Practical 1 Drawing basic shapes

a. Draw a line using built in graphics fucntion

```
import cv2
import numpy as np

# Create a white background image
image = np.ones((400, 600, 3), dtype=np.uint8) * 255

# Draw a line
# Syntax: cv2.line(image, start_point, end_point, color, thickness)
cv2.line(image, (50, 50), (550, 350), (0, 0, 255), 3)  # Red line

# Display the image
cv2.imshow("Line Drawing", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

b. Draw a circle and ellipse with specified coordinates

```
import cv2
import numpy as np

# Create a blank white canvas
img = np.ones((500, 700, 3), dtype=np.uint8) * 255

# ---- Draw a Circle ----
# cv2.circle(image, center_coordinates, radius, color, thickness)
center_circle = (200, 250)   # x, y coordinates of center
radius = 100                 # radius in pixels
color_circle = (255, 0, 0)   # Blue in BGR
thickness_circle = 3         # thickness (use -1 for filled)

cv2.circle(img, center_circle, radius, color_circle, thickness_circle)

# ---- Draw an Ellipse ----
# cv2.ellipse(image, center_coordinates, axes_length, angle,
startAngle, endAngle, color, thickness)
center_ellipse = (500, 250)  # x, y coordinates of center
axes_length = (150, 100)     # major axis length, minor axis length
angle = 30                   # rotation of ellipse in degrees
start_angle = 0              # starting angle of arc
end_angle = 360              # ending angle of arc (360 = full ellipse)
color_ellipse = (255, 0, 255)  # Red in BGR
thickness_ellipse = 2        # thickness (use -1 for filled)

cv2.ellipse(img, center_ellipse, axes_length, angle, start_angle,
end_angle, color_ellipse, thickness_ellipse)

# Display the result
cv2.imshow("Circle and Ellipse", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

c. Create a polygon with 5 sides

```
import cv2
import numpy as np

# Create a blank white canvas
img = np.ones((500, 700, 3), dtype=np.uint8) * 255

# Define points of a pentagon (x, y) coordinates
# You can adjust values to change size and position
points = np.array([
    [350, 100],   # Top vertex
    [500, 200],
    [450, 350],
    [250, 350],
    [200, 200]
], np.int32)
```

```
# Reshape for OpenCV (each point as separate row)
points = points.reshape((-1, 1, 2))

# Draw polygon
# cv2.polylines(image, [points], isClosed, color, thickness)
cv2.polylines(img, [points], True, (0, 0, 255), 3)  # Red outline

# Optional: Fill the polygon
cv2.fillPoly(img, [points], (255, 255, 0))  # Yellow fill

# Display the image
cv2.imshow("Pentagon", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

d. Draw a fill rectangle with color.

```
import cv2
import numpy as np

# Create a blank white canvas
img = np.ones((400, 600, 3), dtype=np.uint8) * 255

# Rectangle coordinates
start_point = (100, 100)   # Top-left corner (x, y)
end_point = (400, 300)     # Bottom-right corner (x, y)

# Color in BGR format
color = (255, 0, 255)  # Green

cv2.rectangle(img, start_point, end_point, color, thickness=-1)

# Display the image
cv2.imshow("Filled Rectangle", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Practical 2 Line Drawing algorithm

a. Write code for DDA line drawing algorithm

```
import matplotlib.pyplot as plt

def dda_line(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1

    if abs(dx) > abs(dy):
        steps = abs(dx)
    else:
        steps = abs(dy)

    x_increment = dx / steps
    y_increment = dy / steps

    x = x1
    y = y1

    x_coords = []
    y_coords = []
    for _ in range(steps + 1):
        x_coords.append(int(x))
        y_coords.append(int(y))
        x += x_increment
        y += y_increment

    return x_coords, y_coords

# Draw the line
x1, y1 = 10, 10
x2, y2 = 20, 20
x_coords, y_coords = dda_line(x1, y1, x2, y2)
```

```
# Plot the line
plt.figure(figsize=(8, 8))
plt.plot(x_coords, y_coords, 'bo-')
plt.plot(x1, y1, 'go')  # starting point
plt.plot(x2, y2, 'ro')  # ending point
plt.title("DDA Line Drawing")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(True)
plt.show()
```

b. Implement Bresenhams algorithm

```
def bresenham_line(x1, y1, x2, y2):
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x2 > x1 else -1
    sy = 1 if y2 > y1 else -1
    err = dx - dy

    points = []
    while True:
        points.append((x1, y1))
        if x1 == x2 and y1 == y2:
            break
        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x1 += sx
        if e2 < dx:
            err += dx
            y1 += sy

    return points

# Draw the line
```

```
x1, y1 = 10, 10
x2, y2 = 20, 20
line_points = bresenham_line(x1, y1, x2, y2)

# Print the points
for point in line_points:
    print(point)

# Plot the line
import matplotlib.pyplot as plt
x_coords = [point[0] for point in line_points]
y_coords = [point[1] for point in line_points]
plt.plot(x_coords, y_coords, 'bo-')
plt.plot(x1, y1, 'go')  # starting point
plt.plot(x2, y2, 'ro')  # ending point
plt.show()
```

c. Draw a circle using midpoint algorithm

```
import matplotlib.pyplot as plt

def midpoint_circle(x0, y0, r):
    x = 0
    y = r
    d = 1 - r

    points = []
    while x <= y:
        points.append((x0 + x, y0 + y))
        points.append((x0 - x, y0 + y))
        points.append((x0 + x, y0 - y))
        points.append((x0 - x, y0 - y))
        points.append((x0 + y, y0 + x))
        points.append((x0 - y, y0 + x))
        points.append((x0 + y, y0 - x))
        points.append((x0 - y, y0 - x))
        if d < 0:
            d += 2 * x + 3
        else:
            d += 2 * (x - y) + 5
            y -= 1
        x += 1

    return points

# Draw the circle
x0, y0 = 20, 20
r = 10
circle_points = midpoint_circle(x0, y0, r)

# Plot the circle
x_coords = [point[0] for point in circle_points]
y_coords = [point[1] for point in circle_points]
plt.plot(x_coords, y_coords, 'bo')
plt.gca().set_aspect('equal')
plt.show()
```

Practical 3 2D transformation

a. Draw a triangle and translate its right by 50 units. Show original and translated image

```
import matplotlib.pyplot as plt

def draw_triangle(x, y):
    points = [(x, y), (x + 50, y + 100), (x + 100, y)]
    return points

def translate(points, tx, ty):
    translated_points = [(x + tx, y + ty) for x, y in points]
    return translated_points

def main():
    original_points = draw_triangle(50, 50)

    translated_points = translate(original_points, 50, 0)

    original_x = [point[0] for point in original_points]
    original_y = [point[1] for point in original_points]
    original_x.append(original_points[0][0])
    original_y.append(original_points[0][1])

    translated_x = [point[0] for point in translated_points]
    translated_y = [point[1] for point in translated_points]
    translated_x.append(translated_points[0][0])
    translated_y.append(translated_points[0][1])

    plt.plot(original_x, original_y, 'bo-')
    plt.plot(translated_x, translated_y, 'ro-')
    plt.gca().set_aspect('equal')
    plt.show()

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

Practical 6 Interactive Graphics with python

a. Move a rectangle using arrow keys

```
import pygame
import sys

# Initialize Pygame
pygame.init()

# Screen setup
width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Move Rectangle with Arrow Keys")

# Colors
WHITE = (255, 255, 255)
BLUE = (0, 0, 255)

# Rectangle properties
rect_x, rect_y = width // 2, height // 2
rect_width, rect_height = 100, 50
speed = 5  # Movement speed

# Main loop
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    # Key press detection
    keys = pygame.key.get_pressed()
    if keys[pygame.K_UP]:
        rect_y -= speed
    if keys[pygame.K_DOWN]:
        rect_y += speed
    if keys[pygame.K_LEFT]:
        rect_x -= speed
    if keys[pygame.K_RIGHT]:
        rect_x += speed

    # Drawing
    screen.fill(WHITE)
    pygame.draw.rect(screen, BLUE, (rect_x, rect_y, rect_width,
rect_height))
    pygame.display.flip()
```

b. Change the color of shape on key press

```
import pygame
import sys

pygame.init()

width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Move Rectangle and Change Color")

WHITE = (255, 255, 255)
BLUE = (0, 0, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
YELLOW = (255, 255, 0)

rect_x, rect_y = width // 2, height // 2
rect_width, rect_height = 100, 50
speed = 5

rect_color = BLUE  # Initial color
```

```
clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            # Change color only on KEYDOWN event
            if event.key == pygame.K_r:
                rect_color = RED
            elif event.key == pygame.K_g:
                rect_color = GREEN
            elif event.key == pygame.K_b:
                rect_color = BLUE
            elif event.key == pygame.K_y:
                rect_color = YELLOW

    keys = pygame.key.get_pressed()
    if keys[pygame.K_UP]:
        rect_y -= speed
    if keys[pygame.K_DOWN]:
        rect_y += speed
    if keys[pygame.K_LEFT]:
        rect_x -= speed
    if keys[pygame.K_RIGHT]:
        rect_x += speed

    screen.fill(WHITE)
    pygame.draw.rect(screen, rect_color, (rect_x, rect_y, rect_width,
rect_height))
    pygame.display.flip()
    clock.tick(60)
```

Practical 7 Basic Animation and object movement

a. Animate a ball moving horizontally across the screen.

```
import pygame
import sys

pygame.init()
pygame.mixer.init()

width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Horizontal Ball Animation")

WHITE = (255, 255, 255)
RED = (255, 0, 0)
movie_sound=pygame.mixer.Sound('C:/Users/HP/Downloads/file_example_WAV_1MG.wav')
ball_radius = 30
x = ball_radius  # Start at left edge
y = height // 2  # Vertical center
speed = 5

clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    x += speed
    if x - ball_radius > width:
        x = -ball_radius  # Reset to left beyond the window for smooth loop

    screen.fill(WHITE)
    pygame.draw.circle(screen, RED, (x, y), ball_radius)
    pygame.display.flip()
    clock.tick(60)
```

b. Animate multiple balls moving independently with different speeds

```
import pygame
import sys
import random

pygame.init()

width, height = 800, 600
screen = pygame.display.set_mode((width, height))
pygame.display.set_caption("Multiple Balls Animation")

WHITE = (255, 255, 255)
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 165, 0), (128, 0, 128)]

ball_radius = 20

# Create multiple balls with random starting x, fixed y, random speeds, and colors
balls = []
for i in range(5):
    x = random.randint(ball_radius, width - ball_radius)
    y = (i + 1) * (height // 6)
    speed = random.randint(2, 8)
    color = colors[i % len(colors)]
    balls.append({'x': x, 'y': y, 'speed': speed, 'color': color})

clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    screen.fill(WHITE)

    # Update and draw each ball independently
    for ball in balls:
        ball['x'] += ball['speed']
        if ball['x'] - ball_radius > width:
            ball['x'] = -ball_radius  # Loop back to left
        pygame.draw.circle(screen, ball['color'], (int(ball['x']),
ball['y']), ball_radius)

    pygame.display.flip()
    clock.tick(60)
```

Practical 9 Color Models and Curve Animation

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Create the figure and axis
fig, ax = plt.subplots(figsize=(7, 5))
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1.5, 1.5)

# Initial curve data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# Initial line object
```

```
(line,) = ax.plot(x, y, lw=2)

# Function to update curve and color for each frame
def animate(frame):
    y = np.sin(x + frame * 0.05)             # Curve animates
horizontally
    # Animate color: map frame number to RGB color
    r = (np.sin(frame * 0.03) + 1) / 2
    g = (np.sin(frame * 0.05 + 2) + 1) / 2
    b = (np.sin(frame * 0.07 + 4) + 1) / 2
    color = (r, g, b)
    line.set_ydata(y)
    line.set_color(color)
    return (line,)

# Create animation
anim = FuncAnimation(fig, animate, frames=100, interval=50,
blit=True)
plt.title("Curve Animation with Color Model (RGB)")
plt.show()
```