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

```
    import turtle

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

    x_step = 1 if x1 < x2 else -1
    y_step = 1 if y1 < y2 else -1

    error = dx - dy

    line_points = []

    x, y = x1, y1

    while True:
        line_points.append((x, y))

        if x == x2 and y == y2:
            break

        e2 = 2 * error
        if e2 > -dy:
            error -= dy
            x += x_step
        if e2 < dx:
            error += dx
            y += y_step

    return line_points

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)

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

turtle.exitonclick()
```

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

```
 import turtle
import math

screen = turtle.Screen()
screen.bgcolor("white")

t = turtle.Turtle()
t.speed(1)
t.pensize(2)

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)

def draw_circle(x, y, radius, color):
    t.penup()
    t.goto(x, y - radius)
    t.pendown()
    t.color(color)
    t.circle(radius)


def translate(x, y, dx, dy):
    return x + dx, y + dy

def rotate(x, y, angle):
    radians = math.radians(angle)
    new_x = x * math.cos(radians) - y * math.sin(radians)
    new_y = x * math.sin(radians) + y * math.cos(radians)
    return new_x, new_y

def scale(x, y, sx, sy):
    return x * sx, y * sy

x, y = -200, 0
draw_rectangle(x, y, 100, 50, "blue")
```

```
x, y = translate(x, y, 200, 0)
draw_rectangle(x, y, 100, 50, "blue")

x, y = rotate(x, y, 45)
draw_rectangle(x, y, 100, 50, "blue")

x, y = scale(x, y, 2, 2)
draw_rectangle(x, y, 100, 50, "blue")

x, y = 100, 100
draw_circle(x, y, 50, "red")

x, y = translate(x, y, 200, 0)
draw_circle(x, y, 50, "red")

x, y = rotate(x, y, 45)
draw_circle(x, y, 50, "red")

x, y = scale(x, y, 2, 2)
draw_circle(x, y, 50, "red")

turtle.done()
```

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

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

scene = canvas(width=800, height=600, background=color.white)

def draw_cuboid(pos, length, width, height, color):
    cuboid = box(pos=vector(*pos), length=length, width=width, height=height, color=color)
    return cuboid

def draw_cylinder(pos, radius, height, color):
    cyl = cylinder(pos=vector(*pos), radius=radius, height=height, color=color)
    return cyl

def translate(obj, dx, dy, dz):
    obj.pos += vector(dx, dy, dz)

def rotate(obj, angle, axis):
    obj.rotate(angle=angle, axis=vector(*axis))

def scale(obj, sx, sy, sz):
    obj.size = vector(obj.size.x * sx, obj.size.y * sy, obj.size.z * sz)
```

```
cuboid = draw_cuboid((-2, 0, 0), 2, 2, 2, color.blue)

translate(cuboid, 4, 0, 0)

rotate(cuboid, angle=45, axis=(0, 1, 0))

scale(cuboid, 1.5, 1.5, 1.5)

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

translate(cyl, 0, -2, 0)

rotate(cyl, angle=30, axis=(1, 0, 0))

scale(cyl, 1.5, 1.5, 1.5)

while True:
    rate(30)
```

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

```
import cv2
import numpy as np

canvas_width = 500
canvas_height = 500

canvas = np.ones((canvas_height, canvas_width, 3), dtype=np.uint8) * 255

obj_points = np.array([[100, 100], [200, 100], [200, 200], [100, 200]], dtype=np.int32)

translation_matrix = np.float32([[1, 0, 100], [0, 1, 50], [0, 0, 1]])
rotation_matrix = cv2.getRotationMatrix2D((150, 150), 45, 1)
scaling_matrix = np.float32([[1.5, 0, 0], [0, 1.5, 0], [0, 0, 1]])

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(np.vstack([rotation_matrix, [0, 0, 1]]), [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)

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)

cv2.imshow("2D Transformations", canvas)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

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

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
import numpy as np

pygame.init()

display_width = 800
display_height = 600
display = pygame.display.set_mode((display_width, display_height), DOUBLEBUF |
OPENGL)
pygame.display.set_caption("3D Transformations")

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)

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)

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

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

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    glLoadIdentity()
    glMultMatrixf(translation_matrix)
    glRotatef(angle, 1, 1, 0)
    glMultMatrixf(rotation_matrix)
    glMultMatrixf(scaling_matrix)


    glBegin(GL_LINES)
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()

    angle += 1
    if angle >= 360:
        angle -= 360

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

pygame.quit()
```

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

```
import pygame
import random

pygame.init()

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

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

num_objects = 10
objects = []

for _ in range(num_objects):
    x = random.randint(50, screen_width - 50)
    y = random.randint(50, screen_height - 50)
    radius = random.randint(10, 30)
    color = random.choice([RED, GREEN, BLUE])
    speed_x = random.randint(-5, 5)
    speed_y = random.randint(-5, 5)
    objects.append({"x": x, "y": y, "radius": radius, "color": color, "speed_x": speed_x,
"speed_y": speed_y})

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

    screen.fill(WHITE)

    for obj in objects:
        obj["x"] += obj["speed_x"]
        obj["y"] += obj["speed_y"]

        if obj["x"] - obj["radius"] < 0 or obj["x"] + obj["radius"] > screen_width:
            obj["speed_x"] = -obj["speed_x"]
```

```
        if obj["y"] - obj["radius"] < 0 or obj["y"] + obj["radius"] > screen_height:
            obj["speed_y"] = -obj["speed_y"]

        pygame.draw.circle(screen, obj["color"], (obj["x"], obj["y"]), obj["radius"])

    pygame.display.flip()

    clock.tick(60)

pygame.quit()
```

**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
import matplotlib.pyplot as plt

image = cv2.imread(r'C:\Users\angad\OneDrive\Documents\download.jpg')

if image is None:
    print("Error: Unable to read the image.")
else:
    height, width, _ = image.shape
    center_x, center_y = width // 2, height // 2

    top_left = image[0:center_y, 0:center_x]
    top_right = image[0:center_y, center_x:width]
    bottom_left = image[center_y:height, 0:center_x]
    bottom_right = image[center_y:height, center_x:width]

    plt.figure(figsize=(10, 10))
    plt.subplot(2, 2, 1)
```

```
    plt.title('Top Left')
    plt.imshow(cv2.cvtColor(top_left, cv2.COLOR_BGR2RGB))

    plt.subplot(2, 2, 2)
    plt.title('Top Right')
    plt.imshow(cv2.cvtColor(top_right, cv2.COLOR_BGR2RGB))

    plt.subplot(2, 2, 3)
    plt.title('Bottom Left')
    plt.imshow(cv2.cvtColor(bottom_left, cv2.COLOR_BGR2RGB))

    plt.subplot(2, 2, 4)
    plt.title('Bottom Right')
    plt.imshow(cv2.cvtColor(bottom_right, cv2.COLOR_BGR2RGB))

    plt.show()
```

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

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path = r'C:\Users\angad\OneDrive\Documents\download.jpg'
img = cv2.imread(image_path)

if img is None:
    print("Error: Unable to load image.")
else:
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    height, width, _ = img.shape
    rotation_matrix = cv2.getRotationMatrix2D((width / 2, height / 2), 45, 1)
```

```
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)

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))

rotated_img_rgb = cv2.cvtColor(rotated_img, cv2.COLOR_BGR2RGB)
scaled_img_rgb = cv2.cvtColor(scaled_img, cv2.COLOR_BGR2RGB)
translated_img_rgb = cv2.cvtColor(translated_img, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(img_rgb)
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title("Rotated Image")
plt.imshow(rotated_img_rgb)
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title("Scaled Image")
plt.imshow(scaled_img_rgb)

plt.axis('off')
plt.subplot(2, 2, 4)
plt.title("Translated Image")
plt.imshow(translated_img_rgb)
plt.axis('off')
plt.show()
```

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

```
import cv2
import numpy as np

image_path = "C:/Users/R_K_0203/Documents/cg_project/brain1.jpg"
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not found.")
    exit()

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 100, 200)
kernel = np.ones((5, 5), np.float32) / 25
texture = cv2.filter2D(gray, -1, kernel)

cv2.imshow("Original Image", img)
cv2.imshow("Edges", edges)
cv2.imshow("Texture", texture)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

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

```
import cv2

image_path = 'C:/Users/R_K_0203/Documents/cg_project/brain1.jpg'
image = cv2.imread(image_path)
```

```
if image is None:
    print(f"Error loading image at {image_path}")
else:
    gaussian_blur = cv2.GaussianBlur(image, (5, 5), 0)
    median_blur = cv2.medianBlur(image, 5)
    bilateral_filter = cv2.bilateralFilter(image, 9, 75, 75)

    cv2.imshow('Original Image', image)
    cv2.imshow('Gaussian Blur', gaussian_blur)
    cv2.imshow('Median Blur', median_blur)
    cv2.imshow('Bilateral Filter', bilateral_filter)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

**11. Write a program to contour an image**.

```
import cv2
import numpy as np

image = cv2.imread('C:/Users/R_K_0203/Documents/cg_project/brain1.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)


contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
contour_image = image.copy()
cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
```

```
cv2.imshow('Original Image', image)
cv2.imshow('Contours', contour_image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

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

```
import cv2

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
image = cv2.imread('C:/Users/R_K_0203/Documents/cg_project/face.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.imshow('Face Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```