<table>
<tr><td>Politechnika Wrocławska</td><td>Autor: Piotr Józefek 272311</td><td>Wydział Informatyki i Telekomunikacji<br>Rok akademicki: 3</td></tr>
<tr><td colspan="3">Grafika komputerowa i komunikacja człowiek – komputer</td></tr>
<tr><td>Data ćwiczenia:<br>15.10.2024</td><td rowspan="2">Temat ćwiczenia laboratoryjnego:<br>Rysowanie obiektów 2-D przy pomocy tworów pierwotnych biblioteki graficznej OpenGL</td><td>Ocena:</td></tr>
<tr><td>Nr ćwiczenia:<br>2</td><td>Prowadzący:<br>dr inż. arch. Tomasz Zamojski</td></tr>
</table>

## 1. Wstęp

OpenGL, czyli Open Graphics Library, to otwarty standard interfejsu API do tworzenia grafiki komputerowej, zapoczątkowany przez firmę Silicon Graphics Inc. na początku lat 90. XX wieku. Powstał jako rozwinięcie ich wcześniejszego, własnościowego systemu IRIS GL (Integrated Raster Imaging System Graphics Library), który stworzono na potrzeby wysokowydajnych stacji graficznych. W 1992 roku, po usunięciu elementów własnościowych, OpenGL stał się otwartym standardem, oferującym zunifikowany zestaw funkcji do generowania grafiki komputerowej, bez względu na platformę sprzętową. Dzięki temu, OpenGL szybko zyskał uznanie, gdyż spełniał rosnące wymagania branży na wydajną i przenośną grafikę. Formalnie OpenGL nie jest biblioteką, lecz specyfikacją API, która definiuje, w jaki sposób mają być realizowane operacje graficzne, pozostawiając implementację odpowiednim sterownikom sprzętu. OpenGL pozwala programistom na efektywne tworzenie i zarządzanie trójwymiarowymi obrazami, opierając się na prymitywach graficznych – podstawowych jednostkach renderingu, takich jak punkty, linie i trójkąty. Do generowania obrazu stosuje się proces rasteryzacji, czyli przekształcania prymitywów w zbiór pikseli na ekranie. Wspierany przez wiodące systemy operacyjne, OpenGL umożliwia renderowanie zarówno prostych obiektów, jak i złożonych scen 3D. Konwencje nazewnictwa funkcji, jak choćby glColor3ub() czy glColor3fv(), pozwalają na jednoznaczne przekazywanie parametrów, co znacznie ułatwia pracę programistów. Obecnie rozwój standardu jest nadzorowany przez Khronos Group, a OpenGL Architectural Review Board (ARB) ustala jego przyszłe wersje i ulepszenia. OpenGL obejmuje nie tylko tworzenie obrazu, ale także pracę z modelami 3D, kolorami, teksturami oraz transformacjami geometrycznymi. Współczesne aplikacje wykorzystujące OpenGL mogą korzystać z dodatkowych narzędzi, takich jak GLFW do zarządzania oknem wyświetlania, czy zestawów funkcji matematycznych, które wspierają transformacje we współczesnych wersjach standardu.

## 2. Cel ćwiczenia

- Zapoznanie się z podstawowymi elementami grafiki komputerowej.
- Zgrubne zrozumienie procesu powstawania obrazu w komputerze.
- Oswojenie się z interfejsem OpenGL, na przykładach 2-wymiarowych.

## 3. Wykonane zadania

Na laboratoriach udało się zrealizować: przygotowanie podstawowego środowiska pracy,  funkcję rysującą prostokąt w podanym miejscu, wprowadzić losowość kolorów i deformacje w prostokącie, narysować fraktal – prostokątny dywan Sierpińskiego oraz fraktal płatek śniegu Kocha

## 4. Prezentacja i omówienie funkcjonalności

4.1 Przygotowanie środowiska pracy

Celem zadania było narysowanie trójkąta z każdym wierzchołkiem innego koloru oraz ustawienie odpowiednich parametrów rzutni. Do tego celu została utworzona funkcja render(time). Najpierw czyści ona ekran za pomocą glClear(GL_COLOR_BUFFER_BIT), co przygotowuje obszar do rysowania. Następnie, glBegin(GL_TRIANGLES) rozpoczyna rysowanie trójkąta, w którym przed każdym dodaniem wierzchołka używana jest funkcja glColor3f() do nadania nowego koloru. glVertex2f() ustala położenie wierzchołków. Na zakończenie glEnd() kończy definiowanie kształtu, a glFlush() powoduje wyświetlenie efektu. Efektem jest utworzenie trójkąta którego każdy wierzchołek ma inny kolor: czerwony, zielony i niebieski.

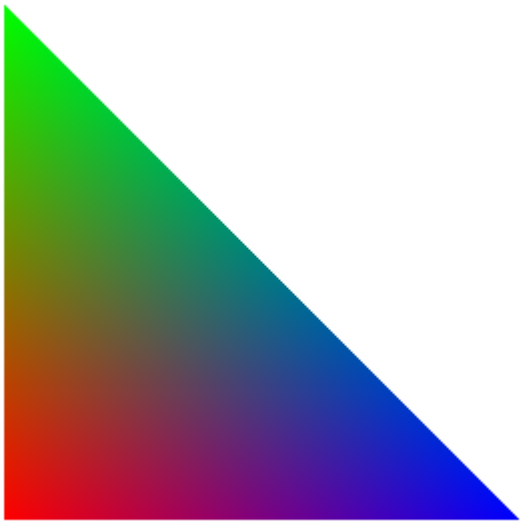
```
def render(time):
    glClear(GL_COLOR_BUFFER_BIT)

    glBegin(GL_TRIANGLES)
    glColor3f(1, 0, 0)
    glVertex2f(0.0, 0.0)
    glColor3f(0, 1, 0)
    glVertex2f(0.0, 50.0)
    glColor3f(0, 0, 1)
    glVertex2f(50.0, 0.0)
    glEnd()
    glFlush()
```

*Rys 1 Funkcja odpowiedzialna za kolorowy trójkąt*

Funkcja update_viewport(window, width, height) dostosowuje parametry rzutni w odpowiedzi na zmianę rozmiaru okna. Na początku, jeśli szerokość lub wysokość są równe zero, przypisuje im wartość 1, aby uniknąć błędów. Następnie oblicza współczynnik proporcji (aspect_ratio) i przełącza macierz na tryb projekcji za pomocą glMatrixMode(GL_PROJECTION).Funkcja ustawia obszar widoku do rozmiaru okna przy użyciu glViewport(0, 0, width, height) i resetuje macierz przez glLoadIdentity(). W zależności od proporcji, ustawia granice widoku za pomocą glOrtho(...), aby odpowiednio skalować widok. Na końcu przywraca macierz modelowania, resetując ją ponownie, co zapewnia prawidłowe wyświetlanie sceny w nowym rozmiarze okna.
Dzięki temu zakresu -1 do 1 odpowiada nowemu zakresowi -100 do 100

```python
def update_viewport(window, width, height):
    if width == 0:
        width = 1
    if height == 0:
        height = 1
    aspect_ratio = width / height

    glMatrixMode(GL_PROJECTION)
    glViewport(0, 0, width, height)
    glLoadIdentity()

    if width <= height:
        glOrtho(-100.0, 100.0, -100.0 / aspect_ratio, 100.0 / aspect_ratio,
                1.0, -1.0)
    else:
        glOrtho(-100.0 * aspect_ratio, 100.0 * aspect_ratio, -100.0, 100.0,
                1.0, -1.0)

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
```

*Rys 2 Funkcja odpowiedzialna za ustawienie parametrów rzutni*



*Rys 3 Narysowany trójkąt z różnokolorowymi wierzchołkami*

4.2 Rysowanie prostokąta

Funkcja draw_rectangle(x, y, a, b) rysuje prostokąt w oknie OpenGL, a jej parametry to: x, czyli współrzędna X środka prostokąta; y, czyli współrzędna Y środka prostokąta; a, która określa szerokość prostokąta; oraz b, która określa wysokość prostokąta. Działanie funkcji rozpoczyna się od obliczenia współrzędnych czterech wierzchołków prostokąta. Wierzchołki są definiowane jako: lewy dolny (bottom_left), lewy górny (top_left), prawy górny (top_right) oraz prawy dolny (bottom_right). Obliczenia te bazują na podanych współrzędnych środka oraz wartościach szerokości i wysokości.

```
def draw_rectangle(x, y, a, b):

    bottom_left = (x - a / 2, y - b / 2)
    top_left = (x - a / 2, y + b / 2)
    top_right = (x + a / 2, y + b / 2)
    bottom_right = (x + a / 2, y - b / 2)
    glColor3f(1, 0, 0)
    glBegin(GL_TRIANGLES)
    glVertex2f(*bottom_left)
    glVertex2f(*top_left)
    glVertex2f(*bottom_right)
    glVertex2f(*top_left)
    glVertex2f(*top_right)
    glVertex2f(*bottom_right)
    glEnd()

def render_rectangle():
    glClear(GL_COLOR_BUFFER_BIT)
    draw_rectangle(0, 0, 100.0, 50.0)
    glFlush()
```

*Rys 4 Funkcja odpowiedzialna za rysowanie prostokąta*

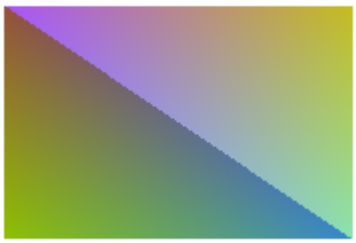
*Rys 5 Narysowany prostokąt*

4.3 Rysowanie zdeformowanego prostokąta

Funkcja draw_rectangle_random(x, y, a, b, d) rysuje prostokąt z losowymi modyfikacjami jego wymiarów i kolorów. Dodatkowym parametrem w tej funkcji jest d, który określa maksymalne odchylenie od oryginalnych wymiarów prostokąta. Wewnątrz funkcji, szerokość a oraz wysokość b są modyfikowane, aby dodać losowy element. Podczas rysowania prostokąta, kolory jego wierzchołków są również losowo generowane. Dla każdego wierzchołka używana jest funkcja random.random(), aby ustawić kolor w zakresie od 0 do 1 dla wartości czerwonej, zielonej i niebieskiej, co skutkuje tym, że każdy z wierzchołków prostokąta może mieć inny, przypadkowy kolor. Funkcja render_rectangle_random(random_seed) przygotowuje losowość przez zainicjowanie generatora liczb pseudolosowych przy użyciu random.seed(random_seed).

```
def draw_rectangle_random(x, y, a, b, d):

    a = a * (1 + random.uniform(-d,d))
    b = b * (1 + random.uniform(-d,d))

    bottom_left = (x - a / 2, y - b / 2)
    top_left = (x - a / 2, y + b / 2)
    top_right = (x + a / 2, y + b / 2)
    bottom_right = (x + a / 2, y - b / 2)


    glBegin(GL_TRIANGLES)
    glColor3f(random.random(), random.random(), random.random())
    glVertex2f(*bottom_left)
    glColor3f(random.random(), random.random(), random.random())
    glVertex2f(*top_left)
    glColor3f(random.random(), random.random(), random.random())
    glVertex2f(*bottom_right)
    glColor3f(random.random(), random.random(), random.random())
    glVertex2f(*top_left)
    glColor3f(random.random(), random.random(), random.random())
    glVertex2f(*top_right)
    glColor3f(random.random(), random.random(), random.random())
    glVertex2f(*bottom_right)
    glColor3f(random.random(), random.random(), random.random())
    glEnd()


def render_rectangle_random(random_seed):
    random.seed(random_seed)
    glClear(GL_COLOR_BUFFER_BIT)
    draw_rectangle_random(0, 0, 100.0, 50.0, 0.2)
    glFlush()
```

*Rys 6 Funkcja odpowiedzialna za rysowanie prostokąta oraz generowanie seeda*



*Rys 7 Zdeformowany prostokąt*

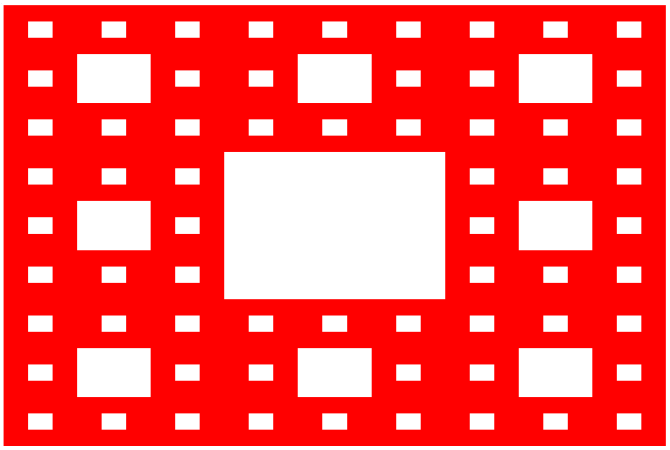
*Rys 8 Zdeformowany prostokąt na innym losowo wygenerowanym seedzie*

### 4.4 Fraktal – prostokątny dywan Sierpińskiego

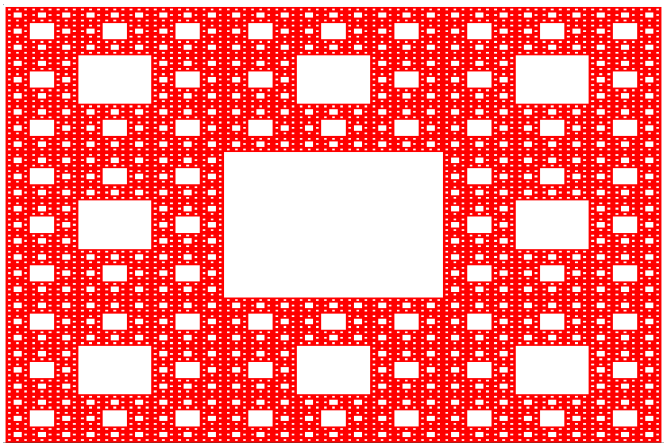
Funkcja carpet(x, y, a, b, depth) generuje fraktalny wzór znany jako prostokątny dywan Sierpińskiego. Parametry x i y definiują środek prostokąta, a to jego szerokość, b wysokość, a depth wskazuje na poziom rekurencji. Gdy depth osiągnie zero, funkcja wywołuje draw_rectangle(x, y, a, b), co rysuje prostokąt w określonej lokalizacji. Jeśli depth jest większe od zera, funkcja dzieli szerokość i wysokość prostokąta na trzy równe części, co pozwala na stworzenie mniejszych prostokątów. Następnie, za pomocą podwójnej pętli, iteruje przez 8 pozycji wokół środka prostokąta, pomijając centralny element. Dla każdej z pozostałych pozycji rekurencyjnie wywołuje samą siebie, przesuwając się do nowych współrzędnych i zmniejszając rekurencje o 1.

```python
def carpet(x, y, a, b, depth):
    if depth == 0:
        draw_rectangle (x, y, a, b)
    else:
        new_a = a/3
        new_b = b/3
        for i in range(-1,2):
            for j in range (-1,2):
                if i==0 and j==0:
                    continue
                carpet(x+i*new_a, y+j*new_b, new_a, new_b, depth-1)

def render_carpet(depth):
    glClear(GL_COLOR_BUFFER_BIT)
    carpet(0, 0 ,300, 200, depth)
    glFlush()
```

*Rys 9 funkcja odpowiedzialna za rysowanie* dywanu Sierpińskiego



*Rys 10 D*ywan Sierpińskiego dla parametru n=3



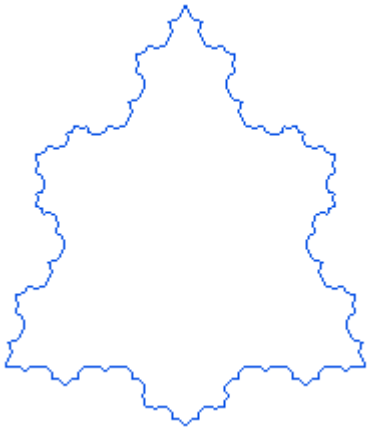
*Rys 11 D*ywan Sierpińskiego dla parametru n=5

4.5 Fraktal - płatek śniegu Kocha

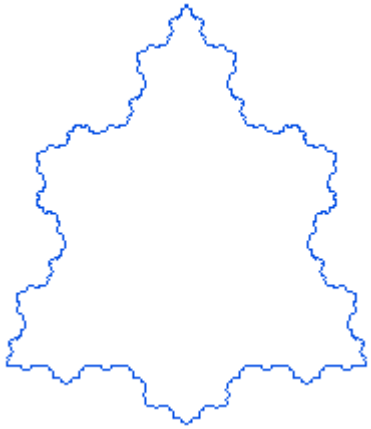
Funkcja koch_curve(x, y, depth) generuje fraktal znany jako krzywa Kocha. Parametry x i y reprezentują współrzędne punktów, między którymi rysowana jest krzywa, a depth wskazuje głębokość rekurencji. Gdy depth wynosi zero, funkcja ustawia kolor linii na niebieski i rysuje prostą linię między punktami x i y przy użyciu glBegin(GL_LINES) i glEnd(). Jeśli depth jest większe od zera, funkcja dzieli odcinek na trzy równe części, obliczając współrzędne punktów a1 i a2 dla dwóch końców. Następnie oblicza położenie punktu a3, który tworzy nowy wierzchołek w kształcie trójkąta, używając funkcji trygonometrycznych do obliczenia nowych współrzędnych. Funkcja następnie rekurencyjnie wywołuje samą siebie cztery razy, aby narysować mniejsze segmenty krzywej. Najpierw rysuje odcinek od punktu x do a1, a następnie przekształca segment między a1 a nowo obliczonym punktem a3. Potem przechodzi do odcinka między a3 a a2, a na końcu łączy a2 z punktem y. W każdym z tych wywołań głębokość rekurencji jest zmniejszana o jeden, co pozwala na stopniowe tworzenie coraz mniejszych detali fraktala w miarę postępu rysowania.

```
def koch_curve(x, y, depth):
    if depth == 0:
        glColor3f(0.0, 0.3, 0.9)
        glBegin(GL_LINES)
        glVertex2f(*x)
        glVertex2f(*y)
        glEnd()
    else:
        a1 = (2 * x[0] + 1 * y[0]) / 3
        b1 = (2 * x[1] + 1 * y[1]) / 3
        a2 = (1 * x[0] + 2 * y[0]) / 3
        b2 = (1 * x[1] + 2 * y[1]) / 3

        angle = math.radians(90)
        a3 = (a1 + a2) / 2 + (math.cos(angle) * (a2 - a1) / 2) - (math.sin(angle) * (b2 - b1) / 2)
        b3 = (b1 + b2) / 2 + (math.sin(angle) * (a2 - a1) / 2) + (math.cos(angle) * (b2 - b1) / 2)

        koch_curve(x, (a1, b1), depth - 1)
        koch_curve((a1, b1), (a3, b3), depth - 1)
        koch_curve((a3, b3), (a2, b2), depth - 1)
        koch_curve((a2, b2), y, depth - 1)

def render_koch(depth):
    glClear(GL_COLOR_BUFFER_BIT)


    cord_x = (45, 0)
    cord_y = (-45, 0)
    cord_z = (0, 90)
    glBegin(GL_TRIANGLES)
    glColor3f(1.0, 1.0, 1.0)
    glVertex2f(*cord_x)
    glVertex2f(*cord_y)
    glVertex2f(*cord_z)
    glEnd()
```

*Rys 12 Funkcja odpowiedzialna za rysowanie* krzywej von Kocha



*Rys 13 Płatek śniegu dla n=3*



*Rys 14 Płatek śniegu dla n=5*

Generowane przez moją funkcje płatki śniegu są lekko zdeformowane na miejscu połączeń krawędzi przez co różnią się od zamieszczonych w materiałach, lecz przez to tworzą równie interesujący obraz bliższy do trójkąta. Różnice mogą być spowodowane innym podejściem do ograniczania obszaru.

**5. Wnioski**

Udało się zrealizować wszystkie zadania. Były one dobrym wstępem do zapoznania się z OpenGl. W zadaniu z rysowaniem zdeformowanego trojkata gdy nie generowalem wczesniej seeda tylko mialem sama funkcje random to prostokąt “migał” poprzez dynamiczne zmienianie proporcji i kolorów.

**6. Literatura**

- Szymon Datko [GK] Lab2 :: Podstawy OpenGL