Vijayanagar Educational Trust ®

EAST WEST COLLEGE OF ENGINEERING

(Affiliated to VTU, Belagavi, approved by AICTE, New Delhi, Recognized by Govt. of Karnataka)

13, 13th 'A' Main, Sector A, Yelahanka new Town, Bengaluru – 560064

Department of Computer Science and Engineering

LAB MANUAL

On

Computer Graphics and Image Processing(21CSL66), VI Semester

By

Anitha L

Assistant Professor

Computer Science and Engineering Department

East West College of Engineering, Yelahanka New Town, Bangalore - 560064

VI Semester

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY			
Course Code	21CSL66	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Total Marks	100
Credits	1	Exam Hours	03

Course Objectives:

CLO 1: Demonstrate the use of Open GL.
CLO 2: Demonstrate the different geometric object drawing using openGL
CLO 3: Demonstration of 2D/3D transformation on simple objects.
CLO 4: Demonstration of lighting effects on the created objects.
CLO 5: Demonstration of Image processing operations on image/s.

Sl. No.	Practise Programs
	• Installation of OpenGL /OpenCV/ Python and required headers • Simple programs using OpenGL (Drawing simple geometric object like line, circle, rectangle, square) • Simple programs using OpenCV (operation on an image/s)
	PART A ***List of problems for which student should develop program and execute in the Laboratory using openGL/openCV/ Python***
1.	Develop a program to draw a line using Bresenham's line drawing technique
2.	Develop a program to demonstrate basic geometric operations on the 2D object
3.	Develop a program to demonstrate basic geometric operations on the 3D object
4.	Develop a program to demonstrate 2D transformation on basic objects
5.	Develop a program to demonstrate 3D transformation on 3D objects
6.	Develop a program to demonstrate Animation effects on simple objects.
7.	Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.
8.	Write a program to show rotation, scaling, and translation on an image.
9.	Read an image and extract and display low-level features such as edges, textures using filtering techniques.
10.	Write a program to blur and smoothing an image.
11.	Write a program to contour an image.
12.	Write a program to detect a face/s in an image.
	PART B **Practical Based Learning**
	Student should develop a mini project and it should be demonstrate in the laboratory examination, Some of the projects are listed and it is not limited to: ➢ Recognition of License Plate through Image Processing ➢ Recognition of Face Emotion in Real-Time ➢ Detection of Drowsy Driver in Real-Time ➢ Recognition of Handwriting by Image Processing ➢ Detection of Kidney Stone ➢ Verification of Signature ➢ Compression of Color Image ➢ Classification of Image Category ➢ Detection of Skin Cancer ➢ Marking System of Attendance using Image Processing ➢ Detection of Liver Tumor ➢ IRIS Segmentation ➢ Detection of Skin Disease and / or Plant Disease ➢ Biometric Sensing System . ➢ Projects which helps to formers to understand the present developments in agriculture.

Software Installation for CG&IP Lab programs

OpenGL programs:

Programs from 1 to 6:

IDE(Integrated Development Environment): Codeblocks 16.1v

Programming Language: C

Library: OpenGL

Steps to install codeblocks:

1. Download codeblocks version16.1 or latest
2. Copy glut32.dll in c://windows
3. While installing set file path to c://codeblocks//minGW

Opencv programs:

Programs from 7 to 12:

IDE: Jupyter notebook

Programming language: Python

Library: Opencv

To import Opencv module goto jupyter notebook command prompt and type

pip install opencv-python

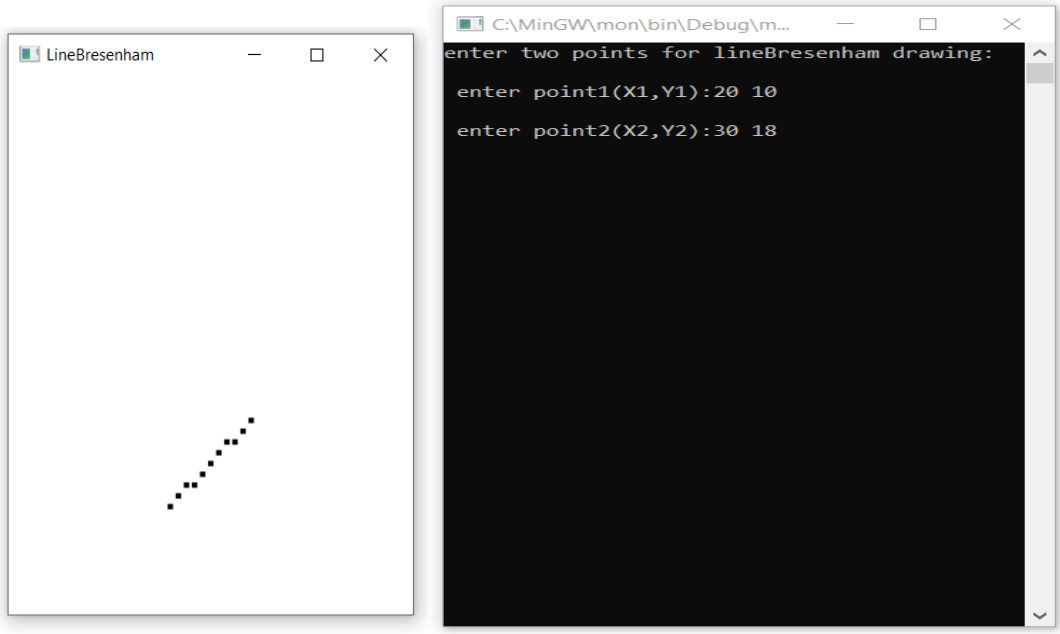
1. Develop a program to draw a line using Bresenham's line drawing technique.

```
#include<stdio.h>
#include<math.h>
#include<gl/glut.h>
GLint X1,Y1,X2,Y2;
void LineBresenham(void)
{
glClear(GL_COLOR_BUFFER_BIT);
int dx=abs(X2-X1),dy=abs(Y2-Y1);
int p=2*dy-dx;
int twoDy=2*dy, twoDyDx=2*(dy-dx);
int x,y;
if(X1>X2)
{
x=X2;
y=Y2;
X2=X1;
}
else
{
x=X1;
y=Y1;
X2=X2;
}
glBegin(GL_POINTS);
glVertex2i(x,y);
while(x<X2)
{
x++;
```

```
if(p<0)
p+=twoDy;
else
{
y++;
p+=twoDyDx;
}
glVertex2i(x,y);
}
glEnd();
glFlush();
}
void Init()
{
glClearColor(1.0,1.0,1.0,0);
glColor3f(0.0,0.0,0.0);
glPointSize(4.0);
glViewport(0,0,50,50);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,50,0,50);
}
int main(int argc,char **argv)
{
printf("enter two points for  lineBresenham drawing:\n");
printf("\n enter point1(X1,Y1):");
scanf("%d%d",&X1,&Y1);
printf("\n enter point2(X2,Y2):");
scanf("%d%d",&X2,&Y2);
glutInit(&argc,argv);
```

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(300,400);
glutInitWindowPosition(0,0);
glutCreateWindow("LineBresenham");
Init();
glutDisplayFunc(LineBres);
glutMainLoop();
}
```

Output:



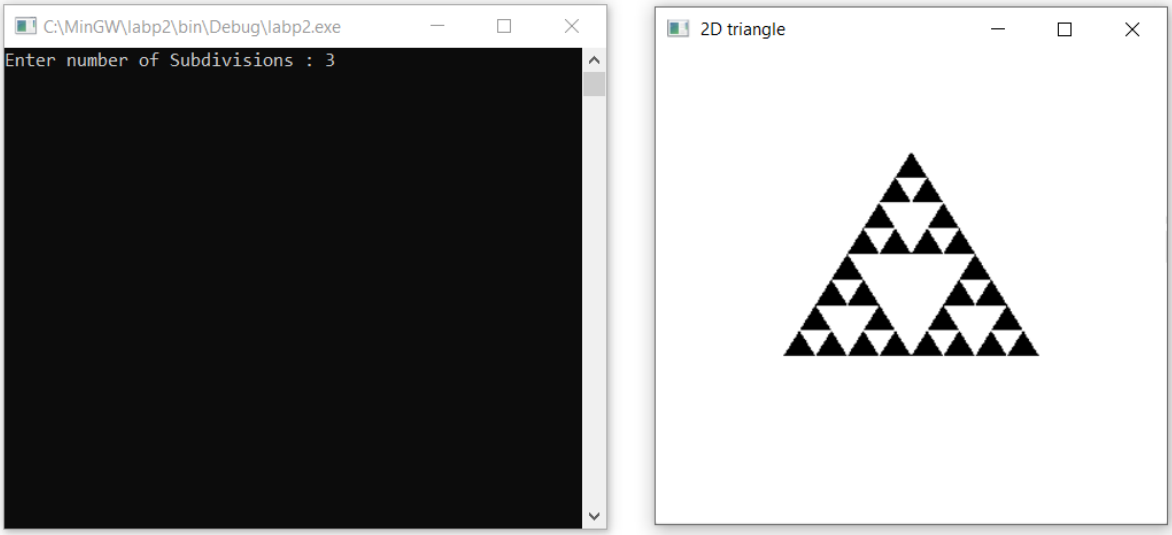
2. Develop a program to demonstrate basic geometric operations on the 2D object.

```
#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);
}
void main(int argc, char **argv)
{
printf("Enter number of Subdivisions : ");
scanf("%d",&n);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
glutInitWindowSize(500, 500);
glutCreateWindow("2D triangle");
glutDisplayFunc(display);
myinit();
glutMainLoop();}
```

Output:


C:\MinGW\labp2\bin\Debug\labp2.exe
Enter number of Subdivisions : 3
2D triangle

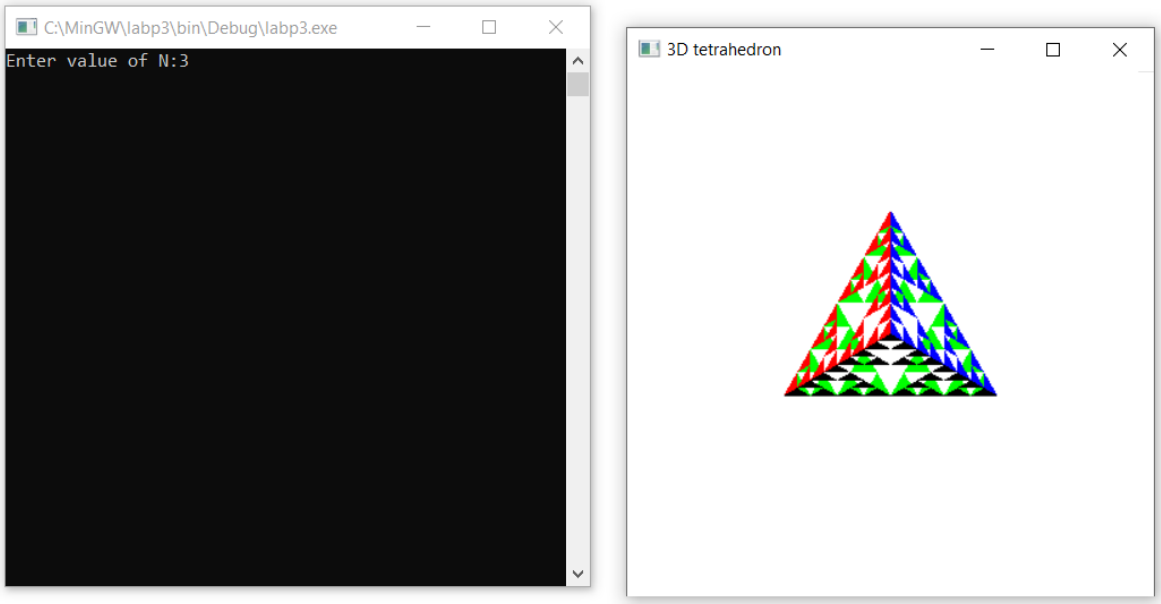
3. Develop a program to demonstrate basic geometric operations on 3D object.

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
typedef float point[3];
/* initial tetrahedron */
point v[]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},
{-0.816497, -0.471405, -0.333333}, {0.816497, -0.471405, -0.333333}};
static GLfloat theta[] = {0.0,0.0,0.0};
int n;
void triangle( point a, point b, point c)
/* display one triangle using a line loop for wire frame, a single
normal for constant shading, or three normals for interpolative shading */
{
glBegin(GL_POLYGON);
glNormal3fv(a);
glVertex3fv(a);
glVertex3fv(b);
glVertex3fv(c);
glEnd();
}
/* triangle subdivision using vertex numbers
righthand rule applied to create outward pointing faces */
void divide_triangle(point a, point b, point c, int m)
{
point v1, v2, v3;459
int j;
if(m>0)
{
```

```
for(j=0; j<3; j++) v1[j]=(a[j]+b[j])/2;
for(j=0; j<3; j++) v2[j]=(a[j]+c[j])/2;
for(j=0; j<3; j++) v3[j]=(b[j]+c[j])/2;
divide_triangle(a, v1, v2, m-1);
divide_triangle(c, v2, v3, m-1);
divide_triangle(b, v3, v1, m-1);
}
else(triangle(a,b,c)); /* draw triangle at end of recursion */
}
/* Apply triangle subdivision to faces of tetrahedron */
void tetrahedron( int m)
{
glColor3f(1.0,0.0,0.0);
divide_triangle(v[0], v[1], v[2], m);
glColor3f(0.0,1.0,0.0);
divide_triangle(v[3], v[2], v[1], m);
glColor3f(0.0,0.0,1.0);
divide_triangle(v[0], v[3], v[1], m);
glColor3f(0.0,0.0,0.0);
divide_triangle(v[0], v[2], v[3], m);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
tetrahedron(3);
glFlush();
}
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);
glutPostRedisplay();
}
int main(int argc, char **argv)
{
int i = 0;
printf("Enter value of N:");
scanf("%d", &i);
n = i;
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(500, 500);
glutCreateWindow("3D tetrahedron ");
glutReshapeFunc(myReshape);
glutDisplayFunc(display);
glEnable(GL_DEPTH_TEST);
glClearColor (1.0, 1.0, 1.0, 1.0);
glutMainLoop();
}
```

Output:


C:\MinGW\labp3\bin\Debug\labp3.exe
Enter value of N:3
3D tetrahedron

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

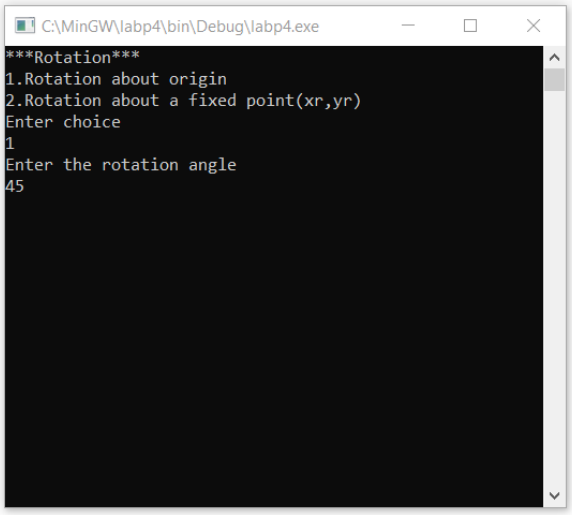
```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
GLfloat t[3][3]={{10.0,30.0,20.0},{20.0,20.0,40.0},{1.0,1.0,1.0}};
GLfloat rotatemat[3][3]={{0},{0},{0}};
GLfloat result[3][9]={{0},{0},{0}};
GLfloat xr=10.0;
GLfloat yr=20.0;
GLfloat theta;
GLint h;
void multiply(){
int i,j,k;
for(i=0;i<3;i++)
for(j=0;j<9;j++){
result[i][j]=0;
for(k=0;k<3;k++)
result[i][j]=result[i][j]+rotatemat[i][k]*t[k][j];
}
}
void rotate_about_origin(){
rotatemat[0][0]=cos(theta);
rotatemat[0][1]=-sin(theta);
rotatemat[0][2]=0;
rotatemat[1][0]=sin(theta);
rotatemat[1][1]=cos(theta);
rotatemat[1][2]=0;
rotatemat[2][0]=0;
```

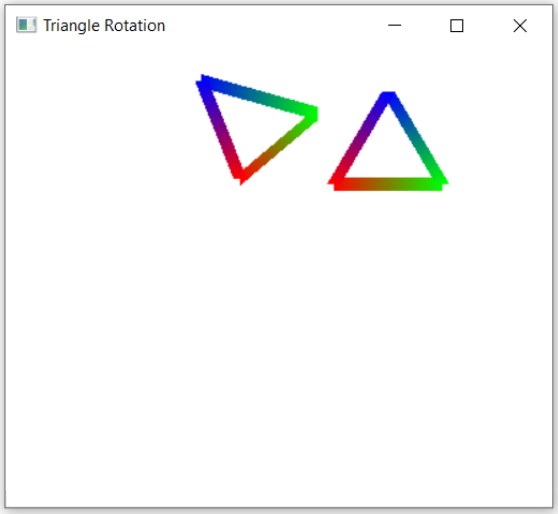
```
rotatemat[2][1]=0;
rotatemat[2][2]=1;
multiply();
}
void rotate_about_fixed_point(){
GLfloat m,n;
m=xr*(1-cos(theta))+yr*sin(theta);
n=yr*(1-cos(theta))-xr*sin(theta);
rotatemat[0][0]=cos(theta);
rotatemat[0][1]=-sin(theta);
rotatemat[0][2]=m;
rotatemat[1][0]=sin(theta);
rotatemat[1][1]=cos(theta);
rotatemat[1][2]=n;
rotatemat[2][0]=0;
rotatemat[2][1]=0;
rotatemat[2][2]=1;
multiply();
}
void draw_triangle(){
glLineWidth(10);
glBegin(GL_LINE_LOOP);
glColor3f(1.0,0.0,0.0);
glVertex2f(t[0][0],t[1][0]);
glColor3f(0.0,1.0,0.0);
glVertex2f(t[0][1],t[1][1]);
glColor3f(0.0,0.0,1.0);
glVertex2f(t[0][2],t[1][2]);
glEnd();
glFlush();
```

```
}
void draw_rotated_triangle(){
glLineWidth(10);
glBegin(GL_LINE_LOOP);
glColor3f(1.0,0.0,0.0);
glVertex2f(result[0][0],result[1][0]);
glColor3f(0.0,1.0,0.0);
glVertex2f(result[0][1],result[1][1]);
glColor3f(0.0,0.0,1.0);
glVertex2f(result[0][2],result[1][2]);
glEnd();
glFlush();
}
void display(){
glClear(GL_COLOR_BUFFER_BIT);
if(ch==1){
draw_triangle();
rotate_about_origin();
draw_rotated_triangle();
glFlush();
}
if(ch==2){
draw_triangle();
rotate_about_fixed_point();
draw_rotated_triangle();
glFlush();
}
}
void myinit(){
glClearColor(1.0,1.0,1.0,1.0);
```

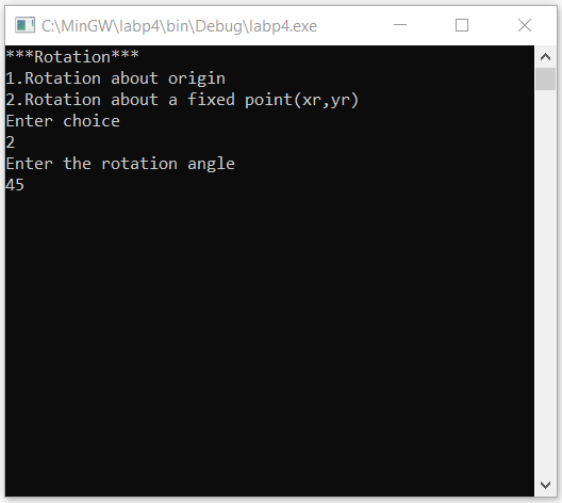
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-50.0,50.0,-50.0,50.0);
}
int main(int argc,char** argv){
printf("***Rotation***\n1.Rotation about origin\n2.Rotation about a fixed point(xr,yr)\n");
printf("Enter choice\n");
scanf("%d",&ch);
printf("Enter the rotation angle\n");
scanf("%f",&theta);
theta=theta*(3.14/180);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Triangle Rotation\n");
glutDisplayFunc(display);
myinit();
glutMainLoop();
return 0;
}
```

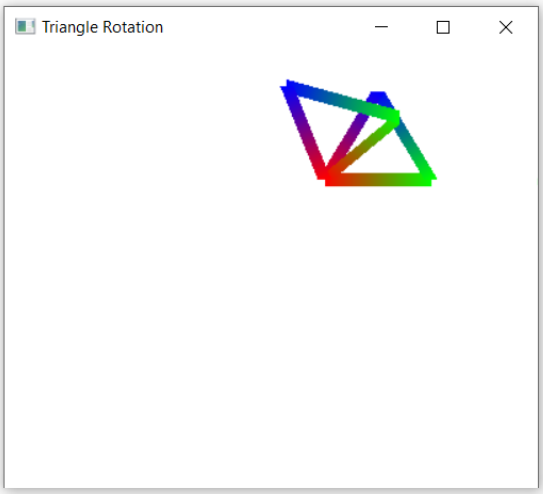
Output:





Output 2:





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

```
#include <stdlib.h>
#include <GL/glut.h>
        GLfloat vertices[][3] = {{-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[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
                               {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
                               {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};

void polygon(int a, int b, int c , int d)
{
        glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
            glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glVertex3fv(vertices[d]);
        glEnd();
}

void colorcube()
{
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
```

```
        polygon(0,1,5,4);
}

static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
static GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */

void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
/* Update viewer position in modelview matrix */
    glLoadIdentity();
    gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
          /* rotate cube */
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);
colorcube();      /* draw the rotated color cube */
glFlush();
glutSwapBuffers();
}

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;
 theta[axis] += 4.0;
if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
display();
}

void keys(unsigned char key, int x, int y)
{
```

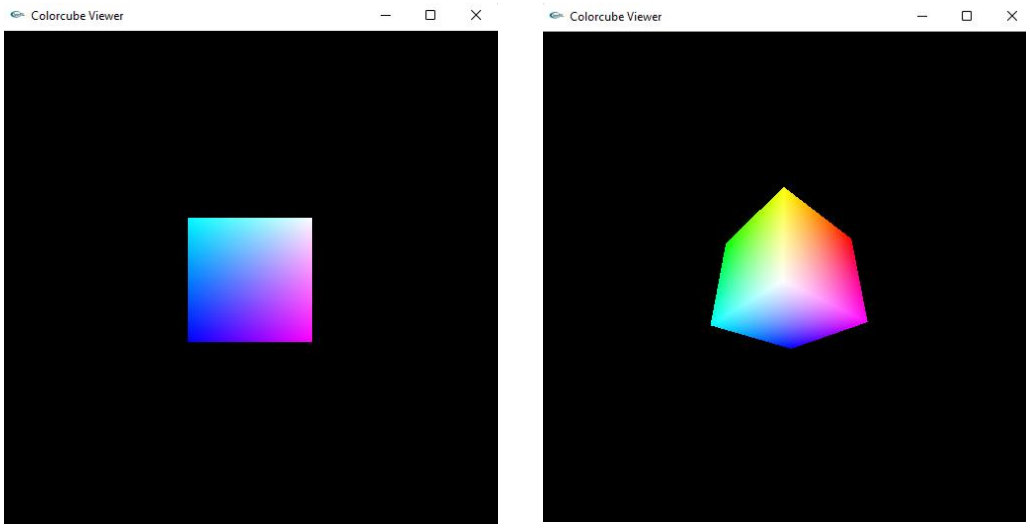
```
/* Use x, X, y, Y, z, and Z keys to move viewer */
   if(key == 'x') viewer[0]-= 1.0;
   if(key == 'X') viewer[0]+= 1.0;
   if(key == 'y') viewer[1]-= 1.0;
   if(key == 'Y') viewer[1]+= 1.0;
   if(key == 'z') viewer[2]-= 1.0;
   if(key == 'Z') viewer[2]+= 1.0;
   display();
}

void myReshape(int w, int h)
{
 glViewport(0, 0, w, h);
/* Use a perspective view */
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
    if(w<=h) glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,
 2.0* (GLfloat) h / (GLfloat) w, 2.0, 20.0);
          else glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,
       2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);
             /* Or we can use gluPerspective that is   gluPerspective(45.0, w/h, -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("Colorcube Viewer");
 glutReshapeFunc(myReshape);
 glutDisplayFunc(display);
glutMouseFunc(mouse);
glutKeyboardFunc(keys);
```

```
glEnable(GL_DEPTH_TEST);
glutMainLoop();
}
```

Output:



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

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

float trianglePosX = -0.5f; // Initial position of the triangle
float triangleSpeed = 0.005f; // Speed of the triangle

void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glLoadIdentity();

   // Draw the triangle
   glBegin(GL_TRIANGLES);
   glColor3f(1.0f, 0.0f, 0.0f);   // Red color
   glVertex2f(trianglePosX, 0.0f);
   glVertex2f(trianglePosX + 0.1f, 0.2f);
   glVertex2f(trianglePosX + 0.2f, 0.0f);
   glEnd();

   glutSwapBuffers();
}

void update(int value)
{
   // Update the position of the triangle
   trianglePosX += triangleSpeed;

   // If the triangle goes beyond the right edge of the window, reset its position
   if (trianglePosX > 1.1f)
      trianglePosX = -0.5f;

   // Redisplay the scene
   glutPostRedisplay();

   // Call update() again after 16 milliseconds
   glutTimerFunc(16, update, 0);
}

void init()
{
   glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // White background color
}
```

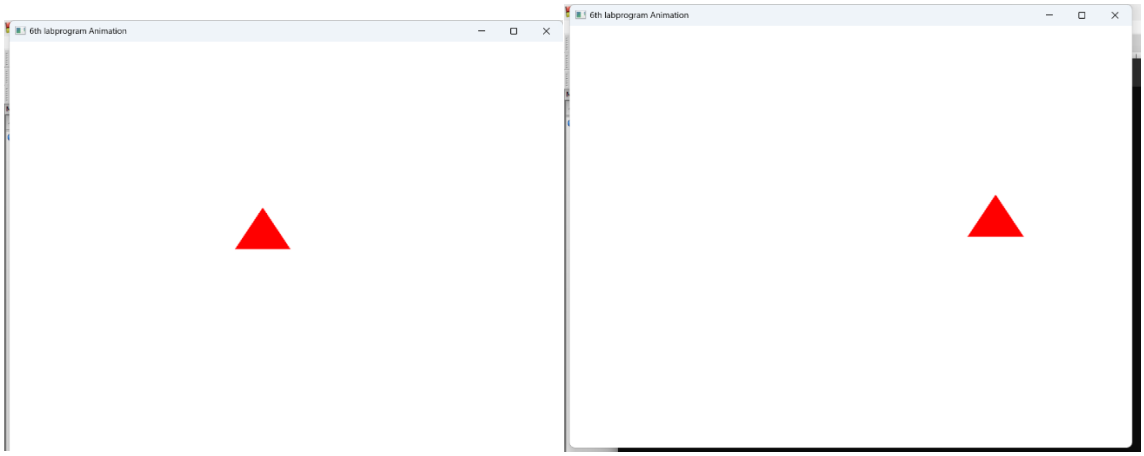
```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize(800, 600); // Window size
   glutCreateWindow("6th labprogram Animation");

   init();

   glutDisplayFunc(display);
   glutTimerFunc(16, update, 0); // Call update() after 16 milliseconds

   glutMainLoop();
   return 0;
}
```

OUTPUT:



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("C:\\users\\hp\\saved pictures\\img.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]
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:

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

```
    #Rotation and scaling of image
import cv2
import numpy as np

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("C:\\Users\\HP\\Pictures\\IMG.jpg")
# Get image dimensions
height, width = image.shape[:2]

# Calculate the center coordinates of the image
center = (width // 2, height // 2)
rotation_value = int(input("Enter the degree of Rotation:"))
scaling_value = int(input("Enter the zooming factor:"))
# 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))
scaled = cv2.getRotationMatrix2D(center=center, angle=0, scale=scaling_value)
scaled_image = cv2.warpAffine(src=rotated_image, M=scaled, dsize=(width, height))
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? "))
translated_image = translate_image(scaled_image, dx=h, dy=v)
cv2.imwrite("final_image.jpg", translated_image)
```

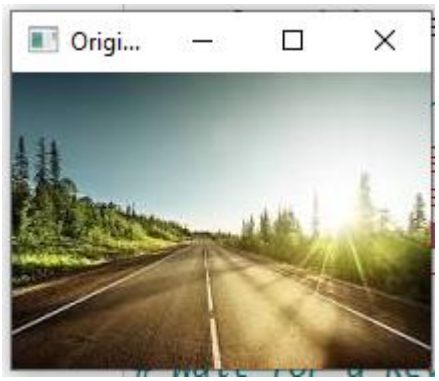
Output:

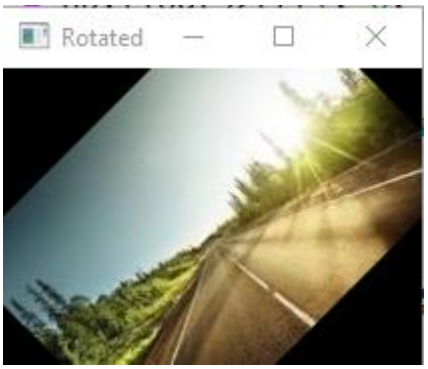
Enter the degree of Rotation: 45

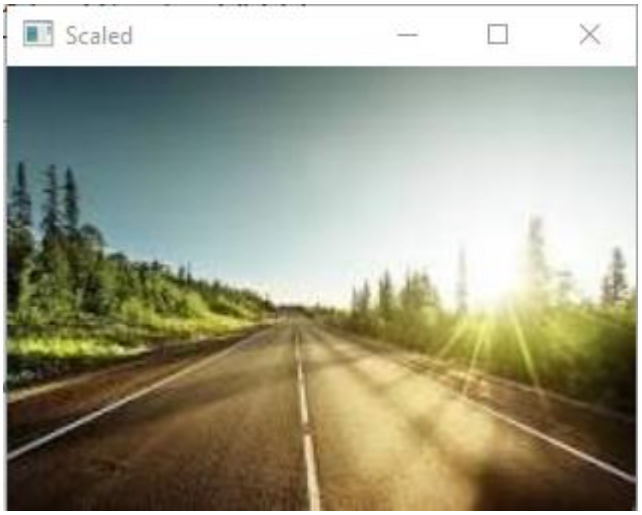
Enter the zooming factor: 200

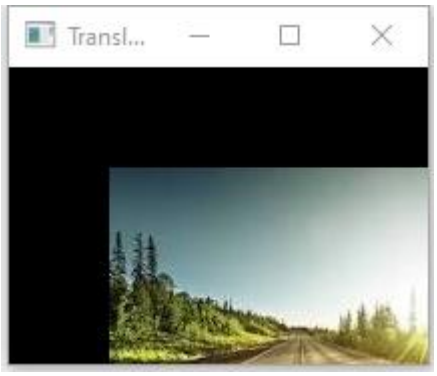
How many pixels you want the image to be translated horizontally? 100

How many pixels you want the image to be translated vertically? 100







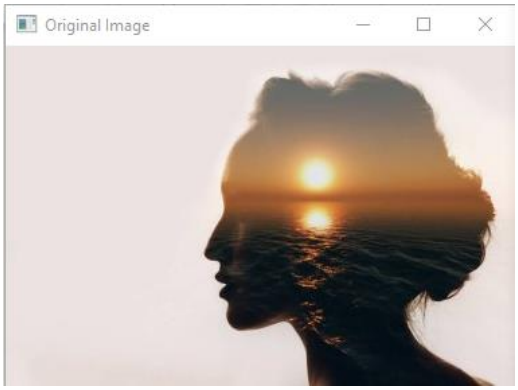


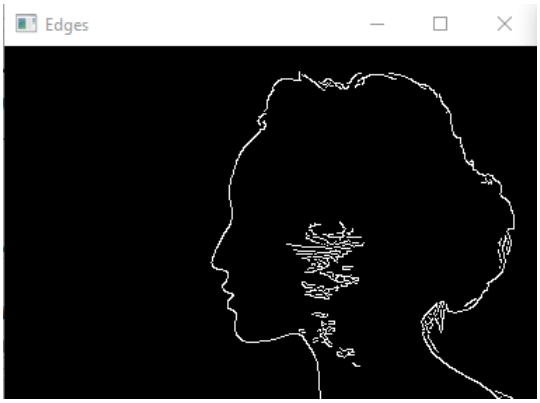
9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.

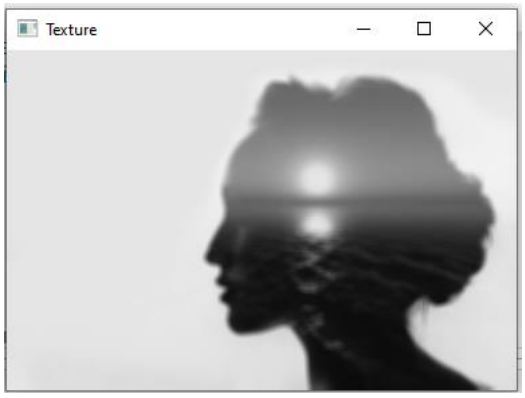
```
import cv2
import numpy as np
# Load the image
image_path = "C:\\Users\\HP\\Pictures\\23.jpg" # Replace with the path to your image
img = cv2.imread(image_path)
# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Edge detection
edges = cv2.Canny(gray, 100, 200) # Use Canny 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:







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

```
import cv2
import numpy as np
img =cv2.imread("C:\\Users\\HP\\Pictures\\IMG.jpg",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")
```

output:

```
[[74 71 73 ... 63 47 44]
 [68 67 69 ... 59 49 59]
 [67 68 71 ... 61 46 56]
 ...
 [ 4  5  3 ...  6 11 14]
 [ 2  5  6 ...  3  4  6]
 [ 3  8 11 ...  8  9 10]]
```

11. Write a program to contour an image.

```
import cv2
import numpy as np

image_path ="C:\\Users\\HP\\Pictures\\IMG.jpg"
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:

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 &#39;your_image.jpg&#39; with the actual image path)

image_path = "C:\\Users\\HP\\Pictures\\IMG.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)

# 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: