CIS 515: COMPUTER GRAPHICS
LAB – 3
UNIVERSITY OF MICHIGAN – DEARBORN
FALL 2024

By,
VISHVENDRA REDDY BHOOMIDI
bhoomidi@umich.edu

Task 1 Measurement of Execution Time

1. A)

```
import pygame
import math

pygame.init()
ticks1 = pygame.time.get_ticks()

for x in range(1000000):
    y = math.sqrt(x)

ticks2 = pygame.time.get_ticks()
totalTime = (ticks2 - ticks1) / 1000

print(f"Execution Time is: {totalTime} seconds")
```

```
/home/vishvendra/Documents/Projects/Python/515/.venv/bin/python /home/vishvendra/Documents/Projects/Python/515/main.py
pygame 2.6.0 (SDL 2.28.4, Python 3.12.3)
Hello from the pygame community. https://www.pygame.org/contribute.html
Execution Time is: 0.184 seconds

Process finished with exit code 0
```

1. B)

```
import pygame
import math

pygame.init()
clock = pygame.time.Clock()
totalTime1 = clock.tick()

for x in range(1000000):
    y = math.sqrt(x)

totalTime2 = clock.tick()
totalTime = (totalTime2 - totalTime1) / 1000

print(f"Execution Time is: {totalTime} seconds")
```

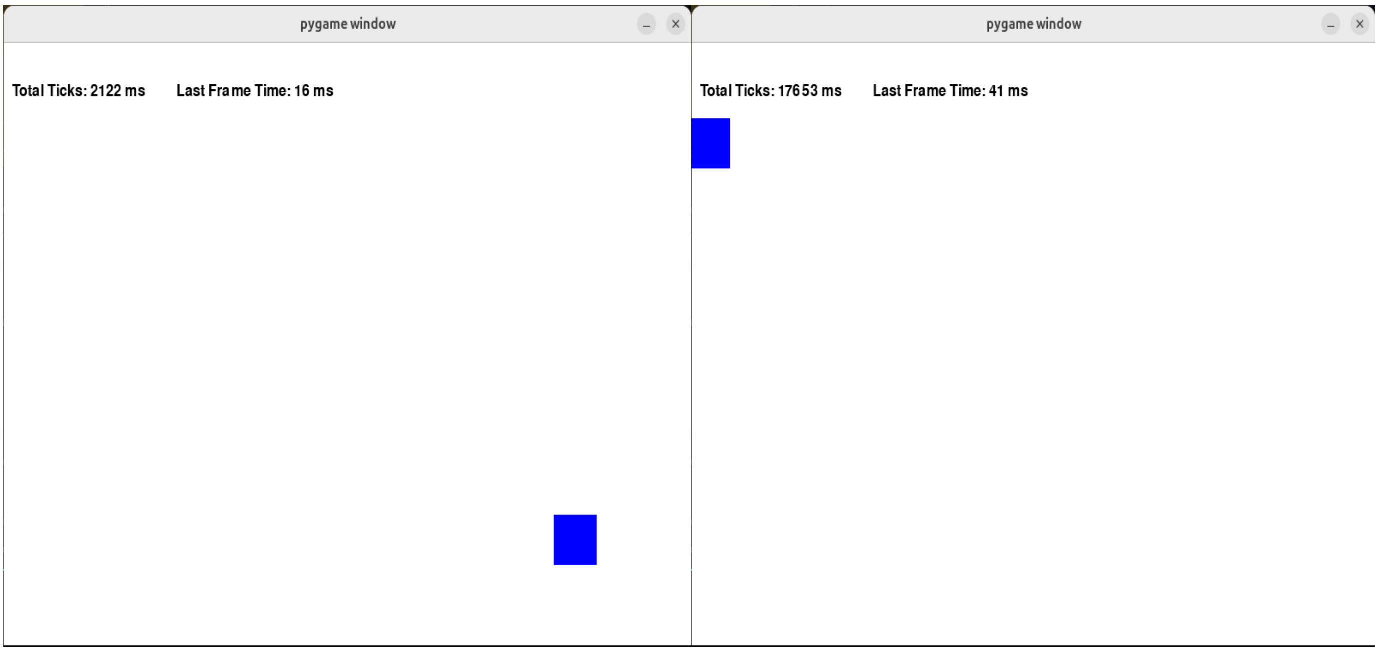
```
/home/vishvendra/Documents/Projects/Python/515/.venv/bin/python /home/vishvendra/Documents/Projects/Python/515/main.py
pygame 2.6.0 (SDL 2.28.4, Python 3.12.3)
Hello from the pygame community. https://www.pygame.org/contribute.html
Execution Time is: 0.163 seconds

Process finished with exit code 0
```

Task 2 Controlling of Frame Rate

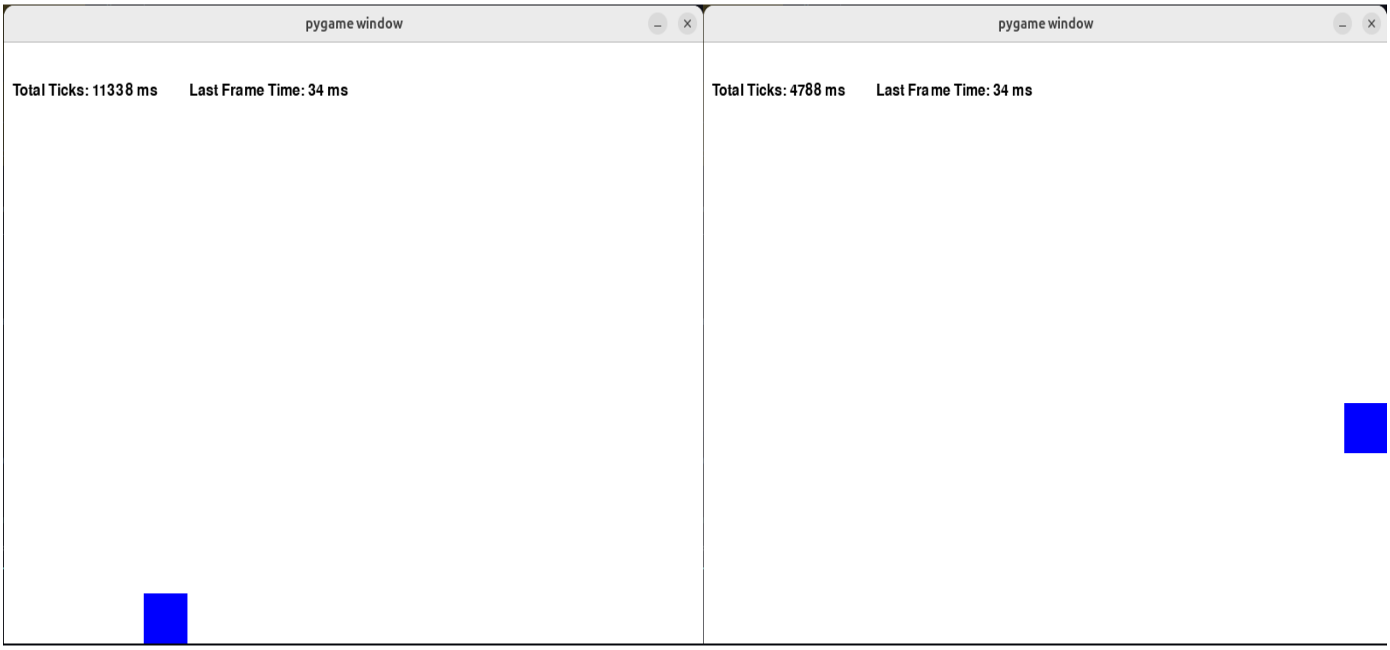
```
import pygame

pygame.init()

FPS = 60

pygame.font.init()
font = pygame.font.SysFont(None, 24)

screen_width, screen_height = 800, 600
screen = pygame.display.set_mode((screen_width, screen_height))

WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
BLACK = (0, 0, 0)

clock = pygame.time.Clock()

rect_x, rect_y = 50, 50
rect_speed_x, rect_speed_y = 5, 5

running = True
lastTicks = 0
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    rect_x += rect_speed_x
    rect_y += rect_speed_y

    if rect_x > screen_width - 50 or rect_x < 0:
        rect_speed_x = -rect_speed_x
    if rect_y > screen_height - 50 or rect_y < 0:
        rect_speed_y = -rect_speed_y

    screen.fill(WHITE)
    pygame.draw.rect(screen, BLUE, (rect_x, rect_y, 50, 50))

    totalTicks = pygame.time.get_ticks()
    lastFrameTime = totalTicks - lastTicks

    text = font.render(f'Total Ticks: {totalTicks} ms        Last Frame Time: {lastFrameTime} ms',
True, BLACK)
    screen.blit(text, (10, 40))
```

```
    clock.tick(FPS)
    lastTicks = totalTicks

    pygame.display.flip()

pygame.quit()
```

WHEN FPS = 60



WHEN FPS = 30

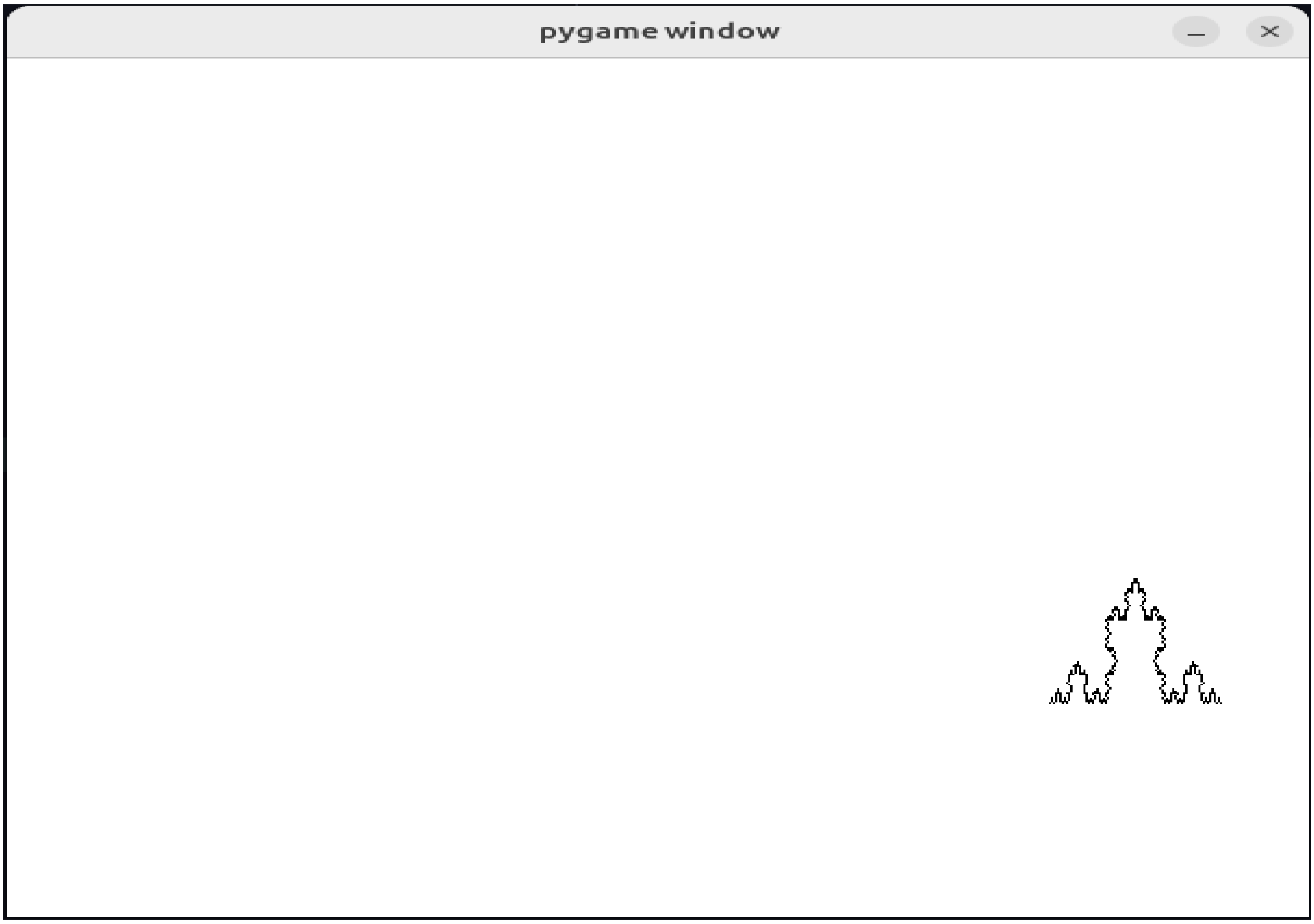


Task 3 2D Transformation

5. A)

```
import pygame
from OpenGL.GL import *
from OpenGL.GLU import *
from math import *


def draw_koch_curve(p1, p2, iterations):
    if iterations == 0:
        glBegin(GL_LINES)
        glVertex2f(p1[0], p1[1])
        glVertex2f(p2[0], p2[1])
        glEnd()
    else:
        one_third = [(2 * p1[0] + p2[0]) / 3, (2 * p1[1] + p2[1]) / 3]
        two_third = [(p1[0] + 2 * p2[0]) / 3, (p1[1] + 2 * p2[1]) / 3]
        dx = p2[0] - p1[0]
        dy = p2[1] - p1[1]
        length = sqrt(dx ** 2 + dy ** 2) / 3
        angle = atan2(dy, dx) + pi / 3
        peak = [one_third[0] + length * cos(angle), one_third[1] + length * sin(angle)]

        draw_koch_curve(p1, one_third, iterations - 1)
        draw_koch_curve(one_third, peak, iterations - 1)
        draw_koch_curve(peak, two_third, iterations - 1)
        draw_koch_curve(two_third, p2, iterations - 1)


def init_window():
    pygame.init()
    display = (600, 600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
    glClearColor(1, 1, 1, 1)
    gluOrtho2D(-1.5, 6, -1, 1)

def main():
    init_window()

    iterations = 4

    vertices = [
        [-0.5, -0.5],
        [0.5, -0.5]
    ]
```

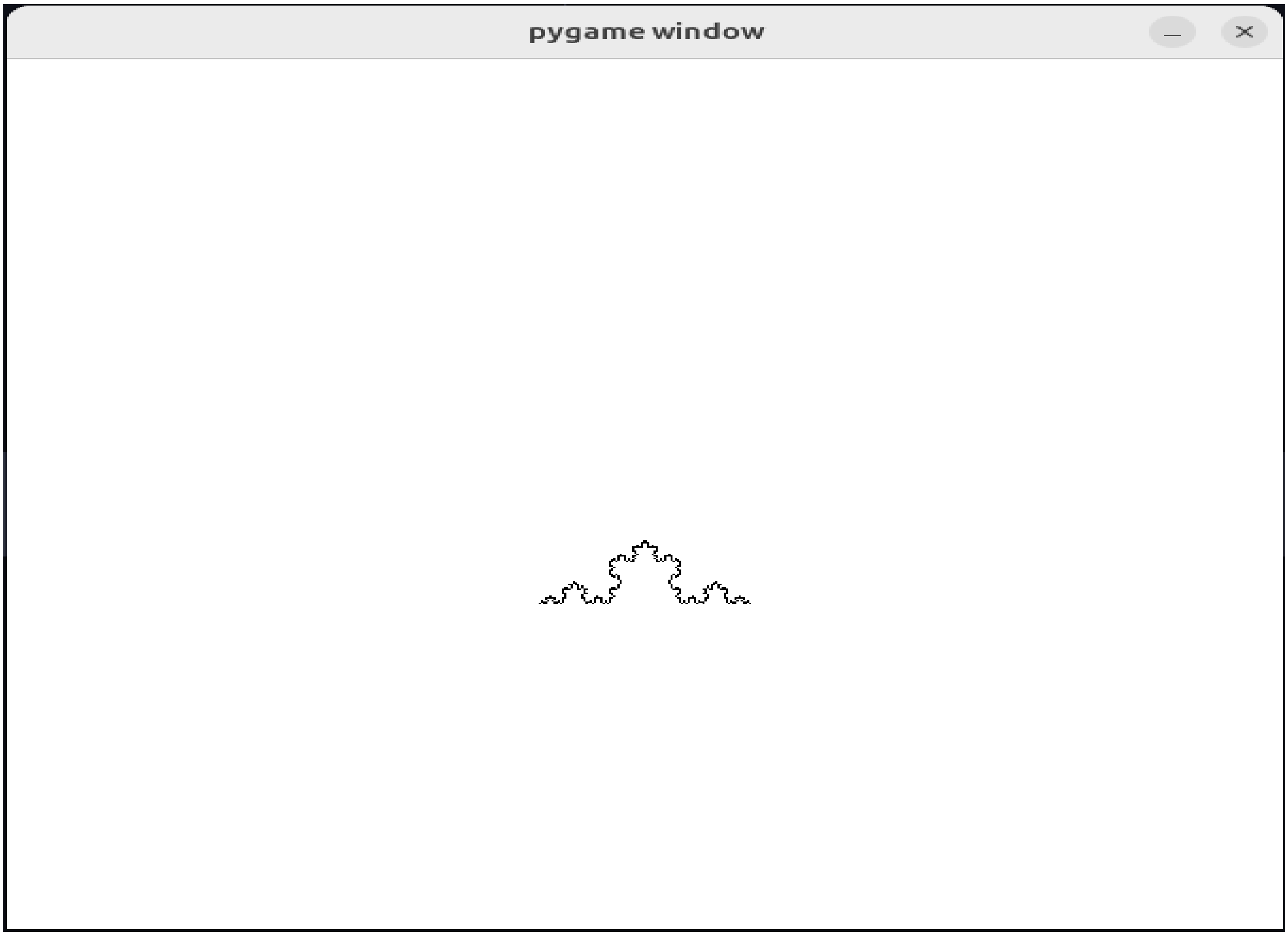
```
    running = True
    while running:
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glColor3f(0, 0, 0)

        glPushMatrix()
        glTranslatef(5, 0, 0)
        draw_koch_curve(vertices[0], vertices[1], iterations)
        glPopMatrix()

        pygame.display.flip()
        pygame.time.wait(10)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

    pygame.quit()


if __name__ == "__main__":
    main()
```



5. B)

```
import pygame
from OpenGL.GL import *
from OpenGL.GLU import *
from math import *


def draw_koch_curve(p1, p2, iterations):
    if iterations == 0:
        glBegin(GL_LINES)
        glVertex2f(p1[0], p1[1])
        glVertex2f(p2[0], p2[1])
        glEnd()
    else:
        one_third = [(2 * p1[0] + p2[0]) / 3, (2 * p1[1] + p2[1]) / 3]
        two_third = [(p1[0] + 2 * p2[0]) / 3, (p1[1] + 2 * p2[1]) / 3]
        dx = p2[0] - p1[0]
        dy = p2[1] - p1[1]
        length = sqrt(dx ** 2 + dy ** 2) / 3
        angle = atan2(dy, dx) + pi / 3
        peak = [one_third[0] + length * cos(angle), one_third[1] + length * sin(angle)]

        draw_koch_curve(p1, one_third, iterations - 1)
        draw_koch_curve(one_third, peak, iterations - 1)
        draw_koch_curve(peak, two_third, iterations - 1)
        draw_koch_curve(two_third, p2, iterations - 1)


def init_window():
    pygame.init()
    display = (600, 600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
    glClearColor(1, 1, 1, 1)
    gluOrtho2D(-1.5, 1.5, -1, 1)


def main():
    init_window()

    iterations = 4

    vertices = [
        [-0.5, -0.5],
        [0.5, -0.5]
    ]

    running = True
```

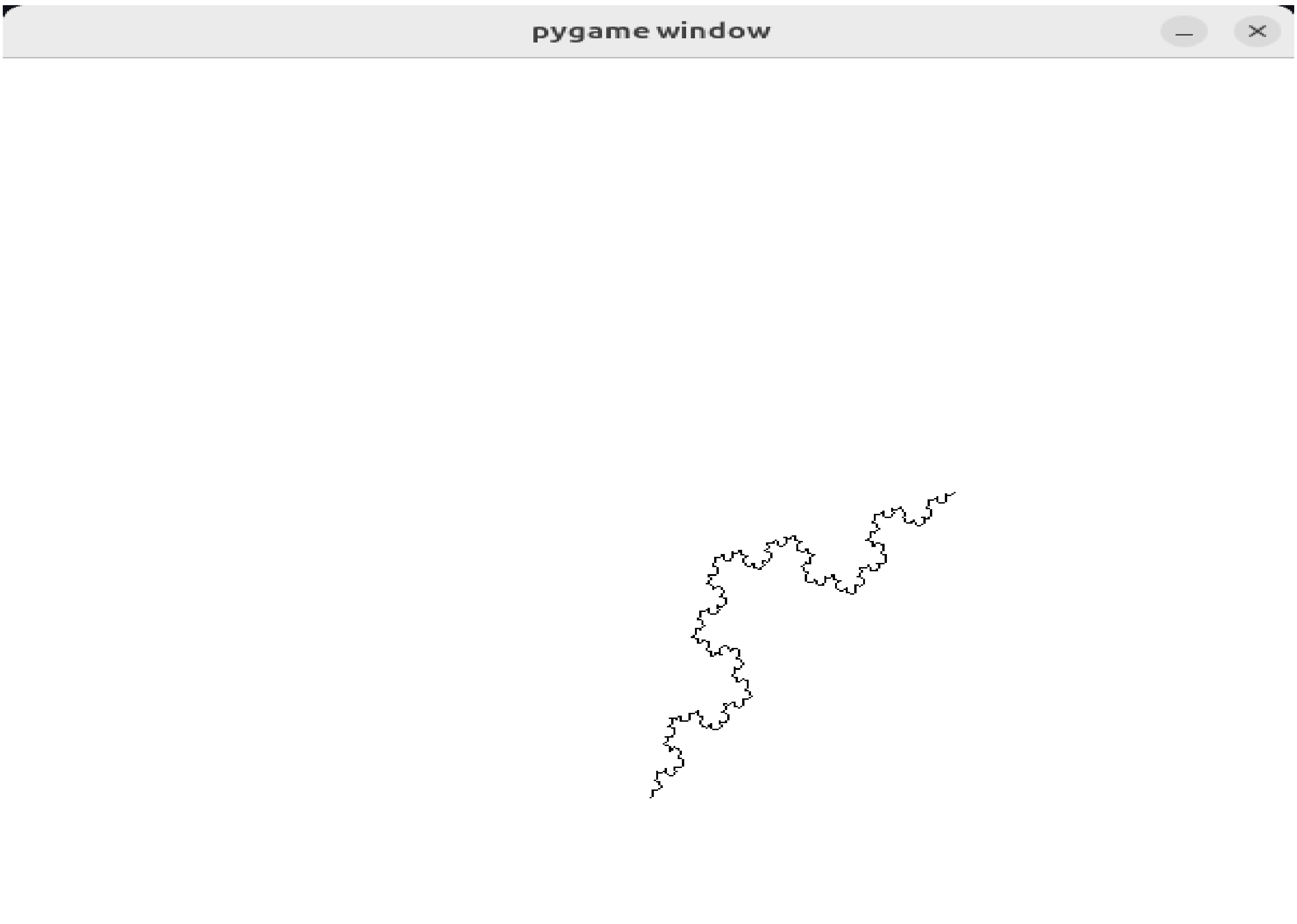
```
    while running:
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glColor3f(0, 0, 0)

        glPushMatrix()
        glScalef(0.5, 0.5, 1.0)
        draw_koch_curve(vertices[0], vertices[1], iterations)
        glPopMatrix()

        pygame.display.flip()
        pygame.time.wait(10)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

    pygame.quit()


if __name__ == "__main__":
    main()
```



5.C)

```
import pygame
from OpenGL.GL import *
from OpenGL.GLU import *
from math import *


def draw_koch_curve(p1, p2, iterations):
    if iterations == 0:
        glBegin(GL_LINES)
        glVertex2f(p1[0], p1[1])
        glVertex2f(p2[0], p2[1])
        glEnd()
    else:
        one_third = [(2 * p1[0] + p2[0]) / 3, (2 * p1[1] + p2[1]) / 3]
        two_third = [(p1[0] + 2 * p2[0]) / 3, (p1[1] + 2 * p2[1]) / 3]
        dx = p2[0] - p1[0]
        dy = p2[1] - p1[1]
        length = sqrt(dx ** 2 + dy ** 2) / 3
        angle = atan2(dy, dx) + pi / 3
        peak = [one_third[0] + length * cos(angle), one_third[1] + length * sin(angle)]

        draw_koch_curve(p1, one_third, iterations - 1)
        draw_koch_curve(one_third, peak, iterations - 1)
        draw_koch_curve(peak, two_third, iterations - 1)
        draw_koch_curve(two_third, p2, iterations - 1)


def init_window():
    pygame.init()
    display = (600, 600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
    glClearColor(1, 1, 1, 1)
    gluOrtho2D(-1.5, 1.5, -1, 1)


def main():
    init_window()

    iterations = 4

    vertices = [
        [-0.5, -0.5],
        [0.5, -0.5]
    ]
```

```
    running = True
    while running:
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glColor3f(0, 0, 0)

        glPushMatrix()
        glRotatef(45, 0, 0, 1)
        draw_koch_curve(vertices[0], vertices[1], iterations)
        glPopMatrix()

        pygame.display.flip()
        pygame.time.wait(10)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

    pygame.quit()


if __name__ == "__main__":
    main()
```



Task 4 Derive the 2D rotation matrix

A 2D rotation matrix for rotating a point (x, y) by an angle θ is given as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Where,

$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ and θ is the angle of rotation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta) \times \begin{bmatrix} x \\ y \end{bmatrix}$$

Task 5 Homogeneous Coordinates

Homogeneous coordinates handle translations as a matrix multiplication instead of adding a vector. This simplifies combining multiple transformations into a single matrix operation. When representing a 2D point (x, y) as (x, y, 1) in homogeneous coordinates, translation can be expressed using a 3×3 matrix. This allows translation to be combined with other transformations such as rotation and scaling in a single matrix operation, simplifying the process of performing multiple transformations. Moreover, it offers a unified view of transformation as matrix multiplication, which is easier in hardware and software.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{xx} & a_{xy} & b_x \\ a_{yx} & a_{yy} & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$