1. Program to draw points , line, circle, Polygon and rectangle on a plane using OpenGL

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

// Function to initialize OpenGL
void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0);   // Set background to white
    glColor3f(0.0, 0.0, 0.0);          // Set drawing color to black
    glPointSize(5.0);                  // Set point size
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 500.0, 0.0, 500.0); // Set 2D coordinate system
}

// Function to draw a circle
void drawCircle(float cx, float cy, float r, int segments) {
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < segments; i++) {
        float theta = 2.0f * 3.1415926f * i / segments;
        float x = r * cosf(theta);
        float y = r * sinf(theta);
        glVertex2f(cx + x, cy + y);
    }
    glEnd();
}
```

```
// Display callback function
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // 1. Draw a point
    glBegin(GL_POINTS);
    glVertex2i(100, 100);
    glEnd();

    // 2. Draw a line
    glBegin(GL_LINES);
    glVertex2i(150, 150);
    glVertex2i(300, 150);
    glEnd();

    // 3. Draw a rectangle using GL_LINE_LOOP
    glBegin(GL_LINE_LOOP);
    glVertex2i(100, 200);
    glVertex2i(300, 200);
    glVertex2i(300, 300);
    glVertex2i(100, 300);
    glEnd();
```

```
    // 4. Draw a polygon (triangle)
    glBegin(GL_POLYGON);
    glVertex2i(350, 100);
    glVertex2i(400, 200);
    glVertex2i(300, 200);
    glEnd();

    // 5. Draw a circle
    drawCircle(200, 400, 50, 100);

    glFlush();
}

// Main function
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("2D Shapes in OpenGL (C)");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

2. Program to demonstrate DDA Line Drawing Algorithm

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

// Function to initialize OpenGL settings
void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0);   // White background
    glColor3f(0.0, 0.0, 0.0);          // Black drawing color
    glPointSize(3.0);                  // Point size
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 500, 0, 500);        // Set 2D orthographic projection
}

// Function to plot a point
void drawPixel(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
}
```

```
// DDA Line Drawing Algorithm
void drawLineDDA(int x0, int y0, int x1, int y1) {
    float dx = x1 - x0;
    float dy = y1 - y0;

    float steps = fabs(dx) > fabs(dy) ? fabs(dx) : fabs(dy);
    float x_inc = dx / steps;
    float y_inc = dy / steps;

    float x = x0;
    float y = y0;

    for (int i = 0; i <= steps; i++) {
        drawPixel(round(x), round(y));
        x += x_inc;
        y += y_inc;
    }
}

// Display callback
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Example: Draw a line from (50, 50) to (400, 300)
    drawLineDDA(50, 50, 400, 300);
```

```
    glFlush();
}

// Main function
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("DDA Line Drawing Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

3. Program to demonstrate Bresenham's Circle Drawing Algorithm

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

// Function to plot a single pixel (x, y)
void plotPixel(int x, int y, int xc, int yc) {
    glBegin(GL_POINTS);
        glVertex2i(xc + x, yc + y);
        glVertex2i(xc - x, yc + y);
        glVertex2i(xc + x, yc - y);
        glVertex2i(xc - x, yc - y);
        glVertex2i(xc + y, yc + x);
        glVertex2i(xc - y, yc + x);
        glVertex2i(xc + y, yc - x);
        glVertex2i(xc - y, yc - x);
    glEnd();
}

// Bresenham's Circle Drawing Algorithm
void drawCircle(int xc, int yc, int r) {
    int x = 0, y = r;
    int d = 3 - 2 * r;

    while (x <= y) {
```

```
        plotPixel(x, y, xc, yc);
        if (d < 0)
            d = d + 4 * x + 6;
        else {
            d = d + 4 * (x - y) + 10;
            y--;
        }
        x++;
    }
}

// Display callback
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0); // white color

    int xc = 250, yc = 250, r = 100; // center and radius
    drawCircle(xc, yc, r);

    glFlush();
}

// Initialization
void init() {
    glClearColor(0.0, 0.0, 0.0, 0.0); // black background
```

```
    glColor3f(1.0, 1.0, 1.0);         // drawing color = white
    glPointSize(2.0);                 // pixel size
    gluOrtho2D(0, 500, 0, 500);       // coordinate system
}

// Main function
int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Bresenham's Circle Drawing Algorithm");

    init();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```

4.**Program to fill any given polygon using scan-line area filling algorithm**

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

#define MAX_VERTICES 20

int vertex_count;
int poly_x[MAX_VERTICES], poly_y[MAX_VERTICES];

int max_y = 0, min_y = 10000;

// Function to initialize OpenGL settings
void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0); // White background
    glColor3f(0.0, 0.0, 0.0);        // Black drawing color
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 500, 0, 500);
}

// Function to draw a pixel
void draw_pixel(int x, int y) {
    glBegin(GL_POINTS);
    glVertex2i(x, y);
```

```
    glEnd();
}

// Scan-line fill algorithm
void scanline_fill() {
    int i, j, k;
    int inter_x[MAX_VERTICES];  // Stores x-intersections

    for (int y = min_y; y <= max_y; y++) {
        k = 0;

        // Find intersections with edges
        for (i = 0; i < vertex_count; i++) {
            j = (i + 1) % vertex_count;

            int x1 = poly_x[i], y1 = poly_y[i];
            int x2 = poly_x[j], y2 = poly_y[j];

            if (y1 < y2) {
                int tmp;
                tmp = x1; x1 = x2; x2 = tmp;
                tmp = y1; y1 = y2; y2 = tmp;
            }

            if (y >= y2 && y < y1 && y1 != y2) {
```

```
            inter_x[k++] = (int)(x1 + ((float)(x2 - x1) / (float)(y2 - y1)) * (y - y1));
        }
    }

    // Sort the intersection points
    for (i = 0; i < k - 1; i++) {
        for (j = 0; j < k - i - 1; j++) {
            if (inter_x[j] > inter_x[j + 1]) {
                int temp = inter_x[j];
                inter_x[j] = inter_x[j + 1];
                inter_x[j + 1] = temp;
            }
        }
    }

    // Fill pixels between pairs of intersections
    for (i = 0; i < k; i += 2) {
        for (j = inter_x[i]; j <= inter_x[i + 1]; j++) {
            draw_pixel(j, y);
        }
    }
  }
}
```

```
// Function to draw the polygon outline
void draw_polygon() {
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < vertex_count; i++) {
        glVertex2i(poly_x[i], poly_y[i]);
    }
    glEnd();
}

// Display callback
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0); // Black border
    draw_polygon();
    glColor3f(1.0, 0.0, 0.0); // Red fill
    scanline_fill();
     glFlush();
}

// Main
int main(int argc, char **argv) {
    // Example polygon: user can modify or take input dynamically
    vertex_count = 4;
    poly_x[0] = 100; poly_y[0] = 100;
    poly_x[1] = 200; poly_y[1] = 300;
```

```
    poly_x[2] = 300; poly_y[2] = 300;
    poly_x[3] = 400; poly_y[3] = 100;

    // Find min_y and max_y
    for (int i = 0; i < vertex_count; i++) {
        if (poly_y[i] > max_y) max_y = poly_y[i];
        if (poly_y[i] < min_y) min_y = poly_y[i];
    }

    // GLUT init
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Scan-Line Polygon Fill Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();

    return 0;
}
```

5.Program to draw a color cube and spin it using open GL transformation matrices.

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[]={-1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0,-1.0,1.0,-1.0,-1.0,-1.0,1.0,1.0,
-1.0,1.0,1.0,1.0,1.0,-1.0,1.0,1.0};
GLfloat normals[]={-1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0,-1.0,1.0,-1.0,-1.0,-1.0,1.0,1.0,
-1.0,1.0,1.0,1.0,1.0,-1.0,1.0,1.0};
GLfloat colors[]={0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,1.0,0.0,1.0,1.0,1.0,
1.0,1.0,0.0,1.0,1.0};
GLubyte CubeIndices[]={0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4};
static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glRotatef(theta[0],1.0,0.0,0.0);
glRotatef(theta[1],0.0,1.0,0.0);
glRotatef(theta[2],0.0,0.0,1.0);
glDrawElements(GL_QUADS,24,GL_UNSIGNED_BYTE,CubeIndices);
glBegin(GL_LINES);
glVertex3f(0.0,0.0,0.0);
glVertex3f(1.0,1.0,1.0);
glEnd();
glFlush();
glutSwapBuffers();
}
void SpinCube()
{
theta[axis]+=5.0;
if(theta[axis]>360.0)theta[axis]-=360.0;
glutPostRedisplay();
}
void mouse(int btn,int state,int x,int y)
{
if(btn==GLUT_LEFT_BUTTON&&state==GLUT_DOWN)axis=0;
if(btn==GLUT_MIDDLE_BUTTON&&state==GLUT_DOWN)axis=1;
if(btn==GLUT_RIGHT_BUTTON&&state==GLUT_DOWN)axis=2;
} COMPUTER GRAPHICS AND VISUALIZATION LAB PROGRAMS
```

```
void myReshape(int w,int h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if(w<=h)
glOrtho(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);
else
glOrtho(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-2.0,2.0,-10.0,10.0);
glMatrixMode(GL_MODELVIEW);
}
void main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutCreateWindow("spin a colorcube");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glutIdleFunc(SpinCube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST);
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3,GL_FLOAT,0,vertices);
glColorPointer(3,GL_FLOAT,0,colors);
glNormalPointer(GL_FLOAT,0,normals);
glColor3f(1.0,1.0,1.0);
glutMainLoop();
}
```

6. Program to create a House like figure and rotate it about a given fixed point using open GL functions.

```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
GLfloat house[3][9]={{100.0,100.0,175.0,250.0,250.0,150.0,150.0,200.0,200.0},{100.0,300.0,
400.0, 300.0,100.0,100.0,150.0,150.0,100.0},{1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0}};
GLfloat rot_mat[3][3]={{0},{0},{0}};
GLfloat result[3][9]={{0},{0},{0}};
GLfloat h=100.0;
GLfloat k=100.0;
GLfloat theta;
void multiply()
{
int i,j,l;
for(i=0;i<3;i++)
for(j=0;j<9;j++)
{
result[i][j]=0;
for(l=0;l<3;l++)
result[i][j]=result[i][j]+rot_mat[i][l]*house[l][j];
}
}
void rotate()
{
GLfloat m,n;
m=-h*(cos(theta)-1)+k*(sin(theta));
n=-k*(cos(theta)-1)-h*(sin(theta));
rot_mat[0][0]=cos(theta);
rot_mat[0][1]=-sin(theta);
rot_mat[0][2]=m;
rot_mat[1][0]=sin(theta);
rot_mat[1][1]=cos(theta);
rot_mat[1][2]=n;
rot_mat[2][0]=0;
rot_mat[2][1]=0;
rot_mat[2][2]=1;
multiply();
}
void drawhouse()
{
glColor3f(0.0,0.0,1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(house[0][0],house[1][0]);
glVertex2f(house[0][1],house[1][1]);
```

```
glVertex2f(house[0][3],house[1][3]);
glVertex2f(house[0][4],house[1][4]);
glEnd();
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(house[0][5],house[1][5]);
glVertex2f(house[0][6],house[1][6]);
glVertex2f(house[0][7],house[1][7]);
glVertex2f(house[0][8],house[1][8]);
glEnd();
glColor3f(0.0,0.0,1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(house[0][1],house[1][1]);
glVertex2f(house[0][2],house[1][2]);
glVertex2f(house[0][3],house[1][3]);
glEnd();
}
void drawrotatedhouse()
{
glColor3f(0.0,0.0,1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(result[0][0],result[1][0]);
glVertex2f(result[0][1],result[1][1]);
glVertex2f(result[0][3],result[1][3]);
glVertex2f(result[0][4],result[1][4]);
glEnd();
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(result[0][5],result[1][5]);
glVertex2f(result[0][6],result[1][6]);
glVertex2f(result[0][7],result[1][7]);
glVertex2f(result[0][8],result[1][8]);
glEnd();
glColor3f(0.0,0.0,1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(result[0][1],result[1][1]);
glVertex2f(result[0][2],result[1][2]);
glVertex2f(result[0][3],result[1][3]);
glEnd();
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
drawhouse();
rotate();
```

```
drawrotatedhouse();
glFlush();
}
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc,char **argv)
{
printf("Enter the rotation angle\n");
scanf("%f",&theta);
theta=theta*3.142/180.0;
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("HOUSE ROTATION");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```

7 Program using OpenGL and GLUT to create a basic graphical window and handle keyboard input events

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

// Window dimensions
int windowWidth = 800;
int windowHeight = 600;

// Function to display contents on the window
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_TRIANGLES);
        glVertex2f(-0.5f, -0.5f);
        glVertex2f(0.5f, -0.5f);
        glVertex2f(0.0f, 0.5f);
    glEnd();
    glFlush();
}

void handleKeypress(unsigned char key, int x, int y) {
    switch (key) {
        case 27:  // ESC key
            printf("Escape pressed. Exiting...\n");
```

```
            exit(0);
            break;
        case 'r':
        case 'R':
            printf("You pressed 'R'.\n");
            glClearColor(1.0, 0.0, 0.0, 1.0); // Change background to red
            glutPostRedisplay();
            break;
        case 'g':
        case 'G':
            printf("You pressed 'G'.\n");
            glClearColor(0.0, 1.0, 0.0, 1.0); // Change background to green
            glutPostRedisplay();
            break;
        case 'b':
        case 'B':
            printf("You pressed 'B'.\n");
            glClearColor(0.0, 0.0, 1.0, 1.0); // Change background to blue
            glutPostRedisplay();
            break;
        default:
            printf("You pressed key: %c\n", key);
            break;
    }
}
```

```
// Function to handle special keys (arrows, F1–F12, etc.)
void handleSpecialKeys(int key, int x, int y) {
    switch (key) {
        case GLUT_KEY_UP:
            printf("Up arrow pressed.\n");
            break;
        case GLUT_KEY_DOWN:
            printf("Down arrow pressed.\n");
            break;
        case GLUT_KEY_LEFT:
            printf("Left arrow pressed.\n");
            break;
        case GLUT_KEY_RIGHT:
            printf("Right arrow pressed.\n");
            break;
    }
}
void init() {
    glClearColor(0.0, 0.0, 0.0, 1.0); // Set initial background color (black)
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-1, 1, -1, 1); // 2D orthographic projection
}
```

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("OpenGL Keyboard Input Example");
    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(handleKeypress
    glutSpecialFunc(handleSpecialKeys);
    glutMainLoop();
    return 0;
}
```

8. Implementation of Basic Relationships Between Pixels

PREREQUSITE

Install OpenCV if not already:

```
pip install opencv-python
```

CODE:

```
import cv2

import numpy as np


# Create a blank 7x7 image (black)

img = np.zeros((7, 7, 3), dtype=np.uint8)


# Define the central pixel

x, y = 3, 3


# Mark the center pixel in RED

img[x, y] = [0, 0, 255]  # BGR (Red)


# Define 4-neighbors (up, down, left, right)

N4 = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]


# Define diagonal neighbors

ND = [(x-1, y-1), (x-1, y+1), (x+1, y-1), (x+1, y+1)]
```

```
# Mark 4-neighbors in GREEN
for (i, j) in N4:
    img[i, j] = [0, 255, 0]

# Mark diagonal neighbors in BLUE
for (i, j) in ND:
    img[i, j] = [255, 0, 0]

# Resize image for better visualization
img = cv2.resize(img, (350, 350), interpolation=cv2.INTER_NEAREST)

# Display the image
cv2.imshow("Basic Pixel Relationships", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

9. Implementation of Image Sampling and Quantization

PREREQUSITE

Install OpenCV if not already:

```
pip install opencv-python
```

Make sure you have an image file (e.g., `your_image.jpg`) in the same directory.

```
import cv2
import numpy as np

# Load image
img = cv2.imread('your_image.jpg')   # Replace with your image path
if img is None:
    raise FileNotFoundError("Image not found. Please check the path.")

# Resize to a fixed size for convenience
img = cv2.resize(img, (512, 512))

# ---------- IMAGE SAMPLING ----------
def image_sampling(image, factor):
    """Reduce the spatial resolution by the given factor."""
    # Downsample
```

```
    sampled = cv2.resize(image, (image.shape[1] // factor, image.shape[0] // factor),
                         interpolation=cv2.INTER_NEAREST)
    # Upsample back to original size (for display)
    sampled_up = cv2.resize(sampled, (image.shape[1], image.shape[0]),
                            interpolation=cv2.INTER_NEAREST)
    return sampled_up

# ---------- IMAGE QUANTIZATION ----------
def image_quantization(image, levels):
    """Reduce number of intensity levels (quantization)."""
    # levels: number of gray levels per channel (e.g., 2, 4, 8, 16, 32, 64, 128, 256)
    quantized = np.floor(image / (256 / levels)) * (256 / levels)
    quantized = np.uint8(quantized)
    return quantized

# Perform Sampling (reduce spatial resolution)
sampled_2x = image_sampling(img, 2)
sampled_4x = image_sampling(img, 4)
sampled_8x = image_sampling(img, 8)

# Perform Quantization (reduce intensity levels)
quant_64 = image_quantization(img, 64)
quant_16 = image_quantization(img, 16)
quant_4 = image_quantization(img, 4)
```

```
# ---------- DISPLAY ----------
cv2.imshow("Original Image", img)
cv2.imshow("Sampling x2", sampled_2x)
cv2.imshow("Sampling x4", sampled_4x)
cv2.imshow("Sampling x8", sampled_8x)
cv2.imshow("Quantization 64 Levels", quant_64)
cv2.imshow("Quantization 16 Levels", quant_16)
cv2.imshow("Quantization 4 Levels", quant_4)

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

10. Contrast stretching of a low contrast image, Histogram, and Histogram Equalization

---

Requirements

Install dependencies:

```
pip install opencv-python matplotlib
```

Use a **low-contrast image**, e.g., a dark or grayish photo saved as `low_contrast.jpg`

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

# --------- Load Image ---------
img = cv2.imread('low_contrast.jpg', 0)   # Load in grayscale
if img is None:
    raise FileNotFoundError("Image not found. Please check the path.")

# --------- 1. CONTRAST STRETCHING ---------
# Get min and max pixel values
r_min, r_max = np.min(img), np.max(img)

# Apply contrast stretching formula
```

```
stretched = ((img - r_min) / (r_max - r_min)) * 255
stretched = np.uint8(stretched)

# --------- 2. HISTOGRAM EQUALIZATION ---------
equalized = cv2.equalizeHist(img)

# --------- 3. HISTOGRAMS ---------
# Function to plot histogram
def plot_histogram(image, title):
    plt.figure(figsize=(5, 3))
    plt.title(title)
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.hist(image.ravel(), 256, [0, 256], color='blue')
    plt.grid(True)
    plt.show()

# --------- DISPLAY RESULTS ---------
cv2.imshow("Original Low Contrast Image", img)
cv2.imshow("Contrast Stretched Image", stretched)
cv2.imshow("Histogram Equalized Image", equalized)

# --------- PLOT HISTOGRAMS ---------
plot_histogram(img, "Original Image Histogram")
plot_histogram(stretched, "Contrast Stretched Histogram")
```

```
plot_histogram(equalized, "Histogram Equalized Histogram")


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

11.