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 = 2 \* dy - dx**

**# Initialize the line points**

**line\_points = []**

**# Start at the first point**

**x, y = x1, y1**

**# Draw the line**

**for \_ in range(dx + 1):**

**# Add the current point to the line**

**line\_points.append((x, y))**

**# Update the error term and adjust the coordinates**

**if error > 0:**

**y += y\_step**

**error -= 2 \* dx**

**error += 2 \* dy**

**x += x\_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()**

1. **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()**

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

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

**# 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 3Dn 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=45, axis=(0, 1, 0))**

**# Scale the cuboid**

**scale(cuboid, 1.5, 1.5, 1.5)**

**# Draw a cylinder**

**cylinder = draw\_cylinder((2, 2, 0), 1, 10, color.red)**

**# Translate the cylinder**

**translate(cylinder, 0, -2, 0)**

**# Rotate the cylinder**

**rotate(cylinder, angle=30, axis=(1, 0, 0))**

**# Scale the cylinder**

**scale(cylinder, 1.5, 1.5, 1.5)**

**while True:**

**rate(30)  # Set the frame rate to 30 frames per second**

1. **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()**

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

**import numpy as np**

**from mpl\_toolkits.mplot3d import Axes3D**

**import matplotlib.pyplot as plt**

**# Define a 3D object (modify these vertices for different shapes)**

**vertices = np.array([**

**[1, 1, 1],  # Top front right**

**[-1, 1, 1],  # Top back right**

**[-1, -1, 1], # Bottom back right**

**[1, -1, 1],  # Bottom front right**

**[1, 1, -1],  # Top front left**

**[-1, 1, -1], # Top back left**

**[-1, -1, -1],# Bottom back left**

**[1, -1, -1],  # Bottom front left**

**])**

**# Define functions for transformations (modify these for different effects)**

**def translate(vertices, tx, ty, tz):**

**"""**

**Translates the object by the specified amounts in x, y, and z directions.**

**"""**

**return vertices + np.array([tx, ty, tz])**

**def rotate\_x(vertices, angle):**

**"""**

**Rotates the object around the X-axis by the given angle in degrees.**

**"""**

**theta = np.radians(angle)**

**rotation\_matrix = np.array([[1, 0, 0], [0, np.cos(theta), -np.sin(theta)], [0, np.sin(theta), np.cos(theta)]])**

**return vertices.dot(rotation\_matrix)**

**def rotate\_y(vertices, angle):**

**"""**

**Rotates the object around the Y-axis by the given angle in degrees.**

**"""**

**theta = np.radians(angle)**

**rotation\_matrix = np.array([[np.cos(theta), 0, np.sin(theta)], [0, 1, 0], [-np.sin(theta), 0, np.cos(theta)]])**

**return vertices.dot(rotation\_matrix)**

**def rotate\_z(vertices, angle):**

**"""**

**Rotates the object around the Z-axis by the given angle in degrees.**

**"""**

**theta = np.radians(angle)**

**rotation\_matrix = np.array([[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], [0, 0, 1]])**

**return vertices.dot(rotation\_matrix)**

**def scale(vertices, sx, sy, sz):**

**"""**

**Scales the object by the specified factors in x, y, and z directions.**

**"""**

**return vertices \* np.array([sx, sy, sz])**

**# Apply transformations (replace with desired operations)**

**transformed\_vertices = translate(vertices, 2, 1, 0)  # Translate object**

**transformed\_vertices = rotate\_y(transformed\_vertices, 60)  # Rotate around Y-axis**

**# Define viewing parameters (optional, adjust for better visualization)**

**view\_elev = 15  # Elevation angle for viewing (in degrees)**

**view\_azim = -60  # Azimuth angle for viewing (in degrees)**

**# Plot the original and transformed object**

**fig = plt.figure(figsize=(10, 6))**

**ax = fig.add\_subplot(121, projection='3d')**

**ax.scatter(vertices[:, 0], vertices[:, 1], vertices[:, 2], c='red', marker='o', label='Original')**

**ax.set\_xlabel("X Label")**

**ax.set\_ylabel("Y Label")**

**ax.set\_zlabel("Z Label")**

**ax = fig.add\_subplot(122, projection='3d')**

**ax.scatter(transformed\_vertices[:, 0], transformed\_vertices[:, 1], transformed\_vertices[:, 2], c='blue', marker='o', label='Transformed')**

**ax.set\_xlabel("X Label")**

**ax.set\_ylabel("Y Label")**

**ax.set\_zlabel("Z Label")**

**# Set viewing angles (optional)**

**ax.view\_init(elev=view\_elev, azim=view\_azim)**

**plt.title("3D Transformation Demonstration")**

**plt.legend()**

**plt.show()**

1. **Develop a program to demonstrate Animation effects on simple objects**

**import pygame**

**import random**

**# Initialize Pygame**

**pygame.init()**

**# Set up the display**

**screen\_width = 800**

**screen\_height = 600**

**screen = pygame.display.set\_mode((screen\_width, screen\_height))**

**pygame.display.set\_caption("Animation Effects")**

**# Define colors**

**BLACK = (0, 0, 0)**

**WHITE = (255, 255, 255)**

**RED = (255, 0, 0)**

**GREEN = (0, 255, 0)**

**BLUE = (0, 0, 255)**

**# Define object properties**

**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})**

**# Main loop**

**running = True**

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

**while running:**

**# Handle events**

**for event in pygame.event.get():**

**if event.type == pygame.QUIT:**

**running = False**

**# Clear the screen**

**screen.fill(WHITE)**

**# Update and draw objects**

**for obj in objects:**

**# Move the object**

**obj["x"] += obj["speed\_x"]**

**obj["y"] += obj["speed\_y"]**

**# Bounce off the edges**

**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"]**

**# Draw the object**

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

**# Update the display**

**pygame.display.flip()**

**clock.tick(60)  # Limit the frame rate to 60 FPS**

**# Quit Pygame**

**pygame.quit()**

1. **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**

**# Define image path (replace with your image path)**

**image\_path = "atc.jpg"**

**# Load the image**

**img = cv2.imread(image\_path)**

**# Check if image loading was successful**

**if img is None:**

**print("Error: Could not load image from", image\_path)**

**exit()**

**# Get the image height, width, and number of channels**

**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()**

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

**import cv2**

**import numpy as np**

**# Load the image**

**image\_path = "atc.jpg"  # Replace with the path to your image**

**img = cv2.imread(image\_path)**

**# 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()**

1. **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 = "atc.jpg"  # Replace with the path to your image**

**img = cv2.imread(image\_path)**

**# Convert the image to grayscale**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**# Edge detection**

**edges = cv2.Canny(gray, 100, 200)  # Use Canny edge detector**

**# Texture extraction**

**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()**

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

**import cv2**

**# Load the image**

**image = cv2.imread('atc.jpg')**

**# 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()**

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

**import cv2**

**import numpy as np**

**# Load the image**

**image = cv2.imread('atc.jpg')**

**# 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()**

1. **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('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()**