Brindavan College of Engineering

BRINDAVAN GROUP OF INSTITUTIONS

Dwarakanagar, Bagalur Main Road, Yelahanka, Bengaluru – 560063

Affiliated to VTU Belagavi, Approved by AICTE, New Delhi, India
Accredited 'B++' level by NAAC

www.brindavancollege.com

VI Semester

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY 21CSL66

ACADEMIC YEAR
2023 – 24

LABORATORY MANUAL

PREPARED BY

Prof. DARSHAN M PATEL

BRINDAVAN GROUP OF INSTITUTIONS

Brindavan College of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VI Semester

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY

21CSL66

ACADEMIC YEAR
2023 – 24

NAME OF THE STUDENT	:	--
UNIVERSITY SEAT NO.	:	--
BATCH	:	--

PREPARED BY

Prof. DARSHAN M PATEL

Brindavan College of Engineering

Department of Computer Science & Engineering

LABORATORY CERTIFICATE

This is to certify that Mr. /Ms.____ bearing USN___________________ of ___________semester and _________section has satisfactorily completed the course of experiments in ***computer graphics and image processing*** *code* ***21CSL66*** *prescribed by the Visvesvaraya Technological University, Belagavi of this Institute for the academic year 2023– 24 .*

MARKS	
Maximum Marks	**Marks Obtained**

Signature of Faculty-In-Charge *Head of the Department*

Brindavan College of Engineering

Dwarakanagar, Bagalur Main Road, Yelahanka, Bengaluru – 560063

Affiliated to VTU Belagavi, Approved by AICTE, New Delhi, India

Accredited B++ level by NAAC

Department of Computer Science & Engineering

DEPARTMENT VISION

"To foster the students into globally competent professionals, ingenious entrepreneurs with ethical values"

DEPARTMENT MISSION

- To produce technologically competent professionals through balanced and dynamic curriculum.
- To facilitate the students to explore creativity, innovations and develop leadership qualities.
- To inculcate the spirit of ethical values contributing to welfare of society.

Department of Computer Science & Engineering

DOs & DON'Ts IN LABORATORY

DOs

- Be on time and students should carry observation and completed records in all aspects.
- Dress code & wearing ID card is compulsory.
- Electronic gadgets are not allowed inside the lab.
- Students should be at their concerned desktop.
- After execution the students should get it verified by the concerned faculty.
- The executed results should be noted in their observations and get it verified by the concerned faculty.
- Observe good housekeeping practices. Keep the equipment's in proper place after the conduction.
- Students must ensure that all the switches are in the OFF position; desktop is shutdown properly after completion of the assignments.
- For circuits lab the components must returned properly.
- For power electronics lab wearing shoes is compulsory.

DON'Ts

- Do not come late to lab.
- Do not touch server computer.
- Do not leave the lab without the permission of the faculty in-charge.
- Do not wear footwear and enter the lab (except power electronics lab).
- Do not insert pen drive/memory card to any computer in the lab.
- Do not upload, delete or alter any software files.

PREFACE

The Computer Graphics and Imaging (CGI) lab introduces the foundational principles and techniques of computer graphics and image processing. It emphasizes both the theoretical aspects and practical applications in creating and manipulating visual content. Topics covered include the mathematics of transformations, lighting and shading models, rendering techniques, and geometric modeling. Students will explore algorithms for rasterization, texture mapping, and 3D modeling, as well as fundamental image processing techniques such as filtering, edge detection, and color manipulation. The lab also delves into the use of graphics APIs, such as OpenGL or DirectX, for implementing these concepts. The aim is to equip students with a robust understanding and skillset to create and analyze graphical and image-based solutions, preparing them for real-world applications in fields like video games, simulations, and digital media.

Prof. DARSHAN M PATEL

ANALYSIS & DESIGN OF ALGORITHMS LAB

Course Code: 21CSL66 **CIE Marks: 50**

Number of Lecture Hours/Week: 02 Hours

SEE Marks: 50 **Exam Hours: 03**

CREDITS: 01 **RBT Levels L1, L2, L3**

Course Objectives

1. Demonstrate the use of Open GL.
2. Demonstrate the different geometric object drawing using openGL
3. Demonstration of 2D/3D transformation on simple objects.
4. Demonstration of lighting effects on the created objects.
5. Demonstration of Image processing operations on images.

Laboratory Experiments

1. Develop a program to draw a line using Bresenham's line drawing technique
2. Develop a program to demonstrate basic geometric operations on the 2D object
3. Develop a program to demonstrate basic geometric operations on the 3D object
4. Develop a program to demonstrate 2D transformation on basic objects
5. Develop a program to demonstrate 3D transformation on 3D objects

6. Develop a program to demonstrate Animation effects on simple objects.

7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

8. Write a program to show rotation, scaling, and translation on an image.

9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.

10. Write a program to blur and smoothing an image.

11. Write a program to contour an image.

12. Write a program to detect a face/s in an image.

Course Outcomes:

1. Use OpenGL /OpenCV for the development of mini-Projects. Analyse the necessity mathematics and design required to demonstrate basic geometric transformation techniques.
2. Demonstrate the ability to design and develop input interactive techniques.
3. Apply the concepts to Develop user friendly applications using Graphics and IP concepts

Program-01

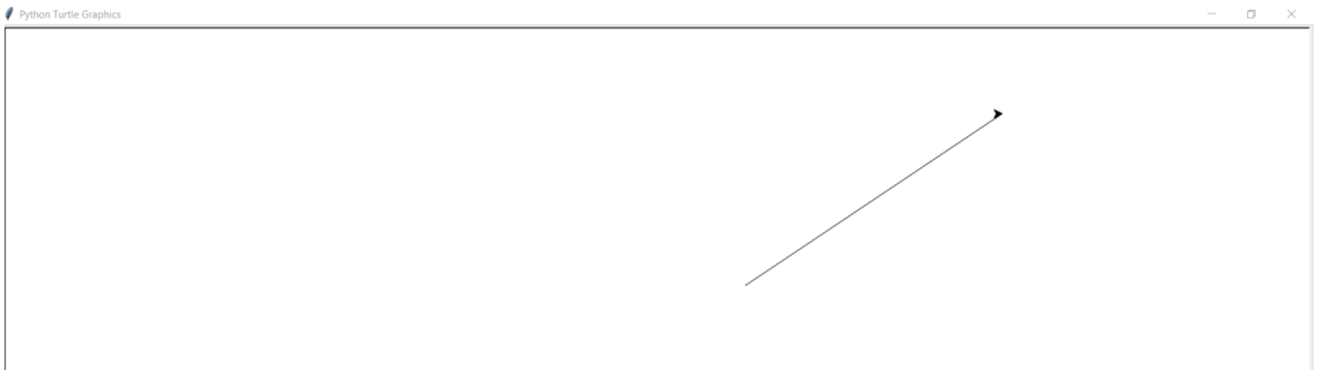
1.Develop a program to draw a line using Bresenham’s line drawing technique

```
import turtle

def bresenham_line(x1, y1, x2, y2):
    # Calculate the deltas
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    # Determine the step direction for each axis
    x_step = 1 if x1 < x2 else -1
    y_step = 1 if y1 < y2 else -1
```

```
# Initialize the error term
error = dx - dy
# Initialize the line points
line_points = []
# Start at the first point
x, y = x1, y1

while True:
    # Add the current point to the line
    line_points.append((x, y))


    # Break the loop if the end point is reached
    if x == x2 and y == y2:
        break
    # Store the error term
    e2 = 2 * error
    # Update the error term and adjust the coordinates
    if e2 > -dy:
        error -= dy
        x += x_step
    if e2 < dx:
        error += dx
        y += y_step

return line_points
```

```
# Example usage
turtle.setup(500, 500)
turtle.speed(0)  # Fastest drawing speed
x1, y1 = 100, 100
x2, y2 = 400, 300
line_points = bresenham_line(x1, y1, x2, y2)

# Draw the line
turtle.penup()
turtle.goto(x1, y1)
turtle.pendown()
for x, y in line_points:
    turtle.goto(x, y)

turtle.exitonclick()
```

OUTPUT:



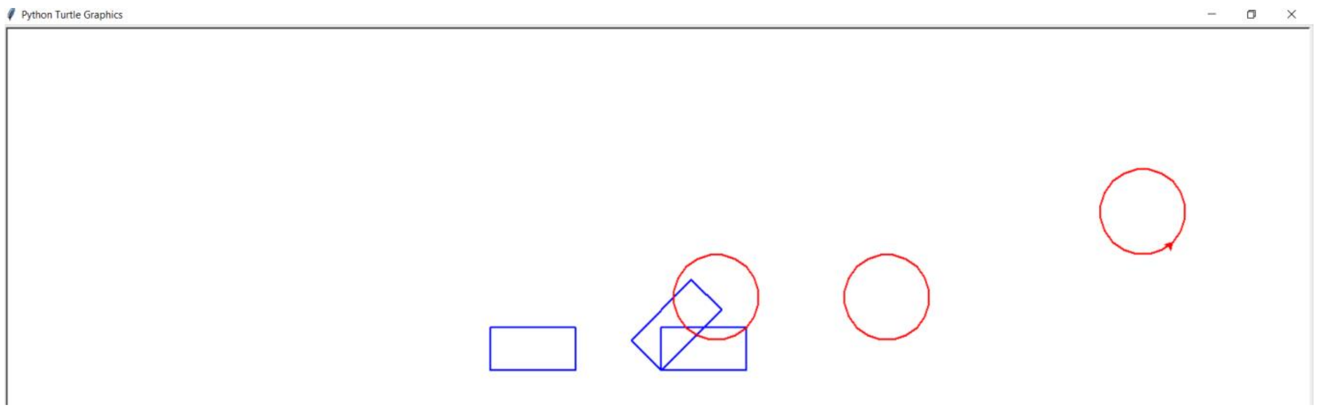
2. Develop a program to demonstrate basic geometric operations on the 2D object

```
import turtle
import math

# Set up the turtle screen
screen = turtle.Screen()
screen.bgcolor("white")

# Create a turtle instance
t = turtle.Turtle()
t.speed(1) # Set the drawing speed (1 is slowest, 10 is fastest)
t.pensize(2) # Set the pen size

# Define a function to draw a rectangle
def draw_rectangle(x, y, width, height, color):
    t.penup()
    t.goto(x, y)
    t.pendown()
    t.color(color)
    for _ in range(2):
        t.forward(width)
        t.left(90)
        t.forward(height)
        t.left(90)

# Define a function to draw a circle
def draw_circle(x, y, radius, color):
    t.penup()
    t.goto(x, y - radius)
    t.pendown()
    t.color(color)
    t.circle(radius)

# Define a function to translate a 2D object
def translate(x, y, dx, dy):
    t.penup()
    t.goto(x + dx, y + dy)
    t.pendown()

# Define a function to rotate a 2D object
def rotate(x, y, angle):
    t.penup()
    t.goto(x, y)
    t.setheading(angle)
    t.pendown()

# Define a function to scale a 2D object
def scale(x, y, sx, sy):
    t.penup()
    t.goto(x * sx, y * sy)
    t.pendown()

# Draw a rectangle
draw_rectangle(-200, 0, 100, 50, "blue")
```

```
# Translate the rectangle
translate(-200, 0, 200, 0)
draw_rectangle(0, 0, 100, 50, "blue")

# Rotate the rectangle
rotate(0, 0, 45)
draw_rectangle(0, 0, 100, 50, "blue")

# Scale the rectangle
scale(0, 0, 2, 2)
draw_rectangle(0, 0, 100, 50, "blue")

# Draw a circle
draw_circle(100, 100, 50, "red")

# Translate the circle
translate(100, 100, 200, 0)
draw_circle(300, 100, 50, "red")

# Rotate the circle
rotate(300, 100, 45)
draw_circle(300, 100, 50, "red")

# Scale the circle
scale(300, 100, 2, 2)
draw_circle(600, 200, 50, "red")
```

```
# Keep the window open until it's closed
turtle.done()
```

OUTPUT:

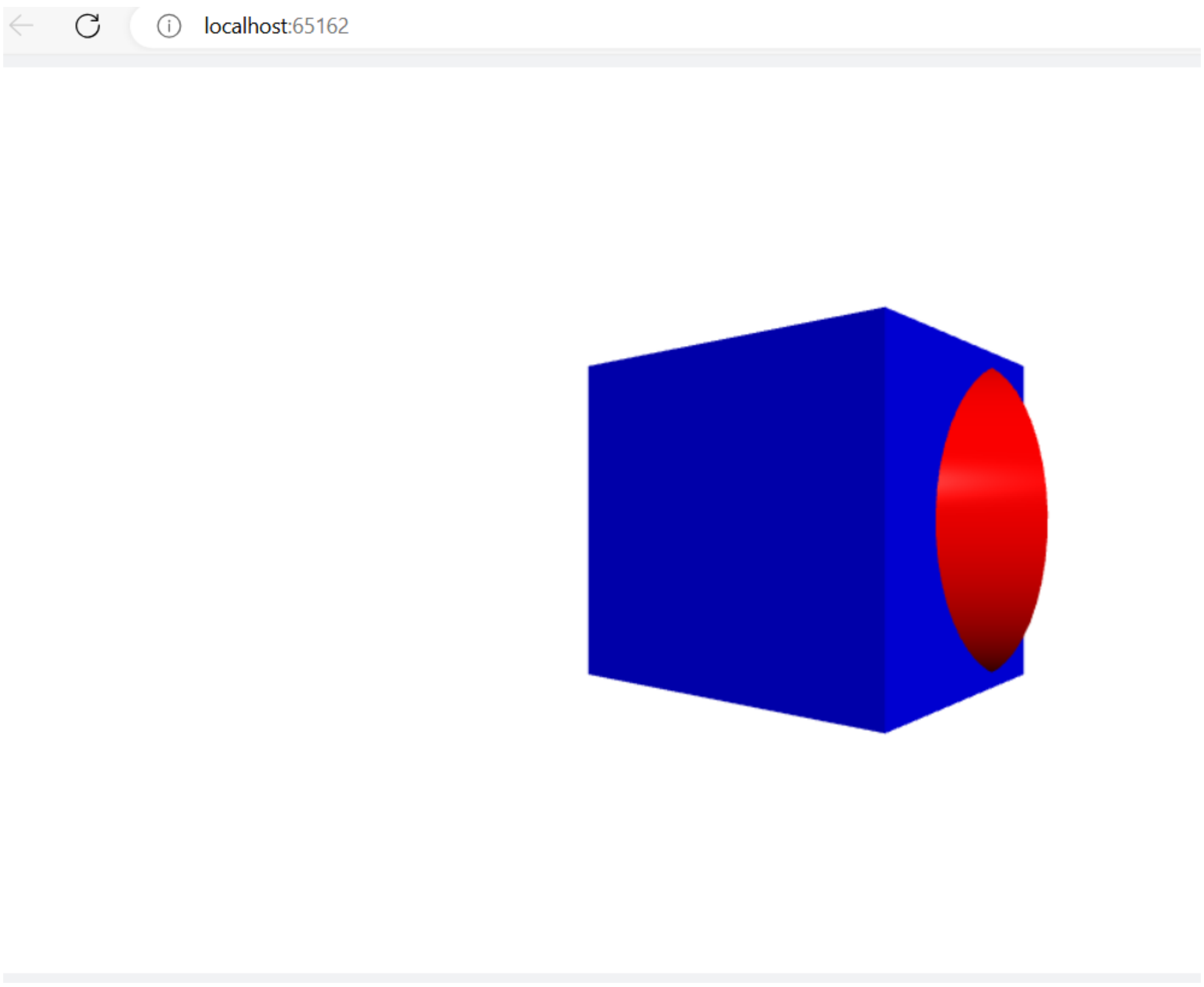


3. Develop a program to demonstrate basic geometric operations on the 3D object

```
from vpython import canvas, box, cylinder, vector, color, rate
import math

# Create a 3D canvas
scene = canvas(width=800, height=600, background=color.white)

# Define a function to draw a cuboid
def draw_cuboid(pos, length, width, height, color):
    cuboid = box(pos=vector(*pos), length=length, width=width, height=height, color=color)
    return cuboid

# Define a function to draw a cylinder
def draw_cylinder(pos, radius, height, color):
    cyl = cylinder(pos=vector(*pos), radius=radius, height=height, color=color)
    return cyl

# Define a function to translate a 3D object
def translate(obj, dx, dy, dz):
    obj.pos += vector(dx, dy, dz)

# Define a function to rotate a 3D object
def rotate(obj, angle, axis):
    obj.rotate(angle=angle, axis=vector(*axis))

# Define a function to scale a 3D object
def scale(obj, sx, sy, sz):
    obj.size = vector(obj.size.x * sx, obj.size.y * sy, obj.size.z * sz)

# Draw a cuboid
cuboid = draw_cuboid((-2, 0, 0), 2, 2, 2, color.blue)

# Translate the cuboid
translate(cuboid, 4, 0, 0)

# Rotate the cuboid
rotate(cuboid, angle=math.radians(45), axis=(0, 1, 0))

# Scale the cuboid
scale(cuboid, 1.5, 1.5, 1.5)

# Draw a cylinder
```

```
cyl = draw_cylinder((2, 2, 0), 1, 10, color.red)

# Translate the cylinder
translate(cyl, 0, -2, 0)

# Rotate the cylinder
rotate(cyl, angle=math.radians(30), axis=(1, 0, 0))

# Scale the cylinder
scale(cyl, 1.5, 1.5, 1.5)

# Keep the 3D scene interactive
while True:
    rate(30) # Set the frame rate to 30 frames per second
```

OUTPUT:

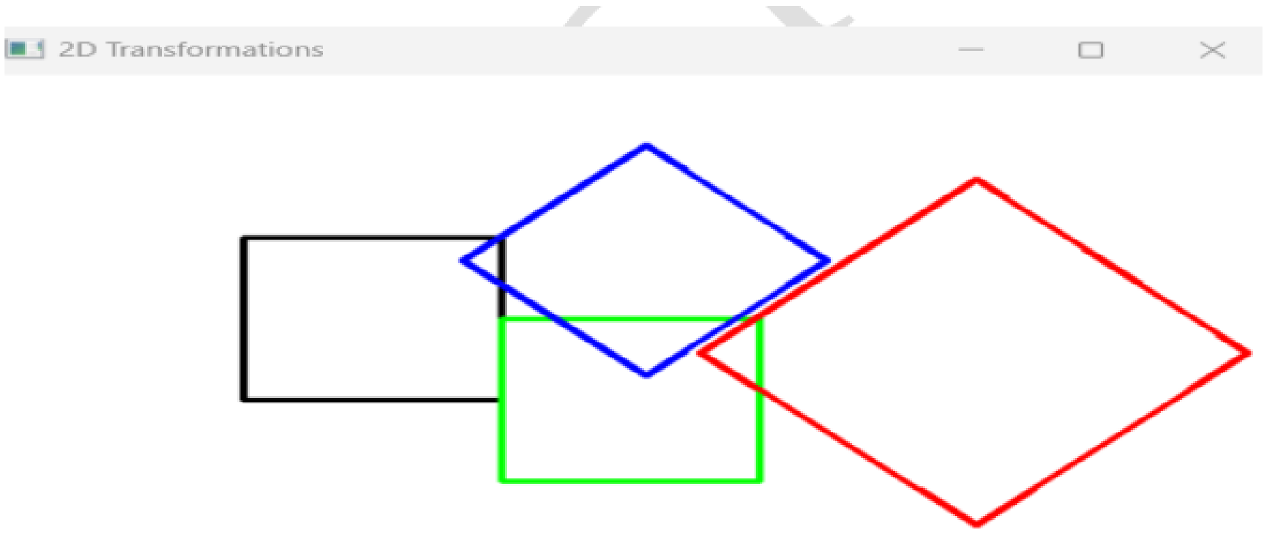


4. Develop a program to demonstrate 2D transformation on basic objects

```
import cv2
import numpy as np
# Define the dimensions of the canvas
canvas_width = 500
canvas_height = 500
# Create a blank canvas
canvas = np.ones((canvas_height, canvas_width, 3), dtype=np.uint8) * 255
# Define the initial object (a square)
obj_points = np.array([[100, 100], [200, 100], [200, 200], [100, 200]], dtype=np.int32)
```

```
# Define the transformation matrices
translation_matrix = np.float32([[1, 0, 100], [0, 1, 50]])
rotation_matrix = cv2.getRotationMatrix2D((150, 150), 45, 1)
scaling_matrix = np.float32([[1.5, 0, 0], [0, 1.5, 0]])
# Apply transformations
translated_obj = np.array([np.dot(translation_matrix, [x, y, 1])[:2] for x, y in
obj_points],
dtype=np.int32)
rotated_obj = np.array([np.dot(rotation_matrix, [x, y, 1])[:2] for x, y in
translated_obj],
dtype=np.int32)
scaled_obj = np.array([np.dot(scaling_matrix, [x, y, 1])[:2] for x, y in
rotated_obj],
dtype=np.int32)
# Draw the objects on the canvas
cv2.polylines(canvas, [obj_points], True, (0, 0, 0), 2)
cv2.polylines(canvas, [translated_obj], True, (0, 255, 0), 2)
cv2.polylines(canvas, [rotated_obj], True, (255, 0, 0), 2)
cv2.polylines(canvas, [scaled_obj], True, (0, 0, 255), 2)
# Display the canvas
cv2.imshow("2D Transformations", canvas)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



5. Develop a program to demonstrate 3D transformation on 3D objects

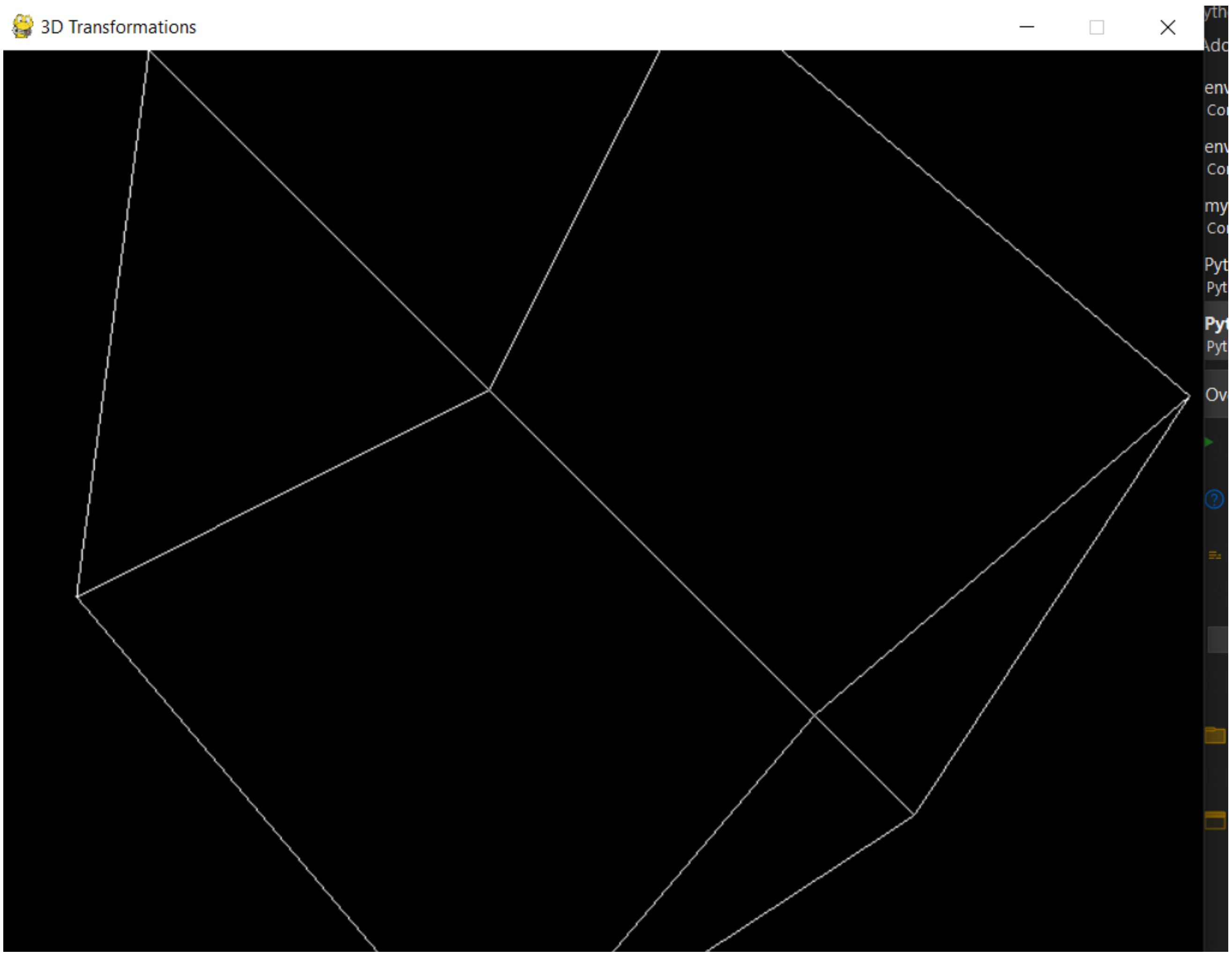
```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
import numpy as np
# Initialize Pygame
pygame.init()
# Set up the display
display_width = 800
display_height = 600
display = pygame.display.set_mode((display_width, display_height),
DOUBLEBUF |
OPENGL)
pygame.display.set_caption("3D Transformations")
```

```
# Set up OpenGL
glClearColor(0.0, 0.0, 0.0, 1.0)
glEnable(GL_DEPTH_TEST)
glMatrixMode(GL_PROJECTION)
gluPerspective(45, (display_width / display_height), 0.1, 50.0)
glMatrixMode(GL_MODELVIEW)
# Define the 3D object (a cube)
vertices = np.array([
    [-1, -1, -1],
[1, -1, -1],
[1, 1, -1],
[-1, 1, -1],
[-1, -1, 1],
[1, -1, 1],
[1, 1, 1],
[-1, 1, 1]
], dtype=np.float32)
edges = np.array([
[0, 1], [1, 2], [2, 3], [3, 0],
[4, 5], [5, 6], [6, 7], [7, 4],
[0, 4], [1, 5], [2, 6], [3, 7]
], dtype=np.uint32)
# Set up the transformation matrices
translation_matrix = np.eye(4, dtype=np.float32)
translation_matrix[3, :3] = [0, 0, -5]
```

```
rotation_matrix = np.eye(4, dtype=np.float32)
scaling_matrix = np.eye(4, dtype=np.float32)
scaling_matrix[0, 0] = 1.5
scaling_matrix[1, 1] = 1.5
scaling_matrix[2, 2] = 1.5
# Main loop
running = True
angle = 0
while running:
for event in pygame.event.get():
  if event.type == pygame.QUIT:
   running = False
# Clear the display
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
# Apply transformations
glLoadIdentity()
glMultMatrixf(translation_matrix)
glRotatef(angle, 1, 1, 0)
glMultMatrixf(rotation_matrix)
glMultMatrixf(scaling_matrix)
# Draw the 3D object
glBegin(GL_LINES)
for edge in edges:
   for vertex in edge:
    glVertex3fv(vertices[vertex])
```

```
glEnd()
# Update the rotation angle
angle += 1
# Swap the front and back buffers
pygame.display.flip()
# Quit Pygame
pygame.quit()
```

OUTPUT:



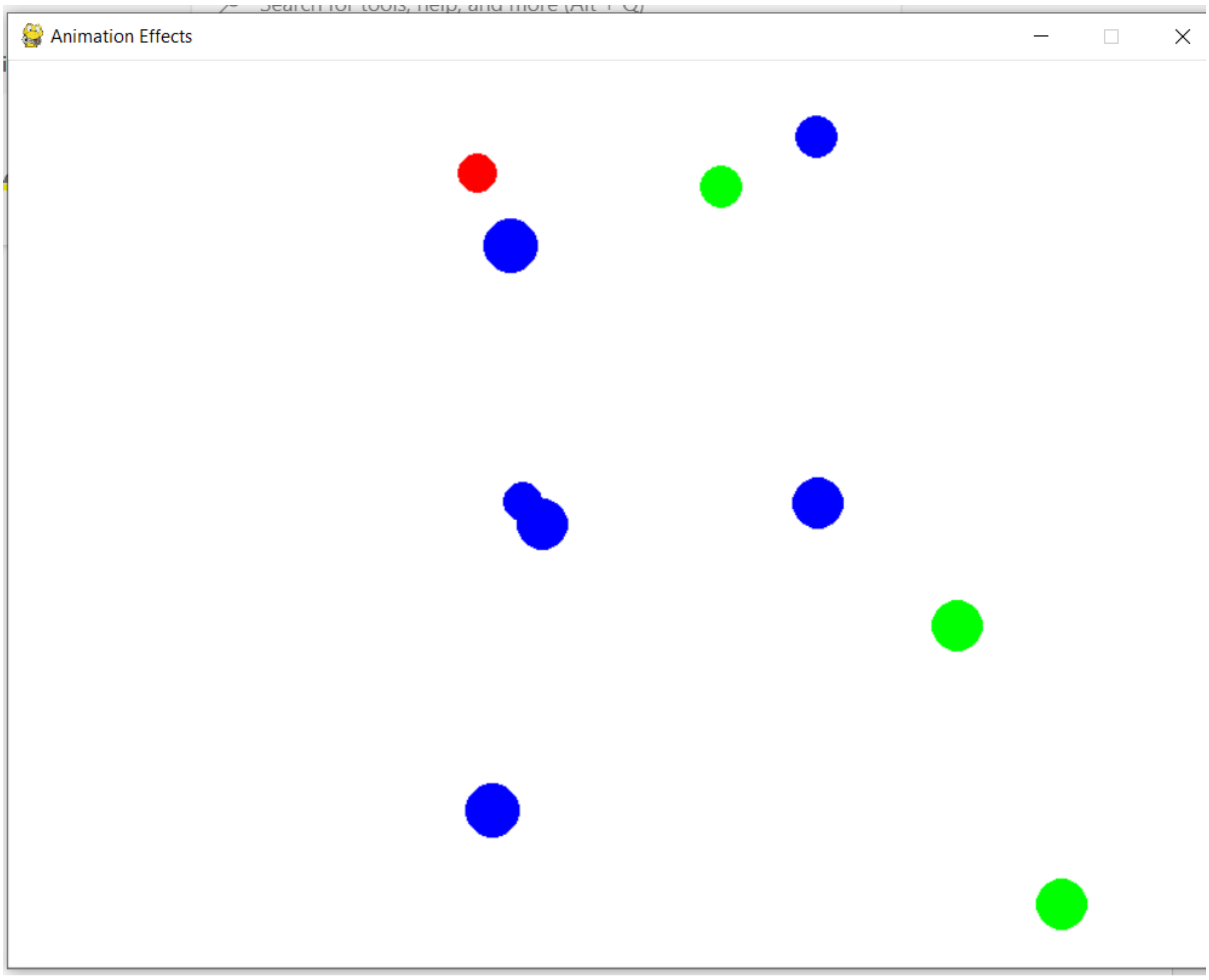
6. Develop a program to demonstrate Animation effects on simple objects.

```
import pygame
import random

# Initialize Pygame
pygame.init()

# Screen dimensions
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption("Animation Effects")
```

```
# Colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)

# Number of objects
num_objects = 10
colors = [RED, GREEN, BLUE]

# Object list
objects = []
for _ in range(num_objects):
    x = random.randint(50, screen_width - 50)
    y = random.randint(50, screen_height - 50)
    radius = random.randint(10, 20)  # Ensure positive radius
    speedx = random.randint(-5, 5)
    speedy = random.randint(-5, 5)
    color = random.choice(colors)
    objects.append({"x": x, "y": y, "radius": radius, "color": color, "speedx": speedx, "speedy": speedy})

# Game loop
running = True
```

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

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

    # Clear the screen
    screen.fill(WHITE)

    for obj in objects:
        # Update object position
        obj["x"] += obj["speedx"]
        obj["y"] += obj["speedy"]

        # Check for collision with screen boundaries
        if obj["x"] - obj["radius"] < 0 or obj["x"] + obj["radius"] > screen_width:
            obj["speedx"] = -obj["speedx"]
        if obj["y"] - obj["radius"] < 0 or obj["y"] + obj["radius"] > screen_height:
            obj["speedy"] = -obj["speedy"]

        # Draw the object
        pygame.draw.circle(screen, obj["color"], (obj["x"], obj["y"]), obj["radius"])
    # Update the display
    pygame.display.flip()

    # Cap the frame rate
    clock.tick(60)

# Quit Pygame
pygame.quit()
```

OUTPUT:

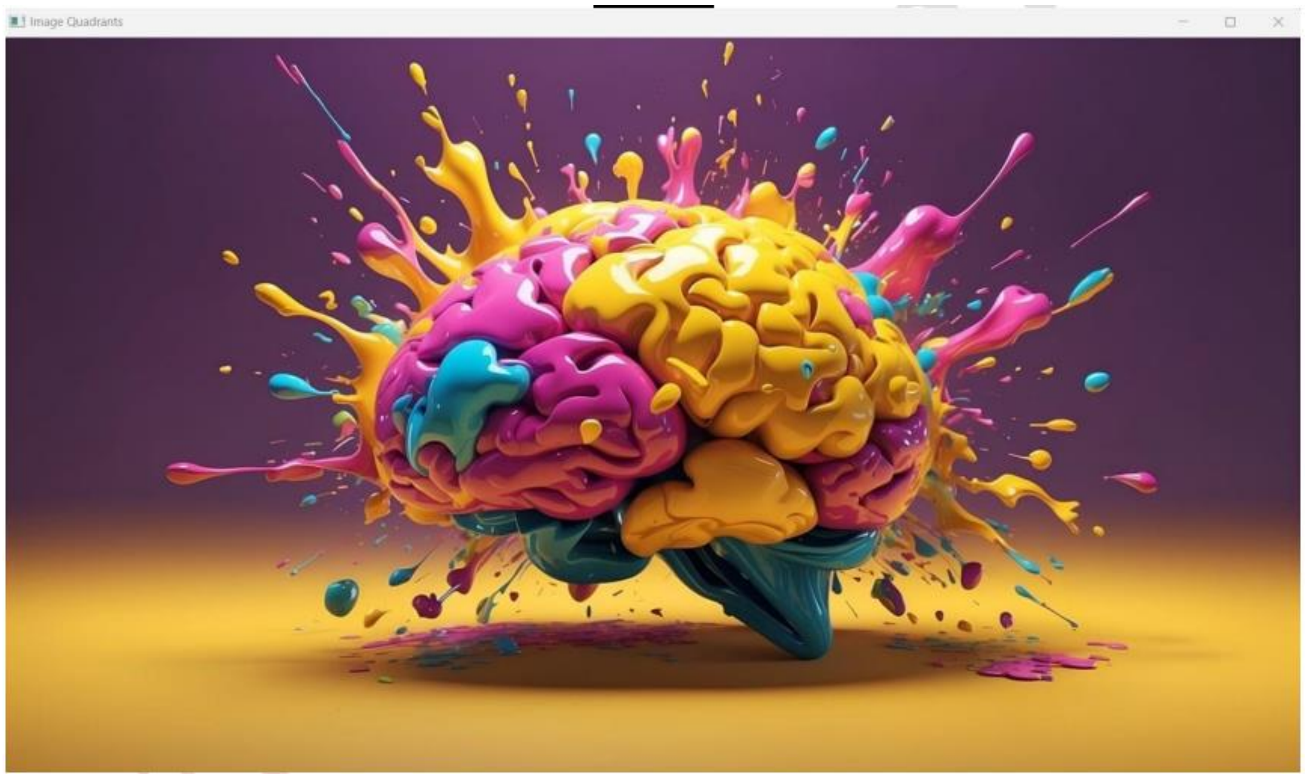


7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

```
import cv2
import numpy as np

# Load the image
image_path = "C:/Users/admin/Downloads/IMG_20221101_102849_795.jpg"  # Replace with the path to your image
img = cv2.imread(image_path)

# Check if the image was successfully loaded
if img is None:
    print("Error: Could not load image.")
    exit()

# Get the height and width of the image
height, width, _ = img.shape

# Split the image into four quadrants
up_left = img[0:height//2, 0:width//2]
up_right = img[0:height//2, width//2:width]
down_left = img[height//2:height, 0:width//2]
down_right = img[height//2:height, width//2:width]

# Create a blank canvas to display the quadrants
canvas = np.zeros((height, width, 3), dtype=np.uint8)

# Place the quadrants on the canvas
```

```
canvas[0:height//2, 0:width//2] = up_left
canvas[0:height//2, width//2:width] = up_right
canvas[height//2:height, 0:width//2] = down_left
canvas[height//2:height, width//2:width] = down_right

# Display the canvas
cv2.imshow("Image Quadrants", canvas)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



8. Write a program to show rotation, scaling, and translation on an image

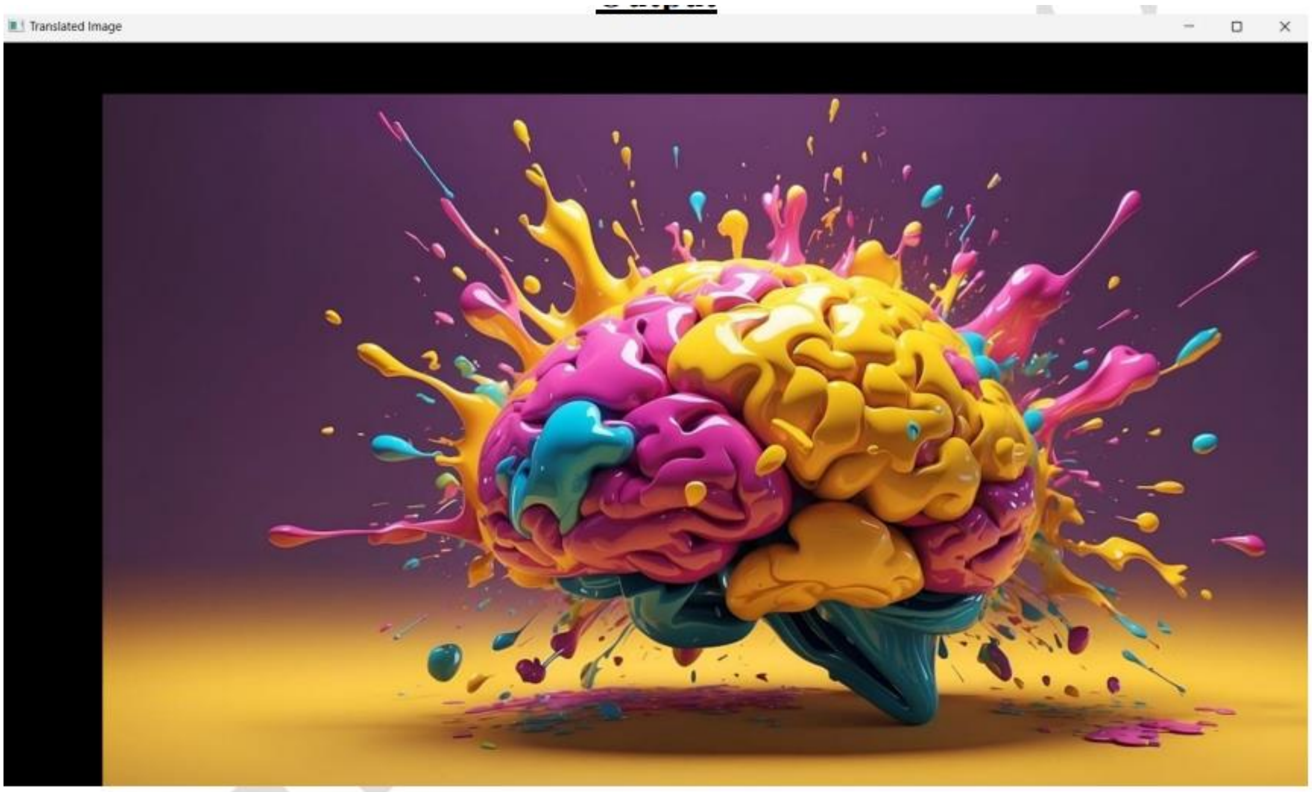
```
import cv2
import numpy as np

# Load the image
```

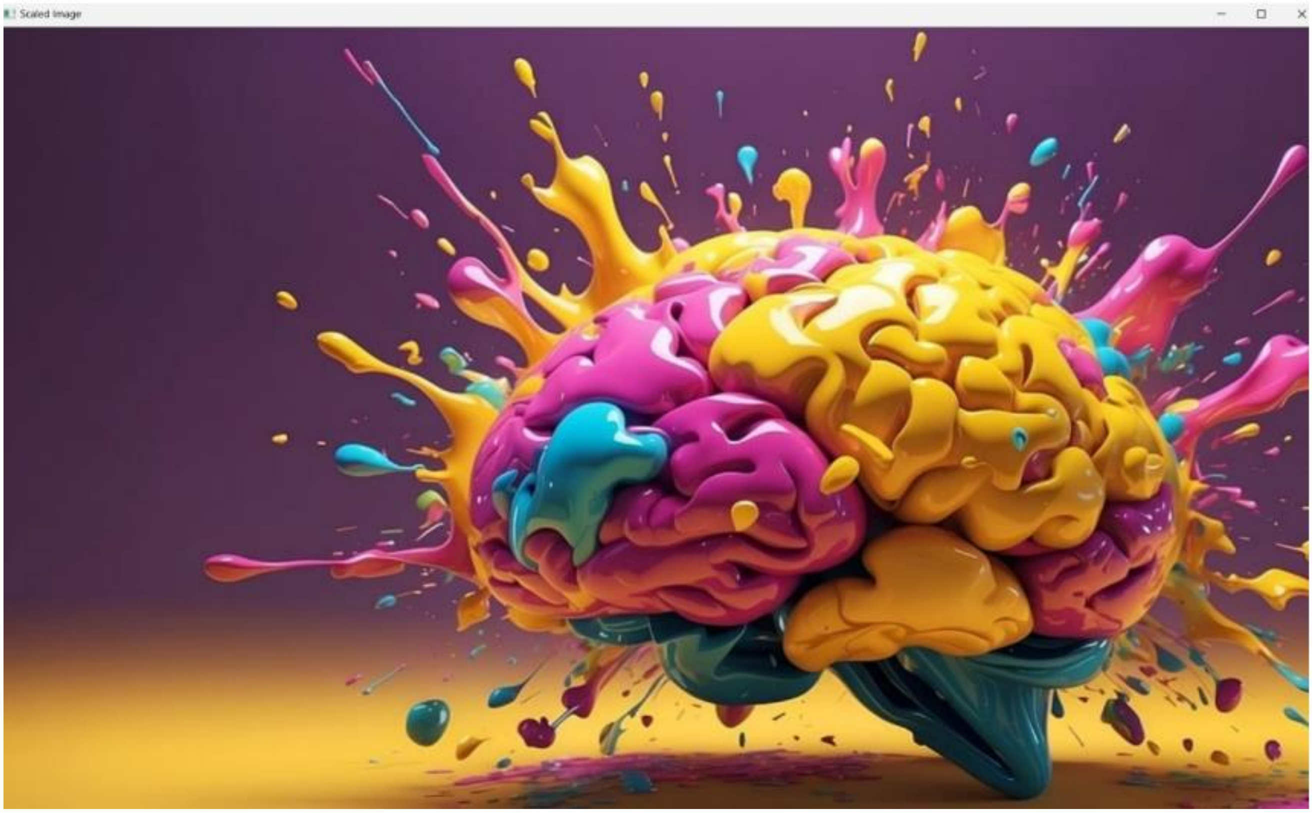
```
image_path = r"C:\Users\admin\Downloads\IMG_20221101_102849_795.jpg"  # Replace with the path to your image
img = cv2.imread(image_path)

# Check if the image was successfully loaded
if img is None:
    print("Error: Could not load image.")
    exit()

# Get the image dimensions
height, width, _ = img.shape

# Define the transformation matrices
rotation_matrix = cv2.getRotationMatrix2D((width / 2, height / 2), 45, 1)  # Rotate by 45 degrees
scaling_matrix = np.float32([[1.5, 0, 0], [0, 1.5, 0]])  # Scale by 1.5x
translation_matrix = np.float32([[1, 0, 100], [0, 1, 50]])  # Translate by (100, 50)

# Apply transformations
rotated_img = cv2.warpAffine(img, rotation_matrix, (width, height))
scaled_img = cv2.warpAffine(img, scaling_matrix, (int(width * 1.5), int(height * 1.5)))
translated_img = cv2.warpAffine(img, translation_matrix, (width, height))

# Display the original and transformed images
```

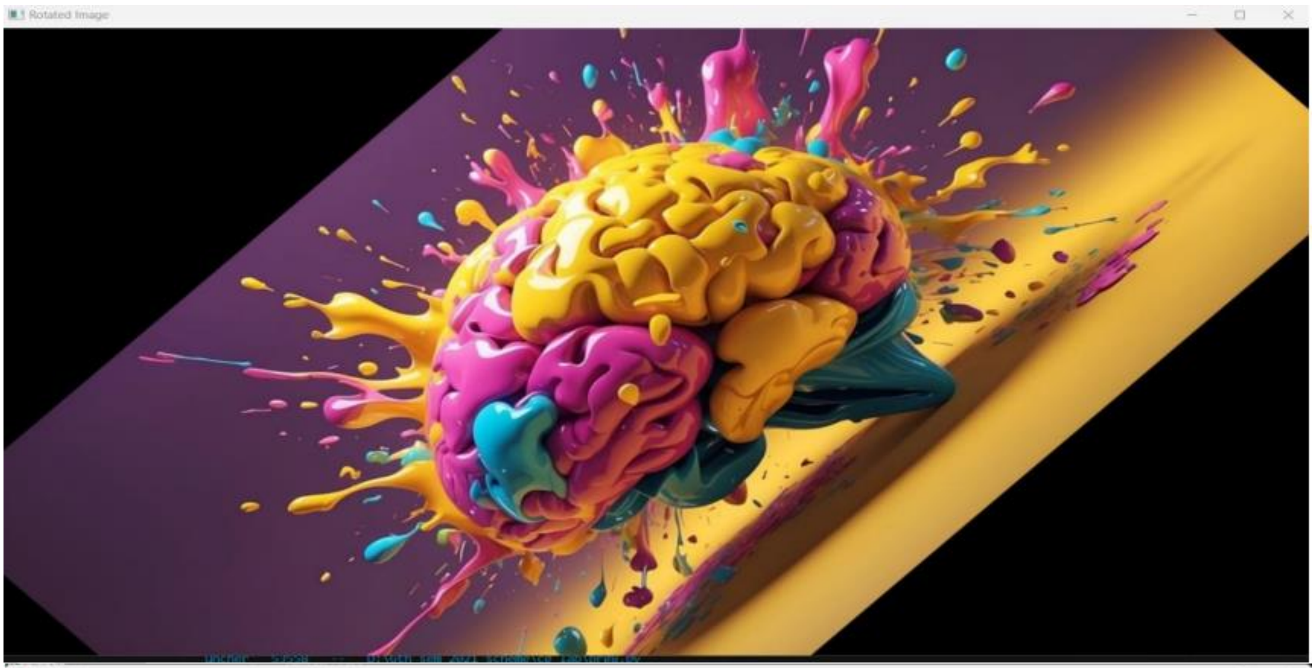
```
cv2.imshow("Original Image", img)
cv2.imshow("Rotated Image", rotated_img)
cv2.imshow("Scaled Image", scaled_img)
cv2.imshow("Translated Image", translated_img)

# Wait for a key press and then close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

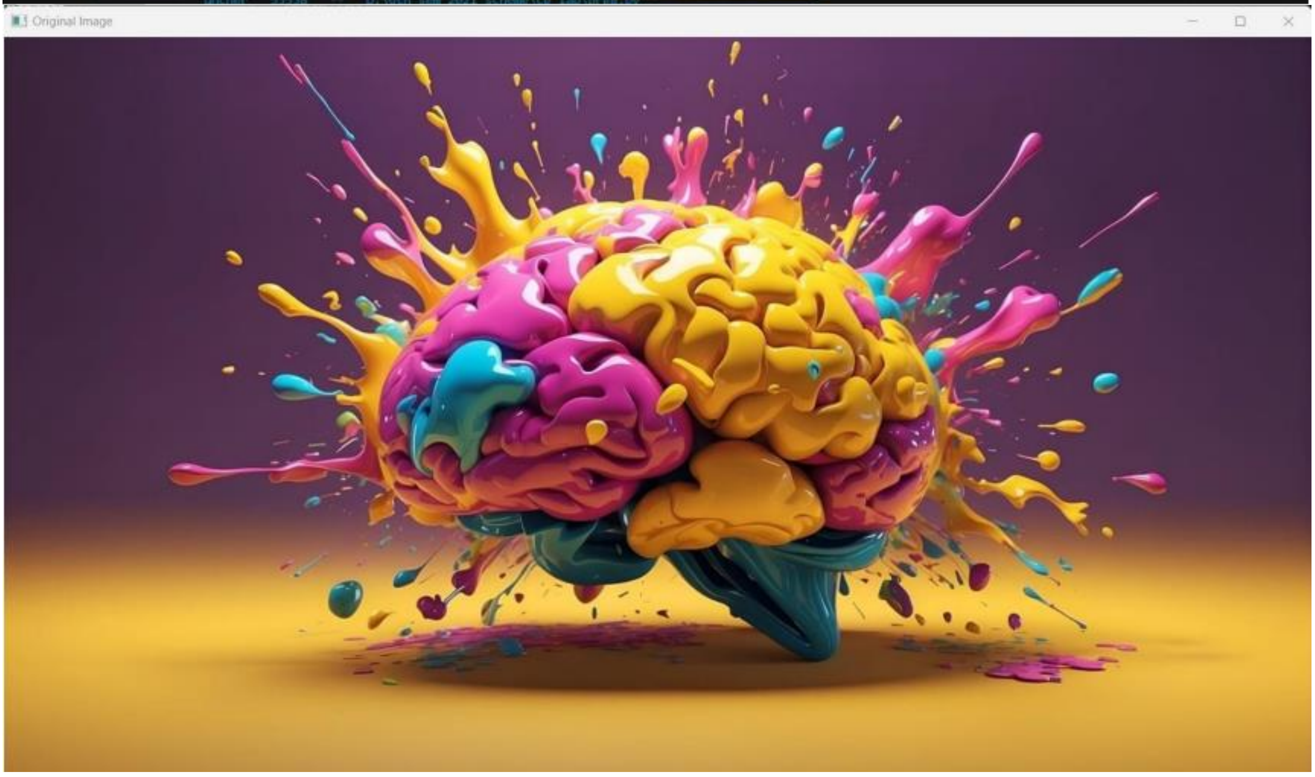
OUTPUT:

Output









9 Read an image and extract and display low-level features such as edges, textures using filtering techniques.

import cv2

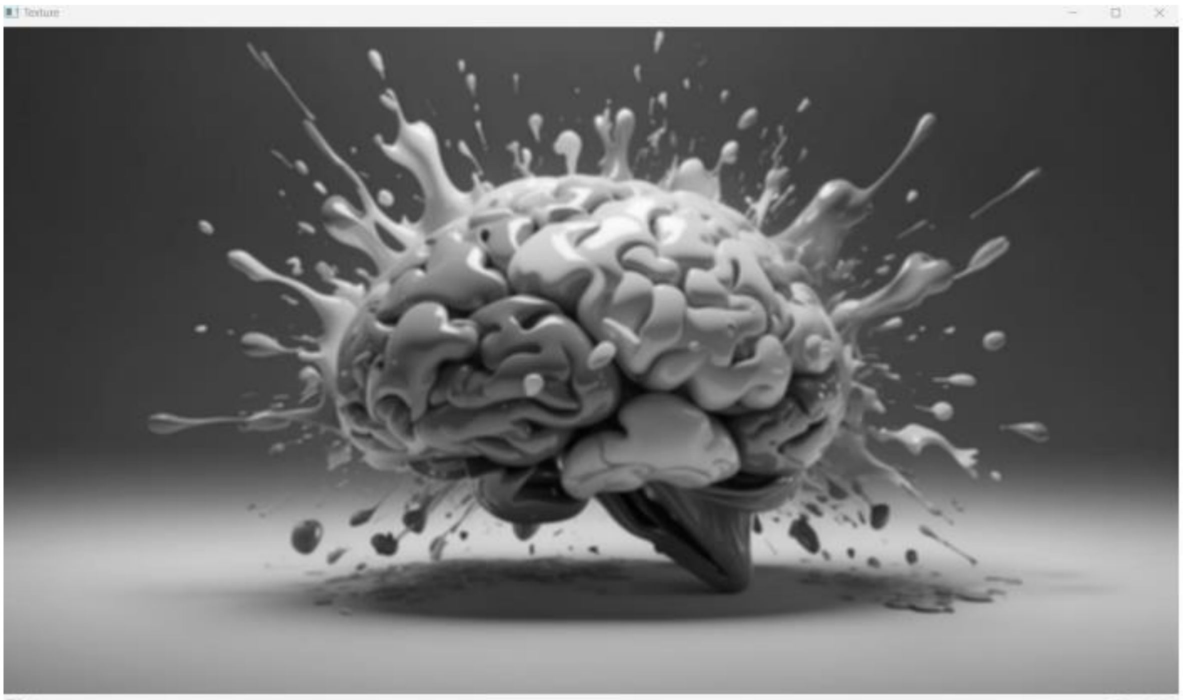
import numpy as np

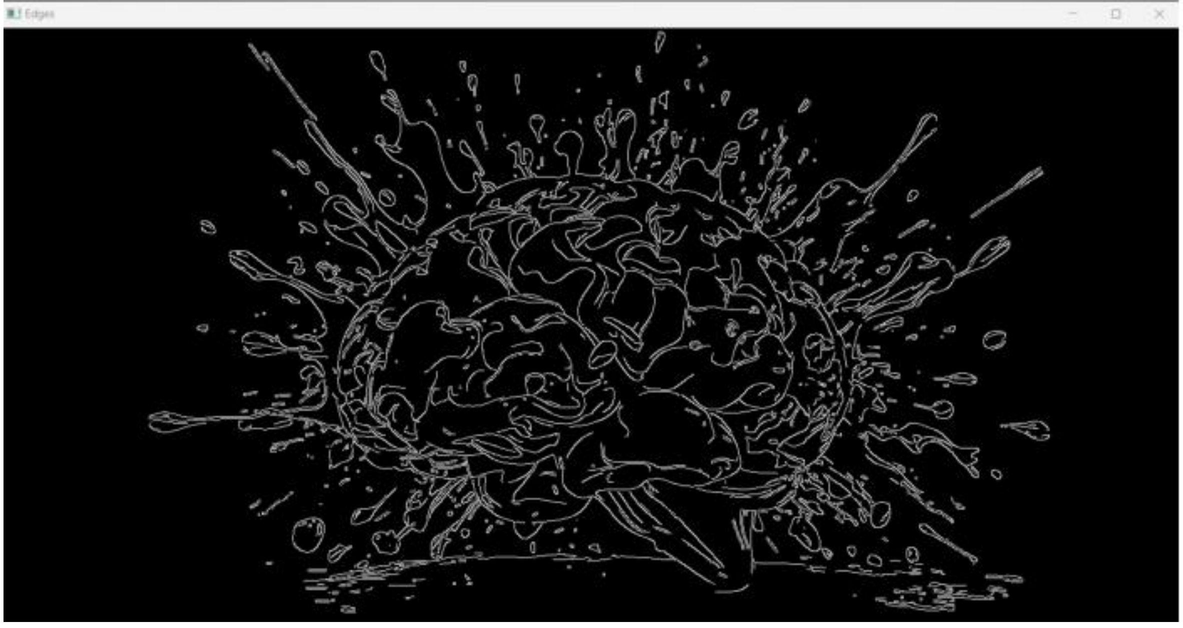
# Load the image

```
image_path = r"C:\Users\admin\Downloads\IMG_20221101_102849_795.jpg"  # Replace with the path to your image
img = cv2.imread(image_path)

# Check if the image was successfully loaded
if img is None:
    print("Error: Could not load image.")
    exit()

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Edge detection using Canny edge detector
edges = cv2.Canny(gray, 100, 200)

# Texture extraction using an averaging filter
kernel = np.ones((5, 5), np.float32) / 25  # Define a 5x5 averaging kernel
texture = cv2.filter2D(gray, -1, kernel)   # Apply the averaging filter for texture extraction

# Display the original image, edges, and texture
cv2.imshow("Original Image", img)
cv2.imshow("Edges", edges)
cv2.imshow("Texture", texture)
```

```
# Wait for a key press and then close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



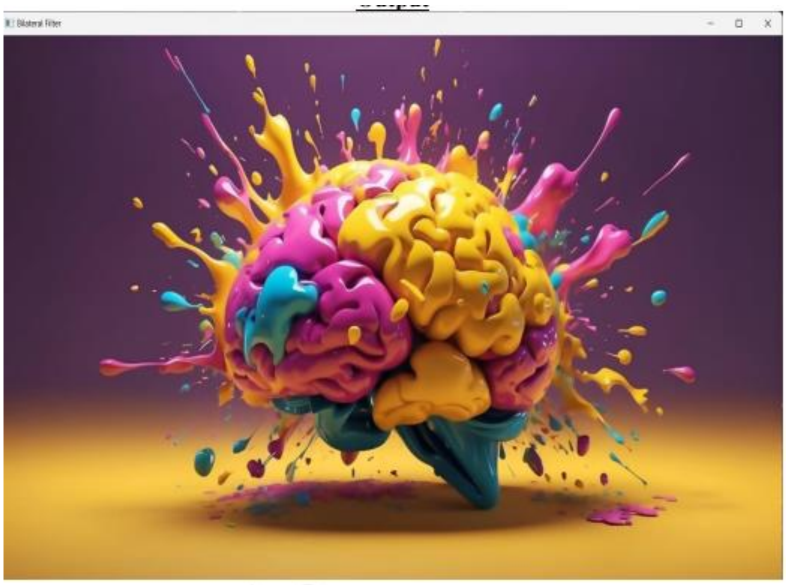


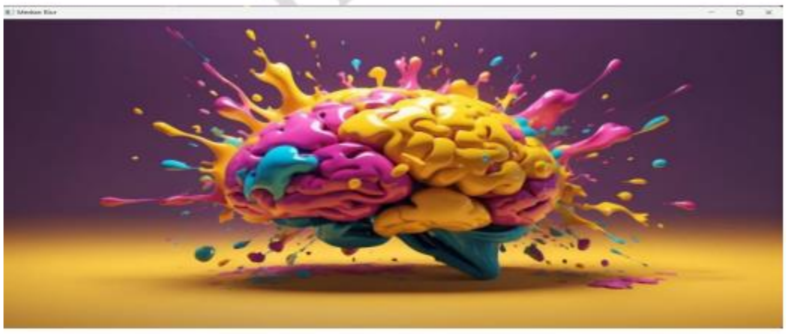
10. Write a program to blur and smoothing an image.

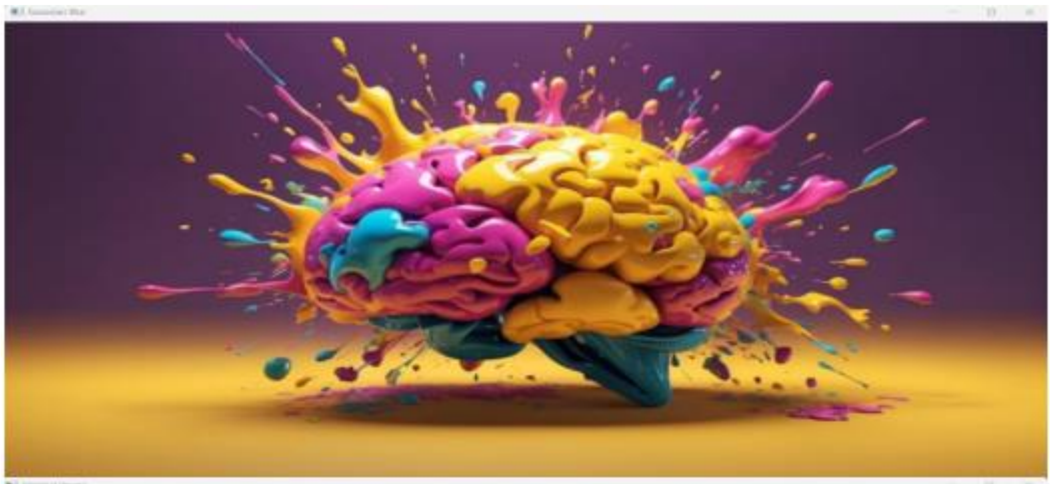
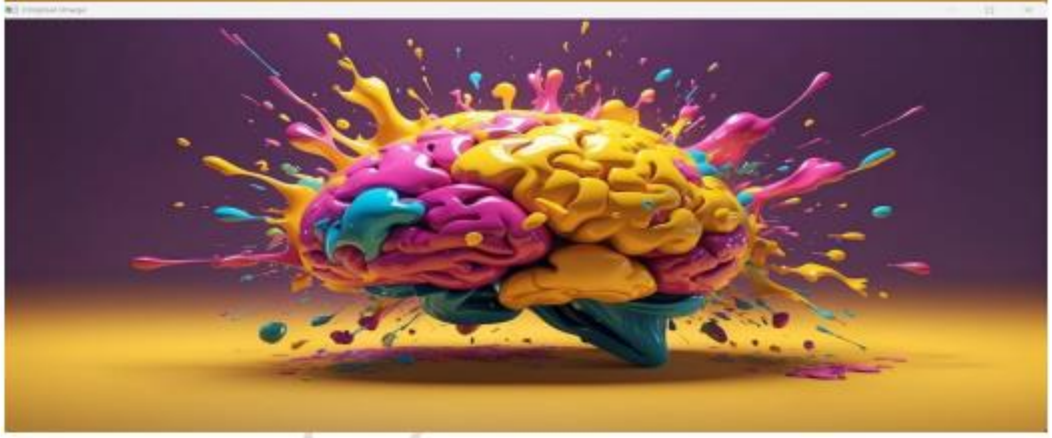
```
import cv2

# Load the image
image = cv2.imread(r'C:\Users\admin\Downloads\IMG_20221101_102849_795.jpg')

# Check if the image was successfully loaded
if image is None:
    print("Error: Could not load image.")
    exit()

# Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image, (5, 5), 0)
```

```
# Median Blur
median_blur = cv2.medianBlur(image, 5)

# Bilateral Filter
bilateral_filter = cv2.bilateralFilter(image, 9, 75, 75)

# Display the original and processed images
cv2.imshow('Original Image', image)
cv2.imshow('Gaussian Blur', gaussian_blur)
cv2.imshow('Median Blur', median_blur)
cv2.imshow('Bilateral Filter', bilateral_filter)

# Wait for a key press to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:


Bilateral Filter


Median Blur

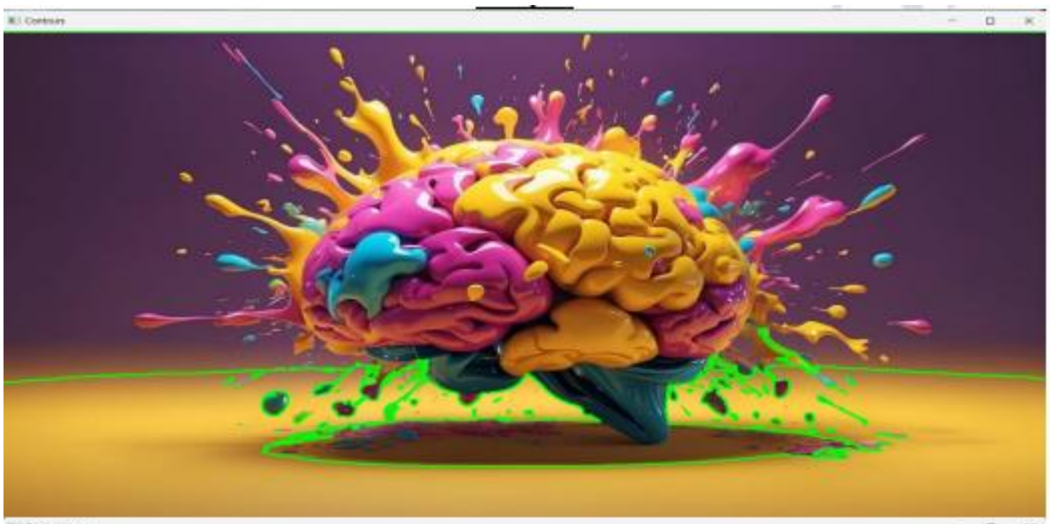



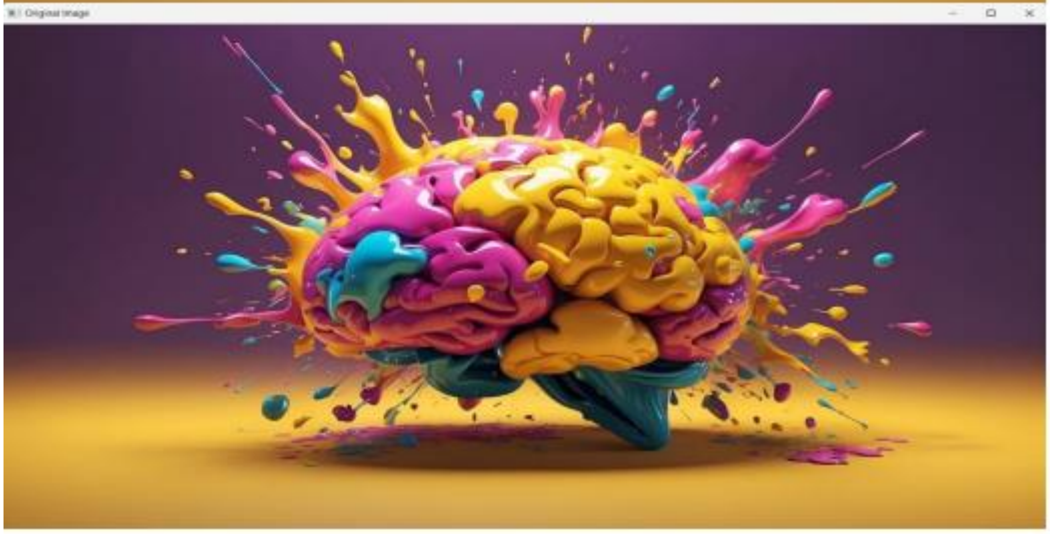
11. Write a program to contour an image.

```
import cv2
import numpy as np

# Load the image
image_path =
r'C:\Users\admin\Downloads\IMG_20221101_102849_795.jpg'
image = cv2.imread(image_path)

# Check if the image was successfully loaded
if image is None:
    print("Error: Could not load image.")
    exit()
```

```
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply binary thresholding
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

# Find contours
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Create a copy of the original image to draw contours on
contour_image = image.copy()

# Draw contours on the image
cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)

# Display the original and contour images
cv2.imshow('Original Image', image)
cv2.imshow('Contours', contour_image)

# Wait for a key press to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



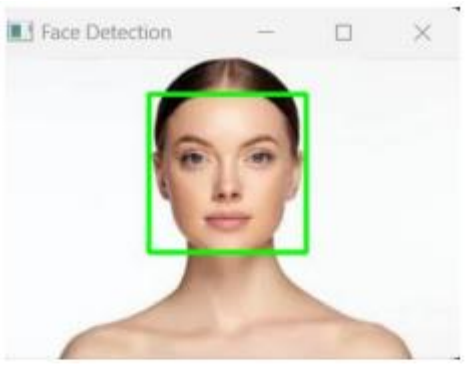


12. Write a program to detect a face/s in an image.

```
import cv2
# Load the cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
# Load the image
image = cv2.imread('image/face.jpeg')
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Detect faces in the grayscale image
```

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5,
minSize=(30, 30))
# Draw rectangles around the detected faces
for (x, y, w, h) in faces:
cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
# Display the image with detected faces
cv2.imshow('Face Detection', image)
# Wait for a key press to close the window
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:


Face Detection