Computer Graphics

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 <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,0},{sinx,cosx,0},{0,0,1}};
matrixmul(rr);
triangle(r);

}


void displayMe()
{

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\n");
scanf("%d",&ch);
```

```
glColor3d(1,1,1);
switch(ch)
{
case 0:
triangle(x);
break;
case 1:
translation();
break;
case 2:
scaling();
break;
case 3:
rotation();
break;
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("Line Drawing Algorithm");
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>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_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);
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_POLYGON);
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(1,1,0); // yellow
glPushMatrix();
glTranslated(100,100,0); //translate back to the original position
glRotated(45,0,0,1); // Rotate degree 45
glTranslated(-100,-100,0); //translate to Origin
drawtriangle();
glPopMatrix();
```

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

}
void pivot_point_scale()
{   glColor3f(1,1,0); // yellow
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(1,0,0); //Red
drawtriangle();
//glutPostRedisplay();
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);
}
//glutPostRedisplay();
}


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[]={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();
}
```

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

```
#include <GL/glut.h>
float ambient[]={1,0,0,1};
float light_pos[]={2,2,2,1};
static float theta[3] = {0,0,0};
int axis = 0;
int ch=1;
void mouse(int button, int state, int x, int y)
```

```
{
if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
axis = 0;
if(button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
axis = 1;
if(button == GLUT_RIGHT_BUTTON && state == GLUT_UP)
axis = 2;
}
void idle(){
theta[axis] += 2;
if(theta[axis] > 360)
theta[axis] = 0;
glutPostRedisplay();
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glClearColor(1,1,1,1);
glLoadIdentity();
glRotatef(theta[0],1,0,0);  // rotation about x
glRotatef(theta[1],0,1,0); // rotate about y
glRotatef(theta[2],0,0,1); // rotate about z
if(ch==1)
glutSolidCube(1);
if(ch==2)
glutSolidTeapot(0.5);
if(ch==3)
glutSolidCone(0.5,0.5,20,20);
glFlush();
glutSwapBuffers();  // use whenever you use double buffer
}
void menu(int id)
{
switch(id)
{
case 1:
ch=1;
break;
case 2:
ch=2;
break;
case 3:
ch=3;
break;
}
}
int main(int argc, char ** argv)
```

```
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(500,500);
glutCreateWindow("Color Cube");
glutCreateMenu(menu);
glutAddMenuEntry("Cube",1);
glutAddMenuEntry("Teapot",2);
glutAddMenuEntry("Cone",3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutDisplayFunc(display);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
glutMouseFunc(mouse);
glutIdleFunc(idle);
glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
return 0;
}
```

IMAGE PROCESSING

P7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

```
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')
```

P8. Write a program to show rotation, scaling, and translation on an image.

```
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()
```

P9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.

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

# Load image and convert to grayscale
image = cv2.cvtColor(cv2.imread('/content/Chess.png'), cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# Sobel edge detection
sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)
sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)
edges = cv2.addWeighted(np.abs(sobel_x), 0.5, np.abs(sobel_y), 0.5, 0)

# Display images
titles = ["Original", "Edges", "Texture"]
images = [image, edges, sobel_x + sobel_y]
for i in range(3):
    plt.subplot(1, 3, i + 1)
    plt.imshow(images[i], cmap='gray' if i != 0 else None)
    plt.title(titles[i])
    plt.axis('off')
plt.show()
```

P10. Write a program to blur and smoothing an image.

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

# Load image and convert to RGB
img = cv.imread('/content/apple.jpeg')
img_mat = img[:, :, ::-1]

# Apply blurring and smoothing
blurred = cv.blur(img_mat, (5, 5))
smoothed = cv.GaussianBlur(img_mat, (5, 5), 0)

# Create a list of images to display
images = [img_mat, blurred, smoothed]
titles = ["Original", "Blurred", "Smoothed"]

# Plot images
plt.figure(figsize=(12, 4))
for i in range(3):
    plt.subplot(1, 3, i + 1)
    plt.imshow(images[i])
    plt.title(titles[i])
```

```
    plt.axis('off')

plt.show()
```

P11. Write a program to contour an image.

```
import cv2 as cv
import matplotlib.pyplot as plt

# Load and convert image
image = cv.imread('Chess.png')
img_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

# Apply Canny edge detection
edges = cv.Canny(img_gray, 100, 200)

# Find and draw contours
contours, _ = cv.findContours(edges, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(image, contours, -1, (0, 0, 255), 2)

# Display the original and contoured images
plt.subplot(1, 2, 1)
plt.imshow(cv.cvtColor(cv.imread('Chess.png'), cv.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
plt.title('Contour Detection')
plt.axis('off')

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

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

```
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread('/content/face.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cascade = cv.CascadeClassifier(cv.data.haarcascades +
'haarcascade_frontalface_default.xml')
faces = cascade.detectMultiScale(gray)
```

```
for (x, y, w, h) in faces:
    cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)

img_rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)

plt.imshow(img_rgb)
plt.axis('off')
plt.show()
```