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

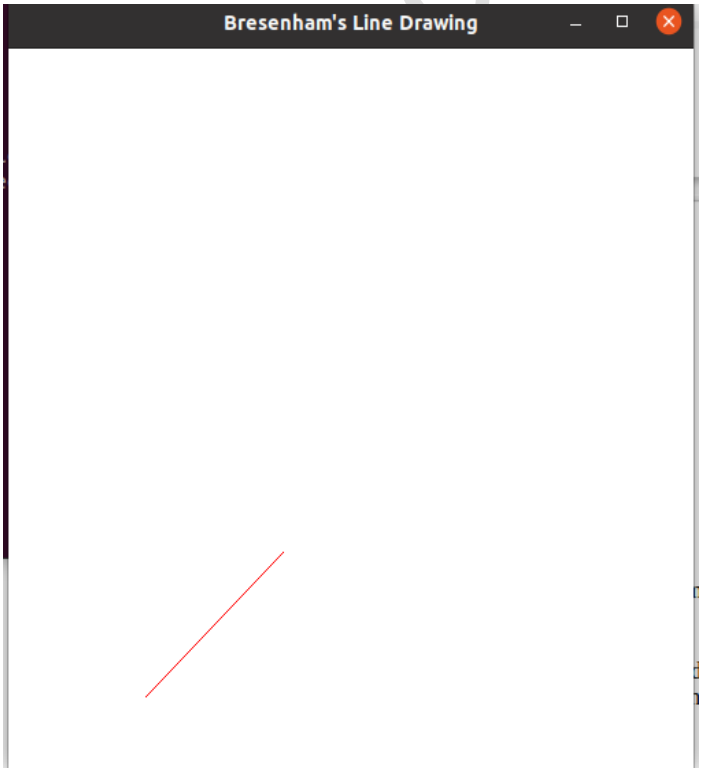
```
#include<GL/glut.h>
#include<stdio.h>
int x1, y1, x2, y2;
void draw_pixel(int x, int y)
{
 glColor3f(1.0,0.0,0.0);
 glBegin(GL_POINTS);
 glVertex2i(x, y);
 glEnd();
}
void bresenhams_line_draw(int x1, int y1, int x2, int y2)
{
 int dx = x2 - x1; // x difference
 int dy = y2 - y1; // y difference
 int m = dy/dx; // slope
 if (m < 1)
 {
 int decision_parameter = 2*dy - dx;
 int x = x1; // initial x
 int y = y1; // initial y
 if (dx < 0) // decide the first point and second point
 {
 x = x2; // making second point as first point
 y = y2;
 x2 = x1;
 }
 draw_pixel (x, y); // plot a point
 while (x < x2) // from 1st point to 2nd point
 {
 if (decision_parameter >= 0)
 {
 x = x+1;
 y = y+1;
 decision_parameter = decision_parameter + 2*dy - 2*dx * (y+1 - y);
 }
 else
 {
 x = x+1;
 y = y;
 decision_parameter = decision_parameter + 2*dy - 2*dx * (y - y);
 }
 draw_pixel (x, y);
 }
 }
```

```
 else if (m > 1)
 {
 int decision_parameter = 2*dx - dy;
 int x = x1; // initial x
 int y = y1; // initial y
 if (dy < 0)
 {
 x = x2;
 y = y2;
 y2 = y1;
 }
 draw_pixel (x, y);
 while (y < y2)
 {
 if (decision_parameter >= 0)
 {
 x = x+1;
 y = y+1;
 decision_parameter = decision_parameter + 2*dx - 2*dy * (x+1 - x);
 }
 else
 {
 y = y+1;
 x = x;
 decision_parameter = decision_parameter + 2*dx - 2*dy * (x- x);
 }
 draw_pixel(x, y);
 }
 }
 else if (m == 1)
 {
 int x = x1;
 int y = y1;
 draw_pixel (x, y);
 while (x < x2)
 {
 x = x+1;
 y = y+1;
 draw_pixel (x, y);
 }
 }
}
void init()
{
 glClearColor(1,1,1,1);
 gluOrtho2D(0.0, 500.0, 0.0, 500.0); // left ->0, right ->500, bottom ->0, top ->500
}
void display()
```

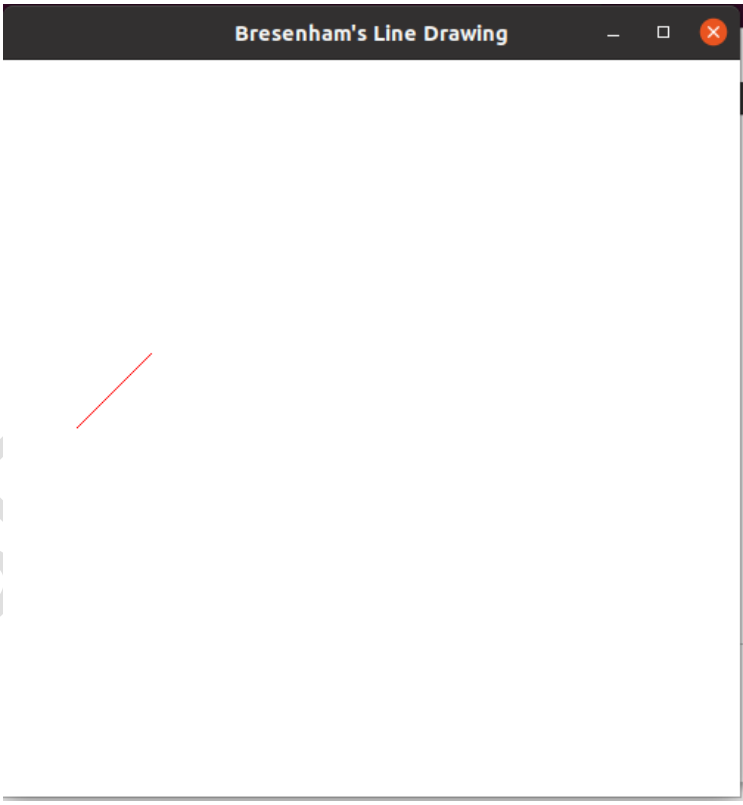
```
{
 glClear(GL_COLOR_BUFFER_BIT);
 bresenhams_line_draw(x1, y1, x2, y2);
 glFlush();
}
int main(int argc, char **argv)
{
 printf( "Enter Start Points (x1,y1)\n");
 scanf("%d %d", &x1, &y1); // 1st point from user
 printf( "Enter End Points (x2,y2)\n");
 scanf("%d %d", &x2, &y2); // 2nd point from user
 glutInit(&argc, argv); // initialize graphics system
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); //single buffered mode with RGB colour
variants
 glutInitWindowSize(500, 500); // 500 by 500 window size
 glutInitWindowPosition(220, 200); // where do you wanna see your window
 glutCreateWindow("Bresenham's Line Drawing"); // the title of your window
 init(); // initialize the canvas
 glutDisplayFunc(display); // call display function
 glutMainLoop(); // run forever
}
```

Output

```
Enter Start Points (x1,y1)
200 100
Enter End Points (x2,y2)
100 50
```



```
Enter Start Points (x1,y1)
100 200
Enter End Points (x2,y2)
50 250
```



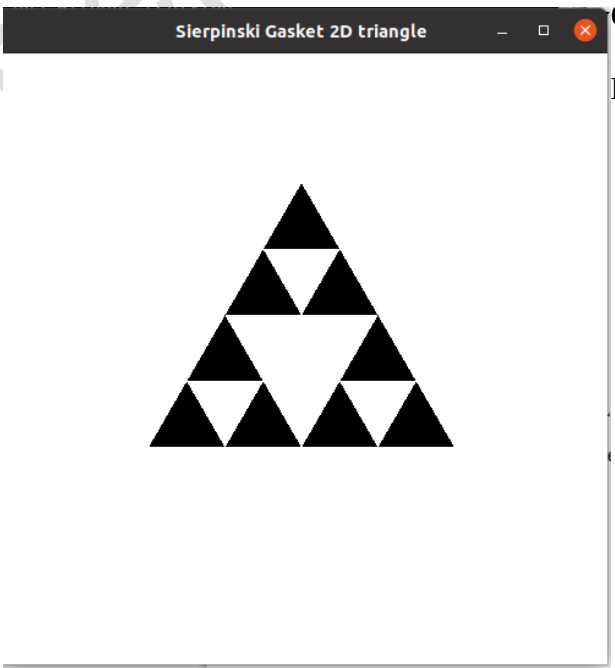
Program 2

Develop a program to demonstrate basic 3D Geometrical operation on 2D object.

```
#include <stdio.h>
#include <GL/glut.h>
typedef float point2[2];

/* initial triangle */

point2 v[]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};
int n;

/* display one triangle  */
void triangle( point2 a, point2 b, point2 c)
{
   glBegin(GL_TRIANGLES);
      glVertex2fv(a);
      glVertex2fv(b);
      glVertex2fv(c);
   glEnd();
}

void divide_triangle(point2 a, point2 b, point2 c, int m)
{
/* triangle subdivision using vertex numbers */

   point2 v0, v1, v2;
   int j;
   if(m>0)
   {
      for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
      for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
      for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
      divide_triangle(a, v0, v1, m-1);
      divide_triangle(c, v1, v2, m-1);
      divide_triangle(b, v2, v0, m-1);
   }
   else(triangle(a,b,c));    /* draw triangle at end of recursion */
}
```

```
void display(void)
{

   glClear(GL_COLOR_BUFFER_BIT);
   divide_triangle(v[0], v[1], v[2], n);
   glFlush();
}

void myinit()
{

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
   glMatrixMode(GL_MODELVIEW);
   glClearColor (1.0, 1.0, 1.0, 1.0);
   glColor3f(0.0,0.0,0.0);
}

int main(int argc, char **argv)
{
  printf(" No. of Subdivisions : ");
  scanf("%d",&n);
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
   glutInitWindowSize(500, 500);
   glutCreateWindow("Sierpinski Gasket 2D triangle");
   glutDisplayFunc(display);
   myinit();
   glutMainLoop();
   return 0;
}
```


Output:



No. of Subdivisions : 2

Program 3

Develop a program to demonstrate basic Geometric operation on 3D object.

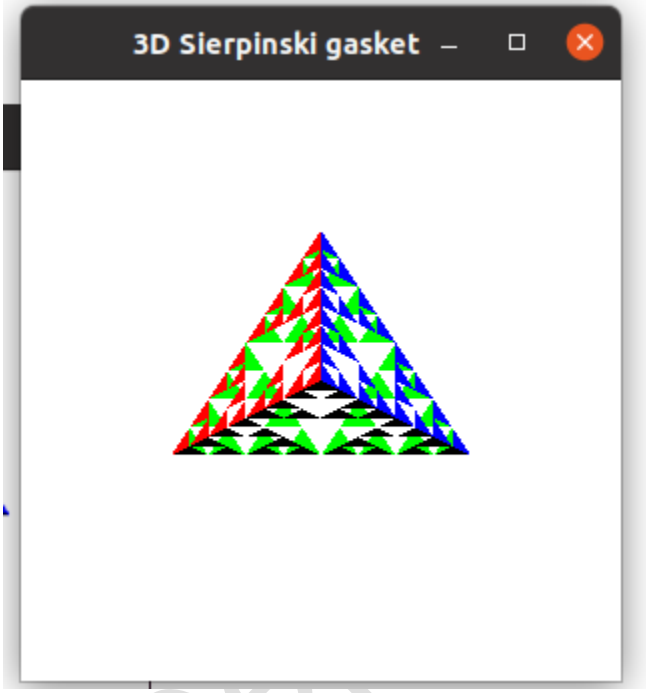
```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
typedef float point[3];
point v[]= {{0, 0, 1}, {0, 1, 0}, {-1, -0.5, 0}, {1, -0.5, 0}};
int n;
void triangle(point a, point b, point c)
{
 glBegin(GL_POLYGON);
 glVertex3fv(a);
 glVertex3fv(b);
 glVertex3fv(c);
 glEnd();
}
void divide_triangle(point a, point b, point c, int n)
{
 point v1,v2,v3;
 int j;
 if(n>0)
 {
 for(j=0; j<3; j++)
 v1[j] = (a[j]+b[j])/2; // calculate mid-point between a and b
 for(j=0; j<3; j++)
 v2[j] = (a[j]+c[j])/2; // calculate mid-point between a and c
 for(j=0; j<3; j++)
 v3[j] = (c[j]+b[j])/2; // calculate mid-point between c and b
 divide_triangle(a,v1,v2,n-1); // divide triangle between points a, ab/2, ac/2 recursively
divide_triangle(c,v2,v3,n-1);
 divide_triangle(b,v3,v1,n-1);
 }
 else
triangle (a,b,c);// draw triangle
}
void tetrahedron(int n)
{
 glColor3f(1, 0, 0); // assign color for each of the side
```

```
 divide_triangle(v[0], v[1], v[2], n); // draw triangle between a, b, c
 glColor3f(0, 1, 0);
 divide_triangle(v[3], v[2], v[1], n);
 glColor3f(0, 0, 1);
 divide_triangle(v[0], v[3], v[1], n);
 glColor3f(0, 0, 0);
 divide_triangle(v[0], v[2], v[3], n);
}
void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glLoadIdentity();
 tetrahedron(n);
 glFlush(); // show the output
}
void myReshape(int w,int h) // please see the earlier program for explanation on this
{
 glViewport(0, 0, w, h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 if(w<=h)
 glOrtho(-2, 2, -2*(GLfloat)h/(GLfloat)w, 2*(GLfloat)h/(GLfloat)w, -10, 10);
 else
 glOrtho(-2*(GLfloat)w/(GLfloat)h, 2*(GLfloat)w/(GLfloat)h, -2, 2, -10, 10);
 glMatrixMode(GL_MODELVIEW);
 glutPostRedisplay();
}
int main(int argc,char ** argv)
{
 printf("No of Recursive steps/Division: ");
 scanf("%d",&n);
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
 glutCreateWindow(" 3D Sierpinski gasket");
 glutReshapeFunc(myReshape);
```


```
 glutDisplayFunc(display); // call display function
 glEnable(GL_DEPTH_TEST); // do depth
comparisons and update the depth buffer.
 glClearColor(1, 1, 1, 0);
 glutMainLoop();
 return 0;
}
```

Output
No of Recursive steps/Division:1

No of Recursive steps/Division:3



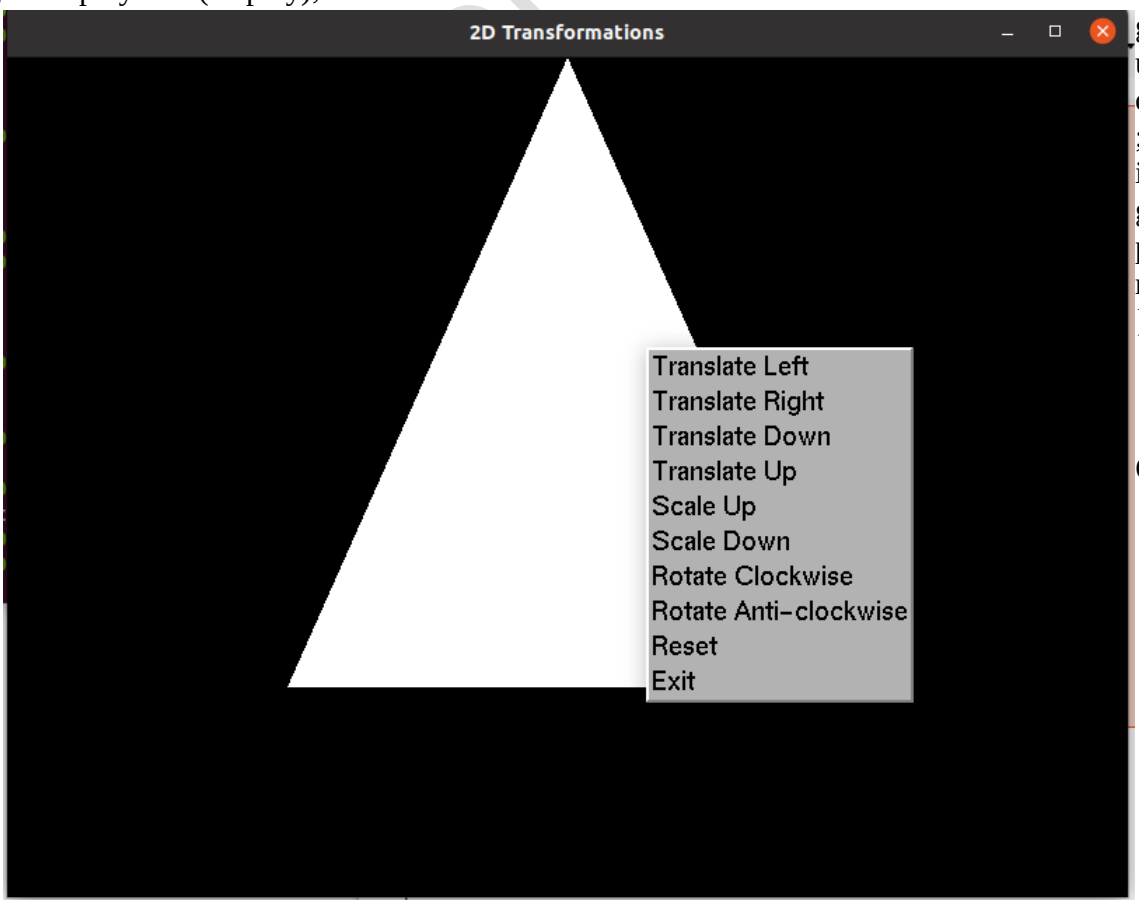
Program 4

Develop a program to demostrate 2D transformations on basic object.

```
#include<GL/glut.h>
#include<math.h>
GLfloat vertices[][2]={
{0.1},{-0.5,-0.5},{0.5,-0.5}
};
GLfloat angle=0.0;
GLfloat transalateX=0.0;
GLfloat transalateY=0.0;
GLfloat scaleX=1.0;
GLfloat scaleY=1.0;
void display ();
void menu(int option);
void createMenu();
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
glTranslatef(transalateX,transalateY,0);
glRotatef(angle,0,0,1);
glScalef(scaleX,scaleY,1);
glColor3f(1,1,1);
glBegin(GL_TRIANGLES);
for (int i=0; i<3; i++)
{
glVertex2fv(vertices[i]);
}
glEnd();
glFlush();
}
void menu (int option)
{
```

```
switch(option)
{
case 1:
transalateX-=0.1;
break;
case 2:
transalateX+=0.1;
break;
case 3:
transalateY-=0.1;
break;
case 4:
transalateY+=0.1;
break;
case 5:
scaleX+=0.1;
scaleY+=0.1;
break;
case 6:
scaleX-=0.1;
scaleY-=0.1;
break;
case 7:
angle+=10.0;
if (angle>360)angle-=360;
break;
case 8:
angle-=10.0;
if (angle<0)angle+=360;
break;
case 9:
angle=0.0;
transalateX=0.0;
transalateY=0.0;
scaleX=1.0;
scaleY=1.0;
break;
case 10:
exit(0);
break;
}
glutPostRedisplay();
}
void createMenu()
{
glutAddMenuEntry("Translate left",1);
glutAddMenuEntry("Translate right",2);
glutAddMenuEntry("Translate Down",3);
```

```
glutAddMenuEntry("Translate Up",4);
glutAddMenuEntry("Scale Up",5);
glutAddMenuEntry("Scale Down",6);
glutAddMenuEntry("Rotate Clockwise",7);
glutAddMenuEntry("Rotate Anticlockwise",8);
glutAddMenuEntry("Reset",9);
glutAddMenuEntry("Exit",10);
glutAttachMenu(GLUT_RIGHT_BUTTON);
}
void init()
{
glClearColor(0,0,0,1);
}
void reshape(int w, int h)
{
glViewport (0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-1,1,-1,1);
glMatrixMode(GL_MODELVIEW);
}
int main( int argc,char**argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(800,600);
glutCreateWindow("2D Transformation");
glutDisplayFunc(display);
```




```
glutReshapeFunc(reshape);
createMenu();
init();
glutMainLoop();
return 0;
}
```

Output

Program 5

Develop a program to demostrate 3D transformations on basic object.

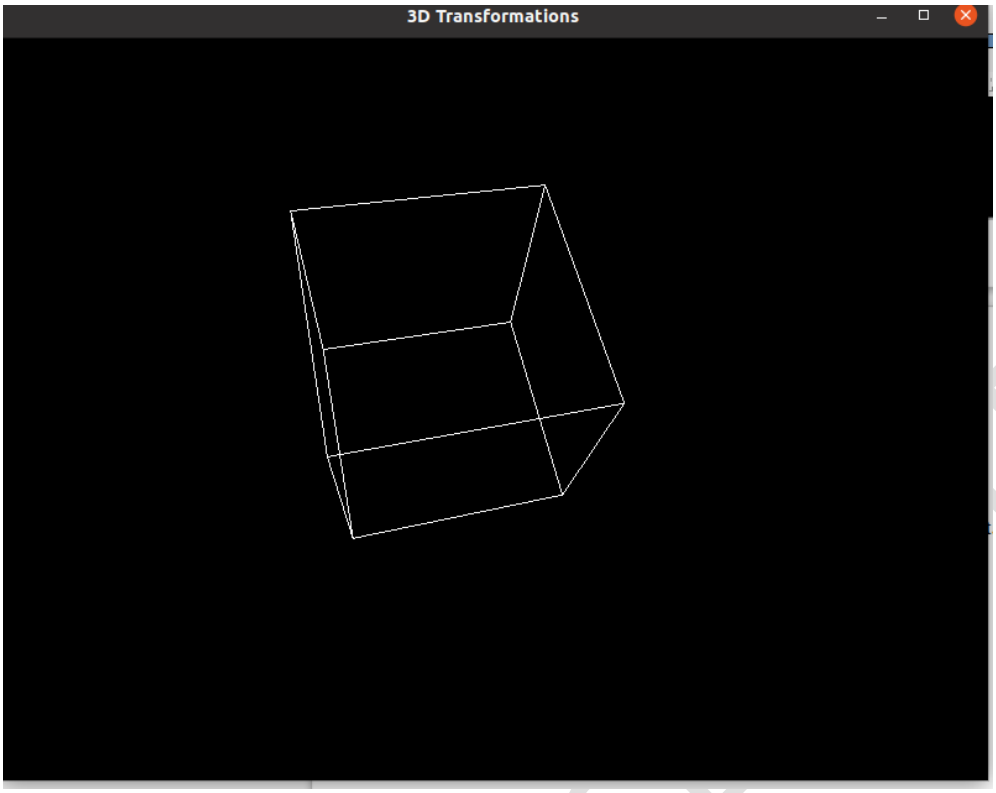
```
#include <stdlib.h>
#include <GL/glut.h>

// Define cube vertices
GLfloat vertices[][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}
};

// Define cube edges
GLint edges[][2] = {
    {0, 1},
    {1, 2},
    {2, 3},
    {3, 0},
    {4, 5},
    {5, 6},
```

```
    {6, 7},
    {7, 4},
    {0, 4},
    {1, 5},
    {2, 6},
    {3, 7}
};

// Define rotation angles
GLfloat angleX = 0.0;
GLfloat angleY = 0.0;
GLfloat angleZ = 0.0;

// Define translation offsets
GLfloat translateX = 0.0;
GLfloat translateY = 0.0;
GLfloat translateZ = 0.0;

// Define scaling factors
GLfloat scaleX = 1.0;
GLfloat scaleY = 1.0;
GLfloat scaleZ = 1.0;

// Display function
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    // Set up perspective projection
    gluLookAt(3, 3, 3, 0, 0, 0, 0, 1, 0);

    // Apply transformations
    glTranslatef(translateX, translateY, translateZ);
    glRotatef(angleX, 1, 0, 0);
    glRotatef(angleY, 0, 1, 0);
    glRotatef(angleZ, 0, 0, 1);
    glScalef(scaleX, scaleY, scaleZ);

    // Draw cube
    glColor3f(1, 1, 1);
    glBegin(GL_LINES);
    for (int i = 0; i < 12; i++) {
        glVertex3fv(vertices[edges[i][0]]);
        glVertex3fv(vertices[edges[i][1]]);
    }
    glEnd();

    glutSwapBuffers();
```

```
}

// Idle function
void idle() {
    angleX += 0.5;
    if (angleX > 360) angleX -= 360;
    angleY += 0.5;
    if (angleY > 360) angleY -= 360;
    angleZ += 0.5;
    if (angleZ > 360) angleZ -= 360;
    glutPostRedisplay();
}

// Keyboard function for scaling and translation
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 'w':
            translateY += 0.1;
            break;
        case 's':
            translateY -= 0.1;
            break;
        case 'a':
            translateX -= 0.1;
            break;
        case 'd':
            translateX += 0.1;
            break;
        case 'q':
            translateZ += 0.1;
            break;
        case 'e':
            translateZ -= 0.1;
            break;
        case '+':
            scaleX += 0.1;
            scaleY += 0.1;
            scaleZ += 0.1;
            break;
        case '-':
            scaleX -= 0.1;
            scaleY -= 0.1;
            scaleZ -= 0.1;
            break;
        case 'r':
            angleX = 0.0;
            angleY = 0.0;
            angleZ = 0.0;
```

```
            translateX = 0.0;
            translateY = 0.0;
            translateZ = 0.0;
            scaleX = 1.0;
            scaleY = 1.0;
            scaleZ = 1.0;
            break;
        case 27: // ESC key
            exit(0);
            break;
    }
    glutPostRedisplay();
}

// Initialization function
void init() {
    glClearColor(0, 0, 0, 1);
    glEnable(GL_DEPTH_TEST);
}

// Reshape function
void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, (float)w / h, 1, 100);
    glMatrixMode(GL_MODELVIEW);
}

// Main function
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("3D Transformations");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(idle);
    glutKeyboardFunc(keyboard);
    init();
    glutMainLoop();
    return 0;
}
```

Output



Program 6

Develop a program to demonstrate animation effects on simple objects

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

int windowWidth = 800;
int windowHeight = 600;
int squareSize = 50;
int squarePosX = 0;
float animationSpeed = 1.0;

void init() {
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, windowWidth, 0.0, windowHeight);
}

void drawSquare() {
```

```
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_QUADS);
    glVertex2i(squarePosX, windowHeight / 2);
    glVertex2i(squarePosX + squareSize, windowHeight / 2);
    glVertex2i(squarePosX + squareSize, windowHeight / 2 + squareSize);
    glVertex2i(squarePosX, windowHeight / 2 + squareSize);
    glEnd();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    drawSquare();

    glutSwapBuffers();
}

void update(int value) {
    // Move the square to the right
    squarePosX += animationSpeed;

    // If the square moves out of the screen, reset its position
    if (squarePosX > windowWidth) {
        squarePosX = -squareSize; // Start from the left edge again
    }

    // Varying animation speed
    animationSpeed += 0.01; // Increase animation speed linearly

    glutPostRedisplay(); // Update the display
    glutTimerFunc(1000 / 60, update, 0); // 60 frames per second
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Animation Effects with Varying Speed");
    init();
    glutDisplayFunc(display);
    glutTimerFunc(0, update, 0);
    glutMainLoop();
    return 0;
}
```

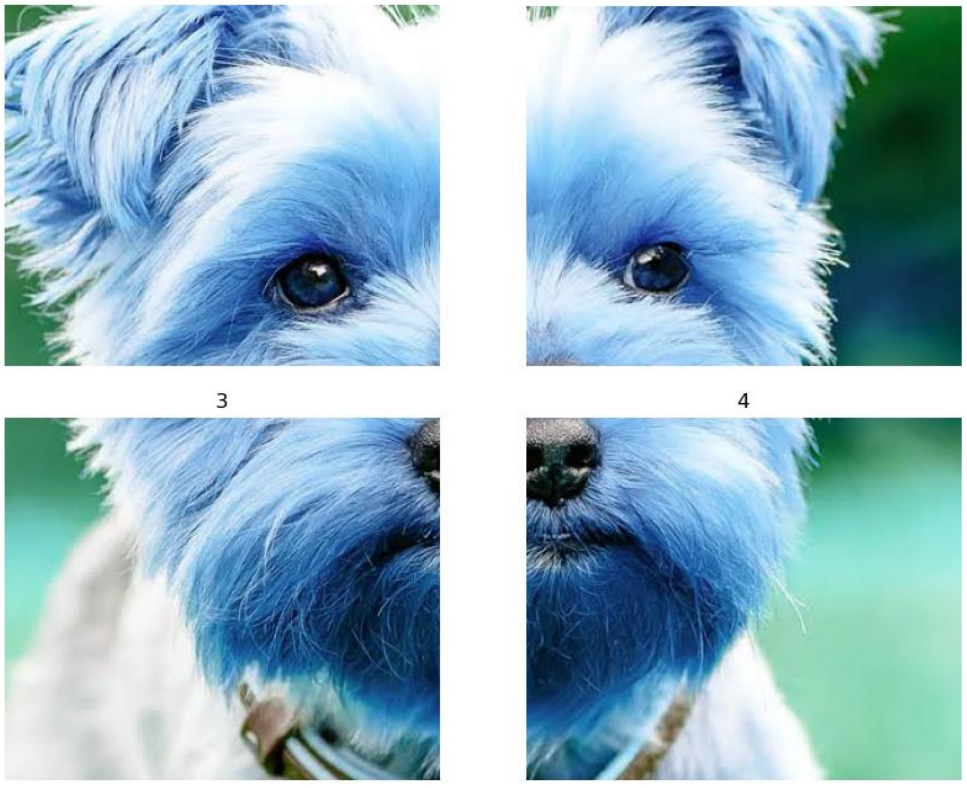
Program 7

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

# Read the image
img = cv2.imread("image_pat.jpeg")

# 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]
quad2 = img[:height//2, width//2:]
quad3 = img[height//2:, :width//2]
quad4 = img[height//2:, width//2:]
plt.figure(figsize=(10, 5))
```

```
plt.subplot(1, 2, 1)
plt.imshow(quad1)
plt.title("1")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(quad2)
plt.title("2")
plt.axis("off")

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(quad3)
plt.title("3")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(quad4)
plt.title("4")
plt.axis("off")

plt.show()
```


Output

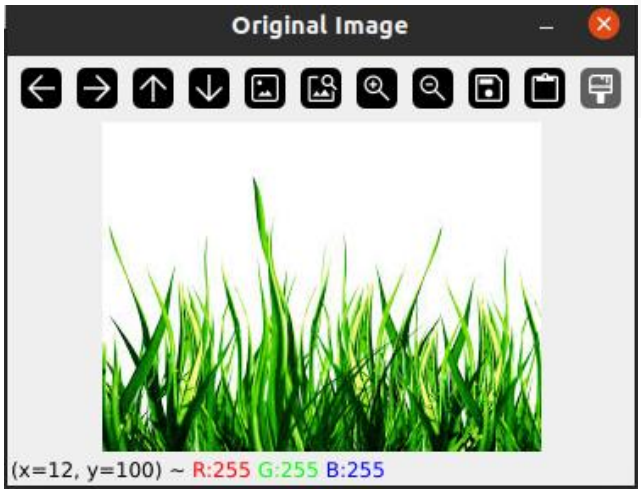


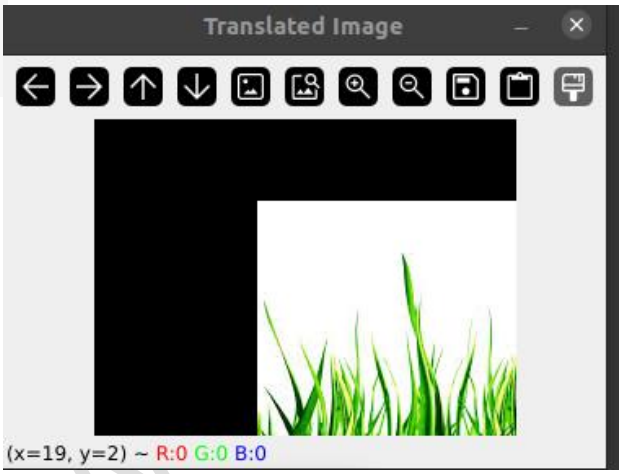
PROGRAM 8

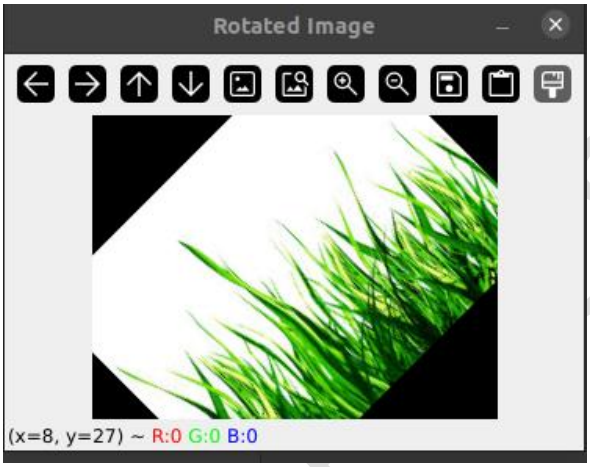
Write a program to show rotation, scaling, and translation on an image.

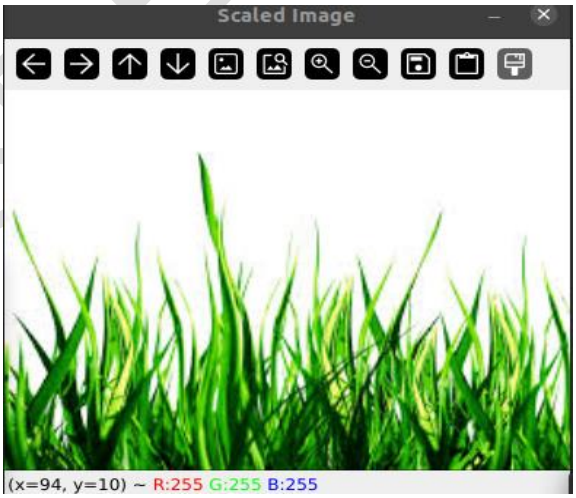
```
import cv2
import numpy as np

# Load the image
image_path = "grass.jpeg"  # 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()
```

Output









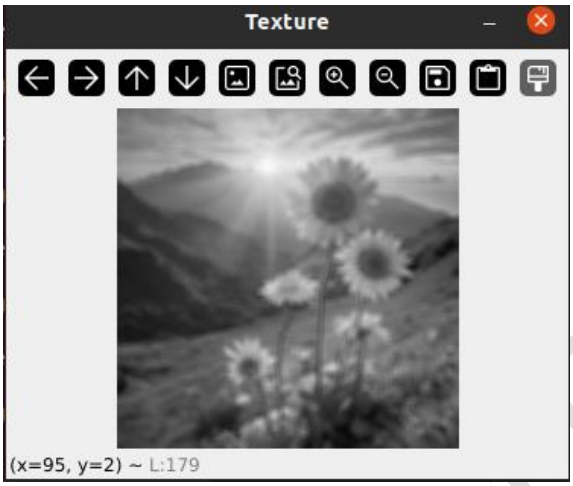
Program 9

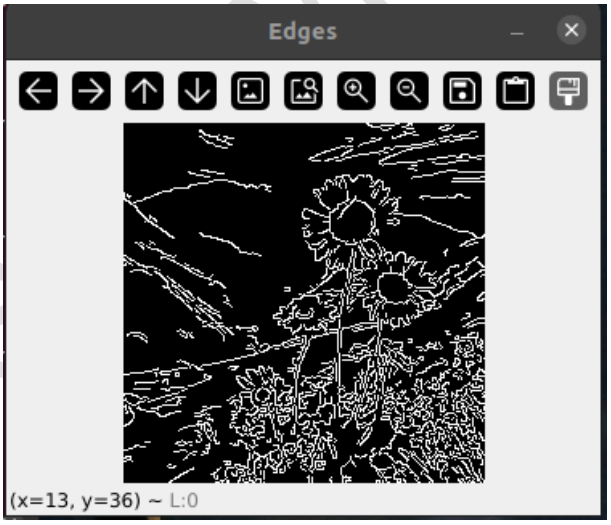
Read an image and extract and display low-level features such as edges, textures using filtering techniques.

```
import cv2
import numpy as np
```

```
# Load the image
image_path = "image/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()
```


Output





Program 10

Write a program to blur and smoothing an image.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
img = cv2.imread("tiger.jpeg",cv2.IMREAD_GRAYSCALE)
image_array = np.array(img)
print(image_array)
def sharpen():
  return np.array([[1,1,1],[1,1,1],[1,1,1]])
def filtering(image, kernel):
    m, n = kernel.shape
    if (m == n):
        y, x = image.shape
        y = y - m + 1 # shape of image - shape of kernel + 1
        x = x - m + 1
        new_image = np.zeros((y,x))
        for i in range(y):
            for j in range(x):
                new_image[i][j] = np.sum(image[i:i+m, j:j+m]*kernel)
    return new_image
# Display the original and sharpened images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_array,cmap='gray')
plt.title("Original Grayscale Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(filtering(image_array, sharpen()),cmap='gray')
plt.title("Blurred Image")
plt.axis("off")
plt.show()
```

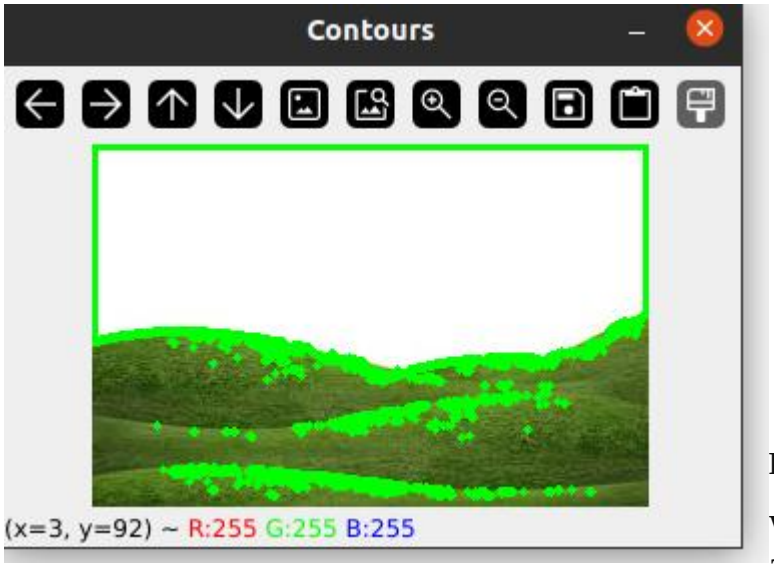

OUTPUT

Program 11

Write a program to contour an image.

```
import cv2
import numpy as np


image_path = '1.png'
image = cv2.imread(image_path)

# Convert the image to grayscale (contours work best on binary images)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding (you can use other techniques like Sobel edges)
_, binary_image = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Find contours
contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Draw all contours on the original image
cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

# Display the result
cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output



Program 12

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(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')

# Read the input image (replace 'your_image.jpg' with the actual image path)
image_path = 'face.jpeg'
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)
# Save or display the result
cv2.imwrite('detected_faces.jpg', image)  # Save the result

cv2.imshow('Detected Faces', image)  # Display the result
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output