Arka Educational & Cultural Trust (Regd.)

Jain Institute of Technology, Davangere

(A UnitofJain Group ofInstitutions, Bengaluru)

# 323, Near Veereshwara Punyashrama, Avaragere, Davangere- 577005.

Department of Computer Science and Engineering

LAB MANUAL

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY – 21CSL66

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

```
#include <GL/glut.h>
#include <iostream>
using namespace std;

// Bresenham's line drawing algorithm
void drawLine(int x0, int y0, int x1, int y1) {
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int sx = (x0 < x1) ? 1 : -1;
    int sy = (y0 < y1) ? 1 : -1;
    int err = dx - dy;

    while (true) {
        glBegin(GL_POINTS);
        glVertex2i(x0, y0);
        glEnd();

        if (x0 == x1 && y0 == y1) break;

        int e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x0 += sx;
        }
        if (e2 < dx) {
            err += dx;
            y0 += sy;
        }
    }
}
```

```
// OpenGL display callback
void display() {
    int x1, x2, y1, y2;

    cout << "Enter coordinates for x1 and y1" << endl;
    cin >> x1 >> y1;
    cout << "Enter coordinates for x2 and y2" << endl;
    cin >> x2 >> y2;

    glClear(GL_COLOR_BUFFER_BIT);

    // Draw line using Bresenham's algorithm
    glColor3f(1.0f, 1.0f, 1.0f);
    drawLine(x1, y1, x2, y2);

    glFlush();
}

// OpenGL initialization
void initializeOpenGL(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Bresenham's Line Algorithm");

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 800, 0, 600);

    glutDisplayFunc(display);
}

// Main function
int main(int argc, char** argv) {

    initializeOpenGL(argc, argv);

    glutMainLoop();
    return 0;
}
```

Output:

```
Enter coordinates for x1 and y1
50 50
Enter coordinates for x1 and y1
500 500
```

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

```
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( )
```

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

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

Develop a program to demonstrate 2D transformation on basic objects Using opengl

```
#include "stdafx.h"
#include <GL/glut.h>
#include <stdio.h>

// Global variables
int width = 800;
int height = 600;
float rectWidth = 100.0f;
float rectHeight = 50.0f;
float rectPositionX = (width - rectWidth) / 2.0f;
float rectPositionY = (height - rectHeight) / 2.0f;
float rotationAngle = 0.0f;
float scaleFactor = 1.0f;

// Function to draw a rectangle
void drawRectangle(float x, float y, float width, float height) {
    glBegin(GL_POLYGON);
    glVertex2f(x, y);
    glVertex2f(x + width, y);
    glVertex2f(x + width, y + height);
    glVertex2f(x, y + height);
    glEnd();
}

// Function to handle display
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Apply transformations
    glTranslatef(rectPositionX, rectPositionY, 0.0f);
    glRotatef(rotationAngle, 0.0f, 0.0f, 1.0f);
    glScalef(scaleFactor, scaleFactor, 1.0f);

    // Draw rectangle
    glColor3f(1.0f, 0.0f, 0.0f); // Red color
    drawRectangle(0.0f, 0.0f, rectWidth, rectHeight);

    glFlush();
}
```

```
// Function to handle keyboard events
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 't':
            // Translate the rectangle by 10 units in the x-direction
            rectPositionX += 10.0f;
            break;
        case 'r':
            // Rotate the rectangle by 10 degrees clockwise
            rotationAngle += 10.0f;
            break;
        case 's':
            // Scale the rectangle by 10% (scaleFactor = 1.1f)
            scaleFactor *= 1.1f;
            break;
        case 'u':
            // Reset transformations (translate back to center, reset rotation and scaling)
            rectPositionX = (width - rectWidth) / 2.0f;
            rectPositionY = (height - rectHeight) / 2.0f;
            rotationAngle = 0.0f;
            scaleFactor = 1.0f;
            break;
        case 27: // Escape key to exit
            exit(0);
            break;
    }

    glutPostRedisplay(); // Trigger a redraw
}

// Function to initialize OpenGL
void initializeOpenGL(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(width, height);
    glutCreateWindow("Geometric Operations in 2D");

    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // White background
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, width, 0, height);

    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
}
```

```
// Main function
int main(int argc, char** argv) {
    initializeOpenGL(argc, argv);
    glutMainLoop();
    return 0;
}
```

Develop a program to demonstrate 3D transformation on 3D objects Using opengl

```
#include "stdafx.h"
#include <GL/glut.h>
#include <stdio.h>

// Global variables
int width = 800;
int height = 600;
GLfloat rotationX = 0.0f;
GLfloat rotationY = 0.0f;
GLfloat scale = 1.0f;

// Function to draw a cube
void drawCube() {
    glBegin(GL_QUADS);

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

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

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

    // Bottom face
    glColor3f(1.0f, 1.0f, 0.0f); // Yellow
    glVertex3f(-0.5f, -0.5f, -0.5f);
    glVertex3f(0.5f, -0.5f, -0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);
    glVertex3f(-0.5f, -0.5f, 0.5f);
    // Right face
    glColor3f(1.0f, 0.0f, 1.0f); // Magenta
    glVertex3f(0.5f, -0.5f, -0.5f);
    glVertex3f(0.5f, 0.5f, -0.5f);
    glVertex3f(0.5f, 0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);

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

    glEnd();
}

// Function to handle display
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Apply transformations
    glTranslatef(0.0f, 0.0f, -3.0f);
    glRotatef(rotationX, 1.0f, 0.0f, 0.0f);
    glRotatef(rotationY, 0.0f, 1.0f, 0.0f);
    glScalef(scale, scale, scale);

    // Draw cube
    drawCube();

    glutSwapBuffers();
}

// Function to handle keyboard events
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 'x':
            rotationX += 5.0f;
            break;
        case 'X':
            rotationX -= 5.0f;
            break;
        case 'y':
            rotationY += 5.0f;
            break;
        case 'Y':
            rotationY -= 5.0f;
            break;
        case '+':
            scale += 0.1f;
            break;
        case '-':
            if (scale > 0.1f)
                scale -= 0.1f;
            break;
        case 27: // Escape key to exit
            exit(0);
            break;
    }

    glutPostRedisplay(); // Trigger a redraw
}

// Function to initialize OpenGL
void initializeOpenGL(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutCreateWindow("Geometric Operations in 3D");

    glEnable(GL_DEPTH_TEST);
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // White background

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, (float)width / (float)height, 1.0f, 100.0f);

    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
}

// Main function
int main(int argc, char** argv) {
    initializeOpenGL(argc, argv);
    glutMainLoop();
    return 0;
}
```

Develop a program to demonstrate Animation effects on simple objects.using openGL

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

const double TWO_PI = 6.2831853;
GLsizei winWidth = 500, winHeight = 500;
GLuint regHex;
static GLfloat rotTheta = 0.0;

// Initial display window size.
// Define name for display list.
class scrPt {
        public:
        GLint x, y;
};

static void init(void)
{
        scrPt hexVertex;
        GLdouble hexTheta;
        GLint k;
        glClearColor(1.0, 1.0, 1.0, 0.0);
        /* Set up a display list for a red regular hexagon.
        * Vertices for the hexagon are six equally spaced
        * points around the circumference of a circle.
        */
        regHex = glGenLists(1);
        glNewList(regHex, GL_COMPILE);
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_POLYGON);
        for(k = 0; k < 6; k++) {
                hexTheta = TWO_PI * k / 6;
                hexVertex.x = 150 + 100 * cos(hexTheta);
                hexVertex.y = 150 + 100 * sin(hexTheta);
                glVertex2i(hexVertex.x, hexVertex.y);
        }
        glEnd( );
```

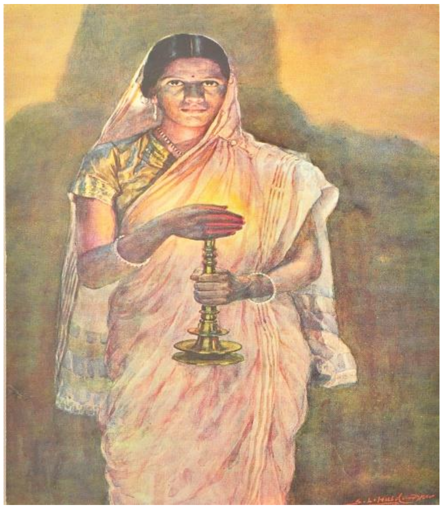
```
    glEndList( );
}

void displayHex(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix( );
    glRotatef(rotTheta, 0.0, 0.0, 1.0);
    glCallList(regHex);
    glPopMatrix( );
    glutSwapBuffers( );
    glFlush( );
}

void rotateHex(void)
{
    rotTheta += 3.0;
    if(rotTheta > 360.0)
    rotTheta -= 360.0;
    glutPostRedisplay( );
}
void winReshapeFcn(GLint newWidth, GLint newHeight)
{
    glViewport(0, 0,(GLsizei) newWidth,(GLsizei) newHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(-320.0, 320.0, -320.0, 320.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity( );
    glClear(GL_COLOR_BUFFER_BIT);
}
void mouseFcn(GLint button, GLint action, GLint x, GLint y)
{
    switch(button) {
        case GLUT_MIDDLE_BUTTON:
            // Start the rotation.
            if(action == GLUT_DOWN)
                glutIdleFunc(rotateHex);
            break;
        case GLUT_RIGHT_BUTTON:
```

```
                        // Stop the rotation.
                        if(action == GLUT_DOWN)
                                glutIdleFunc(NULL);
                        break;
                default:
                        break;
        }
}

int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowPosition(150, 150);
        glutInitWindowSize(winWidth, winHeight);
        glutCreateWindow("Animation Example");
        init( );
        glutDisplayFunc(displayHex);
        glutReshapeFunc(winReshapeFcn);
        glutMouseFunc(mouseFcn);
        glutMainLoop( );
        return 0;
}
```

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

```
import cv2

# Function to split the image into four quadrants
def split_image(image):
    height, width, _ = image.shape
    half_height = height // 2
    half_width = width // 2

    # Split the image into four quadrants
    top_left = image[:half_height, :half_width]
    top_right = image[:half_height, half_width:]
    bottom_left = image[half_height:, :half_width]
    bottom_right = image[half_height:, half_width:]

    return top_left, top_right, bottom_left, bottom_right

# Function to display images
def display_images(images, window_names):
    for img, name in zip(images, window_names):
        cv2.imshow(name, img)

    print("Press any key to terminate.")
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Read the image
image_path = "image.jpg"  # Replace "image.jpg" with the path to your image
image = cv2.imread(image_path)

if image is None:
    print("Failed to load the image.")
else:
    # Split the image into quadrants
    top_left, top_right, bottom_left, bottom_right = split_image(image)

    # Display the quadrants
    display_images([top_left, top_right, bottom_left, bottom_right], ["Top Left", "Top Right",
"Bottom Left", "Bottom Right"])
```

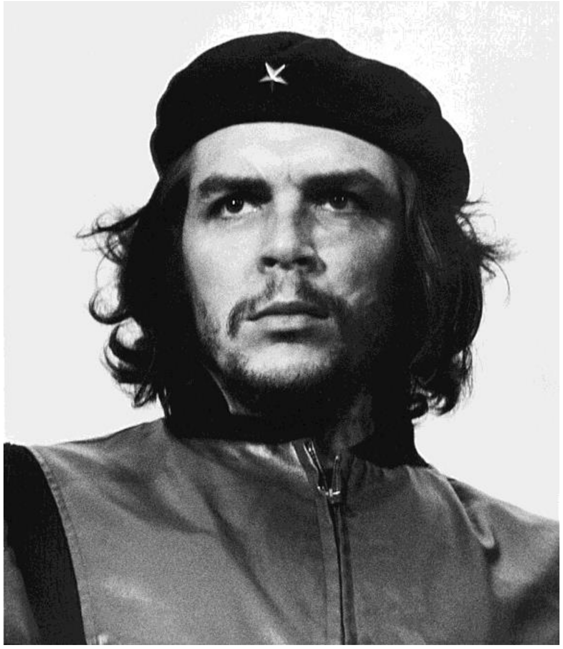
Input image



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

```
import cv2
import numpy as np

# Read the image
image_path = "Che.jpg"  # Replace "your_image.jpg" with the path to your image
image = cv2.imread(image_path)

if image is None:
    print("Failed to load the image.")
else:
    # Display the original image
    cv2.imshow("Original Image", image)

    # Rotation
    angle = 45  # Rotation angle in degrees
    center = (image.shape[1] // 2, image.shape[0] // 2)  # Center of rotation
    rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)  # Rotation matrix
    rotated_image = cv2.warpAffine(image, rotation_matrix, (image.shape[1], image.shape[0]))

    # Scaling
    scale_factor = 0.5  # Scaling factor (0.5 means half the size)
    scaled_image = cv2.resize(image, None, fx=scale_factor, fy=scale_factor)

    # Translation
    translation_matrix = np.float32([[1, 0, 100], [0, 1, -50]])  # Translation matrix (100 pixels right, 50 pixels up)
    translated_image = cv2.warpAffine(image, translation_matrix, (image.shape[1], image.shape[0]))

    # Display the transformed images
    cv2.imshow("Rotated Image", rotated_image)
    cv2.imshow("Scaled Image", scaled_image)
    cv2.imshow("Translated Image", translated_image)

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

Input image



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

```
import cv2
import numpy as np

# Read the image
image_path = "gandhi.jpg"  # Replace "your_image.jpg" with the path to your image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

if image is None:
    print("Failed to load the image.")
else:
    # Display the original image
    cv2.imshow("Original Image", image)

    # Apply Sobel filter to extract edges
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    sobel_edges = cv2.magnitude(sobel_x, sobel_y)
    sobel_edges = cv2.normalize(sobel_edges, None, 0, 255, cv2.NORM_MINMAX,
dtype=cv2.CV_8U)

    # Display edges extracted using Sobel filter
    cv2.imshow("Edges (Sobel Filter)", sobel_edges)

    # Apply Laplacian filter to extract edges
    laplacian_edges = cv2.Laplacian(image, cv2.CV_64F)
    laplacian_edges = cv2.normalize(laplacian_edges, None, 0, 255, cv2.NORM_MINMAX,
dtype=cv2.CV_8U)

    # Display edges extracted using Laplacian filter
    cv2.imshow("Edges (Laplacian Filter)", laplacian_edges)

    # Apply Gaussian blur to extract textures
    gaussian_blur = cv2.GaussianBlur(image, (5, 5), 0)

    # Display image with Gaussian blur
    cv2.imshow("Gaussian Blur", gaussian_blur)

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

Input image

Write a program to blur and smoothing an image.

```
import cv2

# Read the image
image_path = "art.png"  # Replace "your_image.jpg" with the path to your image
image = cv2.imread(image_path)

if image is None:
    print("Failed to load the image.")
else:
    # Display the original image
    cv2.imshow("Original Image", image)

    # Apply blur to the image
    blur_kernel_size = (5, 5)  # Kernel size for blur filter
    blurred_image = cv2.blur(image, blur_kernel_size)

    # Display the blurred image
    cv2.imshow("Blurred Image", blurred_image)

    # Apply Gaussian blur to the image
    gaussian_blur_kernel_size = (5, 5)  # Kernel size for Gaussian blur filter
    gaussian_blurred_image = cv2.GaussianBlur(image, gaussian_blur_kernel_size, 0)

    # Display the Gaussian blurred image
    cv2.imshow("Gaussian Blurred Image", gaussian_blurred_image)

    # Apply median blur to the image
    median_blur_kernel_size = 5  # Kernel size for median blur filter (should be odd)
    median_blurred_image = cv2.medianBlur(image, median_blur_kernel_size)

    # Display the median blurred image
    cv2.imshow("Median Blurred Image", median_blurred_image)

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

Input image

Write a program to contour an image.

```
import cv2

# Read the image
image_path = "annavru.jpeg"  # Replace "your_image.jpg" with the path to your image
image = cv2.imread(image_path)

if image is None:
    print("Failed to load the image.")
else:
    # Convert the image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply adaptive thresholding
    _, thresh = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

    # Find contours in the thresholded image
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Draw contours on the original image
    contour_image = image.copy()
    cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)  # Draw all contours with green
color and thickness 2

    # Display the original image with contours
    cv2.imshow("Image with Contours", contour_image)

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

Input image

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

```
import cv2

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

# Read the image
image_path = "ucl.png"  # Replace "ucl.png" with the path to your image
image = cv2.imread(image_path)

if image is None:
    print("Failed to load the image.")
else:
    # Convert the image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Detect faces in the image
    faces = face_cascade.detectMultiScale(gray_image, 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("Image with Detected Faces", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Input image

unicef