

Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

| Grafika komputerowa i komunikacja człowiek-komputer | | | |
|--|--|--------------------|-------------------------------|
| Temat: | Laboratorium 4 – Interakcja z użytkownikiem, transformacje wierzchołków | | |
| 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 mechanizmem obsługi urządzeń peryferyjnych oraz zrozumienie zasad związanych z przekształcaniem wierzchołków. Implementowano zastosowanie tych transformacji w odpowiedzi na działania użytkownika wykonane z użyciem urządzeń peryferyjnych (na przykład myszy).

2. Kod wspólny dla wszystkich zadań

W skrypcie zapewnionym przez Prowadzącego zostały wprowadzone zmienne pomocnicze oraz funkcje odpowiedzialne za transformacje modelu 3D oraz reakcje na działania użytkownika – ruchy myszą. Funkcja *mouse_button_callback()* odpowiada za reagowanie na akcję – naciśnięcie przycisku myszy przez użytkownika i zmianę wartości zmiennej przechowującej informację o tym fakcie. Funkcja *mouse_motion_callback()* zmienia wartości położenia myszy – przykładowo w poziomie lub w poziomie i pionie, co zależy od konkretnego zadania. Zmienna *viewer* przechowuje informacje o położeniu obserwatora, natomiast zmienna *pix2angle* przechowuje wartość, przez którą przeskalowany jest kąt wynikający z ruchu myszą. Przykładowy model wykorzystany w zadaniach jest tworzony w funkcji *example_object()*. Zostały wykorzystane funkcje pochodzące z implementacji specyfikacji *OpenGL* oraz z biblioteki *GLFW*, która zapewnia obsługę zdarzeń urządzeń peryferyjnych (myszy). Wszystkie skrypty zostały napisane w języku *Python*.

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

W zadaniu tym należało wprowadzić obracanie obiektu wokół osi. W tym celu oprócz zmiennej *theta* przechowującej wartość obrotu wokół osi *Y* wprowadzono zmienną *phi*, która zawiera informację o obrocie wokół osi *X*. Zmienna *left_mouse_button_pressed* zawiera informację, czy lewy klawisz myszy jest wciśnięty. Zmienne *mouse_x_pos_old* i *mouse_y_pos_old* przechowują poprzednią informację o położeniu myszy w poziomie oraz pionie, natomiast zmienne *delta_x* i *delta_y* przechowują wartości przesunięcia w tych kierunkach.

```
16 left_mouse_button_pressed = 0
17 mouse_x_pos_old = 0
18 mouse_y_pos_old = 0
19 delta_x = 0
20 delta_y = 0
```

Rysunek 3.1 Zmienne globalne wykorzystane w zadaniu

W funkcji *mouse_button_callback()* sprawdzane jest, czy lewy klawisz myszy jest wciśnięty – jeżeli tak, zmieniana jest wartość zmiennej, która informuje o tym. Słowo kluczowe *global* umożliwia dostęp do zmiennej spoza zakresu widoczności funkcji.

```
148 def mouse_button_callback(window, button, action, mods):
149     global left_mouse_button_pressed
150
151     if button == GLFW_MOUSE_BUTTON_LEFT and action == GLFW_PRESS:
152         left_mouse_button_pressed = 1
153     else:
154         left_mouse_button_pressed = 0
155
```

Rysunek 3.2 Reakcja na wciśnięcie lewego przycisku myszy

W funkcji *mouse_motion_callback()* w reakcji na ruch myszą korygowane jest położenie w poziomie i pionie.

```
134 def mouse_motion_callback(window, x_pos, y_pos):
135     # Użycie globalnych zmiennych
136     global delta_x
137     global mouse_x_pos_old
138     global delta_y
139     global mouse_y_pos_old
140
141     # Zmiana położenia x, y dla myszy
142     delta_y = y_pos - mouse_y_pos_old
143     mouse_y_pos_old = y_pos
144     delta_x = x_pos - mouse_x_pos_old
145     mouse_x_pos_old = x_pos
```

Rysunek 3.3 Aktualizacja położenia myszy w poziomie i pionie

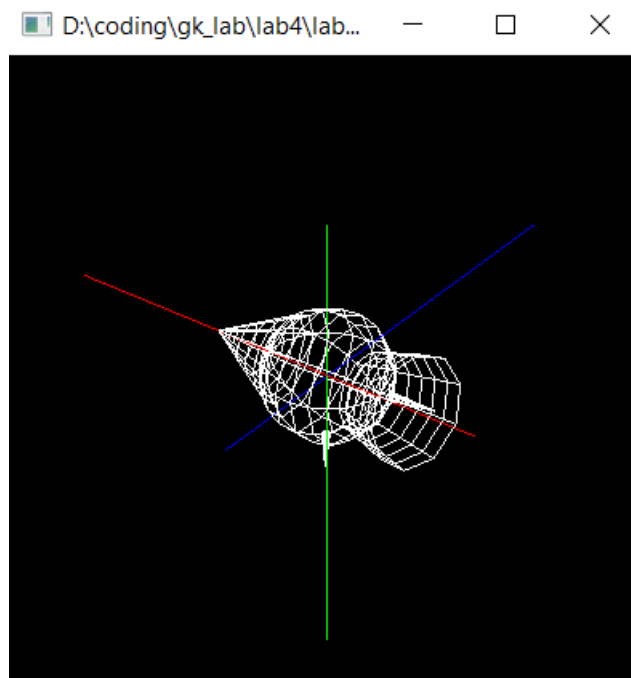
Obroty modelu na podstawie akcji związanych z ruchem myszą dokonywane są w funkcji *render()*. Najpierw ustalane jest przekształcenie patrzenia za pomocą funkcji *gluLookAt()* oraz argumentu – listy *viewer* przechowującej informacje o położeniu obserwatora. Następnie, jeżeli lewy przycisk myszy jest wciśnięty, zmieniane kąty *theta* i *phi* korzystając z wartości przesunięć myszy oraz wartości użytej do przeskalowania (*pix2angle*). Następuje obrót odpowiednio wokół osi X i Y za pomocą wywołań funkcji *glRotatef()* (Rysunek 3.4). Rezultat działania skryptu przedstawia Rysunek 3.5

```

86  def render(time):
87      global theta
88      global phi
89
90      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
91      glLoadIdentity()
92
93      gluLookAt(viewer[0], viewer[1], viewer[2],
94               0.0, 0.0, 0.0, 0.0, 1.0, 0.0)
95
96      # Zmieniamy katy polozenia, gdy LPM wcisniety
97      if left_mouse_button_pressed:
98          phi += delta_y * pix2angle
99          theta += delta_x * pix2angle
100
101      # Obrocenie
102      glRotatef(phi, 1.0, 0.0, 0.0)
103      glRotatef(theta, 0.0, 1.0, 0.0)
104
105      axes()
106      example_object()
107
108      glFlush()

```

Rysunek 3.4 Funkcja render()



Rysunek 3.5 Obrócony obiekt

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

W zadaniu tym należało wprowadzić obsługę drugiego przycisku myszy w celu przeskalowania obiektu. Do zmiennych globalnych z poprzedniego zadania została dodana zmienna *scale*, która

przechowuje liczbową wartość przeskalowania. Postać funkcji *mouse_motion_callback()* nie uległa zmianie, natomiast funkcja *mouse_button_callback()* została zmodyfikowana tak, aby obsłużyć także prawy przycisk myszy – analogicznie jak lewy przycisk (Rysunek 4.1).

```
156 def mouse_button_callback(window, button, action, mods):
157     global left_mouse_button_pressed
158     global right_mouse_button_pressed
159
160     if button == GLFW_MOUSE_BUTTON_LEFT and action == GLFW_PRESS:
161         left_mouse_button_pressed = 1
162     else:
163         left_mouse_button_pressed = 0
164
165     if button == GLFW_MOUSE_BUTTON_RIGHT and action == GLFW_PRESS:
166         right_mouse_button_pressed = 1
167     else:
168         right_mouse_button_pressed = 0
```

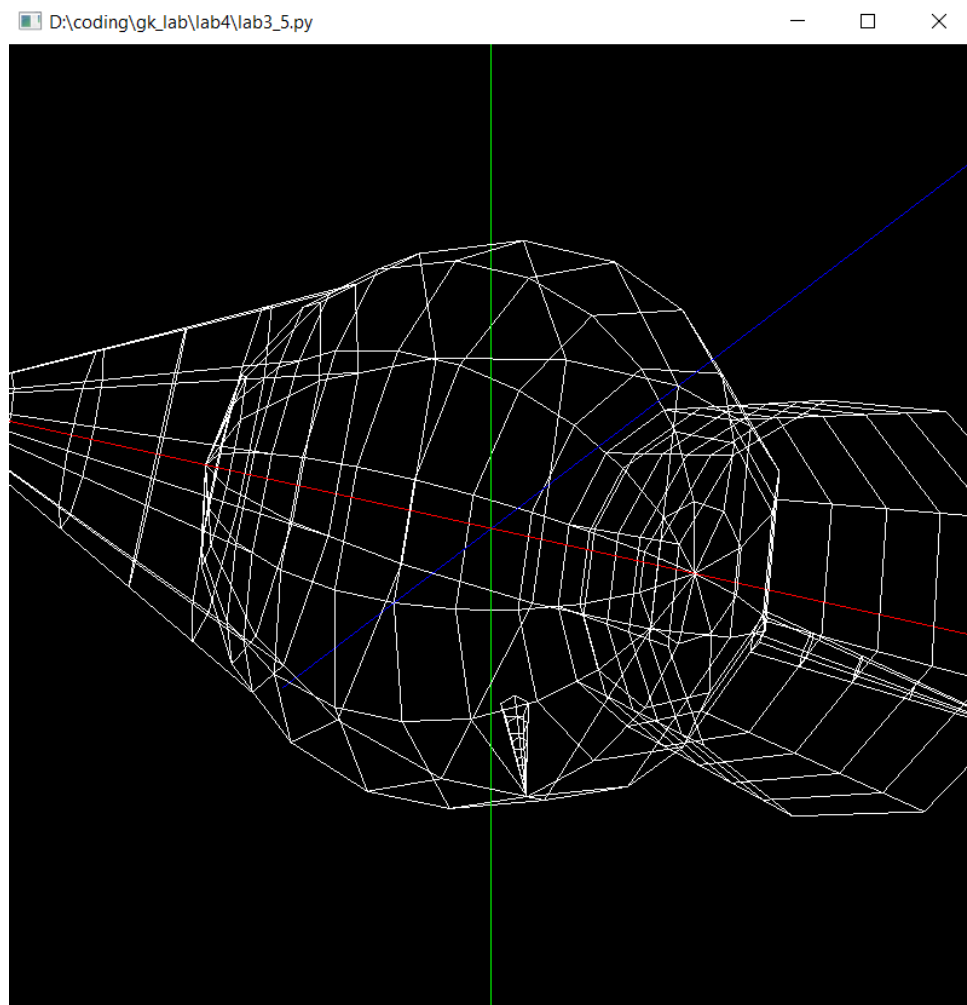
Rysunek 4.1 Dodanie obsługi prawego przycisku

Zmianie uległa funkcja *render()*. Zostało uwzględnione przyciśnięcie prawego przycisku myszy, które skutkuje wyliczeniem wartości *scale*. Do dotychczasowej wartości jest dodawane 0,015 wartości przesunięcia myszy w poziomie, ale tylko pod warunkiem, że wynikowa wartość jest większa od 0. W przeciwnym razie *scale* posiada dotychczasową wartość. Rozwiązanie to, wprowadzone za pomocą jednoliniowej instrukcji warunkowej *if* ma na celu zabezpieczenie przed niepoprawnym skalowaniem, które może prowadzić do uzyskania wartości ujemnych (Rysunek 4.2).

```
99     # Zmieniamy katy polozenia, gdy LPM wcisniety
100     if left_mouse_button_pressed:
101         phi += delta_y * pix2angle
102         theta += delta_x * pix2angle
103     # Zmiana skali, gdy PPM wcisniety
104     if right_mouse_button_pressed:
105         scale = scale + 0.015 * delta_x if scale + 0.015 * delta_x > 0 else scale
106
107     # Obrocenie
108     glRotatef(phi, 1.0, 0.0, 0.0)
109     glRotatef(theta, 0.0, 1.0, 0.0)
110     # Skalowanie
111     glScalef(scale, scale, scale)
```

Rysunek 4.2 Główna, zmodyfikowana część funkcji *render()*

Efekt działania omówionego skryptu w postaci przeskalowanego modelu przedstawia Rysunek 4.3.



Rysunek 4.3 Przeskalowany obiekt