Computer Graphics

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

```
#include <GL/glut.h>
void myinit()
{
   glClear( GL_COLOR_BUFFER_BIT );
   glClearColor( 0, 0, 0, 1 );
   gluOrtho2D(0,500,0,500);
}
void draw_pixel(int x,int y)
{
   glBegin(GL_POINTS);
      glVertex2d(x,y);
   glEnd();
}
void bresenhams(int x1,int y1,int x2,int y2)
{
   int dx,dy,x,y,p0,p,i,incx=1,incy=1;
   dx=abs(x2-x1);
   dy=abs(y2-y1);
   if(x2<x1)incx=-1;
   if(y2<y1)incy=-1;
   x=x1;
   y=y1;
```

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

```
            y=y+incy;
            if(p>=0)
            {
               x=x+incx;
               p=p+(2*dx-2*dy);
            }
            else
            {
               x=x;
               p=p+2*dx;
            }
            draw_pixel(x,y);
         }
      }
}
void display()
{
   glColor3f( 1, 0, 0 );
   bresenhams(20,20,300,50); //Slope <1
   bresenhams(20,20,50,300); //slope >1
   bresenhams(20,20,300,300); //slope=1
   bresenhams(50,300,20,20);  //Negative slope >1
   bresenhams(300,50,20,20);  // Negative slope <1
   glFlush();
}
int main( int argc, char** argv )
```

```
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(100,100);
   glutCreateWindow("Triangle");
   glutDisplayFunc(display);
   myinit();
   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;
}
```

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

```
import cv2 as cv
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
matplotlib.rc('figure',figsize=[15,5])
img = cv.imread('puppy.png')
img_gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
h,w,channels=img.shape
print(h,w,channels)
```

```
img_rgb=img[:,:,::-1]
# plt.imshow(img_rgb)
cy=h//2
cx=w//2
tr=img[0:cy,0:cx]
tl=img[0:cy,cx:w]
br=img[cy:h,0:cx]
bl=img[cy:h,cx:w]
plt.subplot(221);plt.imshow(tr)
plt.subplot(222);plt.imshow(tl)
plt.subplot(223);plt.imshow(br)
plt.subplot(224);plt.imshow(bl)
plt.show()
```

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

```
import cv2 as cv
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
matplotlib.rc('figure',figsize=[10,8])
img = cv.imread('puppy.png')
img_rgb=img[:,:,::-1]
a=cv.rotate(img,cv.ROTATE_90_CLOCKWISE)
b=cv.rotate(img,cv.ROTATE_90_COUNTERCLOCKWISE)
c=cv.rotate(img,cv.ROTATE_180)
h,w,channels=img.shape
d=cv.resize(img_rgb,(w//2,h//2))
```

```
e=cv.resize(img_rgb,None,fx=2,fy=2,interpolation=cv.INTER_LINEAR)
f=cv.resize(img_rgb,None,fx=0.5,fy=2)
tx=100;ty=50
M=np.float32([[1,0,tx],[0,1,ty]])
g=cv.warpAffine(img_rgb,M,(w,h))
list=[a,b,c,d,e,f,g]
for i in range(len(list)):
   plt.subplot(3,3,i+1)
   plt.imshow(list[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 cv2 as cv
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
img_messi=cv.imread('messi.jpeg')
img_messi_gray=cv.cvtColor(img_messi,cv.COLOR_BGR2GRAY)
img_messi_canny=cv.Canny(img_messi_gray,threshold1=100,threshold2=200)
img_chess=cv.imread('chessboard.png')
img_chess_gray=cv.cvtColor(img_chess,cv.COLOR_BGR2GRAY)
img_float=np.float32(img_chess_gray)
dest=cv.cornerHarris(img_float,2,5,0.07)
img_chess[dest>0.01*dest.max()]=[0,0,255]
list=[img_messi,img_messi_canny,img_chess]
for i in range(3):
   plt.subplot(2,2,i+1)
   plt.imshow(list[i])
plt.show()
```

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

```
import cv2 as cv
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
img_messi=cv.imread('messi.jpeg')
img_messi_rgb=img_messi[:,:,::-1]
a=cv.blur(img_messi_rgb,(5,5))
b=cv.GaussianBlur(img_messi_rgb,(5,5),0)
l=[img_messi_rgb,a,b]
for i in range(3):
 plt.subplot(2,2,i+1)
 plt.imshow(a)
plt.show()
```

P11. Write a program to contour an image.

```
import cv2 as cv
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
img_chess=cv.imread('chessboard.png')
w,h,channels=img_chess.shape
img_chess_gray=cv.cvtColor(img_chess,cv.COLOR_BGR2GRAY)
img_canny=cv.Canny(img_chess_gray,threshold1=100,threshold2=200)
c,h=cv.findContours(img_canny,cv.RETR_TREE,cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(img_chess,c,-1,(0,0,255),3)
plt.imshow(img_chess)
plt.show()
```

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

```
import cv2 as cv
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
img_faces=cv.imread('faces.png')
img_face_gray=cv.cvtColor(img_faces,cv.COLOR_BGR2GRAY)
face_cascade=cv.CascadeClassifier(cv.data.haarcascades+'haarcascade_frontal
face_default.xml')
faces=face_cascade.detectMultiScale(img_face_gray,1.3,2)
for x,y,w,h in faces:
   cv.rectangle(img_faces,(x,y),(x+w,y+h),(0,0,255),3)
img_faces=img_faces[:,:,::-1]
plt.imshow(img_faces)
plt.show()
```