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

```
#include <GL/glut.h>
#include <cmath>

void setPixel(int x, int y) {
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
}

void lineBres(int x0, int y0, int xEnd, int yEnd) {
   int dx = fabs(xEnd - x0), dy = fabs(yEnd - y0);
   int p = 2 * dy - dx;
   int twoDy = 2 * dy, twoDyMinusDx = 2 * (dy - dx);
   int x, y;

   if (x0 > xEnd) {
      x = xEnd;
      y = yEnd;
      xEnd = x0;
   }
   else {
      x = x0;
      y = y0;
   }

   setPixel(x, y);
   while (x < xEnd) {
      x++;
      if (p < 0)
         p += twoDy;
      else {
         y++;
         p += twoDyMinusDx;
      }
      setPixel(x, y);
   }
}

void display() {
   glClearColor(0.0, 0.0, 0.0, 0.0); // Black background
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0); // White color

   // Define line coordinates
   int x0 = 50, y0 = 100;
   int xEnd = 200, yEnd = 300;

   lineBres(x0, y0, xEnd, yEnd);
```

```
  glFlush();
}

void init() {

  //glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0.0, 500.0, 0.0, 500.0); // Set the window coordinates
}

int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500); // Set the window size
  glutInitWindowPosition(100, 100); // Set the window position
  glutCreateWindow("Bresenham Line Drawing"); // Create the window
    init();
  glutDisplayFunc(display); // Register the display function
  glutMainLoop();
  return 0;
}
```

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

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>

// Initial triangle vertices
float vertices[3][2] = {
  {0.0, 0.5},
  {-0.5, -0.5},
  {0.5, -0.5}
};

// Transformation parameters
float tx = 0.0, ty = 0.0; // Translation
float sx = 1.0, sy = 1.0; // Scaling
float angle = 0.0;       // Rotation

void display() {
  glClear(GL_COLOR_BUFFER_BIT);

  glBegin(GL_TRIANGLES);
  for (int i = 0; i < 3; i++) {
    // Perform transformations manually
    float x = vertices[i][0];
    float y = vertices[i][1];

    // Scaling
    x *= sx;
```

```
    y *= sy;

    // Rotation
    float rad = angle * 3.141/ 180.0;
    float x_rot = x * cos(rad) - y * sin(rad);
    float y_rot = x * sin(rad) + y * cos(rad);
    x = x_rot;
    y = y_rot;

    // Translation
    x += tx;
    y += ty;

    glVertex2f(x, y);
  }
  glEnd();

  glFlush();
}

void keyboard(unsigned char key, int x, int y) {
  switch (key) {
  case 'w': ty += 0.1; break; // Move up
  case 's': ty -= 0.1; break; // Move down
  case 'a': tx -= 0.1; break; // Move left
  case 'd': tx += 0.1; break; // Move right
  case '+':
    sx += 0.1;
    sy += 0.1;
    if (sx > 2.0) sx = 2.0; // Prevent excessive scaling
    if (sy > 2.0) sy = 2.0;
    break; // Scale up
  case '-':
    sx -= 0.1;
    sy -= 0.1;
    if (sx < 0.1) sx = 0.1; // Prevent negative or too small scaling
    if (sy < 0.1) sy = 0.1;
    break; // Scale down
  case 'r': angle += 5.0; break; // Rotate clockwise
  case 'l': angle -= 5.0; break; // Rotate counterclockwise
  case 27: exit(0); // Escape key to exit
  }
  glutPostRedisplay();
}

void reshape(int width, int height) {
  // Prevent division by zero
  if (height == 0) {
    height = 1;
  }
```

```
    // Set the viewport to cover the new window
    glViewport(0, 0, width, height);

    // Set the aspect ratio of the clipping area to match the viewport
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (width >= height) {
        gluOrtho2D(-1.0 * width / height, 1.0 * width / height, -1.0, 1.0);
    }
    else {
        gluOrtho2D(-1.0, 1.0, -1.0 * height / width, 1.0 * height / width);
    }

    glMatrixMode(GL_MODELVIEW);
}

void init() {
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("2D Transformations - Translation, Scaling, Rotation");

    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);  // Set the reshape callback
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

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

```
#include<stdlib.h>
#include<GL/glut.h>
#include<math.h>
double rot = 0,rot2=0,move_x=0,move_y=0,move_z=0;
void init() {
	glMatrixMode(GL_MODELVIEW);
	glEnable(GL_DEPTH_TEST);
}
void face(float P[8][3], int a,int b,int c,int d) {
	glBegin(GL_QUADS);
	glVertex3fv(P[a]);
```

```
    glVertex3fv(P[b]);
    glVertex3fv(P[c]);
    glVertex3fv(P[d]);
    glEnd();
}
void cube(float tx,float ty,float tz,float rx,float ry,float rz,float scale,float c) {
    float P[8][3] = {
        {-1,-1,-1},
        {-1,-1,+1},
        {+1,-1,+1},
        {+1,-1,-1},
        {-1,+1,-1},
        {-1,+1,+1},
        {+1,+1,+1},
        {+1,+1,-1}
    };
    rx += rot2;
    ry += rot;
    rx = rx * 3.1415 / 180;
    ry = ry * 3.1415 / 180;
    rz = rz * 3.1415 / 180;
    for (int i = 0;i < 8;i++) {
        float x= P[i][0], y= P[i][1], z= P[i][2],t;
        x = x*scale+tx;
        y = y*scale+ty;
        z = z*scale+tz;
        t = y * cos(rx) - z * sin(rx);
        z = y * sin(rx) + z * cos(rx);
        y = t;

        t = z * cos(ry) - x * sin(ry);
        x = z * sin(ry) + x * cos(ry);
        z = t;

        t = x * cos(rz) - y * sin(rz);
        y = x * sin(rz) + y * cos(rz);
        x = t;

        P[i][0] = x +move_x;
        P[i][1] = y +move_y;
        P[i][2] = z +move_z;
    }
    glColor3f(c, 0, 0);
    face(P, 0, 1, 2, 3);
    glColor3f(c, c, 0);
    face(P, 0, 1, 5, 4);
    glColor3f(0, c, 0);
    face(P, 1, 2, 6, 5);
    glColor3f(0, c, c);
    face(P, 2, 3, 7, 6);
```

```
    glColor3f(0, 0, c);
    face(P, 3, 0, 4, 7);
    glColor3f(c, 0, c);
    face(P, 4, 5, 6, 7);
}

void disp() {
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    cube(0, 0, 0, 0, 0, 0, 0.1, 1.0);
    //cube(0.5, 0, 0, 0, 0, 0, 0.2, 0.9);
    //cube(0, 0.5, 0, 0, 0, 0, 0.2, 0.8);
    //cube(0, 0, 0.5, 0, 0, 0, 0.2, 0.7);
    glutSwapBuffers();
}
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
    case 'w':move_y+=0.05;break;
    case 'a':move_x-=0.05;break;
    case 's':move_y-=0.05;break;
    case 'd':move_x+=0.05;break;
    case '.':rot--;break;
    case ',':rot++;break;
    case '[':rot2--;break;
    case ']':rot2++;break;
    case 'r':move_x=move_z=rot=rot2=0;break;
    case 'q': exit(0);
    }
    glutPostRedisplay();
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(300, 300);

    glutCreateWindow("3D Operation");
    init();
    glutDisplayFunc(disp);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

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

```
#include <GL/glut.h>
#include <stdlib.h>

// Initialize the angle, scale, and translation values
float angle = 0.0;
```

```
float scale = 1.0;
float translateX = 0.0;
float translateY = 0.0;
const float xmin = -10.0, xmax = 10.0, ymin = -10.0, ymax = 10.0; // Define your viewport limits

// Function to initialize the OpenGL environment
void initGL() {
  glClearColor(0.0, 0.0, 0.0, 1.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-10.0, 10.0, -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);
}

// Function to handle window reshaping
void reshapefunc(int w, int h) {
  glViewport(0, 0, w, h);
  if (w > h) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(xmin * (float)w / (float)h, xmax * (float)w / (float)h, ymin, ymax);
    glMatrixMode(GL_MODELVIEW);
  }
  else {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(xmin, xmax, ymin * (float)h / (float)w, ymax * (float)h / (float)w);
    glMatrixMode(GL_MODELVIEW);
  }
  glutPostRedisplay();
}

// Function to display the triangle
void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity(); // Reset transformations

  // Apply transformations
  glTranslatef(translateX, translateY, 0.0);
  glScalef(scale, scale, 1.0); // Apply scaling
  glRotatef(angle, 0.0, 0.0, 1.0);

  // Draw a triangle
  glBegin(GL_TRIANGLES);
  glColor3f(1.0, 0.0, 0.0);
  glVertex2f(-5.0, -5.0);
  glColor3f(0.0, 1.0, 0.0);
  glVertex2f(5.0, -5.0);
  glColor3f(0.0, 0.0, 1.0);
  glVertex2f(0.0, 5.0);
```

```
    glEnd();

    glFlush();
}

// Function to handle keyboard input
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
    case 'a':
        angle += 5.0;
        break;
    case 'd':
        angle -= 5.0;
        break;
    case 'w':
        scale += 0.1;
        reshapefunc(glutGet(GLUT_WINDOW_WIDTH), glutGet(GLUT_WINDOW_HEIGHT));
        break;
    case 's':
        if (scale > 0.1) // Ensure scale doesn't go below a minimum value
            scale -= 0.1;
        reshapefunc(glutGet(GLUT_WINDOW_WIDTH), glutGet(GLUT_WINDOW_HEIGHT));
        break;
     case 'i':
        translateY += 0.5;
        break;
    case 'k':
        translateY -= 0.5;
        break;
    case 'j':
        translateX -= 0.5;
        break;
    case 'l':
        translateX += 0.5;
        break;
    case 27: // Escape key
        exit(0);
        break;
    }
    glutPostRedisplay();
}

// Main function
     int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("2D Geometric Operations using builtin functions");

    initGL();
```

```
    glutDisplayFunc(display);
    glutReshapeFunc(reshapefunc); // Use reshapefunc for reshaping
    glutKeyboardFunc(keyboard);

    glutMainLoop();
    return 0;
}
```

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

```
#include<stdlib.h>
#include <GL/glut.h>

float angleX = 0.0, angleY = 0.0, angleZ = 0.0;
float scale = 1.0;
float translateX = 0.0, translateY = 0.0, translateZ = 0.0;

void init() {
  glClearColor(0.0, 0.0, 0.0, 1.0);
  glEnable(GL_DEPTH_TEST);
}

void drawCube() {
  glBegin(GL_QUADS);

  // Front face
  glColor3f(1.0, 0.0, 0.0);
  glVertex3f(-0.5, -0.5, 0.5);
  glVertex3f(0.5, -0.5, 0.5);
  glVertex3f(0.5, 0.5, 0.5);
  glVertex3f(-0.5, 0.5, 0.5);

  // Back face
  glColor3f(0.0, 1.0, 0.0);
  glVertex3f(-0.5, -0.5, -0.5);
  glVertex3f(-0.5, 0.5, -0.5);
  glVertex3f(0.5, 0.5, -0.5);
  glVertex3f(0.5, -0.5, -0.5);

  // Left face
  glColor3f(0.0, 0.0, 1.0);
  glVertex3f(-0.5, -0.5, -0.5);
  glVertex3f(-0.5, -0.5, 0.5);
  glVertex3f(-0.5, 0.5, 0.5);
  glVertex3f(-0.5, 0.5, -0.5);

  // Right face
  glColor3f(1.0, 1.0, 0.0);
  glVertex3f(0.5, -0.5, -0.5);
  glVertex3f(0.5, 0.5, -0.5);
```

```
    glVertex3f(0.5, 0.5, 0.5);
    glVertex3f(0.5, -0.5, 0.5);

    // Top face
    glColor3f(0.0, 1.0, 1.0);
    glVertex3f(-0.5, 0.5, -0.5);
    glVertex3f(-0.5, 0.5, 0.5);
    glVertex3f(0.5, 0.5, 0.5);
    glVertex3f(0.5, 0.5, -0.5);

    // Bottom face
    glColor3f(1.0, 0.0, 1.0);
    glVertex3f(-0.5, -0.5, -0.5);
    glVertex3f(0.5, -0.5, -0.5);
    glVertex3f(0.5, -0.5, 0.5);
    glVertex3f(-0.5, -0.5, 0.5);

    glEnd();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // GL_DEPTH_BUFFER_BIT: Clears the depth buffer. This resets the depth values
    // for each pixel to a large value (usually 1.0)
    // because depth values range from 0.0 (closest) to 1.0 (farthest).
    glLoadIdentity();
    glTranslatef(translateX, translateY, translateZ);
    glRotatef(angleX, 1.0, 0.0, 0.0);
    glRotatef(angleY, 0.0, 1.0, 0.0);
    glRotatef(angleZ, 0.0, 0.0, 1.0);
    glScalef(scale, scale, scale); // Apply uniform scaling

    drawCube();

    glutSwapBuffers();
}

void keyboard(unsigned char key, int x, int y) {
    switch (key) {
    case 'x': angleX += 5.0; break;
    case 'X': angleX -= 5.0; break;
    case 'y': angleY += 5.0; break;
    case 'Y': angleY -= 5.0; break;
    case 'z': angleZ += 5.0; break;
    case 'Z': angleZ -= 5.0; break;
    case 's': scale += 0.1; break;
    case 'S': scale -= 0.1; break;
    case 't': translateX += 0.1; break;
    case 'T': translateX -= 0.1; break;
    case 'u': translateY += 0.1; break;
```

```
  case 'U': translateY -= 0.1; break;
  case 'v': translateZ += 0.1; break;
  case 'V': translateZ -= 0.1; break;
  case 27: exit(0); break; // ESC to exit
  }
  glutPostRedisplay();
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(800, 600);
  glutCreateWindow("3D Transformations");

  init();
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);

  glutMainLoop();
  return 0;
}
```

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

```
#include <math.h>
#include <GL/glut.h>

float theta = 0;
float x, y, r = 50;
bool rotationEnabled = false;  // Flag to control rotation

void init()
{
  gluOrtho2D(-100, 100, -100, 100);
}

void idle()
{
  if (rotationEnabled) {
    theta += 0.01;
    if (theta >= 360)
      theta = 0;
  }
  glutPostRedisplay();
}

void disp()
{
  glClearColor(1, 1, 1, 1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0, 0, 1);
```

```
  x = r * cos(theta * 3.142 / 180);
  y = r * sin(theta * 3.142 / 180);

  glBegin(GL_POLYGON);
  glVertex2f(x, y);
  glVertex2f(-1 * y, x);
  glVertex2f(-1 * x, -1 * y);
  glVertex2f(y, -1 * x);
  glEnd();

  glutSwapBuffers();
}

void mouse(int button, int state, int x, int y)
{
  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
    rotationEnabled = true;
  }
  if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
    rotationEnabled = false;
  }
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(600, 600);
  glutInitWindowPosition(300, 150);
  glutCreateWindow("Idle");
  init();
  glutDisplayFunc(disp);
  glutMouseFunc(mouse);
  glutIdleFunc(idle);
  glutMainLoop();
  return 0;
}
```

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  # Importing matplotlib.pyplot

# Read the image
img = cv2.imread("rnsit.jpg")

# Get the height and width of the image
height, width = img.shape[:2]
```

```
# Split the image into four quadrants
quad1 = img[:height//2, :width//2] # slices the image to get the top-left quadrant.
quad2 = img[:height//2, width//2:] #slices the image to get the top-right quadrant.
quad3 = img[height//2:, :width//2] #slices the image to get the bottom-left quadrant.
quad4 = img[height//2:, width//2:] #slices the image to get the bottom-right quadrant.

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(quad1, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB
plt.title("1")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(quad2, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB
plt.title("2")
plt.axis("off")

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(quad3, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB
plt.title("3")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(quad4, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB
plt.title("4")
plt.axis("off")

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
```

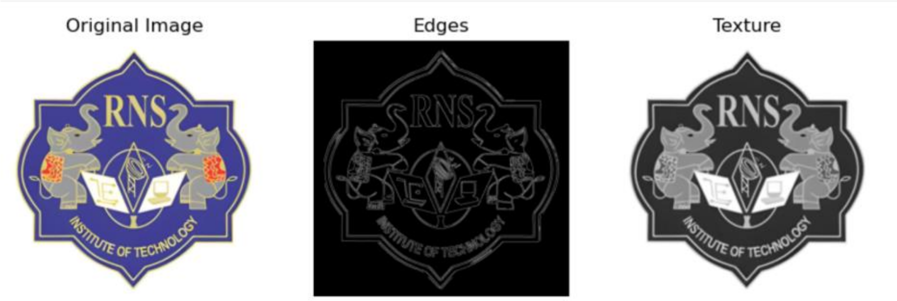
```
def translate_image(image, dx, dy):
    rows, cols = image.shape[:2]
    translation_matrix = np.float32([[1, 0, dx], [0, 1, dy]])
    translated_image = cv2.warpAffine(image, translation_matrix, (cols, rows))
    return translated_image

# Read the image
image = cv2.imread("rnsit.jpg")

# Convert the image from BGR to RGB for correct color display in matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Get image dimensions
height, width = image.shape[:2]

# Calculate the center coordinates of the image
center = (width // 2, height // 2)

# Get rotation and scaling values from the user
rotation_value = int(input("Enter the degree of Rotation (between -180 and 180): "))
while rotation_value < -180 or rotation_value > 180:
    rotation_value = int(input("Invalid input. Enter the degree of Rotation (between -180 and 180): "))

scaling_value = float(input("Enter the zooming factor (between 0.1 and 10): "))
while scaling_value < 0.1 or scaling_value > 10:
    scaling_value = float(input("Invalid input. Enter the zooming factor (between 0.1 and 10): "))

# Create the 2D rotation matrix
rotated = cv2.getRotationMatrix2D(center=center, angle=rotation_value, scale=1)
rotated_image = cv2.warpAffine(src=image, M=rotated, dsize=(width, height))

# Create the 2D scaling matrix
scaled = cv2.getRotationMatrix2D(center=center, angle=0, scale=scaling_value)
scaled_image = cv2.warpAffine(src=rotated_image, M=scaled, dsize=(width, height))

# Get translation values from the user
h = int(input("How many pixels you want the image to be translated horizontally? "))
v = int(input("How many pixels you want the image to be translated vertically? "))

# Translate the image
translated_image = translate_image(scaled_image, dx=h, dy=v)

# Convert the final transformed image from BGR to RGB
translated_image_rgb = cv2.cvtColor(translated_image, cv2.COLOR_BGR2RGB)

# Save the final transformed image
cv2.imwrite('Final_image.png', translated_image)

# Display the original and final transformed images using subplots
plt.figure(figsize=(10, 5))
```

```
# Display the original image
plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")

# Display the final transformed image
plt.subplot(1, 2, 2)
plt.imshow(translated_image_rgb)
plt.title("Final Transformed Image")
plt.axis("off")

plt.show()
```

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

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

# Load the image
image_path = "logo.jpeg"  # 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 using matplotlib
plt.figure(figsize=(10, 5))

plt.subplot(1, 3, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title("Edges")
plt.imshow(edges, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
```

```
plt.title("Texture")
plt.imshow(texture, cmap='gray')
plt.axis('off')

plt.show()
```

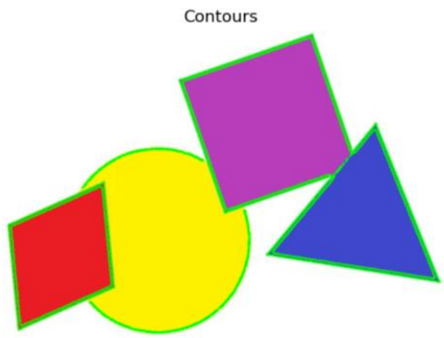


Lab program 10 Write a program to blur and smoothing an image.

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

# Load the image in grayscale
img = cv2.imread("rnsit.jpg", cv2.IMREAD_GRAYSCALE)

# Apply Gaussian blur (blurring)
gaussian_blur = cv2.GaussianBlur(img, (5, 5), 0)

# Apply bilateral filter (smoothening)
bilateral_blur = cv2.bilateralFilter(img, 9, 75, 75)

# Display the original, blurred, and smoothened images using matplotlib
plt.figure(figsize=(25, 10))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Grayscale Image")
plt.axis("off")

plt.subplot(1, 3, 2)
plt.imshow(gaussian_blur, cmap='gray')
plt.title("Blurred Image (Gaussian Blur)")
plt.axis("off")

plt.subplot(1, 3, 3)
plt.imshow(bilateral_blur, cmap='gray')
plt.title("Smoothened Image (Bilateral Filter)")
plt.axis("off")

plt.show()
```

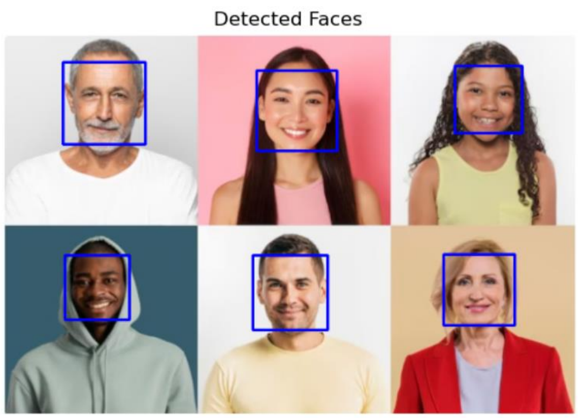
lab program 11 Write a program to contour an image.

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

# Read the image
image = cv2.imread('shapes.jpg')

# 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)

# Apply adaptive thresholding with adjusted parameters
thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 15, 4)

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

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

# Convert BGR image to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image with contours using matplotlib
plt.imshow(image_rgb)
plt.title('Contours')
plt.axis('off')  # Hide axis
plt.show()
```



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

```
import cv2
import matplotlib.pyplot as plt

# Load the pre-trained Haar Cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')

# Read the input image
image_path = 'face.jpg'
image = cv2.imread(image_path)

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

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

# Draw rectangles around detected faces
for (x, y, w, h) in faces:
   cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Convert BGR image to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image with detected faces using matplotlib
plt.imshow(image_rgb)
plt.title('Detected Faces')
plt.axis('off')  # Hide axis
plt.show()
```


Detected Faces