DEERWALK INSTITUTE OF TECHNOLOGY

Tribhuvan University

Faculties of Computer Science

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

Course: Computer Graphics (CSC209)

Year/Semester: II/III

A Lab report on:

Implementation of Line Clipping Algorithm

Submitted by:
Name: Arun Mainali
Roll: 1307

Submitted to:
Binod Sitaula
Department of Computer Science

❖ LAB 6

OBJECTIVE:

Write a program in any high-level language to implement Line Clipping Algorithm:

a. Cohen Sutherland Algorithm
b. Liang Barsky Line Clipping Algorithm

THEORY:

Cohen Sutherland Line Clipping Algorithm:

In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. 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 (x_1,y_2) and B (x_2,y_2) are endpoints of line.

x_{min},x_{max} are coordinates of the window.

y_{min},y_{max} are also coordinates of the window.

$x_1>x_{max}$
$x_2>x_{max}$
$y_1>y_{max}$
$y_2>y_{max}$
$x_1<x_{min}$
$x_2<x_{min}$
$y_1<y_{min}$
$y_2<y_{min}$

4. 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.

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,

$X = x_1 + t(x_2-x_1)$
$Y = y_1 + t(y_2-y_1)$
Where, t is between 0 and 1.

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

$xw_{min} <= x_1 + t(x_2-x_1) <= xw_{max}$
$yw_{min} <= y_1 + t(y_2-y_1) <= yw_{max}$
The above 4 inequalities can be expressed as,

$tp_k <= q_k$
Where k = 1, 2, 3, 4 (correspond to the left, right, bottom, and top boundaries, respectively).

The p and q are defined as,

$p_1 = -(x_2-x_1)$, $q_1 = x_1 - xw_{min}$ (Left Boundary)
$p_2 = (x_2-x_1)$, $q_2 = xw_{max} - x_1$ (Right Boundary)
$p_3 = -(y_2-y_1)$, $q_3 = y_1 - yw_{min}$ (Bottom Boundary)
$p_4 = (y_2-y_1)$, $q_4 = yw_{max} - y_1$ (Top Boundary)
When the line is parallel to a view window boundary, the p value for that boundary is zero.
When $p_k < 0$, as t increase line goes from the outside to inside (entering).
When $p_k > 0$, line goes from inside to outside (exiting).
When $p_k = 0$ and $q_k < 0$ then line is trivially invisible because it is outside view window.
When $p_k = 0$ and $q_k > 0$ then the line is inside the corresponding window boundary.
Parameters t_1 and t_2 can be calculated that define the part of line that lies within the clip rectangle.
When,

1. $p_k < 0$, maximum$(0, q_k/p_k)$ is taken.
2. $p_k > 0$, minimum$(1, q_k/p_k)$ is taken.

If $t_1 > t_2$, 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 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
            $m=(y_2-y_1)(x_2-x_1)$
(a) If bit 1 is "1" line intersects with left boundary of rectangle window
            $y_3=y_1+m(x-X_1)$
            where $X = X_{wmin}$
            where $X_{wmin}$is the minimum value of X co-ordinate of window
(b) If bit 2 is "1" line intersect with right boundary
            $y_3=y_1+m(X-X_1)$
            where $X = X_{wmax}$
            where X more is maximum value of X co-ordinate of the window
(c) If bit 3 is "1" line intersects with bottom boundary
            $X_3=X_1+(y-y_1)/m$
                 where $y = y_{wmin}$
            $y_{wmin}$ is the minimum value of Y co-ordinate of the window

      (d) If bit 4 is "1" line intersects with the top boundary
                 $X_{3=X}1+(y-y_1)/m$
                      where $y = y_{wmax}$
                 $y_{wmax}$ is the maximum value of Y co-ordinate of the window
```

Liang Barsky Algorithm:

```
1. Set $t_{min}=0$, $t_{max}=1$.
2. Calculate the values of t (t(left), t(right), t(top), t(bottom)),
   (i) If $t < t_{min}$ 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 t_{min} = t; if t is existing value, set t_{max} = t.

3. If $t_{min} < t_{max}$, draw a line from $(x_1 + t_{min}(x_2-x_1), y_1 + t_{min}(y_2-y_1))$ to $(x_1 + t_{max}(x_2-x_1), y_1 + t_{max}(y_2-y_1))$
4. If the line crosses over the window, $(x_1 + t_{min}(x_2-x_1), y_1 + t_{min}(y_2-y_1))$ and $(x_1 + t_{max}(x_2-x_1), y_1 + t_{max}(y_2-y_1))$ are the intersection point of line and edge.

PROGRAM CODE:

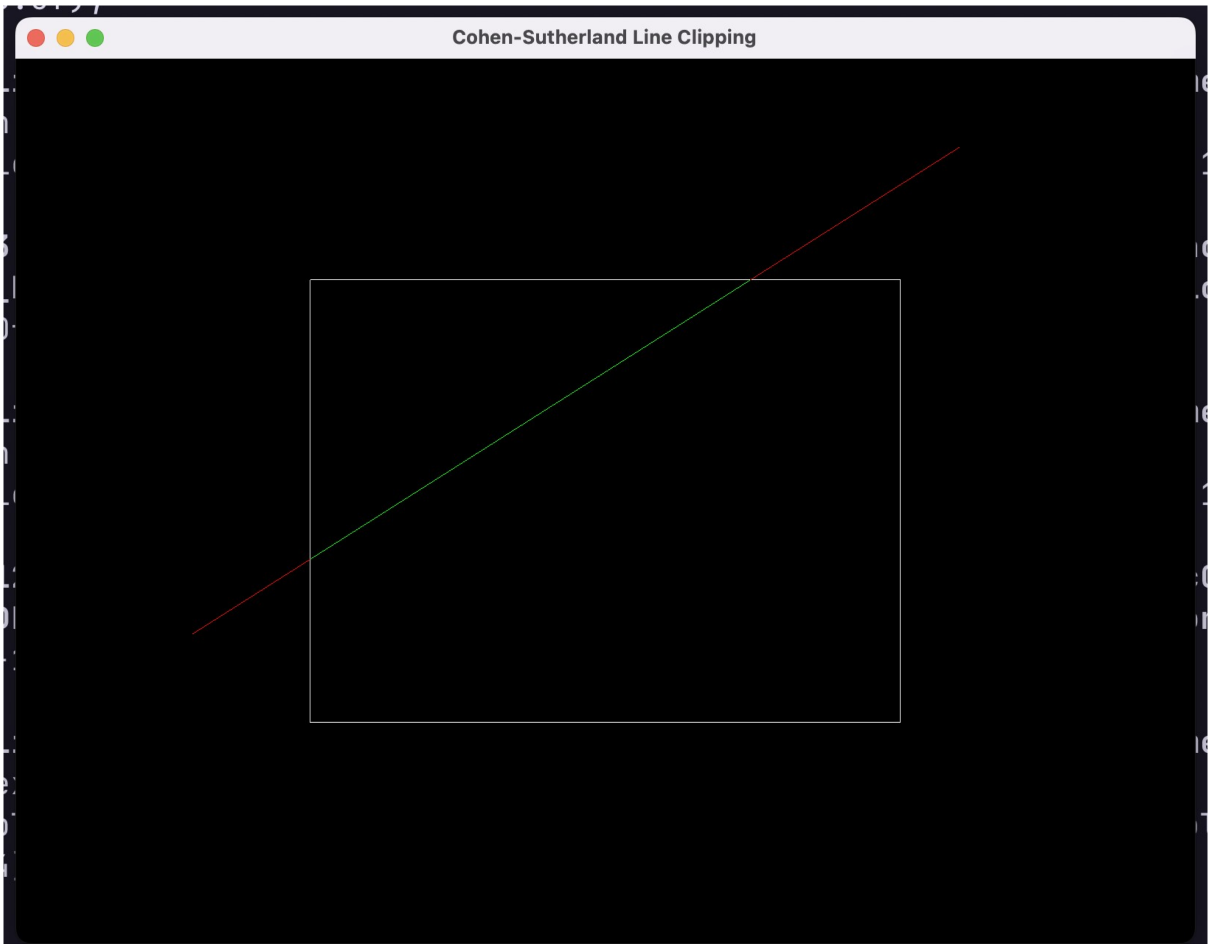
a) COHEN SUTHERLAND LINE CLIPPING ALGORITHM

```
#include <GLFW/glfw3.h>
#include <iostream>

using namespace std;

// Defining 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

// Clipping rectangle boundaries
const float x_min = -0.5f, x_max = 0.5f;
const float y_min = -0.5f, y_max = 0.5f;

// Function to compute region code for a point(x, y)
int computeCode(float x, float 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
bool cohenSutherlandClip(float &x1, float &y1, float &x2, float &y2) {
    int code1 = computeCode(x1, y1);
    int code2 = computeCode(x2, y2);
    bool accept = false;

    while (true) {
        if ((code1 == 0) && (code2 == 0)) {
            accept = true;
            break;
        } else if (code1 & code2) {
            break;
        } else {
            int code_out = (code1 != 0) ? code1 : code2;
            float x, y;

            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);
            }
        }
    }
    return accept;
}

void drawLine(float x1, float y1, float x2, float y2) {
    glBegin(GL_LINES);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glEnd();
}

void drawRectangle() {
    glBegin(GL_LINE_LOOP);
    glVertex2f(x_min, y_min);
    glVertex2f(x_max, y_min);
    glVertex2f(x_max, y_max);
    glVertex2f(x_min, y_max);
    glEnd();
}
```

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 1.0f, 1.0f);
    drawRectangle();

    float x1 = -0.7f, y1 = -0.3f, x2 = 0.6f, y2 = 0.8f;
    glColor3f(1.0f, 0.0f, 0.0f);
    drawLine(x1, y1, x2, y2);

    if (cohenSutherlandClip(x1, y1, x2, y2)) {
        glColor3f(0.0f, 1.0f, 0.0f);
        drawLine(x1, y1, x2, y2);
    }
}

int main() {
    if (!glfwInit()) {
        cerr << "Failed to initialize GLFW" << endl;
        return -1;
    }

    GLFWwindow* window = glfwCreateWindow(800, 600, "Cohen-Sutherland Line
Clipping", NULL, NULL);
    if (!window) {
        cerr << "Failed to create GLFW window" << endl;
        glfwTerminate();
        return -1;
    }

    glfwMakeContextCurrent(window);
    glOrtho(-1, 1, -1, 1, -1, 1);

    while (!glfwWindowShouldClose(window)) {
        display();
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    glfwTerminate();
    return 0;
}
```

OUTPUT:

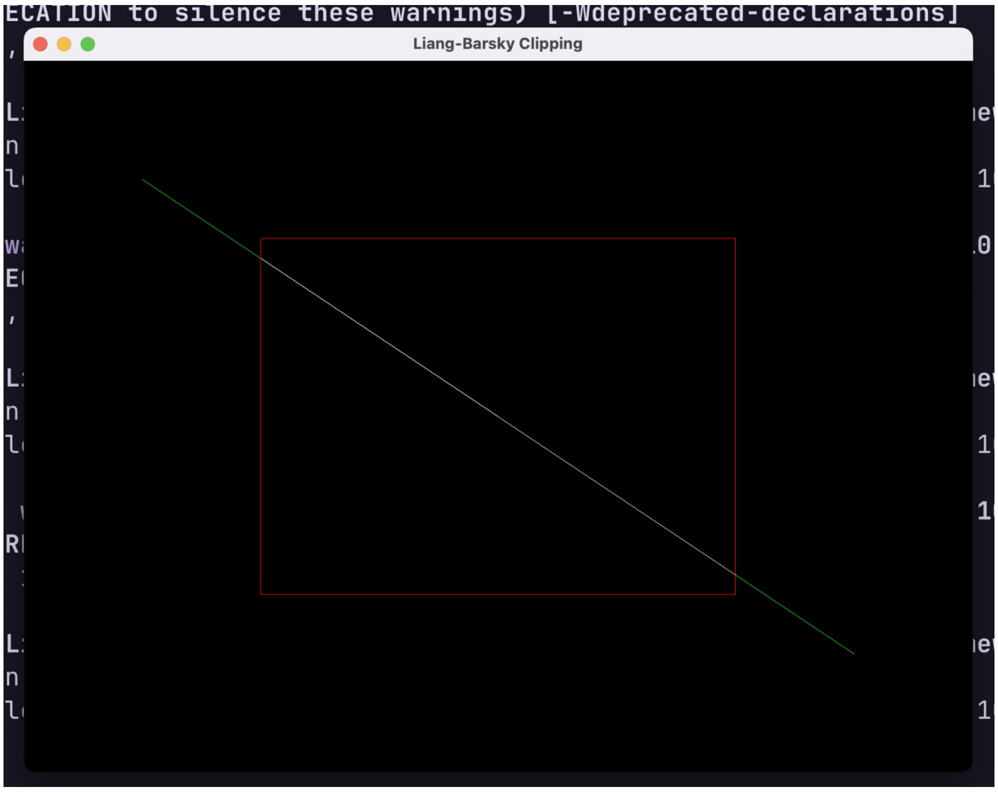


b) LIANG-BARSKY LINE CLIPPING ALGORITHM

```
#include <stdio.h>

#include <stdlib.h>

#include <GLFW/glfw3.h>


const int WIDTH = 800, HEIGHT = 600;


int liangBarsky(float xmin, float ymin, float xmax, float ymax,

            float x1, float y1, float x2, float y2,

            float *xx1, float *yy1, float *xx2, float *yy2) {

    float dx = x2 - x1, dy = y2 - y1;

    float t1 = 0.0, t2 = 1.0;

    float p[4], q[4], t[4];


    p[0] = -dx; q[0] = x1 - xmin;

    p[1] = dx;  q[1] = xmax - x1;

    p[2] = -dy; q[2] = y1 - ymin;

    p[3] = dy;  q[3] = ymax - y1;


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

        if (p[i] != 0) {

            t[i] = (float) q[i] / p[i];

            if (p[i] < 0) {

                if (t[i] > t1) t1 = t[i];

            } else {

                if (t[i] < t2) t2 = t[i];

            }

        } else if (q[i] < 0) {
            return 0; // Line is outside
        }
    }

    if (t1 < t2) {
        *xx1 = x1 + t1 * dx;
        *yy1 = y1 + t1 * dy;
        *xx2 = x1 + t2 * dx;
        *yy2 = y1 + t2 * dy;
        return 1;
    }
    return 0;
}

void drawLine(float x1, float y1, float x2, float y2) {
    glBegin(GL_LINES);
    glVertex2f((2.0 * x1 / WIDTH) - 1.0, 1.0 - (2.0 * y1 / HEIGHT));
    glVertex2f((2.0 * x2 / WIDTH) - 1.0, 1.0 - (2.0 * y2 / HEIGHT));
    glEnd();
}

void drawRectangle(float xmin, float ymin, float xmax, float ymax) {
    glBegin(GL_LINE_LOOP);
    glVertex2f((2.0 * xmin / WIDTH) - 1.0, 1.0 - (2.0 * ymin / HEIGHT));
    glVertex2f((2.0 * xmax / WIDTH) - 1.0, 1.0 - (2.0 * ymin / HEIGHT));
    glVertex2f((2.0 * xmax / WIDTH) - 1.0, 1.0 - (2.0 * ymax / HEIGHT));
    glVertex2f((2.0 * xmin / WIDTH) - 1.0, 1.0 - (2.0 * ymax / HEIGHT));
    glEnd();
}
```

```
int main() {
    if (!glfwInit()) {
        printf("GLFW initialization failed!\n");
        return -1;
    }

    GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Liang-Barsky Clipping",
NULL, NULL);
    if (!window) {
        printf("Window creation failed!\n");
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);

    float xmin = 200, ymin = 150, xmax = 600, ymax = 450;
    float x1 = 100, y1 = 100, x2 = 700, y2 = 500;
    float xx1, yy1, xx2, yy2;
    int isClipped = liangBarsky(xmin, ymin, xmax, ymax, x1, y1, x2, y2, &xx1, &yy1, &xx2,
&yy2);
    while (!glfwWindowShouldClose(window)) {
        glClear(GL_COLOR_BUFFER_BIT);
        glLoadIdentity();
        glColor3f(1.0, 0.0, 0.0); // Red rectangle (clipping window)
        drawRectangle(xmin, ymin, xmax, ymax);

        glColor3f(0.0, 1.0, 0.0); // Green original line
        drawLine(x1, y1, x2, y2);
        if (isClipped) {
            glColor3f(1.0, 1.0, 1.0); // White clipped line
            drawLine(xx1, yy1, xx2, yy2);
        }
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwDestroyWindow(window);
    glfwTerminate();
    return 0;
}
```

OUTPUT:


ECATION to silence these warnings) [-Wdeprecated-declarations]
Liang-Barsky Clipping

CONCLUSION:

Hence, in this lab work we were able to implement Cohen Sutherland Line Clipping Algorithm and Liang-Barsky Line Clipping Algorithm using C programming as the high- level language.