1)

```
from OpenGL.GL import*
from OpenGL.GLUT import*
from OpenGL.GLU import*

def init():
    glClearColor(0.0,0.0,0.0,1.0)
    gluOrtho2D(0,100,0,100)

def plotLine(x1,y1,x2,y2):
    m= 2 * (y2 - y1)
    pk=m - (x2 - x1)
    y=y1
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(1.0,0.0,0.0)
    glPointSize(10.0)
    glBegin(GL_POINTS)
    for x in range(x1,x2+1):
        glVertex2f(x,y)
        pk = pk + m
        if (pk>=0):
            y=y+1
            pk= pk-2*(x2-x1)
    glEnd()
    glFlush()
x1 = int(input("Enter x1:"))
y1 = int(input("Enter y1:"))
x2 = int(input("Enter x2:"))
y2 = int(input("Enter y2:"))
print("starting window.....")
glutInit(sys.argv)
glutInitDisplayMode(GLUT_RGB)
glutInitWindowSize(500,500)
glutInitWindowPosition(0,0)
glutCreateWindow("Bresenham Line Algorithm")
glutDisplayFunc (lambda:plotLine(x1,y1,x2,y2))
init()
glutMainLoop()
```

——

2)

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
```

```
def init():
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluOrtho2D(0, 100, 0, 100)
    glClearColor(1.0, 1.0, 1.0, 1.0)
    glClear(GL_COLOR_BUFFER_BIT)

def draw():
    glColor3f(0.0, 1.0, 0.0)
    glPointSize(10.0)
    glBegin(GL_POLYGON)
    glVertex2i(10, 20)
    glVertex2i(80, 20)
    glVertex2i(80, 60)
    glVertex2i(10, 60)
    glEnd()
    glFlush()

def main():
    glutInit()
    glutInitWindowSize(500, 500)
    glutCreateWindow("OpenGL Window")
    glutDisplayFunc(draw)
    init()
    glutMainLoop()

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

————————————————————————————————————
———————————

3]

```
from OpenGL.GL import*
from OpenGL.GLU import*
from OpenGL.GLUT import*

vertices=(
    (1,-1,-1),
    (1,1,-1),
    (-1,1,-1),
    (-1,-1,-1),
    (1,-1,1),
    (1,1,1),
    (-1,-1,1),
    (-1,1,1)
)

edges=(
   (0,1),
   (1,2),
   (2,3),
   (3,4),
   (5,6),
   (6,7),
   (7,4),
   (0,4),
   (1,5),
   (2,6),
   (3,7)
)

surfaces=(
    (0,1,2,3),
    (3,2,7,6),
    (6,7,5,4),
    (4,5,1,0),
    (1,5,7,2),
    (4,0,3,6)
)

colors=(
    (1, 0, 0),
    (0, 1, 0),
    (0, 0, 1),
    (1, 1, 0),
    (1, 0, 1),
    (0, 1, 1)
)

def Cube():
    glBegin(GL_QUADS)
    for i,surface in enumerate(surfaces):
        glColor3fv(colors[i])
        for vertex in surface:
            glVertex3fv(vertices[vertex])
    glEnd()

    glBegin(GL_LINES)
    glColor3fv((0,0,0,))
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()

def display():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glRotatef(1,3,1,1)
    Cube()
    glutSwapBuffers()
```

```
def timer(value):
   glutPostRedisplay()
   glutTimerFunc(10,timer,0)

def main():
   glutInit()
   glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
   glutInitWindowSize(800,600)
   glutCreateWindow("Rotating Cube")
   glEnable(GL_DEPTH_TEST)
   gluPerspective(45, (800 / 600), 0.1, 50.0)
   glTranslatef(0.0, 0.0, -5)
   glutDisplayFunc(display)
   glutTimerFunc(10, timer, 0)
   glutMainLoop()

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

— —
— — — — — — — — — — — —

4}

```
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *

angle, scale_factor, translate_x, translate_y = 0.0, 1.0, 0.0, 0.0

def init():
   glClearColor(1.0, 1.0, 1.0, 1.0)
   glMatrixMode(GL_PROJECTION)
   glLoadIdentity()
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0)

def draw_square():
   glBegin(GL_POLYGON)
   for vertex in [(-0.1, -0.1), (0.1, -0.1), (0.1, 0.1), (-0.1, 0.1)]:
      glVertex2f(*vertex)
   glEnd()

def display():
   global angle, scale_factor, translate_x, translate_y
   glClear(GL_COLOR_BUFFER_BIT)
   glLoadIdentity()
   glTranslatef(translate_x, translate_y, 0.0)
   glRotatef(angle, 0.0, 0.0, 1.0)
   glScalef(scale_factor, scale_factor, 1.0)
   glColor3f(0.0, 0.0, 1.0)
   draw_square()
   glutSwapBuffers()
```

```
def keyboard(key, x, y):
    global angle, scale_factor, translate_x, translate_y
    key = key.decode("utf-8").lower()
    if key == 'q':
        sys.exit()
    elif key == 'r':
        angle += 10.0
    elif key == 's':
        scale_factor += 0.1
    elif key == 't':
        translate_x += 0.1
    elif key == 'f':
        translate_x -= 0.1
    elif key == 'g':
        translate_y += 0.1
    elif key == 'h':
        translate_y -= 0.1
    glutPostRedisplay()

def main():
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB)
    glutInitWindowSize(500, 500)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"2D Transformation")
    glutDisplayFunc(display)
    glutKeyboardFunc(keyboard)
    init()
    glutMainLoop()

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

—————————————————————————————————————
————————————

5}

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *

angle = 0
scale = 1.0
translation = [0.0, 0.0, 0.0]

vertices = (
    (1, -1, -1),
    (1, 1, -1),
    (-1, 1, -1),
    (-1, -1, -1),
    (1, -1, 1),
    (1, 1, 1),
    (-1, -1, 1),
    (-1, 1, 1)
)

edges = (
   (0, 1),
   (1, 2),
   (2, 3),
   (3, 0),
   (4, 5),
   (5, 6),
   (6, 7),
   (7, 4),
   (0, 4),
   (1, 5),
   (2, 6),
   (3, 7)
)

surfaces = (
   (0, 1, 2, 3),
   (3, 2, 7, 6),
   (6, 7, 5, 4),
   (4, 5, 1, 0),
   (1, 5, 7, 2),
   (4, 0, 3, 6)
)

colors = (
   (1, 0, 0),
   (0, 1, 0),
   (0, 0, 1),
   (1, 1, 0),
   (1, 0, 1),
   (0, 1, 1)
)

def draw_cube():
  glBegin(GL_QUADS)
  for i, surface in enumerate(surfaces):
    glColor3fv(colors[i])
    for vertex in surface:
      glVertex3fv(vertices[vertex])
  glEnd()

  glBegin(GL_LINES)
  glColor3fv((0, 0, 0))
  for edge in edges:
    for vertex in edge:
     glVertex3fv(vertices[vertex])
  glEnd()

def display():
  global angle
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
  glLoadIdentity()
  glTranslatef(0.0, 0.0, -5)
  glScalef(scale, scale, scale)
  glTranslatef(*translation)
  glRotatef(angle, 3, 1, 1)
  draw_cube()
  glutSwapBuffers()
  angle += 0.5

def keyboard(key, x, y):
  global scale, translation
  if key == b'\x1b': # ESC key
   glutLeaveMainLoop()
  elif key == b'+':
   scale += 0.1
  elif key == b'-':
   scale -= 0.1
  elif key == GLUT_KEY_LEFT:
   translation[0] -= 0.1
  elif key == GLUT_KEY_RIGHT:
   translation[0] += 0.1
  elif key == GLUT_KEY_UP:
   translation[1] += 0.1
  elif key == GLUT_KEY_DOWN:
   translation[1] -= 0.1

def reshape(width, height):
  if height == 0:
    height = 1
  glViewport(0, 0, width, height)
  glMatrixMode(GL_PROJECTION)
  glLoadIdentity()
  gluPerspective(45, width / height, 0.1, 50.0)
  glMatrixMode(GL_MODELVIEW)
  glLoadIdentity()

def main():
  glutInit()
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)
  glutInitWindowSize(800, 600)
  glutCreateWindow("Rotating, Scaling, and Translating Cube")
  glEnable(GL_DEPTH_TEST)
  glutDisplayFunc(display)
  glutIdleFunc(display)
  glutReshapeFunc(reshape)
  glutSpecialFunc(keyboard)

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

————————————————————————————————————
———————————

6}

```
from OpenGL.GL import*
from OpenGL.GLU import*
from OpenGL.GLUT import*

vertices=(
    (1,-1,-1),
    (1,1,-1),
    (-1,1,-1),
    (-1,-1,-1),
    (1,-1,1),
    (1,1,1),
    (-1,-1,1),
    (-1,1,1)
)

edges=(
   (0,1),
   (1,2),
   (2,3),
   (3,4),
   (5,6),
   (6,7),
   (7,4),
   (0,4),
   (1,5),
   (2,6),
   (3,7)
)

surfaces=(
    (0,1,2,3),
    (3,2,7,6),
    (6,7,5,4),
    (4,5,1,0),
    (1,5,7,2),
    (4,0,3,6)
)

colors=(
    (1, 0, 0),
    (0, 1, 0),
    (0, 0, 1),
    (1, 1, 0),
    (1, 0, 1),
    (0, 1, 1)
)

def Cube():
    glBegin(GL_QUADS)
    for i,surface in enumerate(surfaces):
        glColor3fv(colors[i])
        for vertex in surface:
            glVertex3fv(vertices[vertex])
    glEnd()

    glBegin(GL_LINES)
    glColor3fv((0,0,0,))
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()

def display():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glRotatef(1,3,1,1)
    Cube()
    glutSwapBuffers()

def timer(value):
    glutPostRedisplay()
    glutTimerFunc(10,timer,0)

def main():
    glutInit()
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
    glutInitWindowSize(800,600)
    glutCreateWindow("Rotating Cube")
    glEnable(GL_DEPTH_TEST)
    gluPerspective(45, (800 / 600), 0.1, 50.0)
    glTranslatef(0.0, 0.0, -5)
    glutDisplayFunc(display)
    glutTimerFunc(10, timer, 0)
    glutMainLoop()

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

— —

7}

```
import cv2
```

```
def split_image(image_path):
    # Read the image
    image = cv2.imread(image_path)

    if image is None:
        raise FileNotFoundError(f"Image at path '{image_path}' not found.")

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

    # Split the image into four quadrants
    top_left = image[0:height//2, 0:width//2]
    top_right = image[0:height//2, width//2:width]
    bottom_left = image[height//2:height, 0:width//2]
    bottom_right = image[height//2:height, width//2:width]

    return top_left, top_right, bottom_left, bottom_right

def display_quadrants(top_left, top_right, bottom_left, bottom_right):
    cv2.imshow('Top Left Quadrant', top_left)
    cv2.imshow('Top Right Quadrant', top_right)
    cv2.imshow('Bottom Left Quadrant', bottom_left)
    cv2.imshow('Bottom Right Quadrant', bottom_right)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def main():
    image_path = "bf.jpeg"  # Replace with your actual image path
    try:
        top_left, top_right, bottom_left, bottom_right = split_image(image_path)
        display_quadrants(top_left, top_right, bottom_left, bottom_right)
    except FileNotFoundError as e:
        print(e)

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

—————————————————————————————————————
——————————

8}

```
import cv2
import numpy as np

def rotate_image(image, angle):
    height, width = image.shape[:2]
    rotation_matrix = cv2.getRotationMatrix2D((width / 2, height / 2), angle, 1)
    rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))
    return rotated_image

def scale_image(image, scale_factor):
    scaled_image = cv2.resize(image, None, fx=scale_factor, fy=scale_factor,
interpolation=cv2.INTER_LINEAR)
    return scaled_image

def translate_image(image, tx, ty):
    translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
    translated_image = cv2.warpAffine(image, translation_matrix, (image.shape[1],
image.shape[0]))
    return translated_image

def main():
    # Read the image
    image_path = "bf.jpeg"  # Replace with the path to your image
    original_image = cv2.imread(image_path)

    if original_image is None:
        raise FileNotFoundError(f"Image at path '{image_path}' not found.")

    # Rotate the image
    rotated_image = rotate_image(original_image, 45)

    # Scale the image
    scaled_image = scale_image(original_image, 1.5)

    # Translate the image
    translated_image = translate_image(original_image, 50, 50)

    # Display the images
    cv2.imshow('Original Image', original_image)
    cv2.imshow('Rotated Image', rotated_image)
    cv2.imshow('Scaled Image', scaled_image)
    cv2.imshow('Translated Image', translated_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

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

— —
— — — — — — — — — — — —

9}

```
import cv2
import numpy as np

def display_image(title, image):
    cv2.imshow(title, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def canny_edge_detection(image):
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply Gaussian blur to reduce noise
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # Perform Canny edge detection
    edges = cv2.Canny(blurred, 50, 150)
    return edges

def texture_filtering(image):
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply Laplacian filter for edge detection
    laplacian = cv2.Laplacian(gray, cv2.CV_64F)
    # Convert the Laplacian result to uint8
    laplacian_uint8 = np.uint8(np.absolute(laplacian))
    return laplacian_uint8

def main():
    # Read the image
    image_path = "bf.jpeg"  # Replace with the path to your image
    image = cv2.imread(image_path)

    if image is None:
        raise FileNotFoundError(f"Image at path '{image_path}' not found.")

    # Apply Canny edge detection
    edges = canny_edge_detection(image)
    display_image('Canny Edge Detection', edges)

    # Apply texture filtering
    textures = texture_filtering(image)
    display_image('Texture Filtering (Laplacian)', textures)

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

————————————————————————————————————
———————————

10}

```
import cv2

def display_image(title, image):
    cv2.imshow(title, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def blur_image(image):
    # Apply Gaussian blur
    blurred = cv2.GaussianBlur(image, (5, 5), 0)
    return blurred

def main():
    # Read the image
    image_path = "bf.jpeg"  # Replace with the path to your image
    image = cv2.imread(image_path)

    if image is None:
        raise FileNotFoundError(f"Image at path '{image_path}' not found.")

    # Blur the image
    blurred_image = blur_image(image)

    # Display the original and blurred images
    display_image('Original Image', image)
    display_image('Blurred Image', blurred_image)

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

——————————————————————————————————
———————————

11}

```
import cv2

def display_image(title, image):
    cv2.imshow(title, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def contour_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    contour_image = image.copy()
    cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
    return contour_image

def main():
    # Read the image
    image_path = "bf.jpeg"  # Replace with the path to your image
    image = cv2.imread(image_path)

    if image is None:
        raise FileNotFoundError(f"Image at path '{image_path}' not found.")

    contoured_image = contour_image(image)
    display_image('Original Image', image)
    display_image('Contoured Image', contoured_image)

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

——————————————————————————————————

————————————

```
12}
import cv2

def display_image(title, image):
    cv2.imshow(title, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def detect_faces(image_path):
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    image = cv2.imread(image_path)

    if image is None:
        raise FileNotFoundError(f"Image at path '{image_path}' not found.")

    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)

    return image

def main():
    image_path = "hf.jpg"  # Replace with the path to your image
    image_with_faces = detect_faces(image_path)
    display_image('Image with Faces Detected', image_with_faces)

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