Wrocław
University
of Science
and Technology

<table>
<tr><td>POLITECHNIKA
WROCŁAWSKA</td><td>Autor:
Daniel Gościński
280878</td><td>Wydział Informatyki i Telekomunikacji

Rok:2025
Rok akadem.: 3</td></tr>
<tr><td colspan="3">Grafika komputerowa i komunikacja człowiek-komputer</td></tr>
<tr><td>Data ćwiczenia:
22.10.2025</td><td rowspan="2">Temat ćwiczenia laboratoryjnego:

Podstawy OpenGL, grafika 2D</td><td rowspan="2">Prowadzący:
Dr inż. Arch.
Tomasz Zamojski</td></tr>
<tr><td>Nr ćwiczenia: 2</td></tr>
</table>

Spis treści

1. Wstęp

Podczas zajęć zapoznano się z podstawowymi operacjami z wykorzystaniem OpenGL poprzez wykonywanie różnych przekształceń obrazu. Zadania zostały wykonane w języku programowania Python z biblioteką PyOpenGL.

Przed przystąpieniem do wykonywania zadań zapoznano się z podstawywmi funkcjami biblioteki, takie jak:

- **glColor3f** - zajmuje się definiowaniem koloru
- **glBegin / glEnd** - zajmują się definiowaniem początku/końca rysowania
- **glVertex2f** - zajmują się definiowaniem położenia punktów

Zainstalowao również wymagane dowiązania (PyOpenGL oraz GLFW)

2. Opis zadań opracowanych w ramach laboratorium

Celem ćwiczenia było zaprezentowanie możliwości biblioteki OpenGL wraz z rozszerzeniem GLUT (GL Utility Toolkit) poprzez wykonanie elementarych operacji pokroju tworzenia prymitywów w przestrzeni 2D oraz deklarowania kolorów wierzchołków. Zadaniem wieńczącym było narysowanie Dywanu Sierpińskiego z zadaną skalą samopodobieństwa.

2.1. Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z podstawowymi operacjami umożliwianymi przez interfejs OpenGL w wymiarze 2D.

2.2. Wykonane zadania

Wykonane zostało 5 zadań z listy zadań dostarczonej przez prowadzącego:

1. Narysowanie trójkąta, którego każdy wierzchołek ma inny kolor.
2. Utworzenie funkcji pozwalającej na wygenerowanie prostokąta w podanym miejscu.
3. Zmodyfikowanie funkcji z zadania 2, rozszerzając ją o losowość kolorów oraz o możliwość modyfikacji rozmiaru prostokąta współczynnikiem skalującym.
4. Narysowanie Dywanu Sierpińskiego, którego stopień samopodobieństwa podawany jest jako argument programu.
5. Narysowanie wybranego drugiego fraktalu, podobnie do zadania 4.

2.3. Prezentacja i omówienie poszczególnych funkcjonalności

Plik źródłowy każdego z zadań bazowany jest na przykładowym pliku dostarczonym przez prowadzacego, na podstawie którego nastepnie wykonywane były zadania

2.3.1. Zadanie 1

Do pliku dołączonego do zadań zmodyfikowano funkcję render, w celu utworzenia pojedyńczego trójkąta.

```
def render(time):
    glClear(GL_COLOR_BUFFER_BIT)

    glBegin(GL_TRIANGLES)
    glColor3f(1,0,0)
    glVertex2f(-50,0)
    glColor3f(0,1,0)
    glVertex2f(50,0)
    glColor3f(0,0,1)
    glVertex2f(0,50)
    glEnd()
    glFlush()
```

Powyższy kod tworzy pojedyńczy trójkąt, którego każdy z wierzchołków ma inny kolor (czerwony, zielony, niebieski), tworząc figurę zawierające różne miszkanki kolorów.

2.3.2. Zadanie 2

W zadaniu zamieniono funkcję render(), która od teraz wykonuje tylko polecenie glFlush(), który wyświetla narysowane obiekty na ekran. Utworzono nową funkcję rectangle, która przyjmuje koordynaty prawego górnego wierzchołka, oraz długość obu boku. Na tej podstawie tworzone są 2 trójkąty, które łącznie tworzą oczekiwany prostokąt.

```
def rectangle(x,y,a,b):
    glClear(GL_COLOR_BUFFER_BIT)

    glColor3f(0,0,100)
    glBegin(GL_TRIANGLES)
    glVertex2f(x,y)
    glVertex2f(x,y-b)
    glVertex2f(x+a,y-b)
    glEnd()

    glBegin(GL_TRIANGLES)
    glVertex2f(x,y)
    glVertex2f(x+a,y)
    glVertex2f(x+a,y-b)
    glEnd()

    render(glfwGetTime())
```

funkcja tworząca wybrany prostokąt, który składa się z 2 trójkątów.

2.3.3. Zadanie 3

Utworzoną funkcję do generowania prostokątów zmodyfikowano o nowy parametr, który modyfikuje wymiary prostokąta. Dodatkowo losowany jest jego kolor, który zmienia się co zadaną jednostkę czasu.

```
    horizontal=(a*d)+glfwGetTime()  #rozszerzają się z czasem trwania
programu
    vertical=(b*d)+glfwGetTime()    #rozszerzają się z czasem trwania
programu
    colorR=random.random()
    colorG=random.random()
    colorB=random.random()
```

Dodatkowo w celu rozszerzenia zadania do wymiarów dodawana jest jednostka czasu, co rozszerza prostokąty co każdy render figur.

Ponieważ uruchomienie programu spowodowałoby bardzo szybkie zmiany kolorów (kolory są generowane co każdą generację obrazu) dodano przed wywołaniem funkcji render() rozkaz **time.sleep(1)**, w celu spowolnienia działania prorgamu

W celu sprawdzenia działania wygenerowano 2 figury o następujących parametrach:

```
rectangle(-50,50,50,25,0.5)
rectangle(-50,-50,50,25,1)
```

2.3.4. Zadanie 4

W celu utworzenia Dywanu Sierpińskiego wykorzystano wcześniej utworzoną funkcję generującą prostokąty, pozbawionej generacji losowych kolorów oraz funkcji sleep (ponieważ kolory nie są zmieniane nie ma potrzebny spowalniać procesu)

```
def Sierpinski(x,y,dl,szer,d,color,stopien):
    if stopien == 0:
        rectangle(x, y, dl, szer, d, color)
        return
    rectangle(x, y, dl, szer, 1.0, (1.0, 1.0, 1.0))
    step = dl / 3.0
    for row in range(3):
        for col in range(3):
            if row == 1 and col == 1:
               continue
            nx = x + col * step
            ny = y - row * step
            Sierpinski(nx, ny, step, step, 1.0, color, stopien - 1)
```

Utworzona funkcja rekurencyjna sprawdza najpierw, czy stopien samopodobieństwa wynosi 0. Jeśli tak, generowany jest zwykły prostokąt. Jeśli wartość jest inna, wymiary dzielone są na 3 równe części, i dla każdego fragmentu wywoływana jest funkcja Sierpinski z nowymi wymiarami, oraz pomniejszonym stopniem o 1. Wynikiem końcowym jest uzyskany prawidłowo fraktal.

Program uruchamiany był poprzez polecenie **py ./zad4.py n**, gdzie n jest liczbą określającą stopien samopodobieństwa.

2.3.5. Zadanie 5

Drugim, wybranym fraktalem został trójkąt Sierpińskiego. W celu jego utworzenia napisano najpierw funkcję pozwalającą na tworzenie trójkątów równobocznych

```
def draw_triangle(center, size):
    cx, cy = center
    h = math.sqrt(3) / 2 * size
    # Wierzchołki trójkąta równobocznego
    return [
        (cx - size / 2, cy - h / 3),
        (cx + size / 2, cy - h / 3),
        (cx, cy + 2 * h / 3)
    ]
```

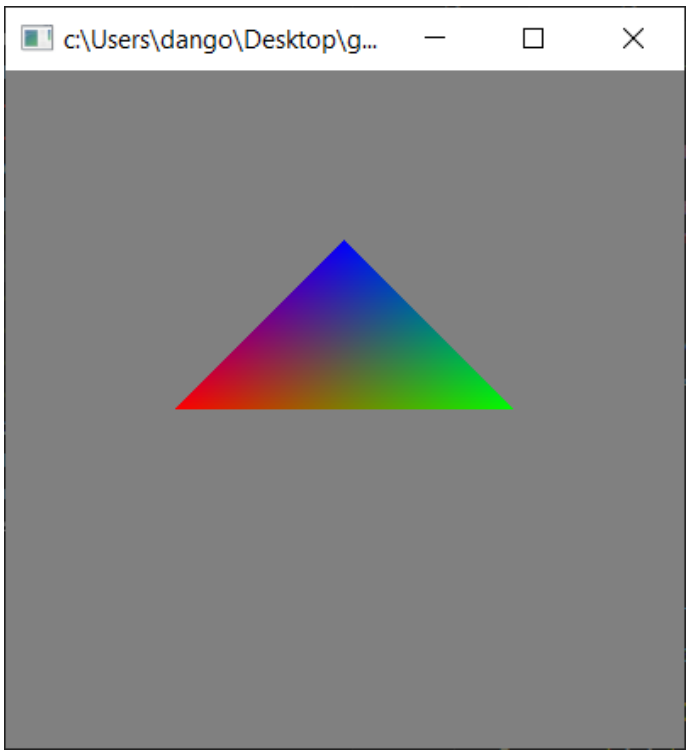
Funkcja zwraca w tablicy wszystkie wierzchołki trójkąta. Następnie utworzono funkcję analogiczną do dywanu Sierpińskeigo, która rekurencyjnie generuje trójkąty

```
def draw_sierpinski(vertices, depth):
    if depth == 0:
        glBegin(GL_TRIANGLES)
        for x, y in vertices:
            glVertex2f(x, y)
        glEnd()
    else:
        # Oblicz punkty środkowe boków
        midpoints = [
            ((vertices[0][0] + vertices[1][0]) / 2, (vertices[0][1] +
vertices[1][1]) / 2),
            ((vertices[1][0] + vertices[2][0]) / 2, (vertices[1][1] +
vertices[2][1]) / 2),
            ((vertices[2][0] + vertices[0][0]) / 2, (vertices[2][1] +
vertices[0][1]) / 2),
        ]
        # Rekurencja dla trzech mniejszych trójkątów
        draw_sierpinski([vertices[0], midpoints[0], midpoints[2]], depth -
1)
        draw_sierpinski([vertices[1], midpoints[1], midpoints[0]], depth -
1)
        draw_sierpinski([vertices[2], midpoints[2], midpoints[1]], depth -
1)
```

Program uruchomiony przez polecenie **py ./zad5.py n** gdzie n jest liczbą określającą stopień samopodobieństwa poprawnie rysuje rekurencyjnie trójkąt Sierpińskiego.

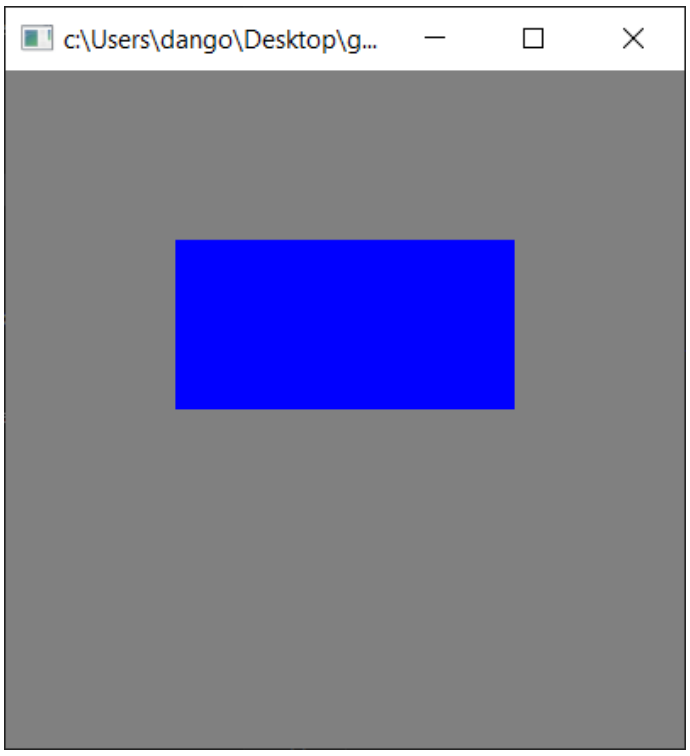
2.4. Efekt wykonanej pracy

2.4.1. Zadanie 1



Zdjęcie 1: Efekt działania programu dla zadania 1

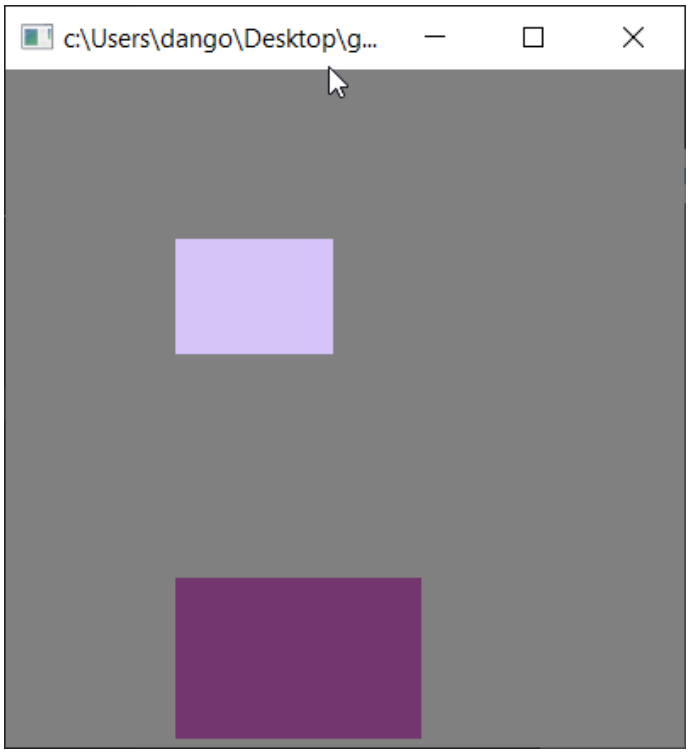
2.4.2. Zadanie 2



Zdjęcie 2: Efekt działania programu dla zadania 2

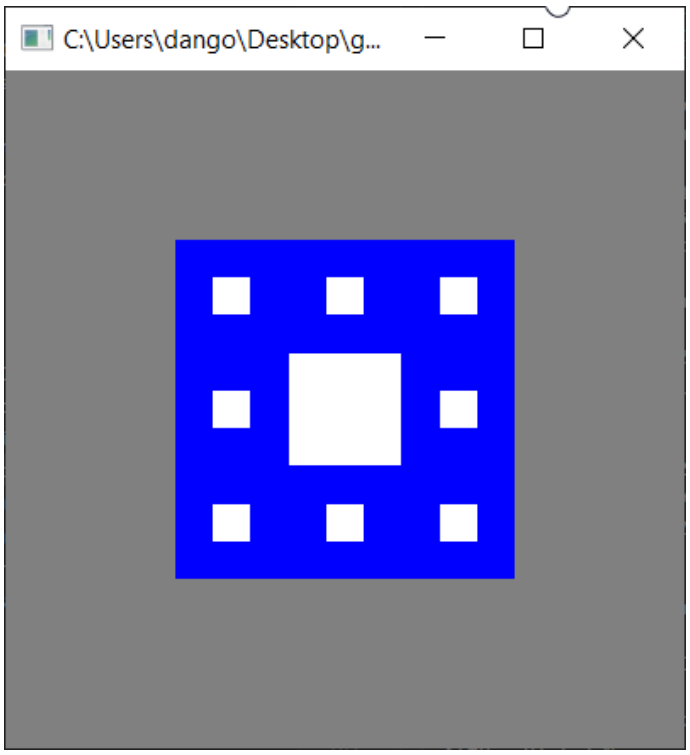
2.4.3. Zadanie 3

Zdjęcie 3: Efekt działania programu dla zadania 3 po uruchomieniu programu

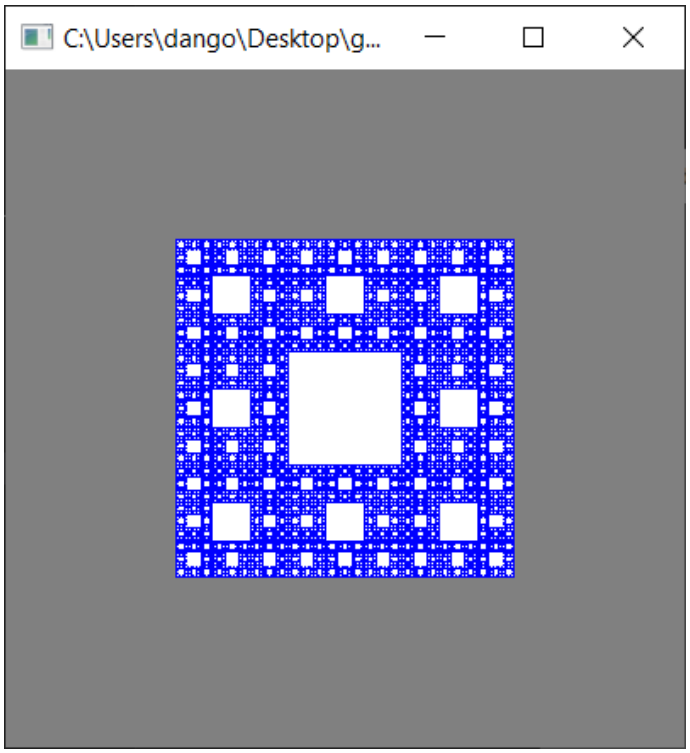


Zdjęcie 4: Efekt działania programu dla zadania 3 po upływie czasu

2.4.4. Zadanie 4

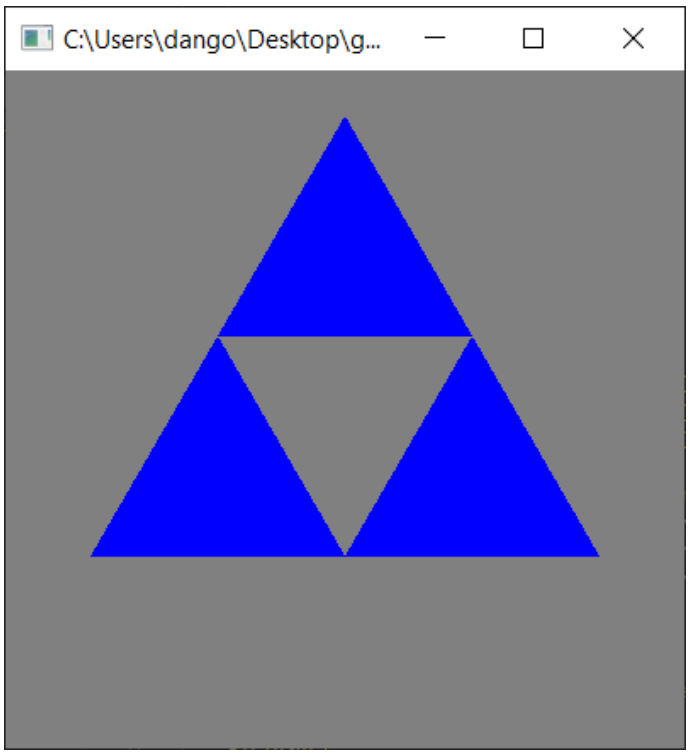


Zdjęcie 5: Efekt działania programu dla zadania 4 dla stopnia saompodobieństwa 2

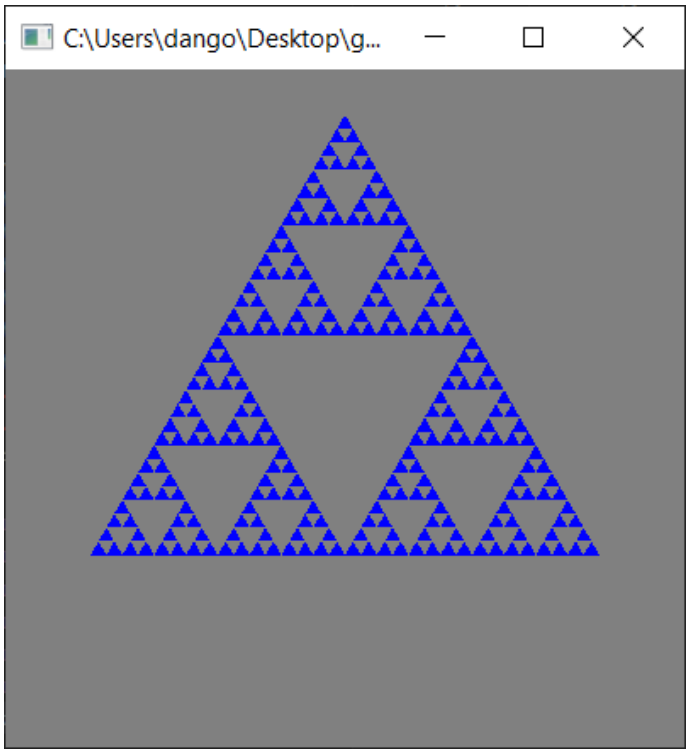


Zdjęcie 6: Efekt działania programu dla zadania 4 dla stopnia saompodobieństwa 5

2.4.5. Zadanie 5



Zdjęcie 7: Efekt działania programu dla zadania 5 dla stopnia saompodobieństwa 1



Zdjęcie 8: Efekt działania programu dla zadania 5 dla stopnia saompodobieństwa 5

3. Podsumowanie

Tworzenie grafik 2D przy pomocy OpenGL wydaje się relatywnie proste, choć nadal potrzebna jest wiedza z zakresu obsługi biblioteki. Atutem jest możliwość szybkiego uruchomienia programu, co pozwala na graficzne zapoznanie się z wynikiem programu, dzięki czemu można szybko zauważyć błędy. Składnia nie wydaje się skomplikowana, pomimo długich nazw funkcji. Wiedza uzyskana w wyniku wykonania zadań pozwoli na późniejsze poszerzenie wiedzy oraz łatwiejsze wykonywanie zadań na następnych zajęciach.

4. Literatura

[1] Handbook of Geometric Programming Using Open Geometry GL

[2] OpenGL Programming Guide. The Official Guide to Learning OpenGL, Version 1.1. OpenGL Architecture Review Board: Mason Woo, Jackie Neider, Tom Davis. Addison-Wesley Developers Press,1997

[3] Strona internetowa dr inż. Szymona Datko: https://datko.pl/GK/

[4] Kanał YouTube Szymona Datko z omówionymi poszczególnymi laboratoriami dot. GRAFIKI KOMPUTEROWEJ: https://youtube.com/playlist?list=PLVFXq_zlHdh_lJQDQD 8AjavN2TmNhFN- P&si=aVqVD4ulVug4VCYI