Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Grafika komputerowa i komunikacja człowiek-komputer			
Temat:	Laboratorium 6 – Teksturowanie obiektów		
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 teksturowania obiektów. Poznawano sposób aplikowania tekstur na zdefiniowane powierzchnie. Analizowano sposoby tworzenia własnych tekstur.

2. Kod wspólny dla wszystkich zadań

W znacznej części został wykorzystany kod z poprzedniego laboratorium służący do obsługi oświetlenia. Zdefiniowane zostały parametry światła, w tym także parametry materiałowe.

W celu realizacji zadań została wykorzystana biblioteka języka *Python* – *Python Imaging Library* (*Pillow*). Umożliwia ona otwieranie i ładowanie plików zawierających tekstury aplikowane na powierzchnie. Użyte tekstury są plikami *tga*. Po załadowaniu, są one definiowane w programie przy użyciu wywołań funkcji *glTexImage2D()*[1] (Rysunek 2.1). Istotne jest także uaktywnienie mechanizmu teksturowania i określenie jego parametrów – linie 59-63 – odpowiedni kod został dostarczony przez Prowadzącego.

```
59      glEnable(GL_TEXTURE_2D)
60      glEnable(GL_CULL_FACE)
61      glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
62      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
63      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
64
65      image = Image.open("tekstura.tga")
66
67      glTexImage2D(
68          GL_TEXTURE_2D, 0, 3, image.size[0], image.size[1], 0,
69          GL_RGB, GL_UNSIGNED_BYTE, image.tobytes("raw", "RGB", 0, -1)
70      )
```

Rysunek 2.1

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

W zadaniu tym należało narysować otekstorowany kwadrat w taki sposób, aby był on widoczny tylko z jednej strony – w celu zaobserwowania zjawiska *face culling*.

W celu realizacji zadania zostały, w funkcji *render()*, narysowane dwa połączone trójkąty, które razem tworzą kwadrat. Wywołania funkcji *glVertex3f()* dla poszczególnych wierzchołków trójkątów zostały umieszczone w odpowiedniej kolejności – przeciwnie do ruchu wskazówek zegara dla obserwatora patrzącego na kwadrat „od przodu”, co powoduje, że jest on dla niego

[1] https://registry.khronos.org/OpenGL-Refpages/gl4/html/glTexImage2D.xhtml

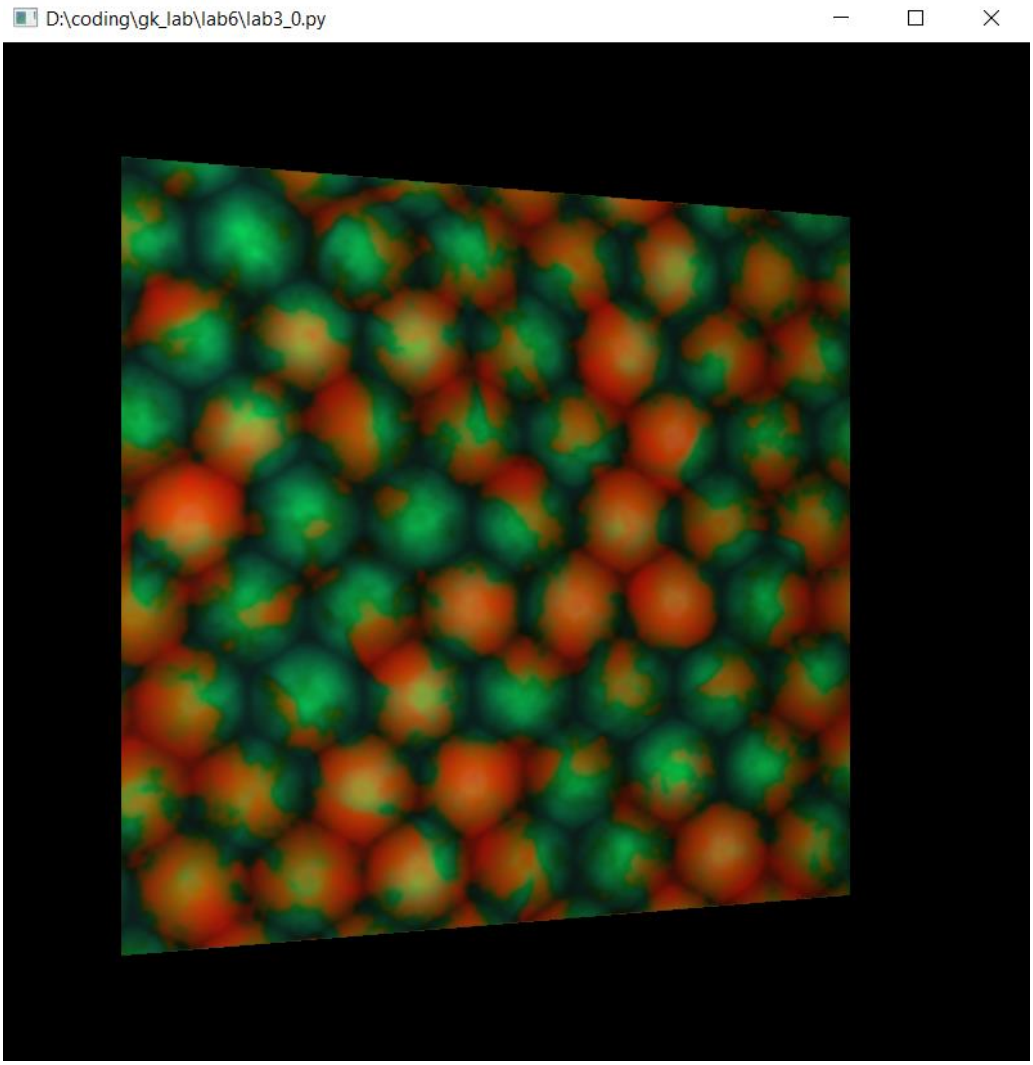
widoczny[2]. Jednocześnie, przed każdym umieszczeniem wierzchołka w pamięci wywoływana jest funkcja *glTexCoord2f()*, która określa współrzędne tekstury nakładanej dla danego punktu (Rysunek 3.1).

```
91      glBegin(GL_TRIANGLES)
92
93      glTexCoord2f(0.0, 0.0)
94      glVertex3f(-5.0, -5.0, 0.0)
95
96      glTexCoord2f(1.0, 0.0)
97      glVertex3f(5.0, -5.0, 0.0)
98
99      glTexCoord2f(0.0, 1.0)
100     glVertex3f(-5.0, 5.0, 0.0)
101
102     glEnd()
103
104     glBegin(GL_TRIANGLES)
105
106     glTexCoord2f(1.0, 0.0)
107     glVertex3f(5.0, -5.0, 0.0)
108
109     glTexCoord2f(1.0, 1.0)
110     glVertex3f(5.0, 5.0, 0.0)
111
112     glTexCoord2f(0.0, 1.0)
113     glVertex3f(-5.0, 5.0, 0.0)
114
115     glEnd()
116     glFlush()
```

Rysunek 3.1 Zmodyfikowany fragment funkcji render()

Efekt działania skryptu przedstawia Rysunek 3.2.

[2] https://learnopengl.com/Advanced-OpenGL/Face-culling



Rysunek 3.2 Efekt działania skryptu na ocenę 3.0

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

W zadaniu tym należało przygotować oteksturowany ostrosłup, którego podstawą miał być kwadrat z zadania poprzedniego. Należało zapewnić możliwość ukrywania przynajmniej jednej ze ścian.

Ściany ostrosłupa zostały utworzone jako trójkąty wychodzące z boków kwadratu. Domyślnie, podobnie jak wcześniej, wywołania funkcji tworzącej wierzchołki są umieszczone w kolejności pozwalającej na widoczność danej ściany dla obserwatora. Funkcja *glTexCoord2f()* jest wywoływana z parametrami takimi, aby każda ściana boczna ostrosłupa zawierała ćwiartkę tekstury – dla wierzchołka ostrosłupa jest ona wywoływana z parametrami *0,5*, *0,5* (Rysunek 4.1).

Dodatkowo, w sposób opisany podczas poprzednich laboratoriów, zostały dodane 3 inne źródła światła o identycznych właściwościach do źródła *GL_LIGHT0*, ale umieszczone równomiernie

z 3 pozostałych stron teksturowanego obiektu – w pozycjach o współrzędnych (0, 0, -10), (10, 0, 0) oraz (-10, 0, 0). Ułatwia to obserwację oteksturowanych powierzchni.

```
110     # Sciany ostroslupa
111     # wierzcholek (0, 0, -3) - szczyt
112
113     # Sciana 1
114     glBegin(GL_TRIANGLES)
115     glTexCoord2f(0.0, 1.0)
116     glVertex3f(-5.0, 5.0, 0.0)
117     glTexCoord2f(0.5, 0.5)
118     glVertex3f(0.0, 0.0, -3.0)
119     glTexCoord2f(0.0, 0.0)
120     glVertex3f(-5.0, -5.0, 0.0)
121     glEnd()
122
123     # Sciana 2
124     glBegin(GL_TRIANGLES)
125     glTexCoord2f(0.0, 0.0)
126     glVertex3f(-5.0, -5.0, 0.0)
127     glTexCoord2f(0.5, 0.5)
128     glVertex3f(0.0, 0.0, -3.0)
129     glTexCoord2f(1.0, 0.0)
130     glVertex3f(5.0, -5.0, 0.0)
131     glEnd()
```

Rysunek 4.1 Tworzenie ścian bocznych 1 i 2

W przypadku ściany bocznej nr 4 zastosowano możliwość jej ukrywania i pokazywania, co ma na celu pokazanie zjawiska *face culling*. Przełączanie pomiędzy stanami odbywa się **poprzez naciśnięcie klawisza H** i związaną z nim zmianę flagi *wall_hidden*. Ukrywanie ściany polega na zmianie kolejności umieszczania wierzchołków w pamięci taki sposób, aby nie odbywało się to przeciwnie do kierunku ruchu wskazówek zegara dla obserwatora (Rysunek 4.3).

W funkcji *keyboard_key_callback()* została wprowadzona obsługa klawisza H.

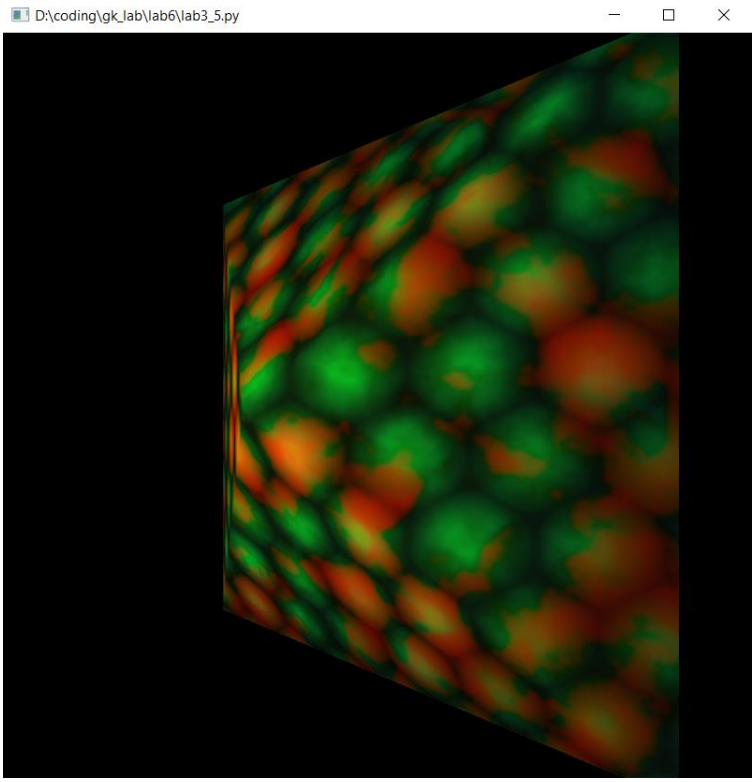
```
216 def keyboard_key_callback(window, key, scancode, action, mods):
217     global wall_hidden
218     if key == GLFW_KEY_ESCAPE and action == GLFW_PRESS:
219         glfwSetWindowShouldClose(window, GLFW_TRUE)
220     if key == GLFW_KEY_H and action == GLFW_PRESS:
221         wall_hidden = not wall_hidden
```

Rysunek 4.2 Obsługa klawisza H

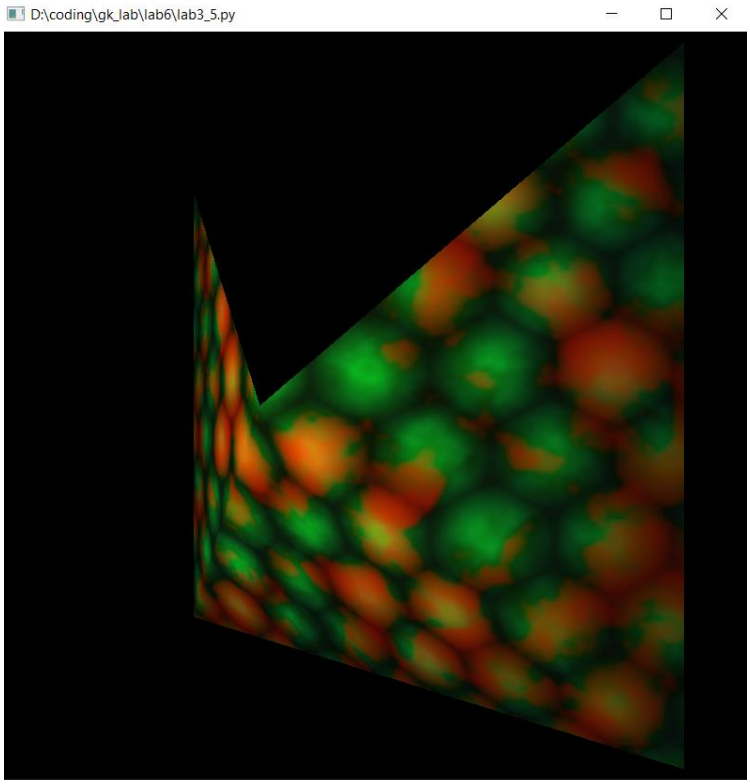
```
    # Sciana 3
    glBegin(GL_TRIANGLES)
    glTexCoord2f(1.0, 0.0)
    glVertex3f(5.0, -5.0, 0.0)
    glTexCoord2f(0.5, 0.5)
    glVertex3f(0.0, 0.0, -3.0)
    glTexCoord2f(1.0, 1.0)
    glVertex3f(5.0, 5.0, 0.0)
    glEnd()

    # Sciana 4
    glBegin(GL_TRIANGLES)
    # Jezeli ukryc sciane - inna kolejnosc wierzcholkow (klawisz H)
    if wall_hidden:
        glTexCoord2f(1.0, 1.0)
        glVertex3f(5.0, 5.0, 0.0)
        glTexCoord2f(0.0, 1.0)
        glVertex3f(-5.0, 5.0, 0.0)
        glTexCoord2f(0.5, 0.5)
        glVertex3f(0.0, 0.0, -3.0)
    # Bez ukrycia - domyslnie
    else:
        glTexCoord2f(1.0, 1.0)
        glVertex3f(5.0, 5.0, 0.0)
        glTexCoord2f(0.5, 0.5)
        glVertex3f(0.0, 0.0, -3.0)
        glTexCoord2f(0.0, 1.0)
        glVertex3f(-5.0, 5.0, 0.0)
    glEnd()
```

Rysunek 4.3 Tworzenie ścian bocznych 3 i 4 oraz ukrywanie ściany 4

Efekty działania skryptu przedstawiają Rysunek 4.4 oraz Rysunek 4.5.



Rysunek 4.4 Efekt działania skryptu na ocenę 3.5 – bez ukrycia ściany

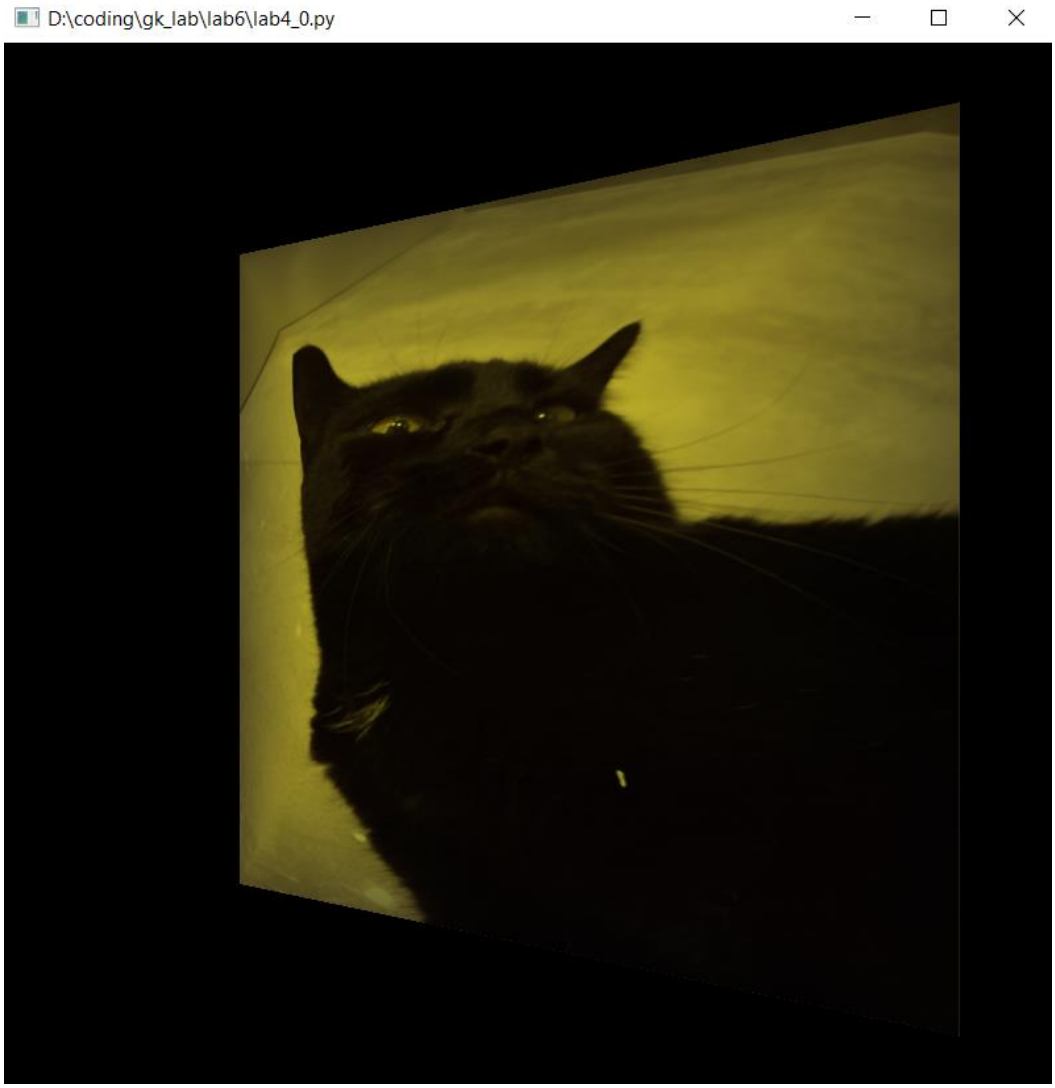


Rysunek 4.5 Ukrycie ściany

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

W zadaniu tym należało stworzyć i zastosować własną teksturę. Mogła ona, przykładowo, przedstawiać wizerunek zwierzęcia.

Zgodnie z instrukcjami dostarczonymi w skrypcie do laboratorium, została utworzona tekstura *kot1.tga* o rozmiarach 512x512. Zachowanie długości boków w postaci potęg liczby 2 jest uważane za pożądaną praktykę[3]. Jedyna zmiana w kodzie, w porównaniu do skryptu na ocenę 3.5, polega na zmianie tekstury przypisywanej do zmiennej *image:* ***image = Image.open("kot1.tga")***. Zachowano algorytmy, w tym ukrywania jednej ze ścian, z poprzedniego zadania. Efekt działania skryptu przedstawia Rysunek 5.1.



Rysunek 5.1 Efekt nałożenia własnej tekstury

[3] https://www.katsbits.com/tutorials/textures/make-better-textures-correct-size-and-power-of-two.php

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

W zadaniu tym należało wprowadzić możliwość przełączania tekstur. Możliwe było zastosowanie przełączania tekstur, przykładowo klawiszem, na jednym obiekcie.

Zdecydowano się na modyfikację skryptu z poprzedniego zadania. Pliki tekstur, podobnie jak wcześniej, są wczytywane do pamięci RAM na początku programu – w funkcji *startup()*. Dodano teksturę *kot2.tga*, także o wymiarach 512x512. Tekstury są przypisywane do zmiennych *image0* oraz *image1* (Rysunek 6.1).

```
41    image0, image1 = Image.open("kot1.tga"), Image.open("kot2.tga")
```

Rysunek 6.1

W funkcji *render()* następuje, z wykorzystaniem funkcji *glTexImage2D(),* podmienienie tekstury w pamięci karty graficznej na podstawie stanu flagi *img_choice* (Rysunek 6.2), która jest zmieniania poprzez **naciśnięcie klawisza C** – analogicznie jak flaga *wall_hidden* (Rysunek 6.3). Domyślnie ustawioną teksturą jest tekstura reprezentowana przez zmienną image1 (*kot2.tga*).

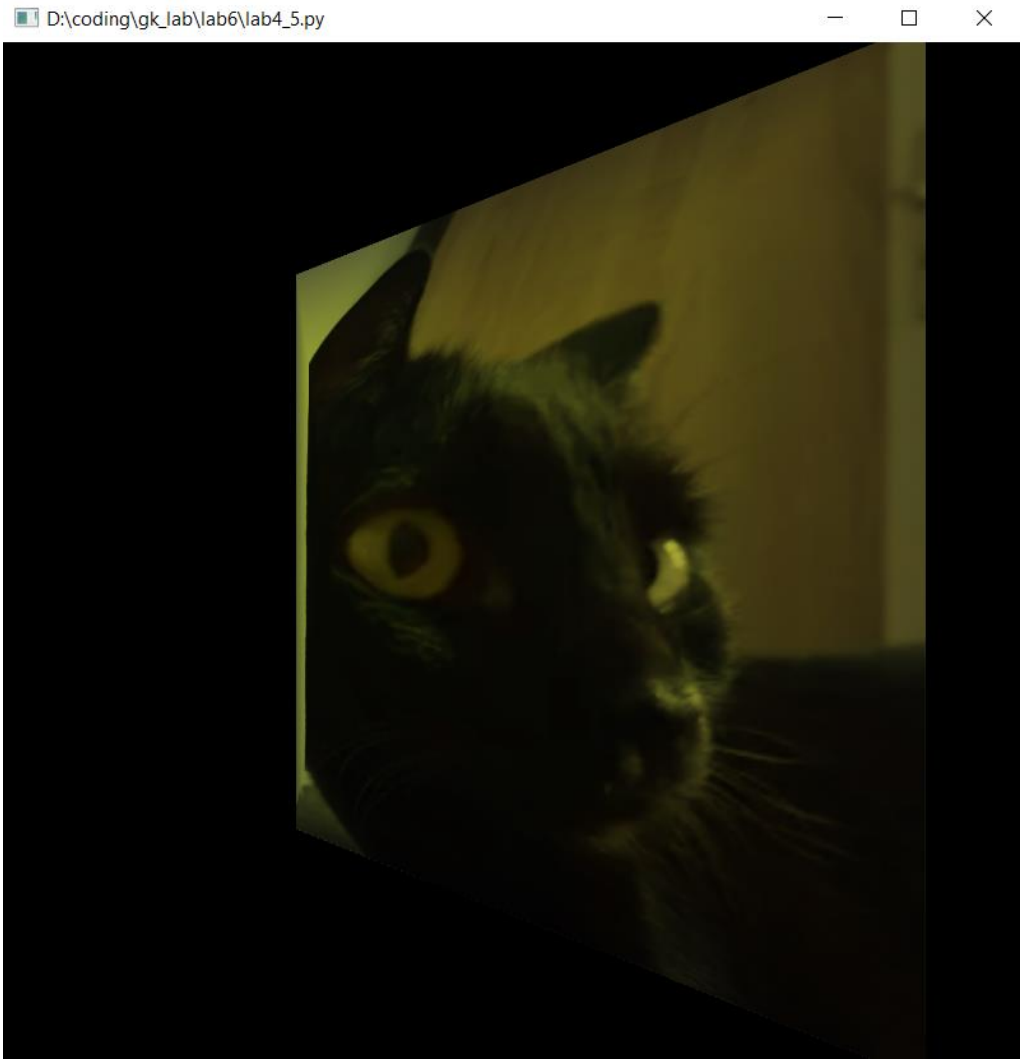
```
127     if img_choice:
128         glTexImage2D(
129         GL_TEXTURE_2D, 0, 3, image0.size[0], image0.size[1], 0,
130         GL_RGB, GL_UNSIGNED_BYTE, image0.tobytes("raw", "RGB", 0, -1)
131         )
132     else:
133         glTexImage2D(
134         GL_TEXTURE_2D, 0, 3, image1.size[0], image1.size[1], 0,
135         GL_RGB, GL_UNSIGNED_BYTE, image1.tobytes("raw", "RGB", 0, -1)
136         )
```

Rysunek 6.2 Zmodyfikowany fragment funkcji render()

```
229 def keyboard_key_callback(window, key, scancode, action, mods):
230     global img_choice, wall_hidden
231     if key == GLFW_KEY_ESCAPE and action == GLFW_PRESS:
232         glfwSetWindowShouldClose(window, GLFW_TRUE)
233     if key == GLFW_KEY_C and action == GLFW_PRESS:
234         img_choice = not img_choice
235     if key == GLFW_KEY_H and action == GLFW_PRESS:
236         wall_hidden = not wall_hidden
```

Rysunek 6.3 Zmodyfikowana funkcja keyboard_key_callback() dla zadania na ocenę 4.5

Efekt działania skryptu w postaci przełączonej tekstury przedstawia Rysunek 6.4.



Rysunek 6.4 Efekt działania skryptu na ocenę 4.5