Experiment No.7
Aim: To implement Line Clipping Algorithm: Cohen Sutherland
Name: Nikhil Kamalaji Shingade
Roll no:57
Date of Performance:
Date of Submission:

Experiment No. 7

Aim: To implement Line Clipping Algorithm: Cohen Sutherland

Objective:

To implement the concept of Cohen Sutherland algorithm to efficiently determine the portions of a line segment that lie within a specified rectangular clipping window. This method is particularly effective to clip line segments against rectangular clipping windows in 2D graphics where visibility needs to be determined quickly.

Theory:

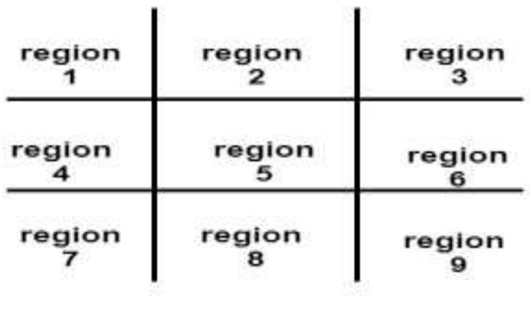
The Cohen-Sutherland algorithm is a popular line clipping algorithm used in computer graphics to determine which portions of a line segment lie within a specified rectangular clipping window. All lines come under any one of the following categories:

1. Visible
2. Not Visible
3. Clipping Case

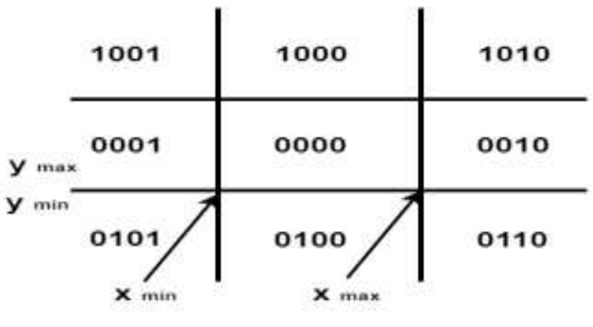
1. Visible: If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.

2. Not Visible: If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible. Let A (x1,y2) and B (x2,y2) are endpoints of line.

3. Clipping Case: If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible.



9 region



bits assigned to 9 regions

Algorithm

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

$m=(y2-y1)(x2-x1)$

(a) 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

(b) 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

(c) 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

(d) If bit 4 is "1" line intersects with the top boundary

$X3=X1+(y-y1)/m$

where $y = ywmax$

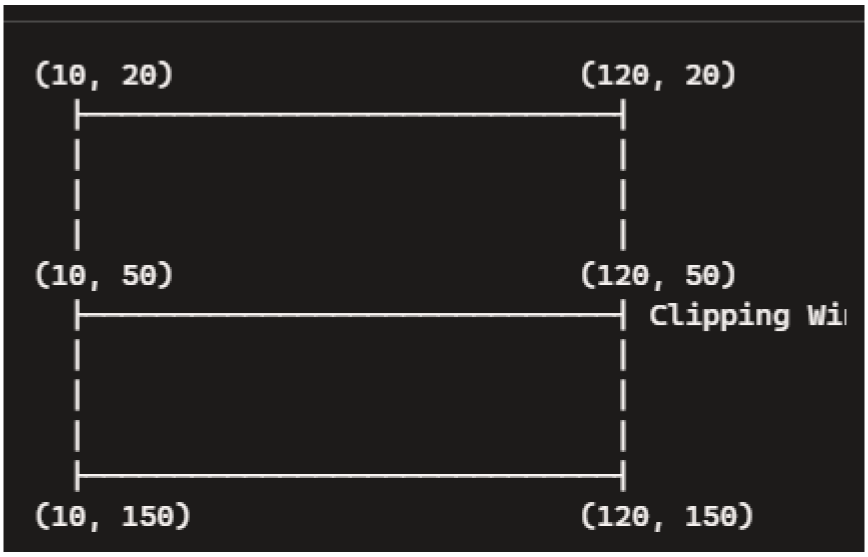
ywmax is the maximum value of Y co-ordinate of the window

Program:

```
#include <stdio.h>

// Define constants for 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

// Define the clipping window
int x_min, y_min, x_max, y_max;

// Function to compute the 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;
}

// Function to implement the 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);
    int accept = 0;

    while (1) {
        if ((code1 == 0) && (code2 == 0)) {
            accept = 1;
            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) {
        printf("Line accepted from (%d, %d) to (%d, %d)\n", x1, y1, x2, y2);
    } else {
        printf("Line rejected\n");
    }
}

int main() {
    x_min = 50, y_min = 50, x_max = 100, y_max = 100;

    int x1 = 10, y1 = 20, x2 = 120, y2 = 150;

    cohenSutherlandClip(x1, y1, x2, y2);
```

```
   return 0;
}
```

Output:



```
(10, 20)                    (120, 20)
  ├────────────────────────────┤
  │                            │
  │                            │
  │                            │
(10, 50)                    (120, 50)
  ├────────────────────────────┤ Clipping Wi
  │                            │
  │                            │
  │                            │
  ├────────────────────────────┤
(10, 150)                   (120, 150)
```

Conclusion:

Comment on: Advantages and Limitations of Cohen Sutherland Algorithm

1.Efficiency:
Uses region codes, which significantly reduce the number of comparisons needed to determ ine whether a line should be clipped, accepted, or rejected.
Ideal for quickly deciding the trivial accept/reject cases without heavy computation.
2.Clipping Precision:
Provides precise clipping of line segments to the boundaries of the rectangular clipping wind ow.
Suitable for graphics applications where exact boundaries are crucial.
3.Simplicity:
Straightforward to understand and implement.
Uses simple bitwise operations, making it efficient in practice.