zawiera współrzędne (x, y, z) dla tego punktu, obliczone za pomocą równań parametrycznych dla kształtu jajka. W kodzie inicjujemy to za pomocą np. zeros ((N, N, 3)) z numpy. Tablice parametrów u i v — każda z zakresem od 0 do 1 — są używane do obliczania współrzędnych powierzchni kształtu, a glVertex3f() renderuje te punkty za pomocą GL_POINTS, aby utworzyć zarys.

Budowanie modelu jajka za pomocą linii:

Następnie łączymy punkty liniami za pomocą GL_LINES, dodając strukturę, aby zarysować krzywiznę jajka. Każdy wierzchołek (i, j) łączy się z sąsiednimi wierzchołkami (i+1, j) i (i, j+1), tworząc strukturę przypominającą siatkę. Aby to obsłużyć, pętle iterują przez i i j, a indeksowanie jest starannie zarządzane, aby uniknąć luk. Tutaj dodajemy również efekt obrotu za pomocą spin(angle), kontrolowany przez angle = time * 180 / π, aby nadać modelowi dynamiczny wygląd 3D.

Budowanie modelu jajka za pomocą trójkątów:

Użycie trójkątów z GL_TRIANGLES wypełnia powierzchnię, tworząc bardziej solidny, realistyczny model. Każdy punkt (i, j) tworzy dwa trójkąty z sąsiadującymi punktami (i+1, j), (i, j+1) i (i+1, j+1), jak widać w funkcji drawEggTriangles(). Ta funkcja ustawia różne kolory dla każdego trójkąta, dzięki czemu jajko wydaje się bardziej teksturowane. Prawidłowe granice i indeksowanie zapewniają, że trójkąty zamykają się płynnie, pokrywając całą powierzchnię modelu.

Współrzędne wierzchołków można określić za pomocą układu równań:

$$x(u, v) = (-90 \cdot u^5 + 225 \cdot u^4 - 270 \cdot u^3 + 180 \cdot u^2 - 45 \cdot u) \cdot \cos(\pi \cdot v),$$

$$y(u, v) = 160 \cdot u^4 - 320 \cdot u^3 + 160 \cdot u^2 - 5,$$

$$z(u, v) = (-90 \cdot u^5 + 225 \cdot u^4 - 270 \cdot u^3 + 180 \cdot u^2 - 45 \cdot u) \cdot \sin(\pi \cdot v),$$

gdzie dziedziny u i v to przedziały $0 \leq u \leq 1$ oraz $0 \leq v \leq 1$.

Opis działanie Programu

Funkcje openGL wykorzystywane w programie:

- glEnable(GL_DEPTH_TEST) – włącza test głębokości, który zapewnia, że powierzchnie zasłonięte przez inne nie są renderowane. Dzięki temu program wyświetla tylko widoczne powierzchnie modelu 3D, tworząc realistyczny efekt głębi.

- glClearColor(0.0, 0.0, 0.0, 1.0) – ustawia kolor tła, który zostanie użyty do czyszczenia ekranu przed każdym renderowaniem. W tym przypadku tło ustawiono na czarne, a czwarty parametr oznacza pełną przezroczystość.

- glRotatef – pozwala na obrót wyświetlanego obiektu o określony kąt wokół wskazanego wektora (X, Y, Z). Funkcja ta jest używana w funkcji spin() do kontroli rotacji modelu na podstawie wartości zmienianych przez użytkownika.

- glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) – argumenty tej funkcji to maski bitowe wskazujące, które bufory należy wyczyścić przed rysowaniem sceny. GL_COLOR_BUFFER_BIT czyści kolor tła, a GL_DEPTH_BUFFER_BIT czyści wartości głębokości, co pozwala na renderowanie świeżego kadru.

- glLoadIdentity – resetuje aktualną macierz (modelu lub projekcji) do macierzy jednostkowej, usuwając poprzednie transformacje. Funkcja ta jest wywoływana przed renderowaniem każdej klatki, aby zapobiec kumulacji transformacji.

- glBegin i glEnd – definiują początek i koniec renderowania prymitywów OpenGL, takich jak punkty (GL_POINTS), linie (GL_LINES) i trójkąty (GL_TRIANGLES). Te funkcje są używane w drawEggPoints, drawEggLines i drawEggTriangles do renderowania modelu jajka w różnych trybach.

- glVertex3f – określa współrzędne wierzchołków, które zostaną wyrenderowane. Współrzędne te pochodzą z macierzy matrix, reprezentującej model jajka.

- glColor3f – ustawia kolor wierzchołków za pomocą trzech parametrów RGB. Kolory są przypisywane z tablicy matrixColor, co pozwala każdej części modelu jajka przyjąć różne barwy w zależności od wybranego trybu rysowania.

- glFlush – wymusza wykonanie wszystkich oczekujących poleceń OpenGL, zapewniając, że cała scena zostanie narysowana przed wyświetleniem użytkownikowi.

- glfwCreateWindow – tworzy nowe okno o określonych wymiarach, tytule i ustawieniach kontekstu. To okno służy jako główna powierzchnia renderowania i punkt interakcji dla użytkownika.

- glfwSetFramebufferSizeCallback – ustawia funkcję update_viewport, która dostosowuje widok i aktualizuje projekcję 3D po zmianie rozmiaru okna przez użytkownika.

- glfwSetKeyCallback – rejestruje funkcję key_callback, pozwalającą programowi przechwytywać naciśnięcia klawiszy. Dzięki temu użytkownik może obracać model jajka i przełączać tryby rysowania.

- glfwSwapBuffers – wykorzystuje technikę podwójnego buforowania, gdzie jeden bufor jest wyświetlany, podczas gdy następna klatka jest rysowana na drugim. Po zakończeniu renderowania bufory są zamieniane, co zapewnia płynne animacje i eliminuje migotanie.

- glViewport – ustawia widoczny obszar dla OpenGL, dopasowując współrzędne rysowania do wymiarów okna.

- glOrtho – konfiguruje projekcję ortograficzną dla sceny, definiując zakres współrzędnych widocznych w oknie. Funkcja ta jest używana w update_viewport do dostosowania projekcji w zależności od proporcji okna.

Opis każdej funkcji w kodzie i fragmenty kodu:

```
8   # Globals for rotation angles, drawing mode, and color toggle
9   rotation_x = 0.0
10  rotation_y = 0.0
11  rotation_z = 0.0
12  draw_mode = 0  # 0 for points, 1 for lines, 2 for triangles
```

Ten fragment kodu definiuje globalne zmienne kontrolujące rotację modelu oraz tryb rysowania. Zmienne ‘rotation_x’, ‘rotation_y’ i ‘rotation_z’ przechowują kąty obrotu wokół osi X, Y i Z, pozwalając na rotację modelu w trakcie działania programu. Zmienna ‘draw_mode’ ustawia tryb rysowania (punkty, linie, lub trójkąty), co umożliwia różne sposoby wizualizacji modelu.

```
14      n = 51
15      matrix = np.zeros((n + 1, n + 1, 3))
16      matrixColor = np.zeros((n + 1, n + 1, 3))
```

Ten fragment kodu tworzy dwie macierze o wymiarach (n + 1, n + 1, 3), gdzie ‘n’ jest ustawione na 51, lub ustawiony przez użytkownika ‘n’. Wybranie rozmiaru (n + 1, n + 1, 3) pozwala na utworzenie siatki 3D, w której każdy punkt siatki jest reprezentowany przez trzy wartości — (x, y, z) w macierzy dla współrzędnych przestrzennych oraz (R, G, B) w ‘matrixColor’ dla wartości koloru. Wprowadzenie ‘n + 1’ zamiast ‘n’ umożliwia włączenie krawędzi siatki, dzięki czemu model jest zamknięty i reprezentowany przez punkty od 0 do ‘n’ na każdym wymiarze, tworząc kompletny obiekt 3D.

```
18  def startup():  1 usage  new *
19      update_viewport( window: None,  width: 400,  height: 400)
20      glClearColor( red: 0.0,  green: 0.0,  blue: 0.0,  alpha: 1.0)
21      glEnable(GL_DEPTH_TEST)
```

Funkcja startup( ):

Funkcja ‘startup( )’ jest wywoływana na początku, aby zainicjować środowisko OpenGL. Na początku wywołuje ‘update_viewport(None, 400, 400)’, która ustawia rozmiar okna na 400x400 pikseli oraz dostosowuje widok, aby renderowanie było prawidłowe. Funkcja ‘glClearColor(0.0, 0.0, 0.0, 1.0)’ ustawia kolor tła okna na czarny z pełną przezroczystością. ‘glEnable(GL_DEPTH_TEST)’ włącza testowanie głębokości, co zapewnia poprawne rysowanie obiektów w scenie na podstawie ich odległości od kamery, zapobiegając niewłaściwemu zakrywaniu bliższych obiektów przez dalsze.

```
24  def spin():  1 usage  new *
25      glRotatef(rotation_x,  x: 1.0,  y: 0.0,  z: 0.0)
26      glRotatef(rotation_y,  x: 0.0,  y: 1.0,  z: 0.0)
27      glRotatef(rotation_z,  x: 0.0,  y: 0.0,  z: 1.0)
```

Funkcja spin( ) stosuje trzy obroty do modelu na podstawie globalnych kątów obrotu rotate_x, rotate_y i rotate_z. Używa glRotatef() do obracania modelu wokół osi X, Y i Z.

Aby jajko kręciło się samo, można wyobrazić to następnym sposobem:

```
def spin (angle):  1 usage  new *
    glRotatef(angle,  x: 1.0,  y: 0.0,  z: 0.0)
    glRotatef(angle,  x: 0.0,  y: 1.0,  z: 0.0)
    glRotatef(angle,  x: 0.0,  y: 0.0,  z: 1.0)
```

Główną różnicą między tymi dwiema funkcjami jest sposób, w jaki obsługują kąt obrotu:

spin(angle) przyjmuje parametr ‘angle’ i stosuje ten sam kąt obrotu do wszystkich trzech osi (X, Y, Z).

spin( ) używa zmiennych globalnych ‘rotate_x’, ‘rotate_y’ i ‘rotate_z’, aby stosować różne kąty obrotu do każdej osi (X, Y, Z) niezależnie.

```
29    def axes():  1 usage  new *
30        glBegin(GL_LINES)
31        glColor3f( red: 1.0,  green: 0.0,  blue: 0.0)
32        glVertex3f(-5.0,  y: 0.0,  z: 0.0)
33        glVertex3f( x: 5.0,  y: 0.0,  z: 0.0)
34        glColor3f( red: 0.0,  green: 1.0,  blue: 0.0)
35        glVertex3f( x: 0.0, -5.0,  z: 0.0)
36        glVertex3f( x: 0.0,  y: 5.0,  z: 0.0)
37        glColor3f( red: 0.0,  green: 0.0,  blue: 1.0)
38        glVertex3f( x: 0.0,  y: 0.0, -5.0)
39        glVertex3f( x: 0.0,  y: 0.0,  z: 5.0)
40        glEnd()
```

Funkcja axes( ) rysuje osie współrzędnych (X, Y i Z), co pomaga zwizualizować orientację renderowanego obiektu.

Oś X: Czerwona linia od (-5, 0, 0) do (5, 0, 0).
Oś Y: Zielona linia od (0, -5, 0) do (0, 5, 0).
Oś Z: Niebieska linia od (0, 0, -5) do (0, 0, 5).

```
42    def matrixValues():  1 usage  new *
43        for i in range(0, n + 1):
44            for j in range(0, n + 1):
45                u = i / n
46                v = j / n
47                matrix[i][j][0] = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) * cos(pi * v)
48                matrix[i][j][1] = 160 * pow(u, 4) - 320 * pow(u, 3) + 160 * pow(u, 2)
49                matrix[i][j][2] = (-90 * pow(u, 5) + 225 * pow(u, 4) - 270 * pow(u, 3) + 180 * pow(u, 2) - 45 * u) * sin(pi * v)
```

Funkcja matrixValues( ) oblicza i przechowuje współrzędne punktów, które definiują kształt jajka. Używa równań matematycznych, aby określić położenie każdego punktu na siatce na podstawie parametrów „u” i „v”, które mieszczą się w zakresie od 0 do 1. Wynikowe współrzędne są przechowywane w tablicy „matrix”, która jest później używana do rysowania kształtu jajka w przestrzeni 3D.

```
51    def matrixColorValues():  1 usage  new *
52        for i in range(0, n + 1):
53            for j in range(0, n + 1):
54                matrixColor[i][j] = [random.random() for _ in range(3)]
55        for i in range(0, int(n / 2) - 1):
56            matrixColor[n - i][n] = matrixColor[i][0]
```

Funkcja matrixColorValues( ) przypisuje losowe kolory do każdego punktu w tablicy matrixColor, co odpowiada współrzędnym 3D kształtu jajka. Dla każdego punktu w macierzy generowany jest kolor poprzez utworzenie listy trzech losowych wartości za pomocą random.random(), która generuje float między 0 a 1. Wartości te reprezentują składowe koloru RGB (czerwony, zielony, niebieski), więc każdy punkt otrzymuje unikalny kolor.

```
matrixColor[i][j] = [random.random() for _ in range(3)]
```

Ta linia wylosuje listę trzech losowych wartości (po jednej dla każdego kanału koloru: czerwony, zielony i niebieski). Dla każdej kombinacji ‘i’ i ‘j’ ta lista jest przypisywana do matrixColor[i][j].

```
for i in range(0, int(n / 2) - 1):
    matrixColor[n - i][n] = matrixColor[i][0]
```

Ta część modyfikuje kolor pewnych punktów w pobliżu krawędzi modelu jajka. Dokładniej odzwierciedla wartości kolorów z pierwszej połowy tablicy (matrixColor[i][0]) do odpowiadających im pozycji w drugiej połowie tablicy.

```
58      def render(time):  1 usage  new *
59          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
60          glLoadIdentity()
61          spin()
62          axes()
63          if draw_mode == 0: #Mode for points
64              drawEggPoints()
65          elif draw_mode == 1: #Mode for lines
66              drawEggLines()
67          elif draw_mode == 2: # mode for triangles
68              drawEggTriangles()
69          elif draw_mode == 3:
70              drawEggWhite()  # mode for full white egg
```

Funkcja render( ) czyści ekran i przygotowuje się do rysowania. Na podstawie wartości draw_mode decyduje, której metody rysowania użyć: punktów, linii, trójkątów lub białego jajka. Funkcja jest odpowiedzialna za renderowanie kształtu jajka w różnych stylach wizualnych w zależności od bieżącego trybu.

```
73      def drawEggPoints():  1 usage  new *
74          glColor3f( red: 0.941,  green: 0.917,  blue: 0.839)
75          for i in range(0, n + 1):
76              for j in range(0, n + 1):
77                  glBegin(GL_POINTS)
78                  glVertex3f(matrix[i][j][0], matrix[i][j][1] - 5, matrix[i][j][2])
79                  glEnd()
```

```
81      def drawEggLines():  1 usage  new *
82          glColor3f( red: 0.7,  green: 0.2,  blue: 0.9)
83          for i in range(0, n):
84              for j in range(0, n):
85                  glBegin(GL_LINES)
86                  glVertex3f(matrix[i][j][0], matrix[i][j][1] - 5, matrix[i][j][2])
87                  glVertex3f(matrix[i + 1][j][0], matrix[i + 1][j][1] - 5, matrix[i + 1][j][2])
88                  glVertex3f(matrix[i][j][0], matrix[i][j][1] - 5, matrix[i][j][2])
89                  glVertex3f(matrix[i][j + 1][0], matrix[i][j + 1][1] - 5, matrix[i][j + 1][2])
90                  glEnd()
```

```python
def drawEggTriangles():  1 usage  new *
    for i in range(0, n):
        for j in range(0, n):
            glBegin(GL_TRIANGLES)

            # First triangle of the square
            glColor3f(*matrixColor[i][j])
            glVertex3f(matrix[i][j][0], matrix[i][j][1] - 5, matrix[i][j][2])

            glColor3f(*matrixColor[i][j + 1])
            glVertex3f(matrix[i][j + 1][0], matrix[i][j + 1][1] - 5, matrix[i][j + 1][2])

            glColor3f(*matrixColor[i + 1][j])
            glVertex3f(matrix[i + 1][j][0], matrix[i + 1][j][1] - 5, matrix[i + 1][j][2])

            glEnd()

            glBegin(GL_TRIANGLES)

            # Second triangle of the square
            glColor3f(*matrixColor[i][j + 1])
            glVertex3f(matrix[i][j + 1][0], matrix[i][j + 1][1] - 5, matrix[i][j + 1][2])

            glColor3f(*matrixColor[i + 1][j])
            glVertex3f(matrix[i + 1][j][0], matrix[i + 1][j][1] - 5, matrix[i + 1][j][2])

            glColor3f(*matrixColor[i + 1][j + 1])
            glVertex3f(matrix[i + 1][j + 1][0], matrix[i + 1][j + 1][1] - 5, matrix[i + 1][j + 1][2])

            glEnd()
```

```python
# Function to render the egg fully in white
def drawEggWhite():  1 usage  new *
    glColor3f( red: 0.949,  green: 0.921,  blue: 0.890)  # Set color to white
    for i in range(0, n):
        for j in range(0, n):
            glBegin(GL_TRIANGLES)

            # First triangle of the square
            glVertex3f(matrix[i][j][0], matrix[i][j][1] - 5, matrix[i][j][2])
            glVertex3f(matrix[i][j + 1][0], matrix[i][j + 1][1] - 5, matrix[i][j + 1][2])
            glVertex3f(matrix[i + 1][j][0], matrix[i + 1][j][1] - 5, matrix[i + 1][j][2])

            glEnd()

            glBegin(GL_TRIANGLES)

            # Second triangle of the square
            glVertex3f(matrix[i][j + 1][0], matrix[i][j + 1][1] - 5, matrix[i][j + 1][2])
            glVertex3f(matrix[i + 1][j][0], matrix[i + 1][j][1] - 5, matrix[i + 1][j][2])
            glVertex3f(matrix[i + 1][j + 1][0], matrix[i + 1][j + 1][1] - 5, matrix[i + 1][j + 1][2])

            glEnd()
```

Za pomocą tych funkcji, rysujemy jajko na różne sposoby. *Funkcja drawEggPoints( )* rysuje jajko za pomocą punktów (wybrany kolor „Eggshell” jako żart, #F0EAD6), *drawEggLines( )* – za pomocą siatki (fioletowy, aby lepiej było widać linii), *drawEggTriangles( )* – za pomocą trójkątów losowych kolorów, *drawEggWhite( )* – białe jajko, jako zwykłe, normalne jajko (kolor – Bone, #F2EBE3)

```python
def update_viewport(window, width, height):  2 usages  new *
    aspect_ratio = width / height if height > 0 else 1
    glMatrixMode(GL_PROJECTION)
    glViewport( x: 0,  y: 0, width, height)
    glLoadIdentity()
    if width <= height:
        glOrtho(-7.5,  right: 7.5, -7.5 / aspect_ratio, 7.5 / aspect_ratio,  zNear: 7.5, -7.5)
    else:
        glOrtho(-7.5 * aspect_ratio, 7.5 * aspect_ratio, -7.5,  top: 7.5,  zNear: 7.5, -7.5)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
```

Funkcja 'update_viewport( )' dostosowuje ustawienia widoku, gdy zmieniany jest rozmiar okna. Oblicza współczynnik proporcji na podstawie nowej szerokości i wysokości, zapewniając, że projekcja nie zostanie zniekształcona. Następnie funkcja aktualizuje macierz projekcji za pomocą 'glOrtho( )', aby ustawić obszar wyświetlania, dostosowując szerokość lub wysokość, w zależności od tego, który jest większy. Na koniec resetuje macierz modelview, aby przygotować się do rysowania. Zapewnia to, że scena jest poprawnie renderowana w nowych wymiarach okna. Ta funkcja jest potrzebna, gdy chcemy powiększyć lub zmniejszyć okno, ponieważ program jest już uruchomiony. Teoretycznie nie jest to konieczne, ale jest zaimplementowane dla bardziej logicznego i ładnego użycia.

```python
def key_callback(window, key, scancode, action, mods):  1 usage  new *
    global rotation_x, rotation_y, rotation_z, draw_mode
    if action == GLFW_PRESS or action == GLFW_REPEAT:
        if key == GLFW_KEY_W:
            rotation_x += 5.0
        elif key == GLFW_KEY_S:
            rotation_x -= 5.0
        elif key == GLFW_KEY_A:
            rotation_y -= 5.0
        elif key == GLFW_KEY_D:
            rotation_y += 5.0
        elif key == GLFW_KEY_Q:
            rotation_z += 5.0
        elif key == GLFW_KEY_E:
            rotation_z -= 5.0
        elif key == GLFW_KEY_LEFT:
            draw_mode = (draw_mode - 1) % 4  # Cycle backward through modes
        elif key == GLFW_KEY_RIGHT:
            draw_mode = (draw_mode + 1) % 4  # Cycle forward through modes
```

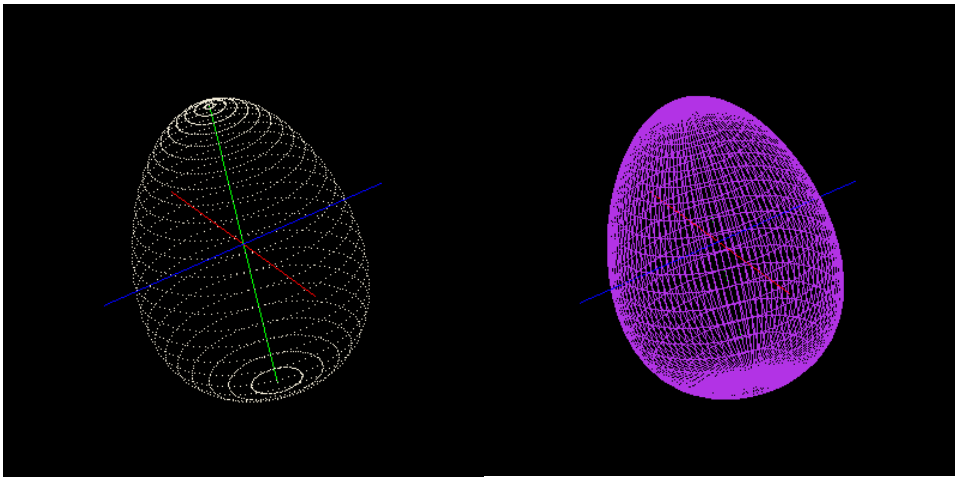
Funkcja key_callback() jest odpowiedzialna za obsługę zdarzeń związanych z naciśnięciem klawiszy na klawiaturze. Reaguje na wciśnięcie klawisza (lub jego powtórzenie) i zmienia odpowiednie wartości globalne, takie jak kąty rotacji (rotation_x, rotation_y, rotation_z) oraz tryb rysowania (draw_mode).

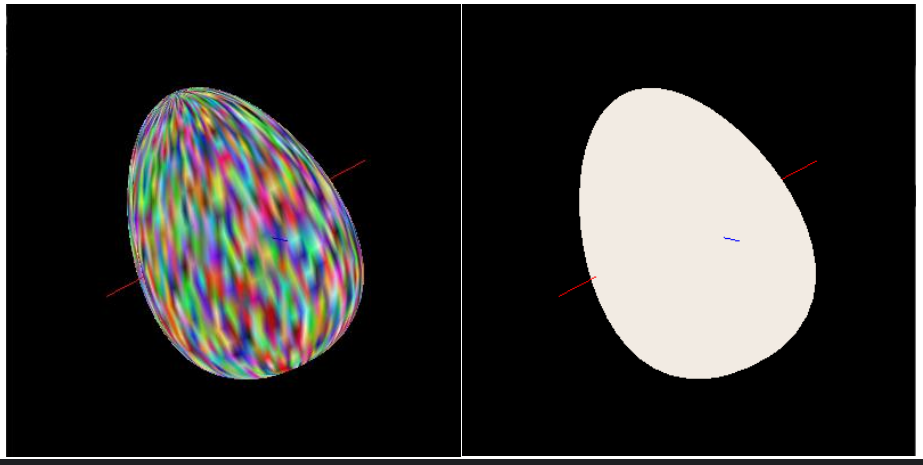
```python
def main():  1 usage  new *
    if not glfwInit():
        sys.exit(-1)

    window = glfwCreateWindow( width: 400,  height: 400,  title: "Egg Rotation and Drawing Mode Switch",  monitor: None,  share: None)
    if not window:
        glfwTerminate()
        sys.exit(-1)

    random.seed(1)
    matrixValues()
    matrixColorValues()

    glfwMakeContextCurrent(window)
    glfwSetFramebufferSizeCallback(window, update_viewport)
    glfwSetKeyCallback(window, key_callback)
    glfwSwapInterval(1)

    startup()
    print("Press Arrow Left or Arrow Right to switch between modes (Points, Lines, Triangles, Full white Egg)")
    print("Press WASDQE to rotate on axes: ")
    print("Axis X: A and D")
    print("Axis Y: W and S")
    print("Axis Z: Q and E")
```

```python
    while not glfwWindowShouldClose(window):
        render(glfwGetTime())
        glfwSwapBuffers(window)
        glfwPollEvents()

    glfwTerminate()
```

Funkcja ‘main()’ inicjalizuje bibliotekę GLFW, tworzy okno aplikacji i ustawia wszystkie niezbędne parametry, takie jak rozmiar okna, callbacki do obsługi zmian rozmiaru okna i naciśnięć klawiszy. Następnie inicjalizuje dane do rysowania (wywołując ‘matrixValues( )’ i ‘matrixColorValues( )’), uruchamia program (funkcja ‘startup( )’), a w pętli głównej renderuje scenę, reaguje na zdarzenia i aktualizuje okno, aż użytkownik zamknie aplikację.

Wynik działanie programu:





```
Press Arrow Left or Arrow Right to switch between modes (Points, Lines, Triangles, Full white Egg)
Press WASDQE to rotate on axes:
Axis X: A and D
Axis Y: W and S
Axis Z: Q and E
Press X to hide the axis
```

Wnioski

Jako niektóre z ulepszeń programu, później dodałem funkcję, która może ukryć oś po naciśnięciu przycisku „X”. Ale jako inne ulepszenia, moglibyśmy dodać przycisk, który sprawi, że jajko samo się obróci, lub poprosić użytkownika o liczbę N ręcznie i zbudować jajko odpowiednio.

Funkcja, która włącza i wyłącza osi (co było dodane):

```
13      show_axes = True  # Start with axes visible
```

```
70          if show_axes:  # Only draw the axes if show_axes is True
71              axes()
```

```
171  # Callback function to handle key events
172  def key_callback(window, key, scancode, action, mods):  1 usage  new *
173      global rotation_x, rotation_y, rotation_z, draw_mode, show_axes
```

```
191                 elif key == GLFW_KEY_X:
192                     show_axes = not show_axes
```

```
213         print("Press X to hide the axis")
```

Wynik działania: