Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

29 September 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 7: Cohen-Sutherland Line Clipping in C++ Using OpenGL

Aim:

Apply Cohen-Sutherland line clipping on a line $(x_1, y_1)(x_2, y_2)$ with respect to a clipping window $(X_W min, Y_W min)(X_W max, Y_W max)$.

After clipping with respect to an edge, display the line segment with the calculated intermediate intersection points and the vertex list.

Input: The clipping window co-ordinates and the line endpoints.

Note: The output should show the clipping window and the line to be clipped in different colors.

You can show the intermediate steps using time delay.

Code: Cohen-Sutherland Algorithm:

```
/*
To perform the Cohen-Sutherland Line Clipping Algorithm on a given line,
based upon a rectangular clipping window
*/

#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#include <iostream>
#include <cstring>

using namespace std;

const int LeftBit = 0x1;
const int RightBit = 0x2;
const int BottomBit = 0x4;
const int TopBit = 0x8;

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;
const int FPS = 60;

class Point
{
private:
    GLfloat x, y;

public:
    Point()
    {
        x = y = 0;
    }

    Point(GLfloat X, GLfloat Y)
    {
        x = X;
        y = Y;
    }

    GLfloat getX()
    {
        return x;
    }

    GLfloat getY()
    {
        return y;
```

```
    }

    void setX(GLfloat X)
    {
        x = X;
    }

    void setY(GLfloat Y)
    {
        y = Y;
    }

    int encode(Point windowMin, Point windowMax)
    {
        int RC = 0x00;

        if (x < windowMin.getX())
        {
            RC = RC | LeftBit;
        }
        if (x > windowMax.getX())
        {
            RC = RC | RightBit;
        }
        if (y < windowMin.getY())
        {
            RC = RC | BottomBit;
        }
        if (y > windowMax.getY())
        {
            RC = RC | TopBit;
        }

        return RC;
    }
};

class Line
{
private:
    Point p, q;

public:
    Line()
    {
        p.setX(0); p.setY(0);
        q.setX(0); q.setY(0);
    }

    Line(float x1, float y1, float x2, float y2)
    {
```

```
        p.setX(x1); p.setY(y1);
        q.setX(x2); q.setY(y2);
    }

    Point getP()
    {
        return p;
    }

    Point getQ()
    {
        return q;
    }

    void setP(float x, float y)
    {
        p.setX(x); p.setY(y);
    }

    void setQ(float x, float y)
    {
        q.setX(x); q.setY(y);
    }

    int isInside(int RC)
    {
        return !RC;
    }

    int trivialReject(int RC1, int RC2)
    {
        return (RC1 & RC2);
    }

    int trivialAccept(int RC1, int RC2)
    {
        return (!(RC1 | RC2));
    }

    void swapPoints()
    {
        Point x;
        x.setX(p.getX()); x.setY(p.getY());
        p.setX(q.getX()); p.setY(q.getY());
        q.setX(x.getX()); q.setY(x.getY());
    }

    bool lineClipCohenSutherland(Point windowMin, Point windowMax)
    {
        int RC1, RC2;
        int plotLine = false, done = false;
```

```
        GLfloat m;

        while (!done)
        {
            RC1 = p.encode(windowMin, windowMax);
            RC2 = q.encode(windowMin, windowMax);

            //cout << "RC1: " << RC1 << "RC2: " << RC2 << endl;

            if (trivialAccept(RC1, RC2))
            {
                //Line coordinates are inside boundary
                done = true;
                continue;
            }

            if (trivialReject(RC1, RC2))
            {
                //Line coordinates are outside boundary
                done = true;
                continue;
            }

            plotLine = true;    //Clipped Line needs to be highlighted

            if (isInside(RC1))
            {
                //If P is inside, then swap P with Q.
                swapPoints();
                RC1 = p.encode(windowMin, windowMax);
                RC2 = q.encode(windowMin, windowMax);
            }

            if (q.getX() != p.getX())
            {
                //Avoid dx = 0 case
                m = (q.getY() - p.getY()) / (q.getX() - p.getX());
            }

            if (RC1 & LeftBit)
            {
                p.setY(p.getY() + (windowMin.getX() - p.getX()) * m);
                p.setX(windowMin.getX());
            }

            else if (RC1 & RightBit)
            {
                p.setY(p.getY() + (windowMax.getX() - p.getX()) * m);
                p.setX(windowMax.getX());
            }

```

```
            else if (RC1 & BottomBit)
            {
                if (p.getX() != q.getX())
                {
                    p.setX(p.getX() + (windowMin.getY() - p.getY()) / m);
                    p.setY(windowMin.getY());
                }
            }

            else if (RC1 & TopBit)
            {
                if (p.getX() != q.getX())
                {
                    p.setX(p.getX() + (windowMax.getY() - p.getY()) / m);
                    p.setY(windowMax.getY());
                }
            }
        }

        return plotLine;
    }
};

void dummyFunction();
void mainLoop(int val);
void initializeDisplay();
void drawLine(Point p, Point q, bool clip = false);
void drawClippingWindow(Point windowMin, Point windowMax);
void getParams();

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");

    cout << "\n\t\t-----[COHEN SUTHERLAND LINE CLIPPING ALGORITHM]-----\n"
    ;

    initializeDisplay();
    glutDisplayFunc(dummyFunction);
    glutTimerFunc(1000/FPS, mainLoop, 0);
    glutMainLoop();

    return 1;
}

void mainLoop(int val)
{
```

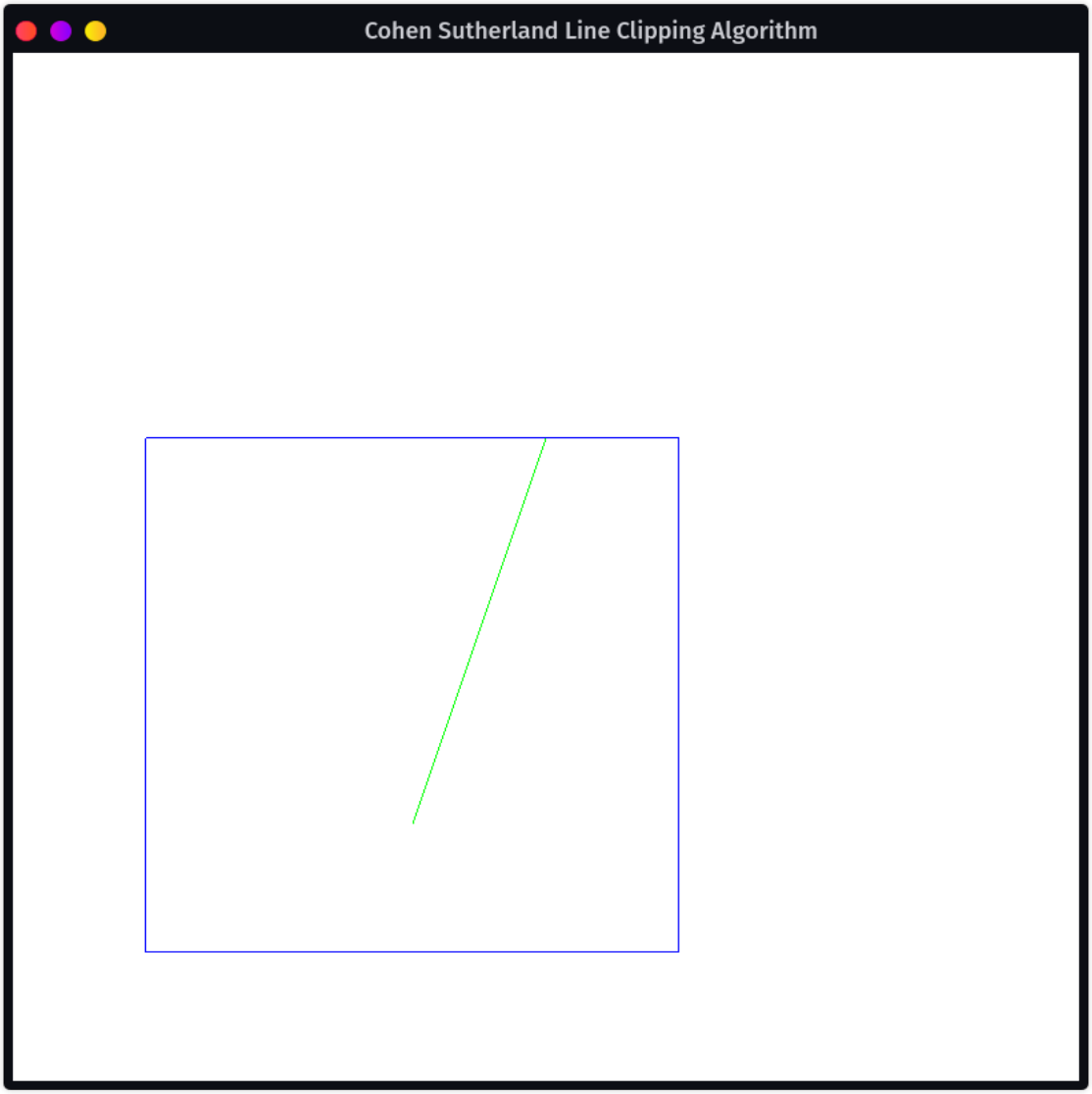
```
    //Render the display using the timer function
    getParams();
}

void dummyFunction()
{
    //Placeholder function
}

void initializeDisplay()
{
    //Initialize the display parameters

    glClearColor(1, 1, 1, 0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT);
    glClear(GL_COLOR_BUFFER_BIT);
}

void getParams()
{
    //To get user parameters for the line clipping process

    Point windowMin = Point(100, 100), windowMax = Point(500, 500);
    Line lineSegment = Line(300, 200, 400, 500);

    int option = 0;

    drawClippingWindow(windowMin, windowMax);
    drawLine(lineSegment.getP(), lineSegment.getQ());
    glFlush();


    while (true)
    {
        //Await user input
        cout << "\n\t1. Set Line Coordinates" << endl;
        cout << "\t2. Set Clipping Window" << endl;
        cout << "\t0. Exit" << endl;
        cout << "\tYour Option -> ";
        cin >> option;

        if (!option)
        {
            cout << "\n\t\t-----[COHEN SUTHERLAND LINE CLIPPING ALGORITHM
    ]-----\n";
            exit(0);
        }

        if (option == 1)
        {
```

```
            float x, y;

            cout << "\n\n\tEnter Point P:" << endl;
            cout << "\t\tEnter X: ";
            cin >> x;
            cout << "\t\tEnter Y: ";
            cin >> y;
            lineSegment.setP(x, y);

            cout << "\n\n\tEnter Point Q:" << endl;
            cout << "\t\tEnter X: ";
            cin >> x;
            cout << "\t\tEnter Y: ";
            cin >> y;
            lineSegment.setQ(x, y);
        }

        if (option == 2)
        {
            float x, y;
            cout << "\n\n\tEnter Window Minimums:" << endl;
            cout << "\t\tEnter X: ";
            cin >> x;
            cout << "\t\tEnter Y: ";
            cin >> y;
            windowMin.setX(x);
            windowMin.setY(y);

            cout << "\n\n\tEnter Window Maximums:" << endl;
            cout << "\t\tEnter X: ";
            cin >> x;
            cout << "\t\tEnter Y: ";
            cin >> y;
            windowMax.setX(x);
            windowMax.setY(y);

            glClear(GL_COLOR_BUFFER_BIT);   //Clear the display window
        }

        drawClippingWindow(windowMin, windowMax);
        drawLine(lineSegment.getP(), lineSegment.getQ());
        glFlush();

        bool plotLine = lineSegment.lineClipCohenSutherland(windowMin, windowMax);

        if (plotLine)
        {
            drawLine(lineSegment.getP(), lineSegment.getQ(), true);
        }

```

```
        glFlush();
    }
}

void drawLine(Point p, Point q, bool clip)
{
    glBegin(GL_LINES);

    if (clip)
    {
        glColor3d(1, 0, 0);
    }
    else
    {
        glColor3d(0, 1, 0);
    }

    glVertex2f(p.getX(), p.getY());
    glVertex2f(q.getX(), q.getY());

    glEnd();
    glFlush();
}

void drawClippingWindow(Point windowMin, Point windowMax)
{
    glBegin(GL_LINE_LOOP);
    glColor3d(0, 0, 1);

    glVertex2f(windowMin.getX(), windowMin.getY());
    glVertex2f(windowMax.getX(), windowMin.getY());
    glVertex2f(windowMax.getX(), windowMax.getY());
    glVertex2f(windowMin.getX(), windowMax.getY());

    glEnd();
    glFlush();
}
```

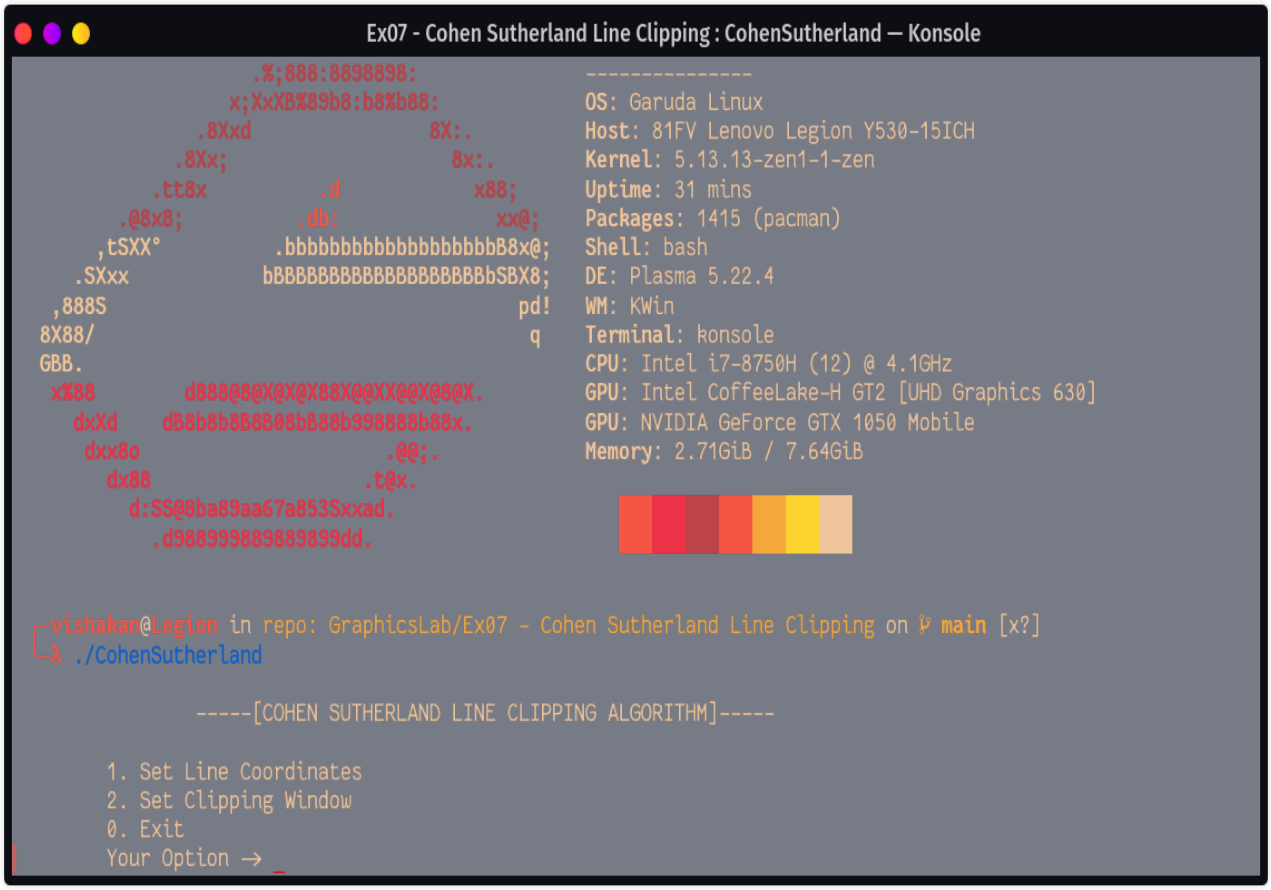
Output: Default Line & Clipping Window

Figure 1: Default Line & Clipping Window



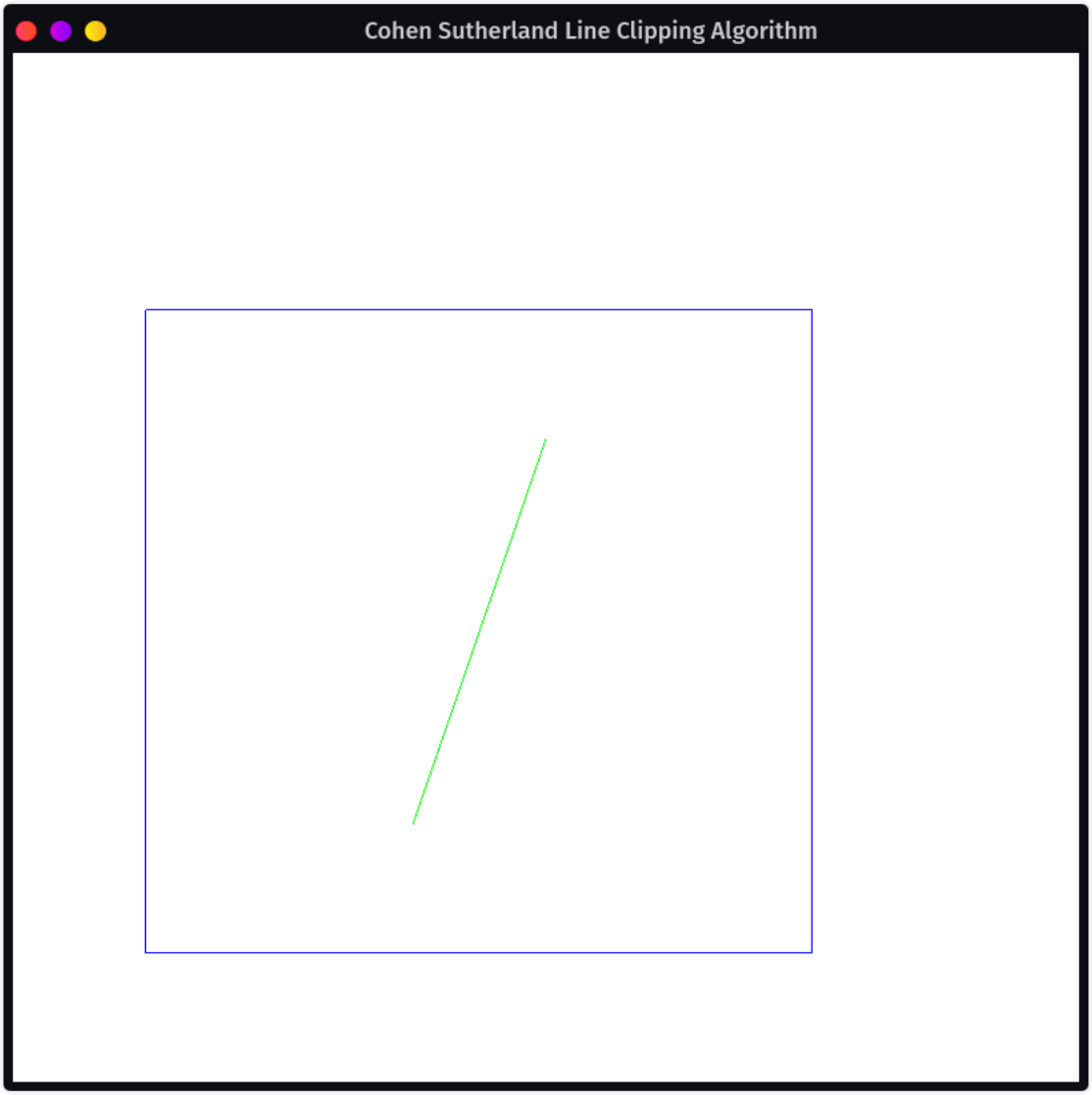
Output: Console, Window[(100, 100), (500, 500)]

Figure 2: Output: Console, Window[(100, 100), (500, 500)]



Output: Window[(100, 100), (600, 600)]

Figure 3: Output: Window[(100, 100), (600, 600)].

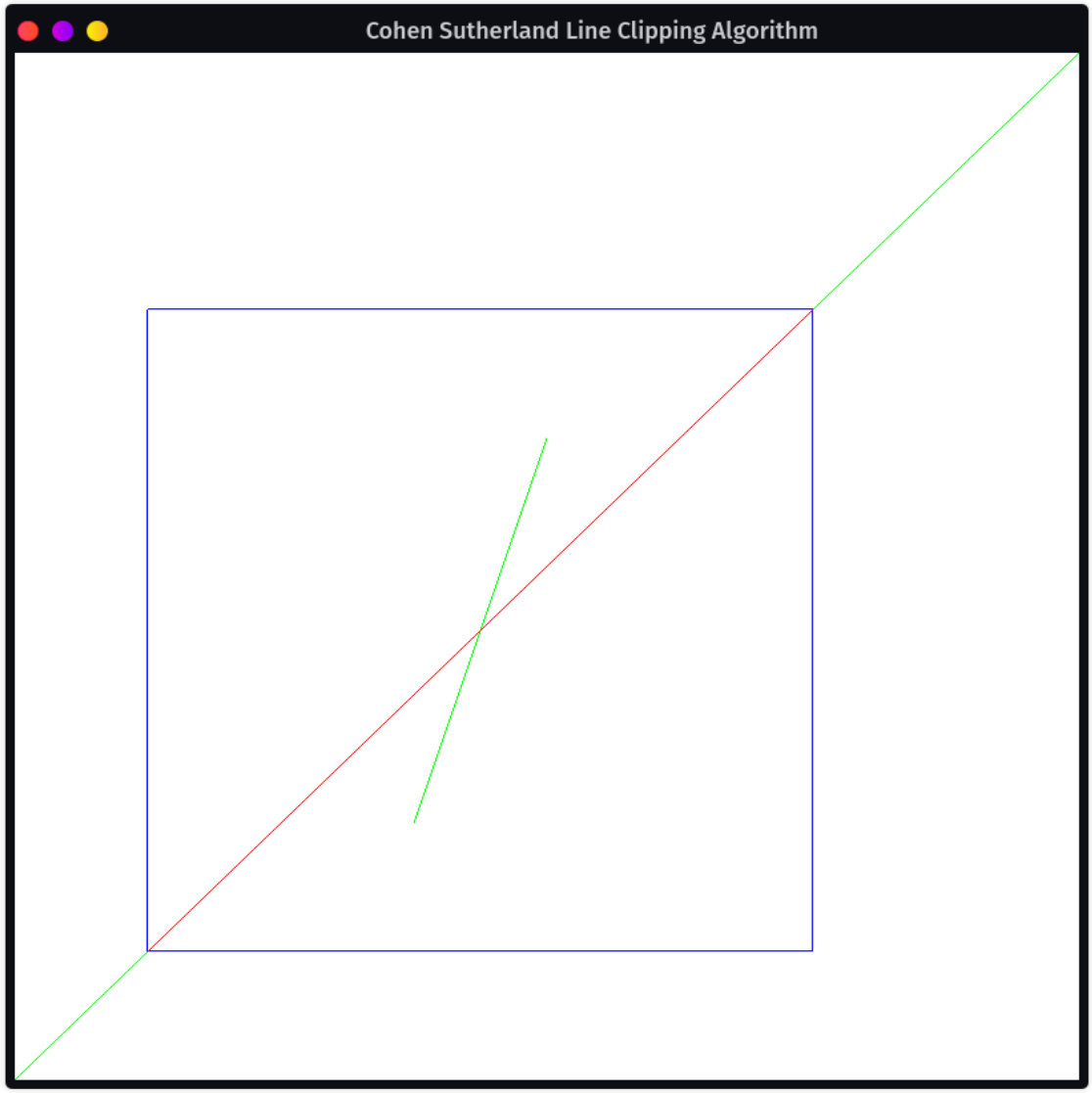


Output: Console, Window[(100, 100), (600, 600)]

Figure 4: Output: Console, Window[(100, 100), (600, 600)].

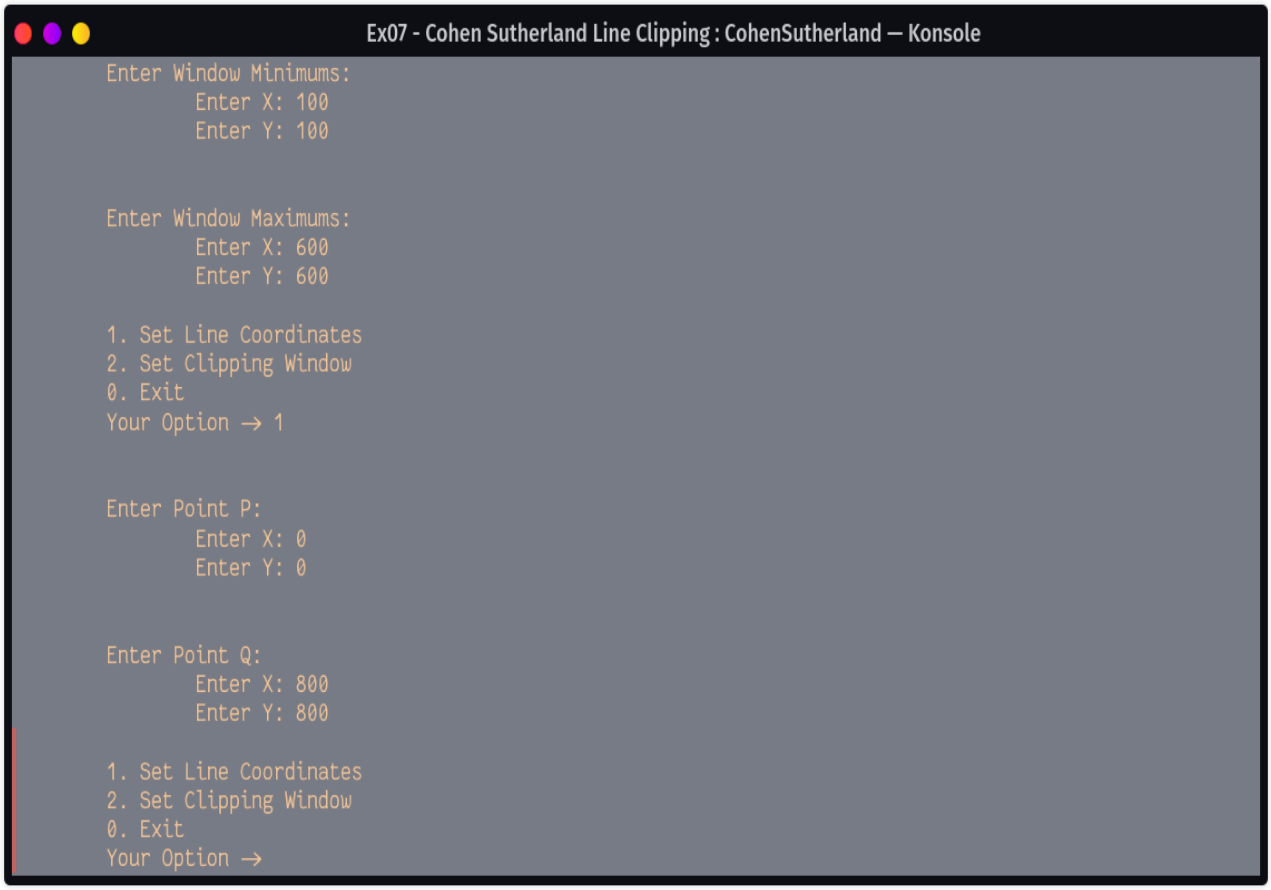
Output: Line[(0, 0), (800, 800)] & Clipping

Figure 5: Output: Line[(0, 0), (800, 800)] & Clipping.



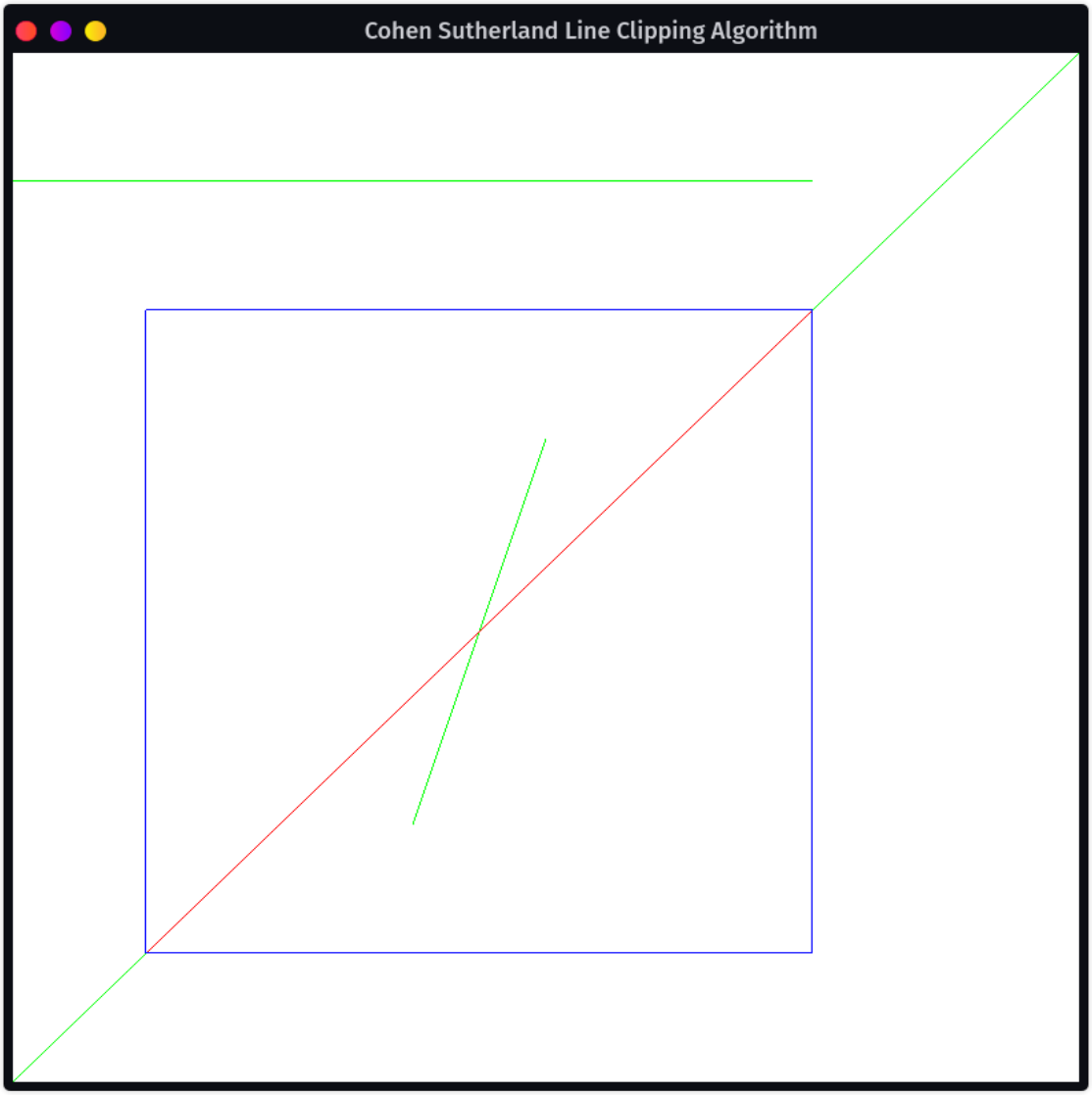
Output: Console, Line[(0, 0), (800, 800)] & Clipping

Figure 6: Output: Console, Line[(0, 0), (800, 800)] & Clipping.



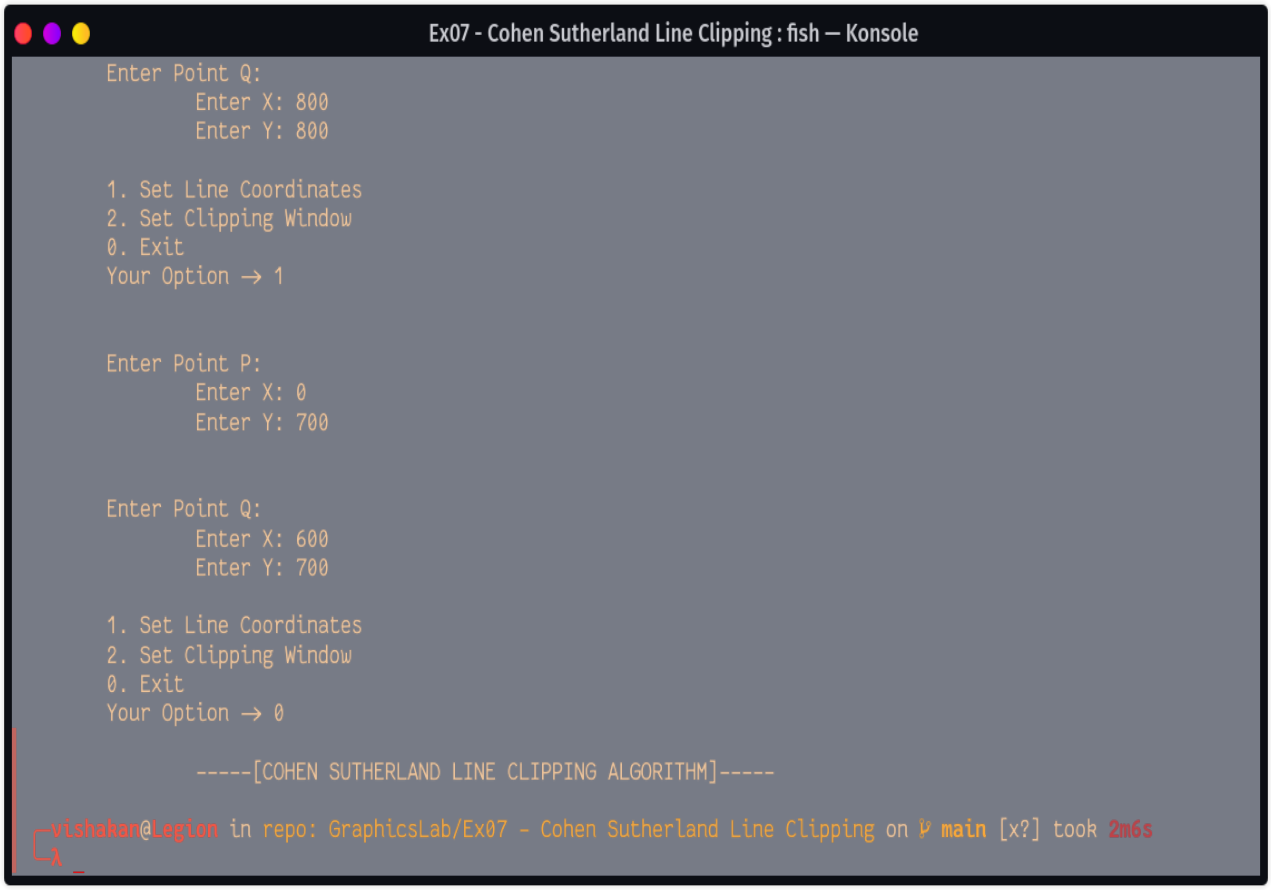
Output: Line[(0, 700), (600, 700)] & Clipping

Figure 7: Output: Line[(0, 700), (600, 700)] & Clipping.



Output: Console, Line[(0, 700), (600, 700)] & Clipping

Figure 8: Output: Console, Line[(0, 700), (600, 700)] & Clipping.



Learning Outcome:

- I learnt about **Cohen-Sutherland Line Clipping Algorithm**'s theoretical basis.
- I learnt how to compute and program **Region Codes** for a given point and clipping window.
- I understood about the advantages and pitfalls of the Cohen-Sutherland Algorithm.
- I understood how to implement **trivial accept & reject** conditions in the algorithm.
- I was able to implement the algorithm for any given line and a rectangular clipping window region.
- I demonstrated the clipped line, the original line and the clipping window using different colors.
- I handled corner cases where $dx = 0$ and thus avoiding divide by zero errors in slope calculation.
- I understood how to swap the points and continue clipping till region codes of both points become 0 for trivial acceptance.