**6.**

#include <stdio.h>

#include <GL/glut.h>

#define outcode GLint

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

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

//GLfloat x1, x0, y1, y0;

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

GLfloat xmin, ymin, xmax, ymax; // Window boundaries

GLfloat xvmin, yvmin, xvmax, yvmax; // Viewport boundaries

GLint n;

struct line\_segment {

GLfloat x0;

GLfloat y0;

GLfloat x1;

GLfloat y1;

};

struct line\_segment ls[10];

const GLint RIGHT = 8;

const GLint LEFT = 2;

const GLint TOP = 4;

const GLint BOTTOM = 1;

//used to compute bit codes of a point

outcode ComputeOutCode(GLfloat x, GLfloat y);

//Cohen-Sutherland clipping algorithm clips a line from

//P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with

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

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

{

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

outcode 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

{

//failed both tests, so calculate the line segment to clip

//from an outside point to an intersection with clip edge

GLfloat x, y;

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

outcodeOut = outcode0 ? outcode0 : outcode1;

//Now find the intersection point;

//use formulas 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

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

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

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

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

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

GLfloat 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();

}

}

//Compute the bit code for a point (x, y) using the clip rectangle

//bounded diagonally by (xmin, ymin), and (xmax, ymax)

outcode ComputeOutCode(GLfloat x, GLfloat y)

{

outcode 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()

{

//GLfloat x0 = 60, y0 = 20, x1 = 80, y1 = 120;

glClear(GL\_COLOR\_BUFFER\_BIT);

//draw the line with red color

glColor3f(1.0, 0.0, 0.0);

//bres(120,20,340,250);

/\*

glBegin(GL\_LINES);

glVertex2d(x0, y0);

glVertex2d(x1, y1);

glEnd();

\*/

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

{

glBegin(GL\_LINES);

glVertex2d(ls[i].x0, ls[i].y0);

glVertex2d(ls[i].x1, ls[i].y1);

glEnd();

}

//draw a blue colored window

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);

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

CohenSutherlandLineClipAndDraw(ls[i].x0, ls[i].y0, ls[i].x1, ls[i].y1);

glFlush();

}

void myinit()

{

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 499.0, 0.0, 499.0);

}

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

{

//int x1, x2, y1, y2;

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

//scanf\_s("%f%f%f%f", &x0, &x1, &y0, &y1);

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

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

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

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

printf("Enter no. of lines:\n");

scanf\_s("%d", &n);

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

{

printf("Enter line-%d endpoints (x1 x2 y1 y2):\n", (i + 1));

scanf\_s("%f%f%f%f", &ls[i].x0, &ls[i].x1, &ls[i].y0, &ls[i].y1);

}

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

glutCreateWindow("Cohen Suderland Line Clipping Algorithm");

glutDisplayFunc(display);

myinit();

glutMainLoop();

return 0;

}

**OUTPUT:**

