

| **TITLE**: Write a program to demonstrate the LINE CLIPPING algorithm |
| --- |

**AIM:**

**Visit Vlab and Explore it**

[**https://cse18-iiith.vlabs.ac.in/exp/clipping-line/**](https://cse18-iiith.vlabs.ac.in/exp/clipping-line/)

Write a program to demonstrate the LINE CLIPPING algorithm

1. Cohen-Sutherland-algorithm
2. Mid-Point Subdivision Line Clipping Algorithm
3. Liang-Barsky Line Clipping Algorithm

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

Understand the basic concepts of computer graphics and OpenGL

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

**Google**

**Geeksforgeeks**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algorithm 1, 2 and 3**

* Cohen-Sutherland-algorithm

1) Assign the region codes to both endpoints.

2) Perform [OR operation](https://www.researchgate.net/figure/Example-of-truth-table-for-logical-OR-operator_tbl1_334130877) on both of these endpoints.

3) if OR = 0000,

then it is completely visible (inside the window).

else

Perform [AND operation](https://homepages.inf.ed.ac.uk/rbf/HIPR2/and.htm) on both these endpoints.

i) if AND ? 0000,

then the line is invisible and not inside the window. Also, it can’t be considered for clipping.

ii) else

AND = 0000, the line is partially inside the window and considered for clipping.

4) After confirming that the line is partially inside the window, then we find the intersection with the boundary of the window. By using the following formula:-

[Slope](https://www.geeksforgeeks.org/how-to-find-the-slope-of-a-line-on-an-excel-graph/):- m= (y2-y1)/(x2-x1)

a) If the line passes through top or the line intersects with the top boundary of the window.

x = x + (y\_wmax – y)/m

y = y\_wmax

b) If the line passes through the bottom or the line intersects with the bottom boundary of the window.

x = x + (y\_wmin – y)/m

y = y\_wmin

c) If the line passes through the left region or the line intersects with the left boundary of the window.

y = y+ (x\_wmin – x)\*m

x = x\_wmin

d) If the line passes through the right region or the line intersects with the right boundary of the window.

y = y + (x\_wmax -x)\*m

x = x\_wmax

5) Now, overwrite the endpoints with a new one and update it.

6) Repeat the 4th step till your line doesn’t get completely clipped

* Mid-Point Subdivision Line Clipping Algorithm  
    
  1) Calculate the position of both endpoints of the line

2) Perform OR operation on both of these endpoints

3) If the OR operation gives 0000  
 then  
 Line is guaranteed to be visible  
 else  
 Perform AND operation on both endpoints.  
 If AND ≠ 0000  
 then the line is invisible  
 else  
 AND=6000  
 then the line is clipped case.

4) For the line to be clipped. Find midpoint  
 Xm=(x1+x2)/2  
 Ym=(y1+y2)/2  
 Xmis midpoint of X coordinate.  
 Ymis midpoint of Y coordinate.

5)Check each midpoint, whether it nearest to the boundary of a window or not.

6) If the line is totally visible or totally rejected not found then repeat step 1-5.

7) Stop

* Liang-Barsky Line Clipping Algorithm  
    
  1) Set tmin=0, tmax=1.

2) Calculate the values of t (t(left), t(right), t(top), t(bottom)), (i) If t < tmin ignore that and move to the next edge. (ii) else separate the t values as entering or exiting values using the inner product. (iii) If t is entering value, set tmin = t; if t is existing value, set tmax = t.

3) If tmin < tmax, draw a line from (x1 + tmin(x2-x1), y1 + tmin(y2-y1)) to (x1 + tmax(x2-x1), y1 + tmax(y2-y1))

4) If the line crosses over the window, (x1 + tmin(x2-x1), y1 + tmin(y2-y1)) and (x1 + tmax(x2-x1), y1 + tmax(y2-y1)) are the intersection point of line and edge.

**Implementation details:**

a )

//Cohen-Sutherland Line Clipping  
#include <iostream>

#include <GL/glut.h>

enum { INSIDE, LEFT, RIGHT, BELOW, ABOVE };

int xmin = 50, ymin = 50, xmax = 400, ymax = 400;

int computeOutCode(double x, double y) {

int code = INSIDE;

if (x < xmin) code |= LEFT;

else if (x > xmax) code |= RIGHT;

if (y < ymin) code |= BELOW;

else if (y > ymax) code |= ABOVE;

return code;

}

void cohenSutherlandClip(double x0, double y0, double x1, double y1) {

int outcode0 = computeOutCode(x0, y0);

int outcode1 = computeOutCode(x1, y1);

bool accept = false;

while (true) {

if (!(outcode0 | outcode1)) {

accept = true;

break;

} else if (outcode0 & outcode1) {

break;

} else {

int outcodeOut = outcode0 ? outcode0 : outcode1;

double x = 0, y = 0;

if (outcodeOut & ABOVE) {

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

y = ymax;

} else if (outcodeOut & BELOW) {

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

y = ymin;

} else if (outcodeOut & RIGHT) {

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

x = xmax;

} else if (outcodeOut & LEFT) {

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

x = xmin;

}

if (outcodeOut == outcode0) {

x0 = x;

y0 = y;

outcode0 = computeOutCode(x0, y0);

} else {

x1 = x;

y1 = y;

outcode1 = computeOutCode(x1, y1);

}

}

}

if (accept) {

glBegin(GL\_LINES);

glVertex2d(x0, y0);

glVertex2d(x1, y1);

glEnd();

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

cohenSutherlandClip(30, 30, 300, 300);

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("Cohen-Sutherland Line Clipping");

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor3f(0.0, 0.0, 0.0);

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

b) Mid-Point Subdivision Line Clipping Algorithm

#include <iostream>

#include <GL/glut.h>

int xmin = 50, ymin = 50, xmax = 400, ymax = 400;

void midpointSubdivisionClip(double x0, double y0, double x1, double y1) {

double mx, my;

while (true) {

if (x0 >= xmin && x0 <= xmax && y0 >= ymin && y0 <= ymax &&

x1 >= xmin && x1 <= xmax && y1 >= ymin && y1 <= ymax) {

break;

}

mx = (x0 + x1) / 2;

my = (y0 + y1) / 2;

if (x1 < xmin || x0 > xmax || y1 < ymin || y0 > ymax) {

x1 = mx;

y1 = my;

} else {

x0 = mx;

y0 = my;

}

}

glBegin(GL\_LINES);

glVertex2d(x0, y0);

glVertex2d(x1, y1);

glEnd();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 1.0, 0.0);

midpointSubdivisionClip(30, 30, 300, 300);

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("Mid-Point Subdivision Line Clipping");

glClearColor(0.0, 0.0, 0.0, 1.0);

glColor3f(0.0, 0.0, 0.0);

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

c) Liang-Barsky Line Clipping Algorithm

#include <iostream>

#include <GL/glut.h>

#include <algorithm>

int xmin = 50, ymin = 50, xmax = 400, ymax = 400;

void liangBarskyClip(double x0, double y0, double x1, double y1) {

double p[4], q[4], t0 = 0.0, t1 = 1.0;

p[0] = -(x1 - x0);

p[1] = x1 - x0;

p[2] = -(y1 - y0);

p[3] = y1 - y0;

q[0] = x0 - xmin;

q[1] = xmax - x0;

q[2] = y0 - ymin;

q[3] = ymax - y0;

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

if (p[i] == 0) {

if (q[i] < 0) return;

} else {

double t = q[i] / p[i];

if (p[i] < 0) {

t0 = std::max(t0, t);

} else {

t1 = std::min(t1, t);

}

}

}

if (t0 > t1) return;

double x0\_new = x0 + t0 \* (x1 - x0);

double y0\_new = y0 + t0 \* (y1 - y0);

double x1\_new = x0 + t1 \* (x1 - x0);

double y1\_new = y0 + t1 \* (y1 - y0);

glBegin(GL\_LINES);

glVertex2d(x0\_new, y0\_new);

glVertex2d(x1\_new, y1\_new);

glEnd();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 1.0);

liangBarskyClip(30, 30, 300, 300);

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow("Liang-Barsky Line Clipping");

glClearColor(0.0, 0.0, 0.0, 1.0);

gluOrtho2D(0.0, 500.0, 0.0, 500.0);

glutDisplayFunc(display);

glutMainLoop();

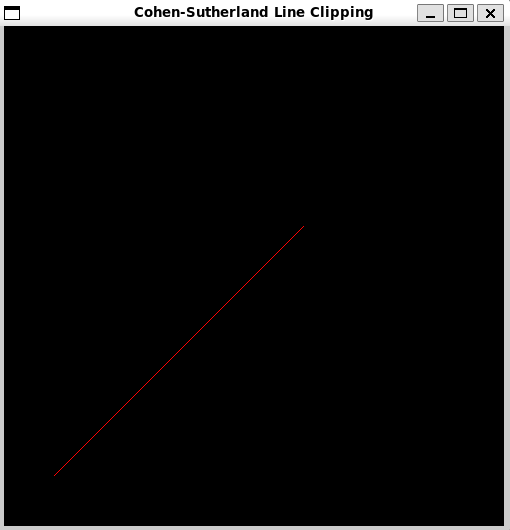
return 0;

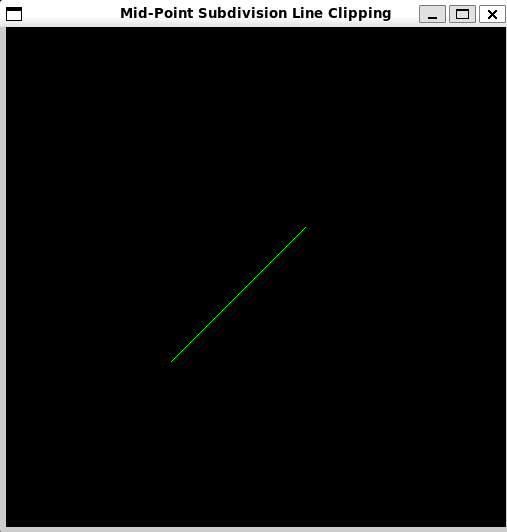
}

**Output(s) (final edited screen shot):**

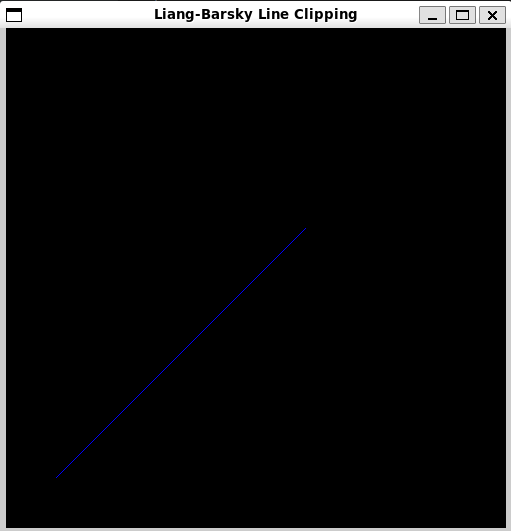
**a) xmin = 50, ymin = 50, xmax = 400, ymax = 400;**



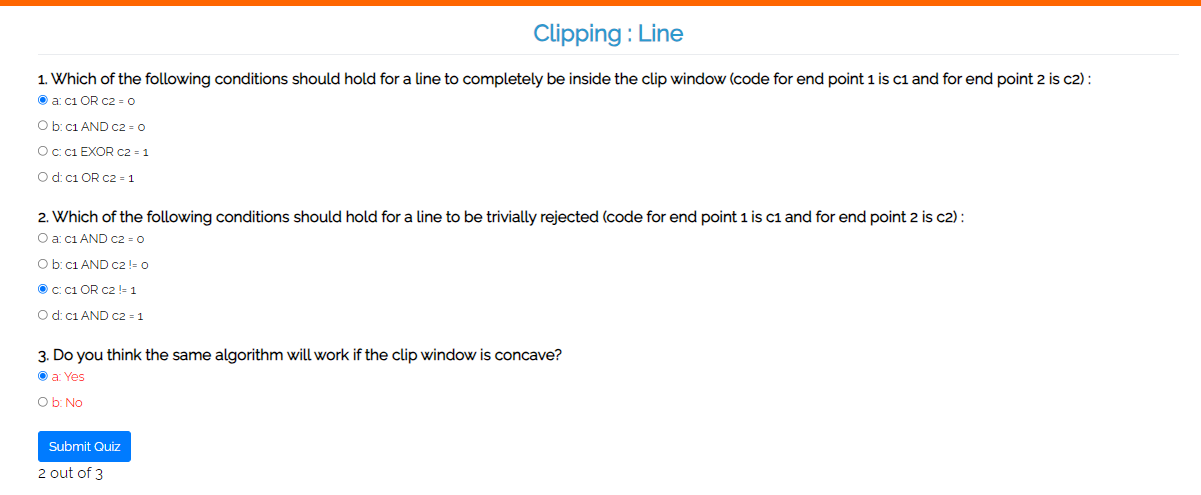
**  
 b) xmin = 50, ymin = 50, xmax = 400, ymax = 400;**

**  
  
  
c) xmin = 50, ymin = 50, xmax = 400, ymax = 400;**

****

****

**Screenshots from VLab(if any):**

****

**Conclusion and discussion (Comparative ):**

* Cohen-Sutherland Algorithm :Simple to implement, efficient for basic cases but can be inefficient with complex cases , redundant checks.
* Mid-Point Subdivision Algorithm : Accurate and handles a variety of clipping scenarios but complex and computationally expensive due to many subdivisions.
* Liang-Barsky Algorithm :Efficient for rectangular clipping windows, directly computes intersections but more complex to implement, less general for non-rectangular clipping regions.

**Date: 21/8/24**

**Signature of faculty in-charge**

**Post Lab**

**What is Turtle in CG, Demonstrate use of Turtle by implementing it?**Turtle was originally used in the Logo programming language, where you control a "turtle" that moves around the screen and draws lines based on the commands you provide.A Turtle graphics system consists of a "turtle" that can move forward, turn, and draw lines. By issuing a sequence of commands, you can create various shapes and patterns.

#include <GL/glut.h>

#include <cmath>

const float colors[6][3] = {

{1.0f, 0.0f, 0.0f},

{0.5f, 0.0f, 0.5f},

{0.0f, 0.0f, 1.0f},

{0.0f, 1.0f, 0.0f},

{1.0f, 0.5f, 0.0f},

{1.0f, 1.0f, 0.0f}

};

float x = 200.0f, y = 270.0f;

float angle = 0.0f;

void drawLine(float x1, float y1) {

glBegin(GL\_LINES);

glVertex2f(x, y);

glVertex2f(x1, y1);

glEnd();

}

void forward(float distance) {

float newX = x + distance \* cos(angle \* M\_PI / 180.0f);

float newY = y + distance \* sin(angle \* M\_PI / 180.0f);

drawLine(newX, newY);

x = newX;

y = newY;

}

void turn(float angleChange) {

angle += angleChange;

}

void drawRainbowBenzene() {

for (int i = 0; i < 360; ++i) {

glColor3f(colors[i % 6][0], colors[i % 6][1], colors[i % 6][2]);

glLineWidth(i / 100.0f + 1.0f);

forward(i);

turn(59);

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glTranslatef(0.0f, 0.0f, 0.0f);

drawRainbowBenzene();

glutSwapBuffers();

}

void init() {

glClearColor(0.0, 0.0, 0.0, 0.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(-400, 400, -400, 400);

glMatrixMode(GL\_MODELVIEW);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(800, 600);

glutCreateWindow("Turtle Programming");

init();

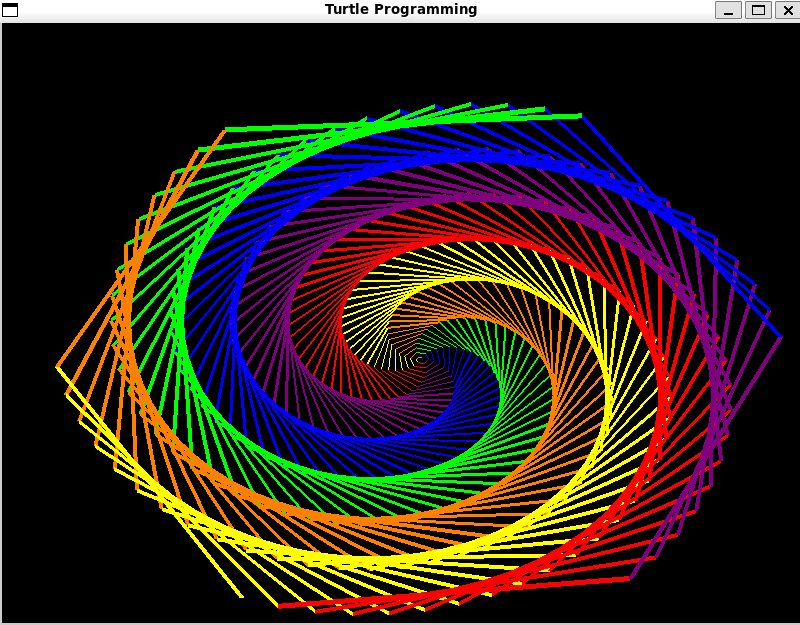
glutDisplayFunc(display);

glutMainLoop();

return 0;

}

****

****