CG PROGRAMS

P1. Develop a program to draw a line using Bresenham's line drawing technique

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

int x1, y11, x2, y2, incx = 1, incy = 1, x, y, p;
float m;

void init()
{
gluOrtho2D(0, 500, 0, 500);
}

void plotpoint(int x, int y)
{
glColor3f(1, 0, 0);

glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
glFlush();
}

void LineBres(int x1, int y11, int x2, int y2)
{
x = x1;
y = y11;
```

```
int dx = abs(y2 - y11);
int dy = abs(x2 - x1);
//Negative Slope
if (x2 < x1)
{
incx = -1;
}
if (y2 < y11)
{
incy = -1;
}
m = dy / dx;
if (m < 1)
{
plotpoint(x1, y11);
p = 2 * dy - dx;
for (int i = 0; i < dx; i++)
{
x = x + incx;
if (p < 0)
{
p = p + 2 * dy;
y = y;
}
else
{
p = p + 2 * dy - 2 * dx;
y += incy;
}
plotpoint(x, y);
}
}
else
```

```
{
plotpoint(x1, y11);
p = 2 * dx - dy;
for (int i = 0; i < dy; i++)
{
y = y + incy;
if (p < 0)
{
p = p + 2 * dx;
x = x;
}
else
{
p = p + 2 * dx - 2 * dy;
x += incx;
}
plotpoint(x, y);
}
}
}

//Positive Slope: m < 1, m > 2, m = 1
//Negative Slope: m < 1, m > 2, m = 1
void Display()
{
glClear(GL_COLOR_BUFFER_BIT);
glClearColor(1, 1, 1, 1);
LineBres(x1, y11, x2, y2);
glEnd();
glFlush();
}

int main(int argc, char **argv)
```

```
{
printf("Enter 2 points: ");
scanf("%d%d%d%d", &x1, &y11, &x2, &y2);
glutInit(&argc, argv);
glutInitWindowPosition(50, 50);
glutInitWindowSize(500, 500);
glutCreateWindow(".Line.");
init();
glutDisplayFunc(Display);
glutMainLoop();
return 0;
```

P2. Develop a program to demonstrate basic geometric operations on the 2D object

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

float x[3][3] = {{0, 100, 50}, {0, 0, 50}, {1, 1, 1}};
float r[3][3];

void myinit() {
glClearColor(1, 1, 1, 0);
gluOrtho2D(-100, 500, -100, 500);
}

void triangle(float x[3][3]) {
glColor4s(1, 1, 1, 0);
glBegin(GL_TRIANGLES);
glVertex2f(x[0][0], x[1][0]);
glVertex2f(x[0][1], x[1][1]);
```

```
glVertex2f(x[0][2], x[1][2]);
glEnd();
}

void matrixmul(float mul[3][3]) {
for(int i=0; i<3; i++) {
for(int j=0; j<3; j++) {
r[i][j] = 0;
for(int k=0; k<3; k++) {
r[i][j] = r[i][j] + mul[i][k]*x[k][j];
}}}}
void translation() {
float t[3][3] = {{1, 0, 100}, {0, 1, 0}, {0, 0, 1}};
printf("Enter the values of Tx and Ty: ");
scanf("%f %f", &t[0][2], &t[1][2]);
matrixmul(t);
triangle(r);
}

void scaling() {
float s[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
printf("Enter the values of Sx and Sy: ");
scanf("%f %f", &s[0][0], &s[1][1]);
matrixmul(s);
triangle(r);
}

void rotation() {
float theta = 0;
printf("Enter the angle: ");
scanf("%f", &theta);
float angle = theta * 3.14 / 180;
float cosx = cos(angle);
float sinx = sin(angle);
float rr[3][3] = {{cosx, -sinx, 30}, {sinx, cosx, 40}, {0, 0, 1}};
matrixmul(rr);
triangle(r);
}

void displayMe() {
while(1) {
glClear(GL_COLOR_BUFFER_BIT);
glColor3d(1, 0, 0);
int ch;
printf("Enter the choice: \n0: For Normal Triangle \n1:
 For Translation \n2: For Scaling \n3: For Rotation \n4: Exit \n");
scanf("%d", &ch);
glColor3d(1, 1, 1); // Color of the Object
switch(ch) {
case 0: triangle(x);
break;
case 1: translation();
break;
case 2: scaling();
break;
case 3: rotation();
break;
case 4: exit(0);
default: printf("Enter a valid choice!");
}
glColor3d(1, 0, 0);
triangle(x);
glFlush();
}
}
```

```
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE);
glutInitWindowSize(500, 500);
glutCreateWindow("Program-2");
myinit();
glutDisplayFunc(displayMe);
glutMainLoop();
return 0;
}
```

P3. Develop a program to demonstrate basic geometric operations on the 3D object

```
#include<GL/glut.h>
#include<stdlib.h>
#include<stdio.h>
typedef float point[3];
point v[]={
{0.0,0.0,1.0},
{0.0,1.0,0.0},
{-1.0,-1.0,0.0},
{1.0,-1.0,0.0}
};
int n;
void triangle(point a,point b,point c){
glBegin(GL_TRIANGLES);
glVertex3fv(a);
glVertex3fv(b);
glVertex3fv(c);
glEnd();
}
```

```
void divide_tri(point a,point b,point c,int m){
point v1,v2,v3;
int j;
if(m>1){
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]=(c[j]+b[j])/2;
divide_tri(a,v1,v2,m-1);
divide_tri(c,v2,v3,m-1);
divide_tri(b,v3,v1,m-1);
}
else{
triangle(a,b,c);
}
}
void tetrahedron(int m){
glColor3f(1.0,0.0,0.0);
divide_tri(v[0],v[1],v[2],m);
glColor3f(0.0,0.0,0.0);
divide_tri(v[3],v[2],v[1],m);
glColor3f(0.0,1.0,0.0);
divide_tri(v[0],v[3],v[1],m);
glColor3f(0.0,0.0,1.0);
divide_tri(v[0],v[2],v[3],m);
}
void display(){
tetrahedron(n);
glFlush();
}
void myinit(){
```

```
glClearColor(1.0,1.0,1.0,1.0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glOrtho(-2.0,2.0,-2.0,2.0,-2.0,2.0);
}
int main(int argc,char **argv){
printf("\nEnter the number of recursive steps you want:");
scanf("%d",&n);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutCreateWindow("Ex 8: 3d Sierpinski's Gasket");
glutDisplayFunc(display);
myinit();
glEnable(GL_DEPTH_TEST);
glutMainLoop();
return 0;
}
```

P4. Develop a program to demonstrate 2D transformation on basic objects.

```
 #include<GL/glut.h>
#include<stdio.h>
void myinit(){
        gluOrtho2D(-500,500,-500,500);
}

void drawTriangle(){
        glBegin(GL_TRIANGLES);
        glVertex2f(100,100);
        glVertex2f(200,100);
        glVertex2f(150,150);
        glEnd();
}

void translate(){
        glPushMatrix();
        glTranslated(100,0,0);
        drawTriangle();
        glPopMatrix();
}

void rotate_triangle(){
        glPushMatrix();
        glRotated(45,0,0,1);
        drawTriangle();
        glPopMatrix();
}

void pivot_point_rotate(){
        glColor3f(0,0,0);
        glPushMatrix();
        glTranslated(100,100,0);
        glRotated(45,0,0,1);
        glTranslated(-100,-100,0);
        drawTriangle();
        glPopMatrix();
}

void scale_triangle(){
        glPushMatrix();
        glScaled(2,2,1);
        drawTriangle();
        glPopMatrix();
}
```

```
void pivot_point_scale(){
        glColor3f(0,0,0);
        glPushMatrix();
        glTranslated(100,100,0);
        glScaled(2,2,1);
        glTranslated(-100,-100,0);
        drawTriangle();
        glPopMatrix();
}

void display(){
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1,1,1,0);
        glColor3f(1,1,0);
        drawTriangle();
        glFlush();
}


void menu_rotate(int id){
        switch(id) {
                case 1:
                        translate();
                        break;
                case 2:
                        rotate_triangle();
                        break;
                case 3:
                        pivot_point_rotate();
                        break;
                case 4:
                        scale_triangle();
                        break;
```

```
			case 5:
				pivot_point_scale();
				break;
			default:
				exit(0);
		}
}

int main(int argc, char **argv){
		glutInit(&argc, argv);
		glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
		glutInitWindowSize(500,500);
		glutCreateWindow("Transformation");
		myinit();
		glutDisplayFunc(display);
		glutCreateMenu(menu_rotate);

		glutAddMenuEntry("Translate",1);
		glutAddMenuEntry("Rotation about origin",2);
		glutAddMenuEntry("Rotation about fixed point",3);
		glutAddMenuEntry("Scale about origin",4);
		glutAddMenuEntry("Scale about fixed point",5);
		glutAddMenuEntry("EXIT",6);
		glutAttachMenu(GLUT_RIGHT_BUTTON);

		glutMainLoop();
		return 0;}
```

P5. Develop a program to demonstrate 3D transformation on 3D objects

```
 #include<GL/glut.h>
float ambient[] = {1, 1, 1, 1};
float light_pos[] = {2, 2, 2, 1};
void align() {
 glRotated(50, 0, 1, 0);
 glRotated(10, -1, 0, 0);
 glRotated(11, 0, 0, -1);
}

void obj(double tx, double ty, double tz, double sx, double sy, double
sz) {
 align();
 glTranslated(tx, ty, tz);
 glScaled(sx, sy, sz);
 glutSolidCube(1);
 glLoadIdentity();
}

void display() {
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 // three walls
 obj(0, 0, 0.5, 1, 1, 0.04);
 obj(-0.5, 0, 0, -0.04, 1, 1);
 obj(0, -0.5, 0, 1, 0.04, 1);
 // table top
 obj(0.2, -0.2, -0.2, 0.6, 0.04, 0.6);
 // four table legs
 obj(-0.2, -0.4, -0.25, 0.02, 0.2, 0.02);
 obj(-0.16, -0.4, 0.16, 0.02, 0.2, 0.02);
 obj(0.2, -0.4, 0.2, 0.02, 0.2, 0.02);
 obj(0.2, -0.4, -0.2, 0.02, 0.2, 0.02);
 align();
 glTranslated(0.3, -0.1, -0.3);
 glutSolidTeapot(0.09);
 glFlush();
 glLoadIdentity();
}

void main(int argc, char **argv) {
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
 glutInitWindowSize(700, 700);
 glutCreateWindow("Teapot");
 glutDisplayFunc(display);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
 glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
 glEnable(GL_DEPTH_TEST);
 glutMainLoop();
}
```

P6. Develop a program to demonstrate Animation effects on simple objects.

```
#include<gl/glut.h>
float ambient[]={1,1,1,1}; float light_pos[]={27,80,2,3};
void obj(double tx,double ty,double tz,double sx,double sy,double sz)
{
glRotated(50,0,1,0); glRotated(10,-1,0,0);
glRotated(11.7,0,0,-1);
glTranslated(tx,ty,tz); glScaled(sx,sy,sz); glutSolidCube(1);
glLoadIdentity();
}
void display()
{
```

```
glViewport(0,0,700,700);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

obj(0,0,0.5,1,1,0.04); // three walls obj(0,-0.5,0,1,0.04,1);
obj(-0.5,0,0,0.04,1,1);

obj(0,-0.3,0,0.02,0.2,0.02);  // four table legs
obj(0,-0.3,-0.4,0.02,0.2,0.02);
obj(0.4,-0.3,0,0.02,0.2,0.02);
obj(0.4,-0.3,-0.4,0.02,0.2,0.02);
obj(0.2,-0.18,-0.2,0.6,0.02,0.6); // table top glRotated(50,0,1,0);
glRotated(10,-1,0,0);
glRotated(11.7,0,0,-1);
glTranslated(0.3,-0.1,-0.3); glutSolidTeapot(0.09);// tea pot glFlush();
glLoadIdentity();
}
void main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(700,700);
glutCreateWindow("Teapot");
glutDisplayFunc(display);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
}
```

#image split cg7

```
import cv2
import matplotlib.pyplot as plt

image=cv2.imread('/content/car_image.jpg')
image_mat=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)

ht,wd,_=image.shape
img_x,img_y=wd//2,ht//2

tl=image_mat[:img_y,:img_x]
tr=image_mat[:img_y,img_x:]
bl=image_mat[img_y:,:img_x]
br=image_mat[img_y:,img_x:]

plt.figure(figsize=(10,10))

plt.subplot(2,2,1)
plt.imshow(tl)
plt.title("top_left")
plt.axis('off')

plt.subplot(2,2,2)
plt.imshow(tr)
plt.title("top_right")
plt.axis('off')

plt.subplot(2,2,3)
plt.imshow(bl)
plt.title("bottom_left")
plt.axis('off')

plt.subplot(2,2,4)
plt.imshow(br)
plt.title("bottom_right")
plt.axis('off')
```

#translate,scale,rotate & cg8

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

image=cv2.imread('/content/car_image.jpg')
image_mat=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)

ht,wd,c=image.shape
center=(ht/2,wd/2)

trans=np.float32([[1,0,50],[0,1,50]])
scale=np.float32([[2,0,0],[0,2,0]])
rotate=cv2.getRotationMatrix2D(center,30,1)

tr=cv2.warpAffine(image_mat,trans,(wd,ht))
sc=cv2.warpAffine(image_mat,scale,(wd*2,ht*2))
r=cv2.warpAffine(image_mat,rotate,(wd,ht))

img=[image_mat,tr,sc,r]
img_t=["Original","translate","scale","rotate"]

fig,axs=plt.subplots(1,4)
fig.tight_layout(pad=1.0)

for i in range(4):
  axs[i].imshow(img[i])
  axs[i].set_title(img_t[i])
plt.show()
```

#edge,texture using filtering,cg9

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

image = cv2.imread('/content/car_image.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, -1]])
sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

edge_x = cv2.filter2D(image_gray, -1, sobel_x)
edge_y = cv2.filter2D(image_gray, -1, sobel_y)

edges = cv2.addWeighted(edge_x, 0.5, edge_y, 0.5, 0)
edges_rgb=cv2.cvtColor(edges,cv2.COLOR_BGR2RGB)

sobel_x = cv2.Sobel(image_gray, cv2.CV_64F, 1, 0, ksize=5)
sobel_y = cv2.Sobel(image_gray, cv2.CV_64F, 0, 1, ksize=5)
texture = sobel_x + sobel_y

images = [image_rgb, edges_rgb, texture]
titles = ["Original", "Edge", "Texture"]

fig, axs = plt.subplots(1, 3, figsize=(15, 5))

for i in range(3):
   axs[i].imshow(images[i])
   axs[i].set_title(titles[i])
   axs[i].axis('off')

plt.show()
```

##blur,smooth,cg10

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim

image = cv2.imread('/content/car_image.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

kernel_size = 9
blurr_size = np.ones((kernel_size, kernel_size),
dtype=np.float32) / (kernel_size * kernel_size)

blur_img = cv2.filter2D(image_rgb, -1, blurr_size)

smoothed_kernel = np.array([[2, 2, 2], [2, 10, 2], [2, 2, 2]])
smooth_img = cv2.filter2D(image_rgb, -1, smoothed_kernel)

gray_original = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)
smooth_original = cv2.cvtColor(smooth_img, cv2.COLOR_RGB2GRAY)

ssim_score_blurred, _ = ssim(image_gray, gray_original, full=True)
ssim_score_smooth, _ = ssim(image_gray, smooth_original, full=True)

print(f"SSIM Score for Blurred Image: {ssim_score_blurred:.4f}")
print(f"SSIM Score for Smooth Image: {ssim_score_smooth:.4f}")

fig, axs = plt.subplots(1, 3, figsize=(15, 5))

images = [image_rgb, blur_img, smooth_img]
titles = ["Original", "Blurred", "Smoothed"]
```

```
for i in range(3):
    axs[i].imshow(images[i])
    axs[i].set_title(titles[i])
    axs[i].axis('off')

plt.show()
```

#contour_image_cg11

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

image_path = '/content/chess.jpg'
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
contoured_image = image_rgb.copy()
cv2.drawContours(contoured_image, contours, -1, (0, 255, 0), 2)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(contoured_image)
plt.title('Contour Detection')
plt.axis('off')
plt.tight_layout()
plt.show()
```

OR

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

image = cv2.imread('/content/car_image.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edged = cv2.Canny(gray, 30, 200)

contours, hierarchy = cv2.findContours(edged.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
print("Number of Contours found = " + str(len(contours)))

cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

cropped_images = []

for i, contour in enumerate(contours):
    mask = np.zeros_like(gray)
    cv2.drawContours(mask, [contour], 0, 255, -1)
    object_extracted = np.zeros_like(image)
    object_extracted[mask == 255] = image[mask == 255]
    object_extracted_rgb = cv2.cvtColor(object_extracted,
cv2.COLOR_BGR2RGB)
    cropped_images.append(object_extracted_rgb)

fig, axs = plt.subplots(1, len(cropped_images)+2, figsize=(12, 4))

axs[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
axs[0].set_title('Original Image')
axs[0].axis('off')

axs[1].imshow(edged, cmap='gray')
axs[1].set_title('Canny Edges')
axs[1].axis('off')

for i in range(len(cropped_images)):
    axs[i+2].imshow(cropped_images[i])
    axs[i+2].set_title(f'Object {i+1}')
    axs[i+2].axis('off')

plt.tight_layout()
plt.show()
```

P12. Write a program to detect a face/s in an image.

```
import cv2

import matplotlib.pyplot as plt


# Load the image

image_path = 'cricket.jpg'

image = cv2.imread(image_path)


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


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))

# Initialize a list to store cropped faces
cropped_faces = []

# Draw rectangles around the detected faces and crop them
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)
    cropped_faces.append(image[y:y+h, x:x+w])

# Display each cropped face separately
plt.figure(figsize=(12, 6))
for i, face in enumerate(cropped_faces):
    plt.subplot(1, len(cropped_faces), i + 1)
    plt.imshow(cv2.cvtColor(face, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title(f'Face {i + 1}')
plt.tight_layout()
plt.show()
```