Sprawozdanie
Grafika komputerowa i komunikacja człowiek-komputer
Laboratorium 3

Katsiaryna Kolyshko 276708

Dr. Inż. Jan Nikodem

Wstęp teoretyczny

Zadanie laboratoryjne polega na dodawanie i rotacji jajka i kamery za pomocą myszy komputerowej.

Musieliśmy przerobić kod, aby za pomocą myszy mogliśmy robić rotacji jak jajko, tak i kamery.

Kamery w Open GL i 3D

W OpenGL i modelowaniu 3D w ogóle zrozumienie, jak działają transformacje, takie jak obrót, skalowanie i translacja, jest podstawą do manipulowania i wyświetlania obiektów 3D. Te transformacje są zazwyczaj reprezentowane za pomocą macierzy.

Podstawowe typy transformacji

Transformacje w grafice 3D są zazwyczaj klasyfikowane jako transformacje afiniczne i nieafiniczne.

Transformacje afiniczne zachowują paralelizm, ale niekoniecznie odległości lub kąty. Obejmują one:

- Tożsamość (brak zmian w obiekcie)
- Skalowanie (powiększanie lub zmniejszanie obiektu)
- Translacja (przesuwanie obiektu)
- Obrót (obrót obiektu wokół osi)

Transformacje afiniczne można przedstawić jako mnożenia macierzy. Na przykład:

- Macierz skalowania: Skaluje obiekt o czynniki Sx, Sy, Sz wzdłuż osi X, Y i Z.
- Macierz translacji: Przesuwa obiekt o przesunięcie wzdłuż osi X, Y i Z.
- Macierze obrotu: Obracają obiekt wokół jednej z trzech osi głównych (X, Y lub Z). Są one przedstawione za pomocą macierzy, które zawierają funkcje trygonometryczne w celu obliczenia nowych współrzędnych po obrocie.

Obrót w przestrzeni 3D

Obrót w przestrzeni 3D może być dość złożony, ponieważ może występować wokół różnych osi. Macierz obrotu służy do obracania punktów lub obiektów w przestrzeni 3D. Obrót wokół osi X, Y lub Z można przedstawić za pomocą określonych macierzy:

Obrót wokół osi X:

$$R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) & 0 \\ 0 & \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ta macierz obraca obiekt o kąt γ wokół osi X.

Obrót wokół osi Y:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Obrót wokół osi Z:

$$R_z(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 & 0 \\ \sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

W OpenGL te transformacje są często łączone ze sobą, aby osiągnąć pożądany rezultat. Na przykład obiekt może być najpierw skalowany, następnie obracany, a na końcu translowany. Kolejność, w jakiej te transformacje są stosowane jest kluczowa, ponieważ mnożenie macierzy nie jest przemienne.

Kamery w grafice 3D

W grafice 3D pojęcie „kamery" jest nieco mylące. Sprzęt graficzny, taki jak GPU, nie używa kamery w sposób, w jaki intuicyjnie myślimy. Zamiast tego GPU renderuje scenę z ustalonego punktu widzenia, zwykle w znormalizowanym układzie współrzędnych. Objętość widoku jest zwykle definiowana przez stożek widzenia, ściętą piramidę, która definiuje region widoczny dla kamery.

Pozycję i orientację kamery w przestrzeni można przedstawić matematycznie za pomocą trzech kluczowych wektorów:

Pozycja oka (e): Pozycja kamery w przestrzeni świata.

Punkt patrzenia (p): Punkt, na który patrzy kamera.

Wektor w górę (u): Wektor skierowany w górę od kamery.

Matryca widoku służy do pozycjonowania i orientowania sceny z perspektywy kamery. Macierz „patrzna" to specyficzny rodzaj transformacji, który zapewnia prawidłowe wyrównanie kamery ze sceną. Po skonstruowaniu macierzy widoku scena jest transformowana względem położenia kamery, tworząc iluzję perspektywy. Macierz kamery można odwrócić, aby przekształcić współrzędne świata we współrzędne kamery.

„Face Culling"

Face culling to technika renderowania stosowana w grafice komputerowej, szczególnie w renderowaniu 3D, w celu poprawy wydajności poprzez odrzucanie (lub „culling") trójkątów lub wielokątów, które nie są widoczne dla kamery. W kontekście obiektów 3D trójkąt ma dwie ściany: przednią i tylną. Dzięki użyciu face culling, OpenGL może uniknąć renderowania ścian obiektów, które są zwrócone od kamery (lub w kierunku przeciwnym do widoku), co zmniejsza liczbę przetwarzanych wielokątów, co prowadzi do lepszej wydajności, szczególnie w złożonych scenach.

Rodzaje ścian

Front Face: Jest to strona wielokąta, która jest ogólnie uważana za „widoczną". Ściana przednia jest zazwyczaj definiowana za pomocą kolejności nawijania wierzchołków przeciwnie do ruchu wskazówek zegara (CCW).

Back Face: Jest to strona przeciwna do ściany przedniej i zazwyczaj nie jest widoczna dla widza, gdy wielokąt jest prawidłowo zorientowany.

Aby użyć „Face culling" w Open GL, możemy używać następne funkcje:

- glEnable(GL_CULL_FACE);
- glCullFace(GL_BACK);
- glCullFace(GL_FRONT);
- glCullFace(GL_FRONT_AND_BACK);

- glDisable(GL_CULL_FACE);

Zalety „Face Culling”

Poprawiona wydajność: Eliminując potrzebę przetwarzania trójkątów, które nie są widoczne, Face Culling zmniejsza ilość geometrii, którą OpenGL musi obsłużyć. Jest to szczególnie przydatne w środowiskach 3D z dużymi obiektami lub złożonymi modelami.

Prędkość renderowania: Ponieważ przetwarzanych jest mniej wielokątów, ogólna wydajność renderowania jest poprawiona, szczególnie podczas korzystania ze złożonych modeli lub scen, w których wiele wielokątów zostałoby odrzuconych.

Optymalizacja: Optymalizuje potok renderowania, oszczędzając zasoby GPU. W scenie może być wiele trójkątów zwróconych w stronę przeciwną do kamery, więc odrzucanie pomaga skupić zasoby GPU na renderowaniu tylko tych, które przyczyniają się do ostatecznego obrazu.

Opis działania kodu

NoweFunkcje openGL wykorzystywane w programie:

- *glScalef* -- Używana do dynamicznej zmiany skali obiektu w odpowiedzi na interakcje użytkownika (np. zmiana rozmiaru obiektu za pomocą myszy). Jest to istotna funkcja do bardziej zaawansowanej manipulacji obiektami 3D.

- *gluLookAt* -- W nowym kodzie pozwala na przekształcenie widoku z kamery, umożliwiając jej obrót wokół obiektu lub poruszanie się w przestrzeni. Wartość theta i phi są obliczane na podstawie interakcji myszy, co pozwala użytkownikowi na pełną kontrolę nad tym, co widzi.

- *keyboard_key_callback* -- Funkcja służy do obsługi naciśnięć klawiszy, takich jak E, G, strzałki, aby zmieniać tryby rysowania obiektu, skalować go, czy zmieniać liczbę segmentów w siatce (n). Dzięki temu użytkownik ma pełną kontrolę nad wyświetlanymi trybami i ustawieniami obiektów w scenie.

- *mouse_button_callback* -- Funkcja do obsługi kliknięć myszy, umożliwiająca przełączanie między trybami interakcji – obracaniem obiektu lub poruszaniem kamerą w przestrzeni. W połączeniu z funkcjami mouse_motion_callback i update_viewport umożliwia bardziej zaawansowaną manipulację widokiem.

- *gl_cull_face* – Używana do włączenia lub wyłączenia culling’u (odrzucania) niepotrzebnych stron obiektów 3D, co zwiększa wydajność renderowania. Aktywacja tej funkcji pozwala na usunięcie tych stron (np. tylnych), które nie są widoczne w danym widoku, co oszczędza zasoby obliczeniowe i poprawia płynność wyświetlania scen.

- *gl_front* – Używana w kontekście culling’u do określenia, że należy odrzucać (cull) strony obiektów, które są uznawane za "przednie" w danej konfiguracji. Zazwyczaj strony przednie to te, które są skierowane ku widzowi. Można ją stosować w specyficznych przypadkach, np. w przypadku renderowania obiektów odwróconych do wewnątrz.

Opis każdej NOWEJ funkcji w kodzie i fragmenty kodu:
Dlatego, że pracujemy na tym samym jajku, to opiszmy tylko nowe funkcji naszego programu.

Zadanie polegało na tym, aby z naciskiem na guziki myszy mogliśmy rotować jajko, lub kamerę.

Zacznijmy od samego początku.

```
11    viewer = [0.0, 0.0, 10.0]
```

Linia „viewer = [0.0, 0.0, 10.0]” definiuje położenie kamery w przestrzeni 3D, a współrzędne są przechowywane na liście „viewer”. Te trzy wartości reprezentują odpowiednio położenia X, Y i Z kamery. W tym przypadku kamera jest umieszczona w początku osi X i Y (0.0, 0.0) i 10 jednostek dalej wzdłuż osi Z (10.0), co oznacza, że znajduje się przed początkiem, patrząc w stronę środka sceny z odległości 10 jednostek.

```
13    theta = 180.0
14    phi = 0.0
15    pix2angle = 1.0
16    piy2angle = 1.0
17    pixs2angle = 1.0
18    scale = 1.0
19    R = 8.0
```

Linie definiują kilka zmiennych, które kontrolują położenie kamery i interakcję ze sceną. theta = 180,0 i phi = 0,0 reprezentują kąty współrzędnych sferycznych kamery, gdzie „theta” kontroluje obrót wokół osi Y, a „phi” kontroluje obrót wokół osi X. „pix2angle = 1,0”, „piy2angle = 1,0” i „pixs2angle = 1,0” to współczynniki służące do konwersji ruchów pikseli myszy na zmiany kąta obrotu kamery. „scale = 1,0” definiuje początkowy współczynnik skalowania obiektu, umożliwiający jego powiększanie lub pomniejszanie, podczas gdy „R = 8,0” ustawia początkową odległość kamery od początku, wpływając na to, jak blisko lub daleko kamera jest umieszczona od sceny.

```
21    left_mouse_button_pressed = 0
22    right_mouse_button_pressed = 0
23    e_button_state = 1
```

Linie inicjują zmienne, które śledzą stan danych wprowadzanych przez użytkownika. „left_mouse_button_pressed = 0” i „right_mouse_button_pressed = 0” reprezentują początkowy stan lewego i prawego przycisku myszy, gdzie wartość 0 oznacza, że przyciski nie są naciśnięte. „e_button_state = 1” oznacza początkowy stan klawisza „E”, gdzie 1 oznacza, że tryb sterowania kamerą jest włączony. Te zmienne są używane do obsługi interakcji użytkownika, takich jak kliknięcia myszą i naciśnięcia klawiszy, aby kontrolować zachowanie kamery i obiektu w scenie 3D.

```
25    mouse_x_pos_old = 0
26    mouse_y_pos_old = 0
27    delta_x = 0
28    delta_y = 0
29    delta_s = 0
30
31    upY = 1.0
```

Ten fragment kodu definiuje zmienne używane do śledzenia i obliczania ruchu myszy i orientacji kamery. „mouse_x_pos_old = 0” i „mouse_y_pos_old = 0” przechowują poprzednią pozycję myszy na ekranie, która jest używana do obliczania ruchu lub delty myszy w następnej klatce. „delta_x =

0” i „delta_y = 0” przechowują zmiany pozycji myszy wzdłuż osi X i Y, które są obliczane na podstawie różnicy między bieżącą a poprzednią pozycją myszy. „delta_s = 0” służy do przechowywania zmiany skali (np. w celu powiększenia), chociaż nie jest bezpośrednio zaimplementowana w tym fragmencie kodu.

„upY = 1.0” określa orientację kierunku „w górę” dla kamery, zapewniając, że oś pionowa kamery jest prawidłowo zorientowana (dodatni kierunek Y).

Teraz opiszemy zmiany w funkcji *render( )*. Niżej jest przedstawiony pełny kod tej funkcji.

```
def render(time):  1 usage  new *
    global theta
    global phi
    global scale
    global R
    global upY
    global e_button_state

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    # klawisz 'e' powoduje zmiane trybu: obracanie obiektu lub tryb poruszania kamera
    if e_button_state:
        xeye = R * cos(2 * pi * theta / 360) * cos(2 * pi * phi / 360)
        yeye = R * sin(2 * pi * phi / 360)
        zeye = R * sin(2 * pi * theta / 360) * cos(2 * pi * phi / 360)

        gluLookAt(xeye, yeye, zeye, 0.0, 0.0, 0.0, 0.0, upY, 0.0)

        if phi > 180:
            phi -= 2 * 180
        elif phi <= -180:
            phi += 2 * 180

        if phi < -180 / 2 or phi > 180 / 2:
            upY = -1.0
        else:
            upY = 1
```

```
        if left_mouse_button_pressed:
            theta += delta_x * pix2angle
            phi += delta_y * piy2angle
        # 'R' moze przyjmowac wartosci od 1 do 10
        if right_mouse_button_pressed:
            if delta_x > 0 and R < 10:
                R += 0.05
            else:
                if R >= 1:
                    R -= 0.05

    else:
        gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)

        if left_mouse_button_pressed:
            theta += delta_x * pix2angle
            phi += delta_y * piy2angle

        glRotatef(theta,  x: 0.0,  y: 1.0,  z: 0.0)
        glRotatef(phi,  x: 1.0,  y: 0.0,  z: 0.0)

        # 'scale' moze przyjmowac wartosci od 0.3 do 3
        if right_mouse_button_pressed:
            if delta_x > 0 and scale < 3:
                scale += 0.050
            else:
                if scale >= 0.3:
                    scale -= 0.050
        glScalef(scale, scale, scale)
```

```
    axes()
    example_object()
    glFlush()
```

Funkcja render( ) jest odpowiedzialna za rysowanie i renderowanie sceny 3D, w której użytkownik może manipulować kamerą i obiektami za pomocą myszy. Na początku funkcja czyści bufor kolorów i głębokości, a następnie resetuje transformacje macierzy. Jeśli przycisk „e” jest wciśnięty, funkcja zmienia tryb z obrotu obiektu na tryb poruszania kamerą. W trybie kamery, pozycja kamery obliczana jest na podstawie kątów theta i phi, które kontrolują kąt widzenia, a promień R decyduje o odległości kamery od punktu środkowego sceny. Funkcja gluLookAt ustawia kamerę w odpowiedniej pozycji, w zależności od tych parametrów. Kamera jest dynamicznie obracana, a także kontrolowana przez ruchy myszy, gdzie zmiany pozycji myszy (delta_x i delta_y) wpływają na kąty theta i phi.

W przypadku, gdy przycisk „e” nie jest wciśnięty, funkcja wchodzi w tryb obrotu obiektu, w którym stosowane są obrót i skalowanie obiektu w przestrzeni 3D. gluLookAt ustawia kamerę w stałej pozycji określonej przez wektor viewer, a obiekt jest obracany wokół osi Y (dla kąta theta) oraz osi X (dla kąta phi). Manipulowanie myszą pozwala na zmianę tych kątów, a także na skalowanie obiektu, gdzie przycisk prawy myszy zmienia wartość skali obiektu w zakresie od 0.3 do 3.0.

Na końcu, funkcja wywołuje funkcje rysujące, takie jak axes( ) i example_object( ), które rysują osie oraz przykładowy obiekt w przestrzeni 3D. Zastosowanie funkcji glFlush( ) kończy proces renderowania, gwarantując, że wszystkie zmiany są natychmiastowo wyświetlane na ekranie.

W porównaniu do programu z laboratorium 2, musimy zakodować mysz komputerową do programu i jak będą działać guziki myszy. W tym jest jedna z różnic kodów, że mamy nie tylko keyboard_callback, ale jesczce i mouse_button_callback.

Niżej jest przedstawiony kod funkcji *mouse_button_callback ( ):*

```
109  def mouse_button_callback(window, button, action, mods):  1 usage  new *
110      global left_mouse_button_pressed
111      global right_mouse_button_pressed
112
113      if button == GLFW_MOUSE_BUTTON_LEFT and action == GLFW_PRESS:
114          left_mouse_button_pressed = 1
115      else:
116          left_mouse_button_pressed = 0
117
118      if button == GLFW_MOUSE_BUTTON_RIGHT and action == GLFW_PRESS:
119          right_mouse_button_pressed = 1
120      else:
121          right_mouse_button_pressed = 0
```

Funkcja mouse_button_callback( ) obsługuje zdarzenia związane z kliknięciem przycisków myszy w aplikacji. Jest to funkcja zwrotna, która jest wywoływana za każdym razem, gdy użytkownik naciśnie lub zwolni przycisk myszy. Parametry wejściowe to: window, który odnosi się do aktywnego okna aplikacji, button, który określa, który przycisk myszy został naciśnięty, action, który wskazuje, czy przycisk został wciśnięty (GLFW_PRESS) lub zwolniony (GLFW_RELEASE), oraz mods, który może zawierać dodatkowe modyfikatory, takie jak np. przytrzymanie klawiszy modyfikujących. Funkcja ta sprawdza, który przycisk myszy został użyty (lewy lub prawy) oraz czy został naciśnięty lub zwolniony. Jeśli użytkownik naciśnie lewy przycisk myszy, zmienna left_mouse_button_pressed zostaje ustawiona na 1, w przeciwnym przypadku na 0. Podobnie, jeśli naciśnie prawy przycisk myszy, zmienna right_mouse_button_pressed jest ustawiana na 1, w przeciwnym przypadku na 0. Te zmienne globalne są później wykorzystywane w innych

częściach programu, aby kontrolować interakcje z obiektami 3D, takie jak obracanie obiektów czy zmiana kamery.

Funkcja mouse_motion_callback( ):

```
124    def mouse_motion_callback(window, x_pos, y_pos):  1 usage  new *
125        global delta_x
126        global mouse_x_pos_old
127        global delta_y
128        global mouse_y_pos_old
129
130        delta_x = x_pos - mouse_x_pos_old
131        mouse_x_pos_old = x_pos
132
133        delta_y = y_pos - mouse_y_pos_old
134        mouse_y_pos_old = y_pos
```

Funkcja mouse_motion_callback( ) obsługuje ruch myszy, śledząc zmiany pozycji kursora. Jest to funkcja zwrotna, która jest wywoływana za każdym razem, gdy użytkownik przemieszcza myszkę nad oknem aplikacji. Parametry wejściowe to: window, które odnosi się do aktywnego okna aplikacji, oraz x_pos i y_pos, które wskazują bieżącą pozycję kursora myszy w osi X i Y. Funkcja oblicza różnicę pomiędzy aktualną pozycją kursora a poprzednią, zapisując tę wartość w zmiennych delta_x i delta_y, które reprezentują odpowiednio zmianę w poziomie (X) i pionie (Y) kursora. Następnie aktualizuje poprzednie pozycje myszy, zapisując bieżące wartości x_pos i y_pos do zmiennych mouse_x_pos_old i mouse_y_pos_old. Zmienne delta_x i delta_y są później wykorzystywane w innych częściach programu, np. do obrotu obiektów lub zmiany widoku kamery w odpowiedzi na ruch myszy.

Dodamy do programu Face Culling. W Open GL jest funkcja, którą możemy użyć, aby zamaskować wewnętrzną część kolorowanego jajka.

Jako parametr, wrzucimy face_culling_enabled, który będzię wczytywał czy jest włączony face culling, albo nie.

```
155    # Global variable to track face culling status
156    face_culling_enabled = False  # Initially off
```

Do funkcji keyboard_key_callback( ) dodaliśmy nowe parametry, jak face culling i takie funkcje jak odkamienianie lub powiększanie krawędzi jajek.

Nowy dodatek do funkcji:

```
158    def keyboard_key_callback(window, key, scancode, action, mods):  1 usage  new *
159        global e_button_state, draw_mode, n, color_change_enabled, face_culling_enabled
160
161        if key == GLFW_KEY_ESCAPE and action == GLFW_PRESS:
162            glfwSetWindowShouldClose(window, GLFW_TRUE)
163        if key == GLFW_KEY_E and action == GLFW_PRESS:
164            if e_button_state:
165                e_button_state = 0
166            else:
167                e_button_state = 1
```

Funkcja sprawdza, czy naciśnięto klawisz "Escape" lub "E". Jeśli naciśnięto "Escape", zamyka okno aplikacji, a jeśli naciśnięto "E", zmienia stan trybu (obracanie obiektu lub poruszanie kamerą) przełączając zmienną e_button_state między 0 a 1.

```
173            elif key == GLFW_KEY_G:
174                n = min(n + 6, 120)
175                matrixValues()
176                matrixColorValues()
177            elif key == GLFW_KEY_H:
178                n = max(n - 6, 6)
179                matrixValues()
180                matrixColorValues()
```

Funkcja reaguje na naciśnięcie klawiszy "G" i "H". Naciśnięcie "G" zwiększa wartość zmiennej n o 6, ale nie przekracza 120, natomiast naciśnięcie "H" zmniejsza wartość n o 6, nie schodząc poniżej 6, a następnie wywołuje funkcje matrixValues() i matrixColorValues(), które przeliczają i aktualizują odpowiednie wartości macierzy.

```
184            # Toggle face culling on/off
185            face_culling_enabled = not face_culling_enabled
186            if face_culling_enabled:
187                glEnable(GL_CULL_FACE)
188                glCullFace(GL_FRONT)  # Cull back faces
189                print("Face Culling Enabled")
190            else:
191                glDisable(GL_CULL_FACE)
192                print("Face Culling Disabled")
```

Funkcja reaguje na naciśnięcie klawisza "F", przełączając stan culling-u twarzy w trybie 3D. Jeśli face culling jest włączony, włącza funkcję glEnable(GL_CULL_FACE) oraz ustawia culling dla twarzy z przodu (glCullFace(GL_FRONT)), a jeśli jest wyłączony, wyłącza funkcję culling-u, co pozwala na rysowanie wszystkich twarzy obiektu.

Zapytając u kolegi z grupy, zmieniłam kod jajka, które rusyjmy za pomocą trójkątów. To jajko wygląda trochę inaczej od mojego, bo są używane trochę inni formuły do budowania jajka, ale jest przeskalowany do rozmiary mojego jajka i też przypomina jajko. Pozbawiliśmy od szwa na jajkie.

Tu jest zmieniona funkcja drawEggTriangles( ).

```python
def drawEggTriangles():  1 usage  new *
    global n
    radiusX = 5.0
    radiusY = 5.0

    # Loop over the stacks (height) of the egg
    for i in range(n):
        y = radiusY - (2.0 * radiusY * i / n)
        yNext = radiusY - (2.0 * radiusY * (i + 1) / n)

        normalizedPosition = float(i) / float(n)
        radiusVariable = 0.7 * radiusX * sqrt(sin(pi * normalizedPosition))
        radiusVariableNext = 0.7 * radiusX * sqrt(sin(pi * (normalizedPosition + 1.0 / n)))

        glBegin(GL_TRIANGLES)

        # Use a smooth color gradient instead of random colors
        color = [0.5 + 0.5 * sin(0.1 * i), 0.5 + 0.5 * cos(0.1 * i), 1.0]  # Smooth color transition

        # Loop through all slices (360 degrees) of the egg
        for j in range(n):
            rotationNow = 2.0 * pi * float(j) / float(n)
            rotationNext = 2.0 * pi * float(j + 1) / float(n)

                # Current stack coordinates
                xNow = radiusVariable * cos(rotationNow)
                zNow = radiusVariable * sin(rotationNow)

                xNext = radiusVariable * cos(rotationNext)
                zNext = radiusVariable * sin(rotationNext)

                # Next stack coordinates
                xNowNextStack = radiusVariableNext * cos(rotationNow)
                zNowNextStack = radiusVariableNext * sin(rotationNow)

                xNextNextStack = radiusVariableNext * cos(rotationNext)
                zNextNextStack = radiusVariableNext * sin(rotationNext)

                # Set the color once per triangle strip
                glColor3f(*color)

                # Triangle 1 (between current and next stack)
                glVertex3f(xNow, y, zNow)
                glVertex3f(xNowNextStack, yNext, zNowNextStack)
                glVertex3f(xNext, y, zNext)


                # Triangle 2 (between next stack and next next stack)
                glVertex3f(xNowNextStack, yNext, zNowNextStack)
                glVertex3f(xNextNextStack, yNext, zNextNextStack)
                glVertex3f(xNext, y, zNext)

            glEnd()
```

Różnica pomiędzy starym a nowym sposobem:

Nowe „drawEggTriangles( )”:

- Dynamicznie oblicza kształt jajka przy użyciu współrzędnych sferycznych, iterując przez pionowe stosy i poziome wycinki, aby obliczyć pozycje wierzchołków. Kolory są generowane przy użyciu płynnego gradientu opartego na indeksie iteracji. Trójkąty są rysowane przez łączenie wierzchołków dla każdego wycinka.

Stare „drawEggTriangles( )”:

- - Ta funkcja używa wstępnie obliczonych danych wierzchołków i kolorów zapisanych w `matrix` i `matrixColor`. Pętla przez macierz, rysując trójkąty między sąsiadującymi wierzchołkami i stosując równomierne przesunięcie do współrzędnej `y`. Kolory są pobierane bezpośrednio z macierzy.

Główne różnice:

- Pierwsza metoda oblicza kształt jajka dynamicznie, podczas gdy druga używa wstępnie zdefiniowanych danych wierzchołków i kolorów.

- Pierwsza metoda generuje kolory na podstawie obliczeń, podczas gdy druga używa przechowywanych wartości.

Krótko mówiąc, pierwsza metoda jest bardziej elastyczna przy generowaniu kształtu, druga natomiast sprawdza się wydajniej przy użyciu wstępnie zdefiniowanych danych siatki.

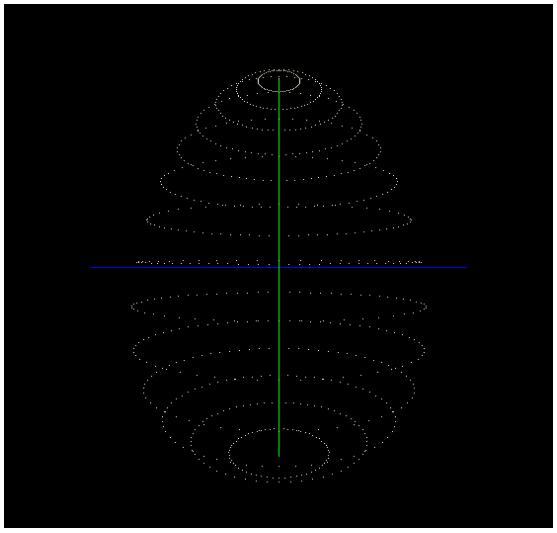
Funkcja startup( ):

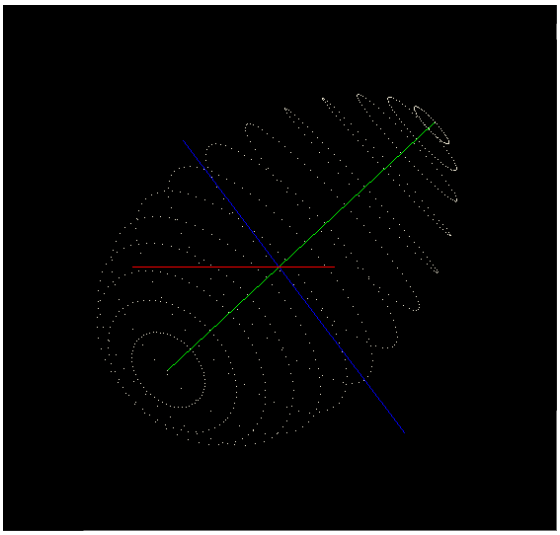
```
352    def startup():  1 usage  new *
353        update_viewport( window: None,  width: 400,  height: 400)
354        glClearColor( red: 0.0,  green: 0.0,  blue: 0.0,  alpha: 1.0)
355
356        # Enable back face culling
357        glEnable(GL_CULL_FACE)
358        glCullFace(GL_FRONT)  # Cull the back faces
359        glFrontFace(GL_CCW)  # Set the front face winding order to counter-clockwise
360
361        glEnable(GL_DEPTH_TEST)
362        print("Press Arrow Left or Arrow Right to switch between modes (Points, Lines, Triangles) ")
363        print("Use Mouse Buttons to Rotate the egg:")
364        print("Left Mouse Button: Rotate the camera/egg by holding the button down")
365        print("Right Mouse Button: Zoom In or Out by dragging the mouse inward or outward")
366        print("Press E to change mode from Camera to Egg")
367        print("Press G to increase the number of n by 6 [cap at 120]")
368        print("Press H to decrease the number of n by 6 [cap at 6]")
```

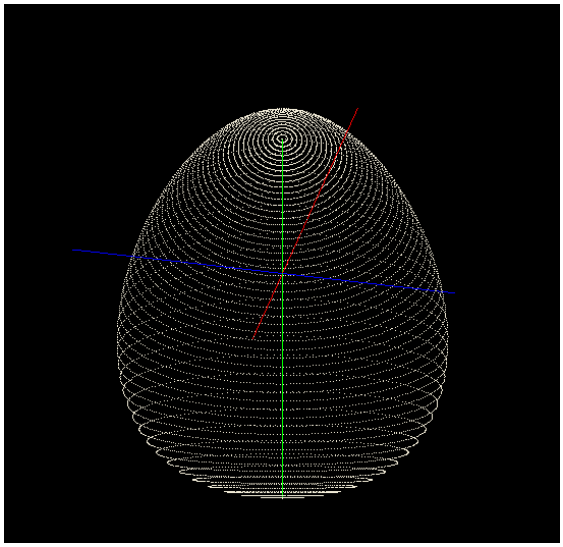
Dodaliśmy nowe polecenia do konsoli i face culling.

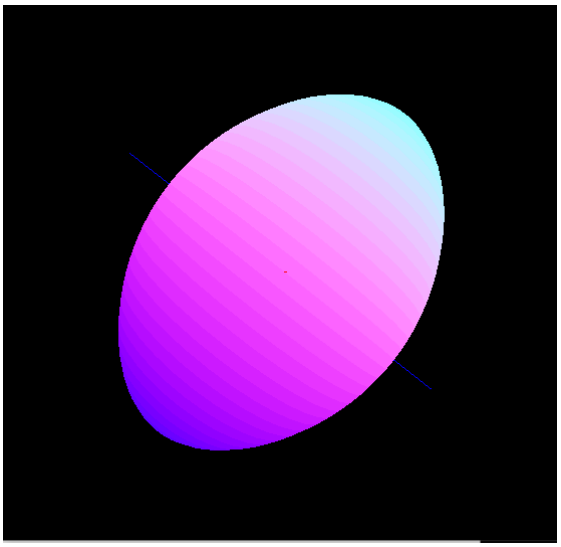
Wyniki działania programu:

```
Press Arrow Left or Arrow Right to switch between modes (Points, Lines, Triangles)
Use Mouse Buttons to Rotate the egg:
Left Mouse Button: Rotate the camera/egg by holding the button down
Right Mouse Button: Zoom In or Out by dragging the mouse inward or outward
Press E to change mode from Camera to Egg
Press G to increase the number of n by 6 [cap at 120]
Press H to decrease the number of n by 6 [cap at 6]
```

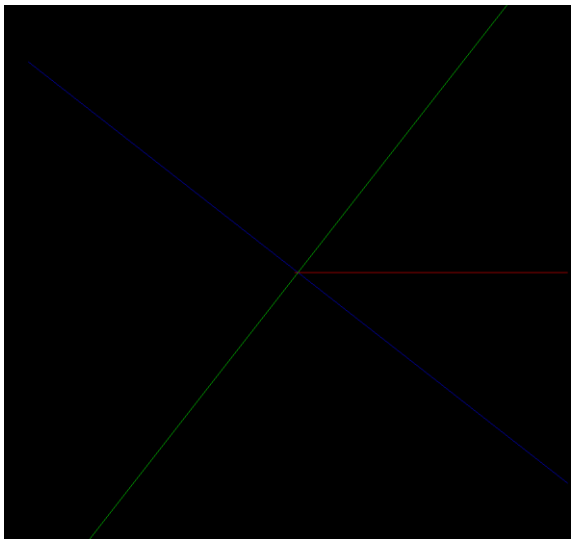




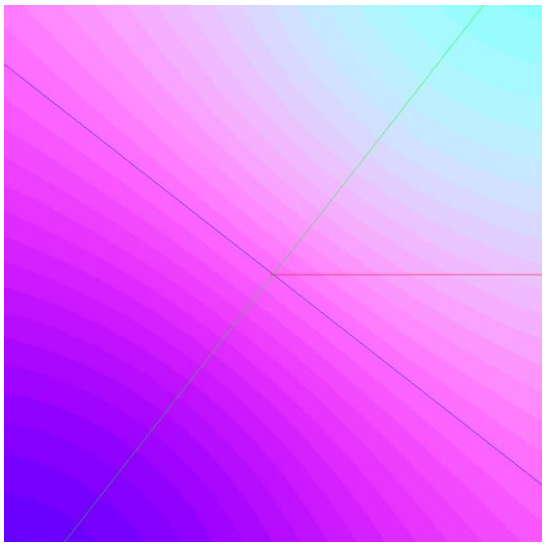




Face culling on:



Face culling off:



Wnioski:

Mam problem z tym, aby wyświetlić czajnik. Postaram się dodać go na następne zajęcie.