**TW-1: Implement Brenham’s line drawing algorithm for all types of slope.**

#include<GL/glut.h>

#include <stdlib.h>

#include<math.h>

void init(void)

{

//set display-window background color to white

glClearColor(1.0,1.0,1.0,0.0);

//set projection paramaters

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0.0,300.0,0.0,300.0);

}

void setPixel(GLint xCoordinate, GLint yCoordinate)

{

glBegin(GL\_POINTS);

glVertex2i(xCoordinate,yCoordinate);

glEnd();

glFlush(); //executes all OpenGL functions as quickl y as possible

}

//Bresenham line-drawing procedure for |m| < 1.0

void lineBres(GLint x0, GLint y0, GLint xEnd, GLint

yEnd)

{

GLint dx = fabs(xEnd - x0);

GLint dy = fabs(yEnd - y0);

GLint p = 2 \* dy - dx;

GLint twoDy = 2 \* dy;

GLint twoDyMinusDx = 2 \* (dy-dx);

GLint x,y;

// determine which endpoint to use as start position

if (x0 > xEnd){

x = xEnd;

y = yEnd;

xEnd = x;

}else{

x = x0;

y = y0;

}

setPixel(x,y);

while(x<xEnd){

x++;

if(p<0)

p += twoDy;

else{

y++;

p += twoDyMinusDx;

}

setPixel(x,y);

}

}

void drawMyLine(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0,0.0,0.0);

glPointSize(4.0);

GLint x0 = 100;

GLint y0 = 150;

GLint xEnd = 200;

GLint yEnd = 200;

lineBres(x0,y0,xEnd,yEnd);

}

void main(int argc, char\*\*argv)

{

//initialize GLUT

glutInit(&argc,argv);

//initialize display mode

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

//set display-window width & height

glutInitWindowSize(400,400);

//set display-window upper-left position

glutInitWindowPosition(0,0);

//create display-window with a title

glutCreateWindow("Bresenham");

//initialze OpenGL

init();

//call graphics to be displayed on the window

glutDisplayFunc(drawMyLine);

//display everything and wait

glutMainLoop();

}

**TW-2: Create and rotate a triangle about the origin and a fixed point**

#include<GL/glut.h>

#include<stdio.h>

float v[]={0,0.6,0.25,0.9,0.5,0.6};

GLubyte list[]={0,1,2};

int g=0;

void dis()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(1,1,1,1);

glColor3f(0,0,0);

glViewport(0,0,700,700);

glDrawElements(GL\_LINE\_LOOP,3,GL\_UNSIGNED\_BYTE,list);

glRotated(g,0,0,1);

glDrawElements(GL\_LINE\_LOOP,3,GL\_UNSIGNED\_BYTE,list);

glFlush();

glLoadIdentity();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

printf("Enter Rotation Angle:");

scanf("%d",&g);

glutCreateWindow("Triangle Rotation");

glutDisplayFunc(dis);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(2,GL\_FLOAT,0,v);

glutMainLoop();

}

**TW-3: Draw a colour cube and spin it using OpenGL transformation matrices.**

#include<GL/glut.h>

float v[]={-0.5,-0.5,-0.5, -0.5,0.5,-0.5, 0.5,0.5,-0.5,

0.5,-0.5,-0.5, -0.5,-0.5,0.5, -0.5,0.5,0.5,

0.5,0.5,0.5, 0.5,-0.5,0.5};

float c[]={0,0,0, 1,0,0, 1,1,0, 0,1,0, 0,0,1, 1,0,1, 1,1,1, 0,1,1,};

GLubyte d[]={0,1,2,3, 2,3,7,6, 4,5,6,7, 4,5,1,0, 5,6,2,1, 0,3,7,4};

int gx=0,gy=0,gz=1.0;

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glRotated(0.2,gx,gy,gz);

glDrawElements(GL\_QUADS,24,GL\_UNSIGNED\_BYTE,d);

glFlush();

}

void mouse(int btn,int state,int x,int y)

{

if(btn==GLUT\_LEFT\_BUTTON&&state==GLUT\_DOWN) {gx=1; gy=0; gz=0;}

if(btn==GLUT\_MIDDLE\_BUTTON&&state==GLUT\_DOWN) {gx=0; gy=1; gz=0;}

if(btn==GLUT\_RIGHT\_BUTTON&&state==GLUT\_DOWN) {gx=0; gy=0; gz=1;}

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

glutCreateWindow("cube");

glutDisplayFunc(display);

glutIdleFunc(display);

glutMouseFunc(mouse);

glEnableClientState(GL\_VERTEX\_ARRAY);

glEnableClientState(GL\_COLOR\_ARRAY);

glVertexPointer(3,GL\_FLOAT,0,v);

glColorPointer(3,GL\_FLOAT,0,c);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**TW-4: Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.**

#include <GL/glut.h>

float v[]={-0.5,-0.5,-0.5, -0.5,0.5,-0.5, 0.5,0.5,-0.5,

0.5,-0.5,-0.5, -0.5,-0.5,0.5, -0.5,0.5,0.5,

0.5,0.5,0.5, 0.5,-0.5,0.5};

float c[]={0,0,0, 0,1,0, 1,1,0, 1,0,0, 0,0,1, 0,1,1, 1,1,1, 1,0,1,};

GLubyte list[]={0,1,2,3, 6,7,3,2, 4,7,6,5, 4,5,1,0, 5,6,2,1, 0,3,7,4};

int gx=0,gy=0,gz=1;

static GLfloat theta[] = {0.0,0.0,0.0};

static GLint axis = 2;

static GLdouble viewer[]= {0.0, 0.0, 5.0};

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

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,list);

glFlush();

}

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] += 2.0;

if( theta[axis] > 360.0 ) theta[axis] -= 360.0;

display();

}

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

{ 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();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(700, 700);

glutCreateWindow("Colorcube Viewer");

glMatrixMode(GL\_PROJECTION);

glFrustum(-2.0, 2.0, -2.0, 2.0, 2.0, 20.0);

glMatrixMode(GL\_MODELVIEW);

glutDisplayFunc(display);

glEnableClientState(GL\_VERTEX\_ARRAY);

glEnableClientState(GL\_COLOR\_ARRAY);

glVertexPointer(3,GL\_FLOAT,0,v);

glColorPointer(3,GL\_FLOAT,0,c);

glutMouseFunc(mouse);

glutKeyboardFunc(keys);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**TW-5: Clip a lines using Cohen-Sutherland algorithm**

#include <GL/glut.h>

double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries

double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries

//bit codes for the right, left, top, & bottom

const int RIGHT = 8;

const int LEFT = 2;

const int TOP = 4;

const int BOTTOM = 1;

//function to compute bit codes of a point

//Compute the bit code for a point (x, y) using the clip rectangle bounded diagonally by (xmin, ymin), and (xmax, ymax)

int ComputeOutCode (double x, double y)

{

int code = 0;

if (y > ymax) //above the clip window

code |= TOP;

else if (y < ymin) //below the clip window

code |= BOTTOM;

if (x > xmax) //to the right of clip window

code |= RIGHT;

else if (x < xmin) //to the left of clip window

code |= LEFT;

return code;

}

/\*Cohen-Sutherland clipping algorithm clips a line from P0 = (x0, y0) to P1 = (x1, y1) against a rectangle \*/

void CohenSutherlandLineClipAndDraw (double x0, double y0,double x1, double y1)

{

//Outcodes for P0, P1 and the point that lies outside the clip rectangle

int outcode0, outcode1, outcodeOut;

bool accept = false, done = false;

//compute outcodes

outcode0 = ComputeOutCode (x0, y0);

outcode1 = ComputeOutCode (x1, y1);

do{

if (!(outcode0 | outcode1)) //logical OR is 0: Trivially accept & exit

{

accept = true;

done = true;

}

else if (outcode0 & outcode1) //logical AND is not 0: Trivially reject and exit

done = true;

else

{

//calculate the line segment to clip from an outside point to an intersection with clip edge

double x, y;

//At least one endpoint is outside the clip rectangle; pick it.

outcodeOut = outcode0? outcode0: outcode1;

//Now find the intersection point with formula: y = y0 + slope \* (x - x0), x = x0 + (1/slope)\* (y - y0)

if (outcodeOut & TOP) //point is above the clip rectangle

{

x = x0 + (x1 - x0) \* (ymax - y0)/(y1 - y0);

y = ymax;

}

else if (outcodeOut & BOTTOM) //point is below the clip rectangle

{

x = x0 + (x1 - x0) \* (ymin - y0)/(y1 - y0);

y = ymin;

}

else if (outcodeOut & RIGHT) //point is to the right of clip rectangle

{

y = y0 + (y1 - y0) \* (xmax - x0)/(x1 - x0);

x = xmax;

}

else //point is to the left of clip rectangle

{

y = y0 + (y1 - y0) \* (xmin - x0)/(x1 - x0);

x = xmin;

}

//Now we move outside point to intersection point to clip

//and get ready for next pass.

if (outcodeOut == outcode0)

{

x0 = x;

y0 = y;

outcode0 = ComputeOutCode (x0, y0);

}

else

{

x1 = x;

y1 = y;

outcode1 = ComputeOutCode (x1, y1);

}

}

}while (!done);

if (accept)

{ // Window to viewport mappings

double sx=(xvmax-xvmin)/(xmax-xmin); // Scale parameters

double sy=(yvmax-yvmin)/(ymax-ymin);

double vx0=xvmin+(x0-xmin)\*sx;

double vy0=yvmin+(y0-ymin)\*sy;

double vx1=xvmin+(x1-xmin)\*sx;

double vy1=yvmin+(y1-ymin)\*sy;

//draw a red colored viewport

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xvmin, yvmin);

glVertex2f(xvmax, yvmin);

glVertex2f(xvmax, yvmax);

glVertex2f(xvmin, yvmax);

glEnd();

glColor3f(0.0,0.0,1.0); // draw blue colored clipped line

glBegin(GL\_LINES);

glVertex2d (vx0, vy0);

glVertex2d (vx1, vy1);

glEnd();

}

}

void display()

{

double x0=60,y0=20,x1=80,y1=120;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2d (x0, y0);

glVertex2d (x1, y1);

glEnd();

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin, ymin);

glVertex2f(xmax, ymin);

glVertex2f(xmax, ymax);

glVertex2f(xmin, ymax);

glEnd();

CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);

glFlush();

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,499.0,0.0,499.0);

glMatrixMode(GL\_MODELVIEW);

}

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

{

//int x1, x2, y1, y2;

//printf("Enter End points:");

//scanf("%d%d%d%d", &x1,&x2,&y1,&y2);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("Cohen Suderland Line Clipping Algorithm");

glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**TW-6: To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.**

#include<GL/glut.h>

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(-10,0,0,1);

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(0,0,0.5,1,1,0.05); // right wall

obj(0,-0.5,0,1,0.05,1); // bottom wall

obj(-0.5,0,0,0.05,1,1); // left wall

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(-10,0,0,1);

glTranslated(0.3,-0.1,-0.3);

glutSolidTeapot(0.1);

glFlush();

glLoadIdentity();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

float ambient[]={1,1,1,1}; //ambient surface in homogeneous: uniform illumination throughout the room

float light\_pos[]={27,80,2,0}; //distant light source

glutInitWindowSize(700,700);

glutCreateWindow("scene");

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();

}

**TW-7: Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.**

#include<GL/glut.h>

#include<stdio.h>

typedef float point[3];

point v[]={{0.0,0.0,-1.0},{0.0,0.9428,-0.3333},{-0.8164,-0.4714,-0.3333},{0.8164,-0.4714,-0.5}};

int n;

void tri(point a,point b,point c)

{ glBegin(GL\_POLYGON);

glVertex3fv(a);

glVertex3fv(b);

glVertex3fv(c);

glEnd();

}

void divide\_tri(point a,point b,point c,int m)

{ point v1,v2,v3;

if(m>0)

{

for(int j=0;j<3;j++)

{

v1[j]=(a[j]+b[j])/2;

v2[j]=(a[j]+c[j])/2;

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 tri(a,b,c);

}

void tet(int m)

{ glColor3f(1,0,0);

divide\_tri(v[0],v[1],v[2],m);

glColor3f(0,1,0);

divide\_tri(v[3],v[2],v[1],m);

glColor3f(0,0,1);

divide\_tri(v[0],v[3],v[1],m);

glColor3f(0,0,0);

divide\_tri(v[0],v[2],v[3],m);

}

void display()

{ glViewport(0,0,700,700);

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

tet(n);

glFlush();

glLoadIdentity();

}

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

{

printf("Enter Divisions: ");

scanf("%d",&n);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(700,700);

glutCreateWindow("Gasket");

glEnable(GL\_DEPTH\_TEST);

glutDisplayFunc(display);

glClearColor(1,1,1,1);

glutMainLoop();

}

**TW-8: Develop a menu driven program to animate a flag using Bezier Curve algorithm**

#include<GL/glut.h>

#include<stdio.h>

#include<math.h>

#define PI 3.1416

GLsizei winWidth = 600, winHeight = 600;

GLfloat xwcMin = 0.0, xwcMax = 130.0;

GLfloat ywcMin = 0.0, ywcMax = 130.0;

typedef struct wcPt3D

{

GLfloat x, y, z;

};

void bino(GLint n, GLint \*C)

{

GLint k, j;

for(k=0;k<=n;k++)

{

C[k]=1;

for(j=n;j>=k+1; j--)

C[k]\*=j;

for(j=n-k;j>=2;j--)

C[k]/=j;

}

}

void computeBezPt(GLfloat u, wcPt3D \*bezPt, GLint nCtrlPts, wcPt3D \*ctrlPts, GLint

\*C)

{

GLint k, n=nCtrlPts-1;

GLfloat bezBlendFcn;

bezPt ->x =bezPt ->y = bezPt->z=0.0;

for(k=0; k< nCtrlPts; k++)

{

bezBlendFcn = C[k] \* pow(u, k) \* pow( 1-u, n-k);

bezPt ->x += ctrlPts[k].x \* bezBlendFcn;

bezPt ->y += ctrlPts[k].y \* bezBlendFcn;

bezPt ->z += ctrlPts[k].z \* bezBlendFcn;

}

}

void bezier(wcPt3D \*ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)

{

wcPt3D bezCurvePt;

GLfloat u;

GLint \*C, k;

C= new GLint[nCtrlPts];

bino(nCtrlPts-1, C);

glBegin(GL\_LINE\_STRIP);

for(k=0; k<=nBezCurvePts; k++)

{

u=GLfloat(k)/GLfloat(nBezCurvePts);

computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);

glVertex2f(bezCurvePt.x, bezCurvePt.y);

}

glEnd();

delete[]C;

}

void displayFcn()

{

GLint nCtrlPts = 4, nBezCurvePts =20;

static float theta = 0;

wcPt3D ctrlPts[4] = {

{20, 100, 0},

{30, 110, 0},

{50, 90, 0},

{60, 100, 0}};

ctrlPts[1].x +=10\*sin(theta \* PI/180.0);

ctrlPts[1].y +=5\*sin(theta \* PI/180.0);

ctrlPts[2].x -= 10\*sin((theta+30) \* PI/180.0);

ctrlPts[2].y -= 10\*sin((theta+30) \* PI/180.0);

ctrlPts[3].x-= 4\*sin((theta) \* PI/180.0);

ctrlPts[3].y += sin((theta-30) \* PI/180.0);

theta+=0.1;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 1.0);

glPointSize(5);

glPushMatrix();

glLineWidth(5);

glColor3f(255/255, 153/255.0, 51/255.0); //Indian flag: Orange color code

for(int i=0;i<8;i++)

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glColor3f(1, 1, 1); //Indian flag: white color code

for(int i=0;i<8;i++)

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glColor3f(19/255.0, 136/255.0, 8/255.0); //Indian flag: green color code

for(int i=0;i<8;i++)

{

glTranslatef(0, -0.8, 0);

bezier(ctrlPts, nCtrlPts, nBezCurvePts);

}

glPopMatrix();

glColor3f(0.7, 0.5,0.3);

glLineWidth(5);

glBegin(GL\_LINES);

glVertex2f(20,100);

glVertex2f(20,40);

glEnd();

glFlush();

glutPostRedisplay();

glutSwapBuffers();

}

void winReshapeFun(GLint newWidth, GLint newHeight)

{

glViewport(0, 0, newWidth, newHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowPosition(50, 50);

glutInitWindowSize(winWidth, winHeight);

glutCreateWindow("Bezier Curve");

glutDisplayFunc(displayFcn);

glutReshapeFunc(winReshapeFun);

glutMainLoop();

}

**TW-9: Develop a menu driven program to fill the polygon using scan line algorithm**

#define BLACK 0

#include <stdlib.h>

#include <stdio.h>

#include <GL/glut.h>

float x1,x2,x3,x4,y1,y2,y3,y4;

void edgedetect(float x1,float y1,float x2,float y2,int \*le,int \*re)

{

float mx,x,temp;

int i;

if((y2-y1)<0)

{

temp=y1;y1=y2;y2=temp;

temp=x1;x1=x2;x2=temp;

}

if((y2-y1)!=0)

mx=(x2-x1)/(y2-y1);

else

mx=x2-x1;

x=x1;

for(i=y1;i<=y2;i++)

{

if(x<(float)le[i])

le[i]=(int)x;

if(x>(float)re[i])

re[i]=(int)x;

x+=mx;

}

}

void draw\_pixel(int x,int y,int value)

{

glColor3f(1.0,1.0,0.0);

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

void scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)

{

int le[500],re[500];

int i,y;

for(i=0;i<500;i++)

{

le[i]=500;

re[i]=0;

}

edgedetect(x1,y1,x2,y2,le,re);

edgedetect(x2,y2,x3,y3,le,re);

edgedetect(x3,y3,x4,y4,le,re);

edgedetect(x4,y4,x1,y1,le,re);

for(y=0;y<500;y++)

{

if(le[y]<=re[y])

for(i=(int)le[y];i<(int)re[y];i++)

draw\_pixel(i,y,BLACK);

}

}

void display()

{

x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=300.0;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(x1,y1);

glVertex2f(x2,y2);

glVertex2f(x3,y3);

glVertex2f(x4,y4);

glEnd();

scanfill(x1,y1,x2,y2,x3,y3,x4,y4);

glFlush();

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,499.0,0.0,499.0);

glMatrixMode(GL\_MODELVIEW);

}

void main(int argc, char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

glutCreateWindow("Filling a Polygon using Scan-line Algorithm");

glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**PART –B ( MINI-PROJECT) :**

Student should develop mini project on the topics mentioned below or similar applications using Open GL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project.