**DEERWALK INSTITUTE OF TECHNOLOGY**

**Tribhuvan University**

**Faculties of Computer Science**

**A logo of a sea creature

Description automatically generated**

**Bachelors of Science in Computer Science and Information Technology (BSc. CSIT)**

**Course: Computer Graphics (CSC214)**

**Year/Semester: II/III**

**A Lab report on:**

**Implementation of Line Clipping Algorithm**

Submitted by:

Name: Parth Poudyal

Roll: 1317

Submitted to:

Binod Sitaula

Department of Computer Science

**Theory**

For a clipping window, of maximum and minimum coordinates of the window be given as

Let be the endpoints of the line to be clipped using the Cohen Sutherland Line Clipping algorithm.

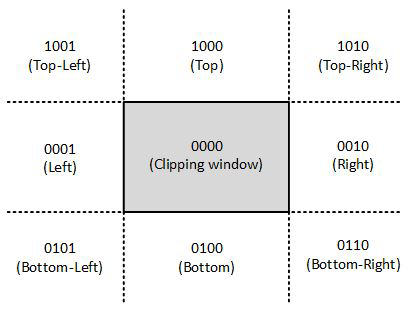
Let the clipping region be allocated into regions where,

Figure 1: Cohen Sutherland Line Clipping Region Area Code

For point (x, y) be a point on the line and provide region code for clipping then:

Clipping case:   
If both endpoints of the line have end bits zero, then the line is visible, no clipping required

If the two region codes for the endpoints are “ANDED” and it provides anything else than 0000, it suggests that the endpoints lie completely outside the clipping region, therefore none of the region is visible.

While “ANDING”, it provides 0000 then,

The line is clipped, and its new vertices are given such as :

* then,
* then,
* then,
* then,

With this the clipped vertices are found.

Advantage of Cohen Sutherland Line Clipping:

* 1. It calculates end-points very quickly and rejects and accepts lines quickly.
  2. It can clip pictures much large than screen size.

Liang-Barsky Algorithm:

The Liang-Barsky algorithm is a line clipping algorithm. This algorithm is more efficient than Cohen–Sutherland line clipping algorithm and can be extended to 3-Dimensional clipping. This algorithm is considered to be the faster parametric line-clipping algorithm. The following concepts are used in this clipping:

* 1. The parametric equation of the line.
  2. The inequalities describing the range of the clipping window which is used to determine the intersections between the line and the clip window.

The parametric equation of a line can be given by,

1. =
2. =

Where, t is between 0 and 1.

Then, writing the point-clipping conditions in the parametric form:

xwmin <= x1 + t(x2-x1) <= xwmax ywmin <= y1 + t(y2-y1) <= ywmax

The above 4 inequalities can be expressed as, tpk <= qk

Where k = 1, 2, 3, 4 (correspond to the left, right, bottom, and top boundaries, respectively). The p and q are defined as,

p1 = -(x2-x1), q1 = x1 - xwmin (Left Boundary) p2 = (x2-x1), q2 = xwmax - x1 (Right Boundary) p3 = -(y2-y1), q3 = y1 - ywmin (Bottom Boundary) p4 = (y2-y1), q4 = ywmax - y1 (Top Boundary)

When the line is parallel to a view window boundary, the p value for that boundary is zero.

When pk < 0, as t increase line goes from the outside to inside (entering). When pk > 0, line goes from inside to outside (exiting). When pk = 0 and qk < 0 then line is trivially invisible because it is outside view window.

When pk = 0 and qk > 0 then the line is inside the corresponding window boundary. Parameters t1 and t2 can be calculated that define the part of line that lies within the clip rectangle.

When,

1. pk < 0, maximum(0, qk/pk) is taken.
2. pk > 0, minimum(1, qk/pk) is taken.

If t1 > t2, the line is completely outside the clip window, and it can be rejected. Otherwise, the endpoints of the clipped line are calculated from the two values of parameter t.

**Algorithm**

Cohen Sutherland Line Clipping

Step1: Calculate positions of both endpoints of the line

Step2: Perform OR operation on both of these end-points

Step3: If the OR operation gives 0000

Then

line is considered to be visible else

Perform AND operation on both endpoints

If And ≠ 0000

then the line is invisible

else

And=0000

Line is considered the clipped case.

Step4: If a line is clipped case, find an intersection with boundaries of the window m=(y2-y1 )(x2-x1)

1. If bit 1 is "1" line intersects with left boundary of rectangle window y3=y1+m(x-X1) where X = Xwmin

where Xwminis the minimum value of X co-ordinate of window

1. If bit 2 is "1" line intersect with right boundary y3=y1+m(X-X1) where X = Xwmax

where X more is maximum value of X co-ordinate of the window

1. If bit 3 is "1" line intersects with bottom boundary X3=X1+(y-y1)/m where y = ywmin

ywmin is the minimum value of Y co-ordinate of the window

1. If bit 4 is "1" line intersects with the top boundary X3=X1+(y-y1)/m where y = ywmax  ywmax is the maximum value of Y co-ordinate of the window Liang Barsky 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.
      1. else separate the t values as entering or exiting values using the inner product.
      2. 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.

**PROGRAM**

#include <iostream>

#include <graphics.h>

using namespace std;

// Clipping window boundaries

const int x\_min = 100, y\_min = 100, x\_max = 400, y\_max = 300;

// Region codes

const int INSIDE = 0; // 0000

const int LEFT = 1;   // 0001

const int RIGHT = 2;  // 0010

const int BOTTOM = 4; // 0100

const int TOP = 8;    // 1000

// Function to compute region code for a point (x, y)

int computeCode(int x, int y) {

    int code = INSIDE;

    if (x < x\_min) code |= LEFT;

    else if (x > x\_max) code |= RIGHT;

    if (y < y\_min) code |= BOTTOM;

    else if (y > y\_max) code |= TOP;

    return code;

}

// Cohen-Sutherland Line Clipping Algorithm

void cohenSutherlandClip(int x1, int y1, int x2, int y2) {

    int code1 = computeCode(x1, y1);

    int code2 = computeCode(x2, y2);

    bool accept = false;

    // Draw original line in red

    setcolor(RED);

    line(x1, y1, x2, y2);

    while (true) {

        if ((code1 == 0) && (code2 == 0)) {

            accept = true;

            break;

        } else if (code1 & code2) {

            break;

        } else {

            int code\_out;

            int x, y;

            if (code1 != 0)

                code\_out = code1;

            else

                code\_out = code2;

            if (code\_out & TOP) {

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

                y = y\_max;

            } else if (code\_out & BOTTOM) {

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

                y = y\_min;

            } else if (code\_out & RIGHT) {

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

                x = x\_max;

            } else if (code\_out & LEFT) {

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

                x = x\_min;

            }

            if (code\_out == code1) {

                x1 = x;

                y1 = y;

                code1 = computeCode(x1, y1);

            } else {

                x2 = x;

                y2 = y;

                code2 = computeCode(x2, y2);

            }

        }

    }

    if (accept) {

        setcolor(GREEN);

        line(x1, y1, x2, y2);

    }

}

int main() {

    int gd = DETECT, gm;

    initgraph(&gd, &gm, (char\*)"");

    // Draw clipping rectangle in white

    setcolor(WHITE);

    rectangle(x\_min, y\_min, x\_max, y\_max);

    // Test cases to clip

    cohenSutherlandClip(50, 150, 450, 250);

    cohenSutherlandClip(120, 80, 380, 320);

    cohenSutherlandClip(200, 200, 300, 250);

    cohenSutherlandClip(50, 50, 450, 350);

    getch();

    closegraph();

    return 0;

}

**A black background with red and green lines

AI-generated content may be incorrect.Output:**