**EXPERIMENT DETAILS**

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

**Bresenham's Line-Drawing Algorithm.**

**Step 1** - Input the two end-points of line, storing the left end-point in (x0,y0).

**Step 2** - Plot the point (x0,y0).

**Step 3** - Calculate the constants dx, dy, 2dy, and (2dy – 2dx) and get the first value for the decision parameter as - p0=2dy-dx

**Step 4** - At each Xk along the line, starting at k = 0, perform the following test - If pk< 0, the next point to plot is (xk+1,yk) and pk+1=pk+2dy Otherwise, (xk,yk+1) pk+1=pk+2dy-2dx

**Step 5** - Repeat step 4 (dx – 1) times.

For m > 1, find out whether you need to increment x while incrementing y each time. After solving, the equation for decision parameter Pk will be very similar, just the x and y in the equation gets interchanged.

**Program:**

#include<math.h>

#include<stdio.h>

#include<GL/glut.h>

int x1, y11, x2, y2,dx,dy;

void display();

void init();

void bresenhams(int,int,int,int);

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

{

glutInit(&argc,argv);

printf("enter the end points of the line");

scanf("%d%d%d%d", &x1, &y11, &x2, &y2);

glutCreateWindow("Bresenhams Line Drawing");

init();

glutDisplayFunc(display);

glutMainLoop();

}

void init()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-500, 500, -500, 500);

glMatrixMode(GL\_MODELVIEW);

}

void display()

{

glClearColor(1, 1, 1, 0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1, 0, 0);

bresenhams(x1, y11, x2, y2);

glFlush();

}

void plotline(int x, int y)

{

glPointSize(2);

glBegin(GL\_POINTS);

glVertex2f(x,y);

glEnd();

}

void bresenhams(int x1, int y11, int x2, int y2)

{

int dx, dy,pk,xinc,yinc,x,y;

dx = x2 - x1;

dy = y2 - y11;

x = x1, y = y11;

plotline(x, y);

if (dx > 0)

xinc = 1;

else

xinc = -1;

if (dy > 0)

yinc = 1;

else

yinc = -1;

if (fabs(dx) > fabs(dy))

{

pk = 2 \* fabs(dy) - fabs(dx);

for (int i = 0; i <= fabs(dx) - 1; i++)

{

if (pk > 0)

{

pk = pk + 2 \* fabs(dy) - 2 \* fabs(dx);

y = y+yinc;

}

else

{

pk = pk + 2 \* fabs(dy);

y = y;

}

x = x + xinc;

plotline(x, y);

}

}

else

{

pk = 2 \* fabs(dx) - fabs(dy);

for (int i = 0; i <= fabs(dy) - 1; i++)

{

if (pk > 0)

{

pk = pk + 2 \* fabs(dx) - 2 \* fabs(dy);

x = x + xinc;

}

else

{

pk = pk + 2 \* fabs(dx);

x = x;

}

y = y + yinc;

plotline(x, y);

}

}

}

**STEPS TO EXECUTE**

1 gedit bresen.c

2 gcc bresen.c -lGL -lGLU -lglut

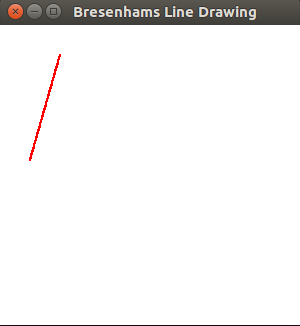
3 ./a.out

**Output 1:**

enter the end points of the line

-400 50

-300 400

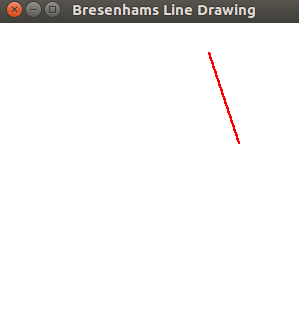


**Output 2:**

enter the end points of the line

200 400

300 100

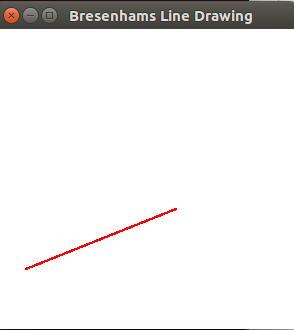


**Output 3:**

enter the end points of the line

-400 -300

100 -100

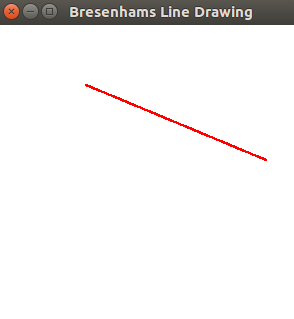


**Output 4:**

enter the end points of the line

400 50

-200 300

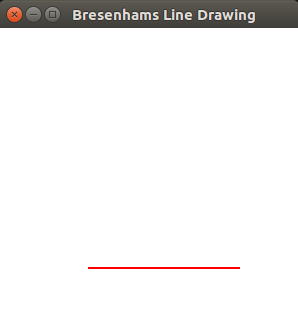


**Output 5:**

enter the end points of the line

-200 -300

300 -300

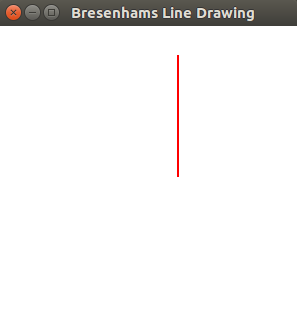


**Output 6:**

enter the end points of the line

100 0

100 400



**2. Create and rotate a triangle about the origin and a fixed point.**

**Program:**

#include<stdio.h>

#include<GL/glut.h>

#include<string.h>

float theta;

void triangle(float \*a,float \*b, float \*c)

{

glPolygonMode(GL\_FRONT\_AND\_BACK,GL\_FILL);

glColor3f(0.9,0.1,0.7);

glBegin(GL\_TRIANGLES);

glVertex2fv(a);

glVertex2fv(b);

glVertex2fv(c);

glEnd();

glPolygonMode(GL\_FRONT\_AND\_BACK,GL\_LINE);

glLineWidth(4);

glColor3f(0.0,0.0,1.0);

glBegin(GL\_TRIANGLES);

glVertex2fv(a);

glVertex2fv(b);

glVertex2fv(c);

glEnd();

}

void init()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-100, 100, -100, 100);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void display()

{

float tri\_fix[3][2]={{-80,-60},{-20,-60},{-40,-20}};

float tri\_org[3][2]={{40,20},{80,20},{60,60}};

glClearColor(1.0,1.0,1.0,1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glEnable(GL\_POLYGON\_STIPPLE);

/\*rotation about origin\*/

triangle(tri\_org[0],tri\_org[1], tri\_org[2]);

glPushMatrix();

glRotatef(theta,0.0,0.0,1.0);

triangle(tri\_org[0],tri\_org[1], tri\_org[2]);

glPopMatrix();

char s[]="Rotation about origin";

glRasterPos2f(-15,10);

for(int i=0;i<strlen(s);i++)

{

glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13,s[i]);

}

/\*rotation about fixed point\*/

triangle(tri\_fix[0],tri\_fix[1], tri\_fix[2]);

glPushMatrix();

glTranslatef(-40,-20,0);

glRotatef(theta,0.0,0.0,1.0);

glTranslatef(40,20,0);

triangle(tri\_fix[0],tri\_fix[1], tri\_fix[2]);

glPopMatrix();

char s1[]="Rotation about fixed point";

glRasterPos2f(-80,-90);

for(int i=0;i<strlen(s1);i++)

{

glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13,s1[i]);

}

glFlush();

}

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

{

printf("Enter the rotation angle\n");

scanf("%f",&theta);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(300,300);

glutInitWindowPosition(10,10);

glutCreateWindow("Triangle Rotation");

init();

glutDisplayFunc(display);

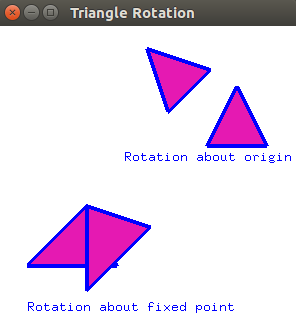
glutMainLoop();

}

**Output 1:**

Enter the rotation angle

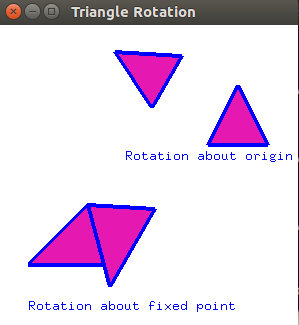
45



**Output 2:**

Enter the rotation angle

60



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

**Program:**

#include<stdio.h>

#include<stdlib.h>

#include<GL/glut.h>

GLfloat v[8][3]={{-200,-200,200},{200,-200,200},{200,200,200},{-200,200,200},{-200,-200,-200}, {200,-200,-200},{200,200,-200},{-200,200,-200}};

void drawcube(GLfloat \*,GLfloat \*,GLfloat \*,GLfloat \*);

float a=0;

void init()

{

glClearColor(0.0,0.0,0.0,0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-500,500,-500,500,-500,2000);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void display()

{

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

glLoadIdentity();

glRotatef(a,1.0,0.0,1.0);

glColor3f(1.0,0.6,0.3);

drawcube(v[0],v[1],v[2],v[3]);

glColor3f(1.0,0.7,0.3);

drawcube(v[1],v[5],v[6],v[2]);

glColor3f(1.0,0.0,0.0);

drawcube(v[3],v[2],v[6],v[7]);

glColor3f(0.0,1.0,0.0);

drawcube(v[4],v[5],v[1],v[0]);

glColor3f(0.0,0.0,1.0);

drawcube(v[7],v[6],v[5],v[4]);

glColor3f(1.0,1.0,0.3);

drawcube(v[3],v[7],v[4],v[0]);

glFlush();

}

void drawcube(GLfloat \*a,GLfloat \*b,GLfloat \*c,GLfloat \*d)

{

glBegin(GL\_POLYGON);

//glColor3f(1.0,0.2,0.5);

glVertex3fv(a);

//glColor3f(0.9,0.1,0.8);

glVertex3fv(b);

//glColor3f(1.0,1.0,1.0);

glVertex3fv(c);

//glColor3f(0.7,0.8,0.6);

glVertex3fv(d);

glEnd();

}

void spincube()

{

a=a+0.1;

if(a>360)

a-=360;

glutPostRedisplay();

}

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

{

glutInit(&argc,argv);

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

glutInitWindowPosition(10,10);

glutInitWindowSize(500,500);

glutCreateWindow("Spinning Color Cube");

init();

glutDisplayFunc(display);

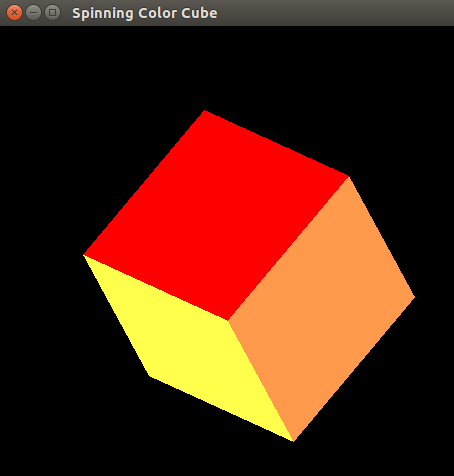
glEnable(GL\_DEPTH\_TEST);

glutIdleFunc(spincube);

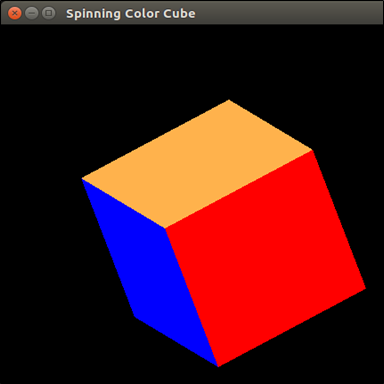
glutMainLoop();

}

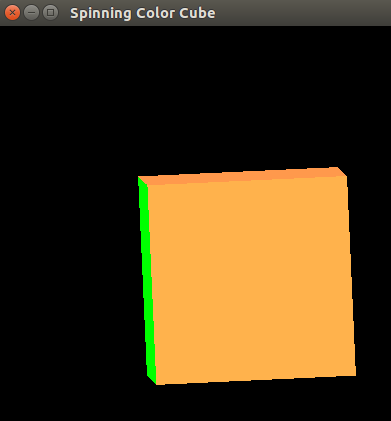
**Output 1:**



**Output 2:**



**Output 3:**



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

**Program:**

#include<stdio.h>

#include<stdlib.h>

#include<GL/glut.h>

GLfloat v[8][3]={{-100,-100,100},{100,-100,100},{100,100,100},{-100,100,100},{-100,-100,-100}, {100,-100,-100},{100,100,-100},{-100,100,-100}};

GLfloat viewer[3]={ 0.0,0.0,500};

void drawcube(GLfloat \*,GLfloat \*,GLfloat \*,GLfloat \*);

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

{

if(k=='x') viewer[0]-=10.0;

if(k=='X') viewer[0]+=10.0;

if(k=='y') viewer[1]+=10.0;

if(k=='Y') viewer[1]-=10.0;

if(k=='z') viewer[2]+=10.0;

if(k=='Z') viewer[2]-=10.0;

glutPostRedisplay();

}

void init()

{

glClearColor(0.0,0.0,0.0,0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(-200,200,-200,200,200,800);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void display()

{

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

glLoadIdentity();

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

glColor3f(1.0,0.6,0.3);

drawcube(v[0],v[1],v[2],v[3]);

glColor3f(1.0,0.7,0.3);

drawcube(v[1],v[5],v[6],v[2]);

glColor3f(1.0,0.0,0.0);

drawcube(v[3],v[2],v[6],v[7]);

glColor3f(0.0,1.0,0.0);

drawcube(v[4],v[5],v[1],v[0]);

glColor3f(0.0,0.0,1.0);

drawcube(v[7],v[6],v[5],v[4]);

glColor3f(1.0,1.0,0.3);

drawcube(v[3],v[7],v[4],v[0]);

glFlush();

}

void drawcube(GLfloat \*a,GLfloat \*b,GLfloat \*c,GLfloat \*d)

{

glBegin(GL\_POLYGON);

//glColor3f(1.0,0.2,0.5);

glVertex3fv(a);

//glColor3f(0.9,0.1,0.8);

glVertex3fv(b);

//glColor3f(1.0,1.0,1.0);

glVertex3fv(c);

//glColor3f(0.7,0.8,0.6);

glVertex3fv(d);

glEnd();

}

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

{

glutInit(&argc,argv);

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

glutInitWindowPosition(10,10);

glutInitWindowSize(500,500);

glutCreateWindow("Perspective View");

init();

glutDisplayFunc(display);

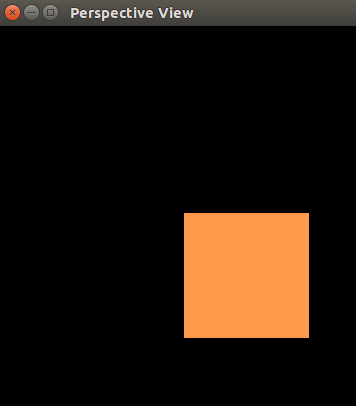
glEnable(GL\_DEPTH\_TEST);

glutKeyboardFunc(keys);

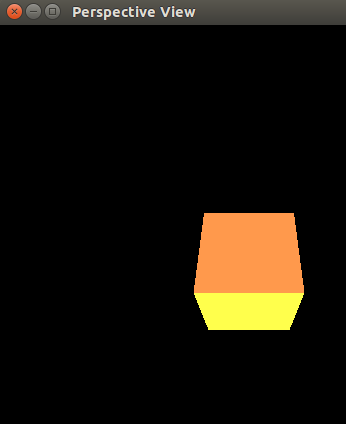
glutMainLoop();

}

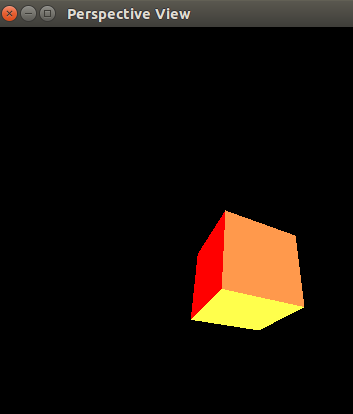
**Output 1:**



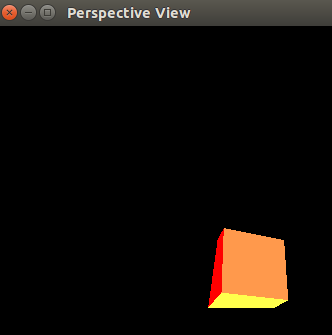
**Output 2:**



**Output 3:**



**Output 4:**



**5. Clip a lines using Cohen-Sutherland algorithm.**

**Cohen-sutherland line clipping Algorithm:**

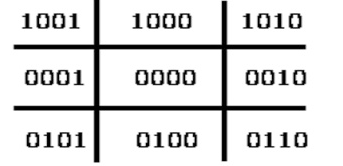
To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies.

1. Given a line segment with endpoint P1=(x1,y1) and P2=( x2,y2).
2. Compute the 4-bit codes for each endpoint.

If both codes are 0000,(bitwise OR of the codes yields 0000 ) line lies completely inside the window: pass the endpoints to the draw routine.

If both codes have a 1 in the same bit position (bitwise AND of the codes is not 0000), the line lies outside the window. It can be trivially rejected.

1. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be clipped at the window edge before being passed to the drawing routine.
2. Examine one of the endpoints, say P1=(x1,y1) . Read P1's 4-bit code in order: Left-to-Right, Bottom-to-Top.
3. When a set bit (1) is found, compute the intersection I of the corresponding window edge with the line from P1 toP2 . Replace P1 with I and repeat the algorithm.



**Program:**

#include<stdio.h>

#include<stdbool.h>

#include<GL/glut.h>

float xmin,ymin,xmax,ymax;//window boundaries

float xvmin,yvmin,xvmax,yvmax; //viewport boundaries

float x1,y1,x2,y2;

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

const int RIGHT=2;

const int LEFT=1;

const int TOP=8;

const int BOTTOM=4;

int computeOutCode(double x, double y);

/\*Cohen-Sutherland clippimg algorithm clips a line from

P0=(x1,y1) to P1=(x2,y2) against a rectangle with

diagonal from (xmin,ymin) to (xmax,ymax).\*/

void cohenSutherlandLineClipAndDraw(double x1,double y1,double x2,double y2)

{

int outcode1,outcode2, outcodeOut;

double x,y;

bool accept=false,done=false;

//compute outcodes

outcode1=computeOutCode(x1,y1);

outcode2=computeOutCode(x2,y2);

do

{

if(!(outcode1|outcode2))

{

//logical or is 0. Trivially accept & exit

accept=true;

done=true;

}

else if(outcode1&outcode2)

//logical and is not 0. Trivially reject and exit

done =true;

else

{

/\*failed both tests, so calculate the line

segment to clip from an outside

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

outcodeOut=outcode1?outcode1:outcode2;

/\*Now find the intersection point; use formulas

y=y1+slope\*(x-x1),x=x1+(1/slope)\*(y-y1)\*/

if(outcodeOut&TOP)

{

//point is above the clip rectangle

x=x1+((x2-x1)/(y2-y1))\*(ymax-y1);

y=ymax;

}

else if(outcodeOut&BOTTOM)

{

//point is below the clip rectangle

x=x1+((x2-x1)/(y2-y1))\*(ymin-y1);

y=ymin;

}

else if(outcodeOut&RIGHT)

{

//point is to the right of clip rectangle

y=y1+((y2-y1)/(x2-x1))\*(xmax-x1);

x=xmax;

}

else

{

//point is to the left of clip rectangle

y=y1+((y2-y1)/(x2-x1))\*(xmin-x1);

x=xmin;

}

/\*Now we move outside point to intersection point

to clip and get ready for next pass.\*/

if(outcodeOut==outcode1)

{

x1=x;

y1=y;

outcode1=computeOutCode(x1,y1);

}

else

{

x2=x;

y2=y;

outcode2=computeOutCode(x2,y2);

}

}

}while(!done);

if(accept)

{

//window to viewport mappings

float sx=(xvmax-xvmin)/(xmax-xmin); //scale parameters

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

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

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

float vx2=xvmin+(x2-xmin)\*sx;

float vy2=yvmin+(y2-ymin)\*sy;

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);

glBegin(GL\_LINES);

glVertex2f(vx1,vy1);

glVertex2f(vx2,vy2);

glEnd();

}

}

/\*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;

}

void display()

{

glClearColor(1.0,1.0,1.0,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,499.0,0.0,499.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2f(x1,y1);

glVertex2f(x2,y2);

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(x1,y1,x2,y2);

glFlush();

}

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

{

printf("Enter the boundaries of window for clipping\n");

printf("Enter xmin, ymin, xmax, ymax :\n");

scanf("%f%f%f%f",&xmin,&ymin,&xmax,&ymax);

printf("Enter the boundaries of view port\n");

printf("Enter xvmin, yvmin, xvmax, yvmax :\n");

scanf("%f%f%f%f",&xvmin,&yvmin,&xvmax,&yvmax);

printf("Enter the vertices of line (x1,y1,x2,y2):\n");

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

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowPosition(10,10);

glutInitWindowSize(500,500);

glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");

glutDisplayFunc(display);

glutMainLoop();

}

**Output 1:**

sahyadrids1@sahyadrids1-Veriton-Series:~/cg\_16$ gcc cohen.c -o cohen -lGL -lGLU -lglut

sahyadrids1@sahyadrids1-Veriton-Series:~/cg\_16$ ./cohen

Enter the boundaries of window for clipping

Enter xmin, ymin, xmax, ymax :

50 50 100 100

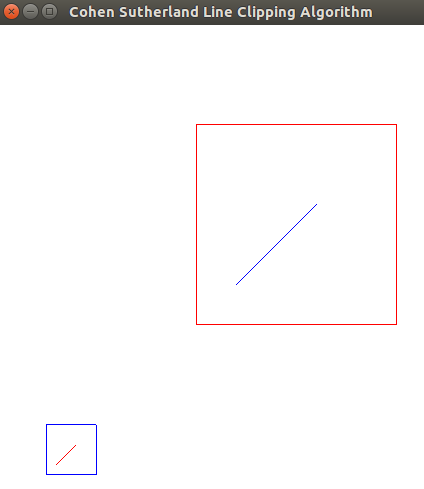
Enter the boundaries of view port

Enter xvmin, yvmin, xvmax, yvmax :

200 200 400 400

Enter the vertices of line (x1,y1,x2,y2):

60 60 80 80



**Output 2:**

sahyadrids1@sahyadrids1-Veriton-Series:~/cg\_16$ ./cohen

Enter the boundaries of window for clipping

Enter xmin, ymin, xmax, ymax :

50 50 100 100

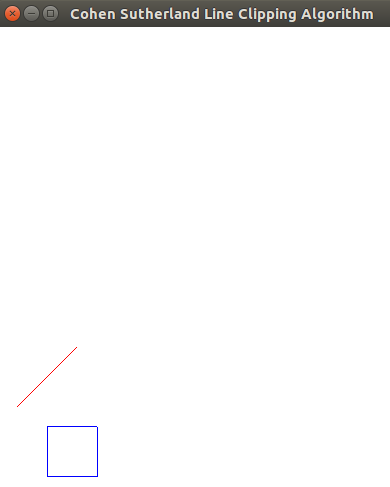
Enter the boundaries of view port

Enter xvmin, yvmin, xvmax, yvmax :

200 200 400 400

Enter the vertices of line (x1,y1,x2,y2):

20 120 80 180



**Output 3:**

sahyadrids1@sahyadrids1-Veriton-Series:~/cg\_16$ ./cohen

Enter the boundaries of window for clipping

Enter xmin, ymin, xmax, ymax :

50 50 150 150

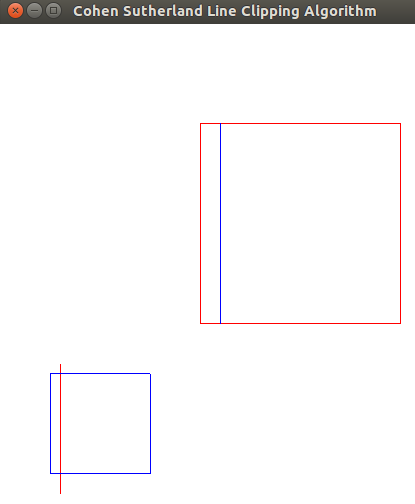
Enter the boundaries of view port

Enter xvmin, yvmin, xvmax, yvmax :

200 200 400 400

Enter the vertices of line (x1,y1,x2,y2):

60 30 60 160



**Output 4:**

sahyadrids1@sahyadrids1-Veriton-Series:~/cg\_16$ ./cohen

Enter the boundaries of window for clipping

Enter xmin, ymin, xmax, ymax :

50 50 100 100

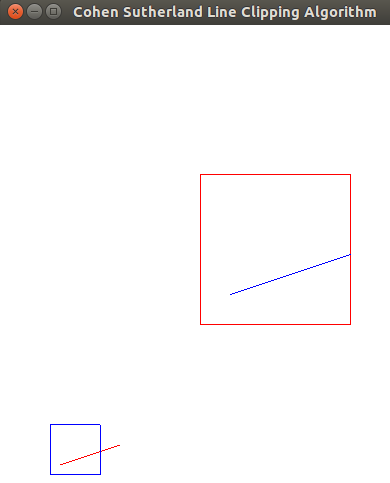
Enter the boundaries of view port

Enter xvmin, yvmin, xvmax, yvmax :

200 200 350 350

Enter the vertices of line (x1,y1,x2,y2):

60 60 120 80



**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.**

**Program:**

#include<GL/glut.h>

void displaySolid(void)

{

glClearColor(0.5,0.5,0.1,0.0);

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-100,100,-100,100,-100,100);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

//set properties of the surface material

GLfloat mat\_ambient[]={0.0f,1.0f,1.0f,1.0f};

GLfloat mat\_diffuse[]={1.0f,0.5f,1.0f,1.0f};

GLfloat mat\_specular[]={0.5f,0.5f,1.0f,1.0f};

GLfloat mat\_shininess[]={25.0f};

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_AMBIENT,mat\_ambient);

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_DIFFUSE,mat\_diffuse);

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_SPECULAR,mat\_specular);

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_SHININESS,mat\_shininess);

//set the light source properties

GLfloat lightIntensity[]={1.0f,0.7f,0.7f,1.0f};

GLfloat light\_position[]={25.0f,50.0f,50.0f,1.0f};

glLightfv(GL\_LIGHT0, GL\_POSITION,light\_position);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,lightIntensity);

glPushMatrix();

glTranslated(0,30,0);

glRotatef(35,1,0.5,0);

//glScaled(1,8,1);

glutSolidTeapot(10);

//glutWireTeapot(10);

glPopMatrix();

GLfloat mat\_ambient1[]={1.0f,0.0f,0.0f,1.0f};

GLfloat mat\_diffuse1[]={1.0f,1.0f,0.0f,1.0f};

GLfloat mat\_specular1[]={1.0f,1.0f,0.5f,1.0f};

GLfloat mat\_shininess1[]={25.0f};

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_AMBIENT,mat\_ambient1);

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_DIFFUSE,mat\_diffuse1);

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_SPECULAR,mat\_specular1);

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_SHININESS,mat\_shininess1);

//set the light source properties

GLfloat lightIntensity1[]={0.5f,0.5f,0.5f,1.0f};

GLfloat light\_position1[]={25.0f,50.0f,50.0f,1.0f};

glLightfv(GL\_LIGHT0, GL\_POSITION,light\_position1);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,lightIntensity1);

//top surface

glPushMatrix();

glTranslated(0,20,0);

glRotatef(-80,1,0.5,0.8);

//glRotatef(,0,0,1);

glScalef(1.5,1.5,0.1);

glutSolidCube(50);

glPopMatrix();

//First Leg

glPushMatrix();

glTranslated(-45,-10,-5);

glRotatef(45,0,1,0);

glScalef(0.4,5.5,0.4);

glutSolidCube(10);

glPopMatrix();

//Second Leg

glPushMatrix();

glTranslated(-10,-25,5);

glRotatef(45,0,1,0);

glScalef(0.4,4.5,0.4);

glutSolidCube(10);

glPopMatrix();

//Third Leg

glPushMatrix();

glTranslated(45,-5,-10);

glRotatef(45,0,1,0);

glScaled(0.4,5.5,0.4);

glutSolidCube(10);

glPopMatrix();

//Fourth Leg

glPushMatrix();

glTranslated(10,5,-35);

glRotatef(45,0,1,0);

glScalef(0.4,6,0.4);

glutSolidCube(10);

glPopMatrix();

glFlush();

}

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

{

glutInit(&argc,argv);

glutInitWindowSize(600,600);

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

glutInitWindowPosition(10,10);

glutInitWindowSize(500,500);

glutCreateWindow("Simple shaded scene consisting of a tea pot on a table");

glutDisplayFunc(displaySolid);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_NORMALIZE);

glutMainLoop();

}

**Output:**



**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.**

**Program:**

#include<stdio.h>

#include<GL/glut.h>

GLfloat v[4][3]={{0.0,0.0,0.5},{-0.5,-0.5,-0.5},{0.5,-0.5,-0.5},{0.0,0.5,-0.5}};

GLfloat color[4][3] = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0},{0.0, 0.0, 1.0}, {0.5, 0.5, 0.0}};

int n;

void divide\_tetra(GLfloat \*a,GLfloat \*b,GLfloat \*c,GLfloat \*d,int m);

void draw\_tri(GLfloat \*a,GLfloat \*b,GLfloat \*c);

void draw\_tetra(GLfloat \*a,GLfloat \*b,GLfloat \*c,GLfloat \*d)

{

glColor3fv(color[0]);

draw\_tri(a,b,c);

glColor3fv(color[1]);

draw\_tri(a,c,d);

glColor3fv(color[2]);

draw\_tri(a,d,b);

glColor3fv(color[3]);

draw\_tri(b,c,d);

}

void draw\_tri(GLfloat \*a,GLfloat \*b,GLfloat \*c)

{

glBegin(GL\_TRIANGLES);

glVertex3fv(a);

glVertex3fv(b);

glVertex3fv(c);

glEnd();

}

void divide\_tetra(GLfloat \*a,GLfloat \*b,GLfloat \*c,GLfloat \*d,int m)

{

GLfloat mid[6][3];

int i;

if(m>0)

{

for(i=0;i<3;i++)

{

mid[0][i]=(a[i]+b[i])/2;

mid[1][i]=(a[i]+c[i])/2;

mid[2][i]=(b[i]+c[i])/2;

mid[3][i]=(d[i]+c[i])/2;

mid[4][i]=(a[i]+d[i])/2;

mid[5][i]=(d[i]+b[i])/2;

}

divide\_tetra(a,mid[0],mid[1],mid[4],m-1);

divide\_tetra(mid[0],b,mid[2],mid[5],m-1);

divide\_tetra(mid[1],mid[2],c,mid[3],m-1);

divide\_tetra(mid[4],mid[5],mid[3],d,m-1);

}

else

draw\_tetra(a,b,c,d);

}

void display()

{

glClearColor(1.0,1.0,1.0,1.0);

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1.0,1.0,-1.0,1.0,-10.0,10.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

divide\_tetra(v[0],v[1],v[2],v[3],n);

glFlush();

}

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

{

printf("enter the number of divisions:");

scanf("%d",&n);

glutInit(&argc,argv);

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

glutInitWindowPosition(10,10);

glutInitWindowSize(500,500);

glutCreateWindow("3d Sierpinski gasket");

glutDisplayFunc(display);

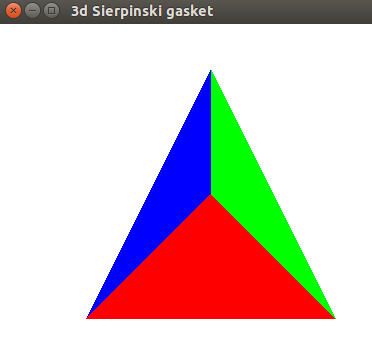
glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

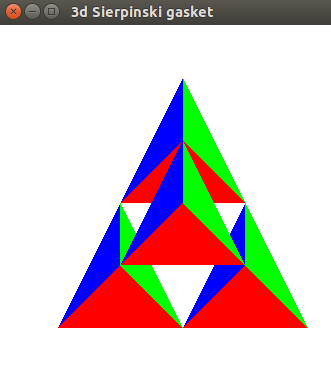
**Output 1:**

enter the number of divisions:0



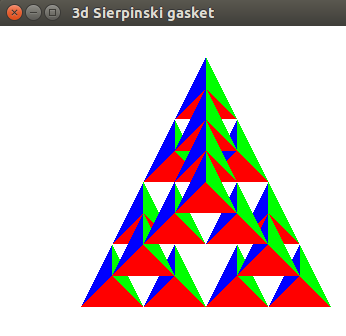
**Output 2:**

enter the number of divisions:1



**Output 3:**

enter the number of divisions:2



**8. Develop a menu driven program to animate a flag using Bezier Curve algorithm**

**Program:**

#include<GL/glut.h>

void display();

void init();

int counter=1;

float grid1[4][4][3] ={

{{-300,225,0},{-250,225,0},{-50,225,0},{50,225,0}},

{{-300,250,0},{-250,250,0},{-50,250,0},{50,250,0}},

{{-300,275,0},{-250,275,0},{-50,275,0},{50,275,0}},

{{-300,300,0},{-250,300,0},{-50,300,0},{40,300,0}}

};

float grid2[4][4][3] ={

{{-300,150,0},{-250,150,0},{-50,150,0},{50,150,0}},

{{-300,175,0},{-250,175,0},{-50,175,0},{50,175,0}},

{{-300,200,0},{-250,200,0},{-50,200,0},{50,200,0}},

{{-300,225,0},{-250,225,0},{-50,225,0},{50,225,0}}

};

float grid3[4][4][3] ={

{{-300,75,0},{-250,75,0},{-50,75,0},{40,75,0}},

{{-300,100,0},{-250,100,0},{-50,100,0},{50,100,0}},

{{-300,125,0},{-250,125,0},{-50,125,0},{50,125,0}},

{{-300,150,0},{-250,150,0},{-50,150,0},{50,150,0}}

};

void init()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-500, 500, -500, 500, -500, 500);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glEnable(GL\_MAP2\_VERTEX\_3);

}

void timer(int value)

{

int i;

for(i=0;i<4;i++)

{

if(counter==1)

{

grid1[i][1][1]+=1;

grid2[i][1][1]+=1;

grid3[i][1][1]+=1;

grid1[i][2][1]-=1;

grid2[i][2][1]-=1;

grid3[i][2][1]-=1;

}

else if(counter==0)

{

grid1[i][1][1]-=1;

grid2[i][1][1]-=1;

grid3[i][1][1]-=1;

grid1[i][2][1]+=1;

grid2[i][2][1]+=1;

grid3[i][2][1]+=1;

}

}

if(grid3[0][1][1]==125)

counter=0;

if(grid3[0][1][1]==0)

counter=1;

glutTimerFunc(10,timer,20);

glutPostRedisplay();

}

void menu(int id)

{

if (id == 1)

glClearColor(1, 0, 0, 0);

if (id==2)

glClearColor(1, 1, 0, 0);

if (id==3)

glClearColor(0, 1, 1, 0);

glutPostRedisplay();

}

void drawflag()

{

glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);

glEvalMesh2(GL\_FILL, 0, 20, 0, 20);

}

void display()

{

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

glColor3f(0.5, 0.5, 0.5);

glLineWidth(25);

glBegin(GL\_LINES);

glVertex2f(-300, -300);

glVertex2f(-300,315);

glEnd();

glLineWidth(1);

glPushMatrix();

glTranslatef(-300,320,0);

glutSolidSphere(10,10,2);

glPopMatrix();

glBegin(GL\_QUADS);

glColor3f(1, 0.5, 0);

glVertex2f(-350, -300);

glVertex2f(-250, -300);

glVertex2f(-250,-340);

glVertex2f(-350,-340);

glColor3f(1, 1, 1);

glVertex2f(-370, -340);

glVertex2f(-230, -340);

glVertex2f(-230,-380);

glVertex2f(-370,-380);

glColor3f(0, 1, 0);

glVertex2f(-390, -380);

glVertex2f(-210, -380);

glVertex2f(-210,-420);

glVertex2f(-390,-420);

glEnd();

glLineWidth(1);

glColor3f(0, 0, 1);

glPushMatrix();

glTranslatef(-130,186,0);

glutWireSphere(35,24,2);

glPopMatrix();

glColor3f(1, 0.5, 0);

glMap2f(GL\_MAP2\_VERTEX\_3, 0, 1, 3, 4, 0, 1, 4 \* 3, 4, &grid1[0][0][0]);

drawflag();

glColor3f(1, 1, 1);

glMap2f(GL\_MAP2\_VERTEX\_3, 0, 1, 3, 4, 0, 1, 4 \* 3, 4, &grid2[0][0][0]);

drawflag();

glColor3f(0, 1, 0);

glMap2f(GL\_MAP2\_VERTEX\_3, 0, 1, 3, 4, 0, 1, 4 \* 3, 4, &grid3[0][0][0]);

drawflag();

glColor3f(0, 0, 1);

glPushMatrix();

glTranslatef(-130,186,0);

glutWireSphere(35,24,2);

glPopMatrix();

glFlush();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DEPTH);

glutCreateWindow("Animated Flag");

init();

glEnable(GL\_DEPTH\_TEST);

glutDisplayFunc(display);

glutCreateMenu(menu);

glutAddMenuEntry("Red Background", 1);

glutAddMenuEntry("Yellow Background", 2);

glutAddMenuEntry("Cyan Background", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutTimerFunc(10,timer,20);

glutMainLoop();

}

**Output :**





**9. Develop a menu driven program to fill the polygon using scan line algorithm.**

**Scan line algorithm:**

**Step 1** − Find out the Ymin and Ymax from the given polygon.

**Step 2** − ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

**Step 3** − Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

**Step 4** − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

**Program 1:**

#include<stdio.h>

#include<stdlib.h>

#include<GL/glut.h>

float v[20][2];

int n;

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)

{

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

glFlush();

}

void scanfill(float v[20][2])

{

int le[500],re[500];

int j,i;

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

{

le[i]=500;

re[i]=0;

}

for(i=0;i<n-1;i++)

{

edgedetect(v[i][0],v[i][1],v[i+1][0],v[i+1][1],le,re);

}

edgedetect(v[i][0],v[i][1],v[0][0],v[0][1],le,re);

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

{

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

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

draw\_pixel(i,j);

}

}

void init()

{

glClearColor(1.0,1.0,1.0,1.0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,499.0,0.0,499.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glColor3f(1.0,0.0,0.0);

}

void display()

{

int i;

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_LINE\_LOOP);

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

{

glVertex2f(v[i][0],v[i][1]);

}

glEnd();

scanfill(v);

//glFlush();

}

void menu(int id)

{

if (id == 1)

glClearColor(1, 0.5, 0, 0);

if (id==2)

glClearColor(0, 1, 1, 0);

if (id==3)

glColor3f(0, 0, 1);

if (id==4)

glColor3f(1, 1, 0);

glutPostRedisplay();

}

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

{

int i;

printf("Enter the number of verices of Polygon\n");

scanf("%d",&n);

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

{

printf("Enter values for vertex %d:",i+1);

scanf("%f%f",&v[i][0],&v[i][1]);

}

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowPosition(10,10);

glutInitWindowSize(500,500);

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

init();

glutDisplayFunc(display);

int bgcolor=glutCreateMenu(menu);

glutAddMenuEntry("orange",1);

glutAddMenuEntry("cyan",2);

int fillcolor=glutCreateMenu(menu);

glutAddMenuEntry("blue",3);

glutAddMenuEntry("yellow",4);

glutCreateMenu(menu);

glutAddSubMenu("Background",bgcolor);

glutAddSubMenu("Fill color",fillcolor);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

}

**Output1 :**

Enter the number of verices of Polygon

5

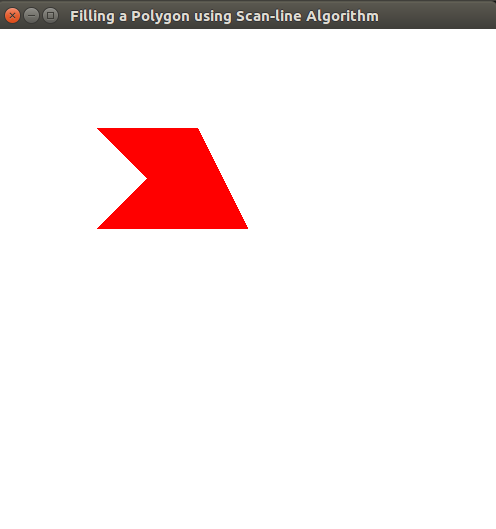
Enter values for vertex 1:100 400

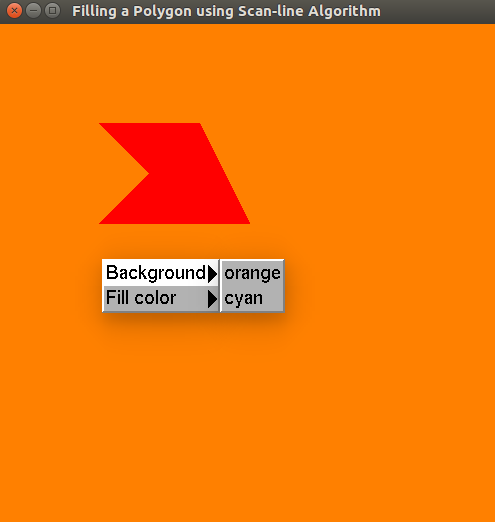
Enter values for vertex 2:200 400

Enter values for vertex 3:250 300

Enter values for vertex 4:100 300

Enter values for vertex 5:150 350







**Program 2:**

#include<stdio.h>

#include<stdlib.h>

#include<GL/glut.h>

#define WINDOW\_HEIGHT 500

/\*The edge data structure\*/

typedef struct tEdge

{

int yUpper;

float xIntersect, dxPerScan;

struct tEdge \* next;

} Edge;

typedef struct tPoint

{

int x;

int y;

} Point;

int n;

Point v[60];

/\* Inserts edge into list in order of increasing xIntersect field. \*/

void insertEdge (Edge \* list, Edge \* edge)

{

Edge \* p, \* q = list;

p = q->next;

while (p != NULL)

{

if (edge->xIntersect < p->xIntersect)

p = NULL;

else

{

q = p;

p = p->next;

}

}

edge->next = q->next;

q->next = edge;

}

/\* Store lower-y coordinate and inverse slope for each edge. Adjust and store upper-y coordinate for edges that are the lower member

of a monotically increasing or decreasing pair of edges \*/

void makeEdgeRec(Point lower, Point upper, int yComp, Edge \* edge, Edge \* edges[])

{

//Edge \*q;

edge->dxPerScan =(float) (upper.x - lower.x) / (upper.y - lower.y);

edge->xIntersect = lower.x;

if (upper.y < yComp)

edge->yUpper = upper.y - 1;

else

edge->yUpper = upper.y;

insertEdge (edges[lower.y], edge);

/\*checking the values inserted into edge records uncomment if you want to check

q=edges[lower.y]->next;

while(q!=NULL)

{

printf("xi=%f\n",q->xIntersect);

q=q->next;

}\*/

}

/\* For an index, return y-coordinate of next nonhorizontal line \*/

int yNext (int k, int n, Point \* v)

{

int j;

if ((k+1) > (n-1))

j = 0;

else

j = k + 1;

while (v[k].y == v[j].y)

if ((j+1) > (n-1))

j = 0;

else

j++;

return (v[j].y);

}

void buildEdgeList (int n, Point \* v, Edge \* edges[])

{

Edge \* edge;

Point v1, v2;

int i, yPrev = v[n - 2].y;

v1.x = v[n-1].x; v1.y = v[n-1].y;

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

{

v2 = v[i];

if (v1.y != v2.y)

{ /\* nonhorizontal line \*/

edge = (Edge \*) malloc (sizeof (Edge));

if (v1.y < v2.y) /\* up-going edge \*/

makeEdgeRec (v1, v2, yNext (i, n, v), edge, edges);

else /\* down-going edge \*/

makeEdgeRec (v2, v1, yPrev, edge, edges);

}

yPrev = v1.y;

v1 = v2;

}

}

void buildActiveList (int scan, Edge \* active, Edge \* edges[])

{

Edge \* p, \* q;

p = edges[scan]->next;

while (p)

{

q = p->next;

insertEdge (active, p);

p = q;

}

}

void fillScan (int scan, Edge \* active)

{

Edge \* p1, \* p2;

int i;

p1 = active->next;

while (p1)

{

p2 = p1->next;

glBegin(GL\_POINTS);

for (i=p1->xIntersect; i<p2->xIntersect; i++)

glVertex2i((int) i, scan);

glEnd();

p1 = p2->next;

}

}

void deleteAfter (Edge \* q)

{

Edge \* p = q->next;

q->next = p->next;

free (p);

}

/\* Delete completed edges. Update 'xIntersect' field for others \*/

void updateActiveList (int scan, Edge \* active)

{

Edge \* q = active, \* p = active->next;

while (p)

if (scan >= p->yUpper)

{

p = p->next;

deleteAfter (q);

}

else

{

p->xIntersect = p->xIntersect +

p->dxPerScan;/\*x=x+1/m\*/

q = p;

p = p->next;

}

}

void resortActiveList (Edge \* active)

{

Edge \* q, \* p = active->next;

active->next = NULL;

while (p)

{

q = p->next;

insertEdge (active, p);

p = q;

}

}

void scanFill (int n, Point \* v)

{

Edge \* edges[WINDOW\_HEIGHT], \* active;

int i, scan;

for (i=0; i<WINDOW\_HEIGHT; i++)

{

edges[i] = (Edge \*) malloc (sizeof (Edge));

edges[i]->next = NULL;

}

buildEdgeList (n, v, edges);

active = (Edge \*) malloc (sizeof (Edge));

active->next = NULL;

for (scan=0; scan<WINDOW\_HEIGHT; scan++)

{

buildActiveList (scan, active, edges);

if (active->next)

{

fillScan (scan, active);

updateActiveList (scan, active);

resortActiveList (active);

}

}

/\* Free edge records that have been malloc'ed ... \*/

free(active);

}

void display()

{

int i;

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_LINE\_LOOP);

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

{

glVertex2i(v[i].x,v[i].y);

}

glEnd();

scanFill(n,v);

glFlush();

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,499.0,0.0,499.0);

glMatrixMode(GL\_MODELVIEW);

glColor3f(1.0,0.0,0.0);

}

void menu(int id)

{

if (id == 1)

glClearColor(1, 0.5, 0, 0);

if (id==2)

glClearColor(0, 1, 1, 0);

if (id==3)

glColor3f(0, 0, 1);

if (id==4)

glColor3f(1, 1, 0);

glutPostRedisplay();

}

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

{

int i;

printf("Enter the no of points\n");

scanf("%d",&n);

printf("Enter the vertices\n");

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

scanf("%d%d",&v[i].x,&v[i].y);

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowPosition(0,0);

glutInitWindowSize(500,500);

glutCreateWindow("Scan Line Area Filling Algorithm");

myinit();

glutDisplayFunc(display);

int bgcolor=glutCreateMenu(menu);

glutAddMenuEntry("orange",1);

glutAddMenuEntry("cyan",2);

int fillcolor=glutCreateMenu(menu);

glutAddMenuEntry("blue",3);

glutAddMenuEntry("yellow",4);

glutCreateMenu(menu);

glutAddSubMenu("Background",bgcolor);

glutAddSubMenu("Fill color",fillcolor);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

}