Ex. No. 7	**Cohen Sutherland's Line Clipping Algorithm**	**Karthik D**
October 12, 2022	**UCS1712 - Graphics and Multimedia Lab**	**195001047**

Aim

To develop a C++ program using the OpenGL framework to implement Cohen Sutherlan's line clipping algorithm, and demonstrate all its output cases.

Question

Apply Cohen Sutherland line clipping on a line (x1,y1) (x2,y2) with respect to a clipping window. After clipping with an edge, display the line segment with the calculated intermediate intersection points.

Cohen Sutherland Clipping Algorithm

1. Calculate positions of both endpoints of the line.
2. Perform Bitwise-OR operation on both of these end-points.
3. If the OR operation gives 000,
    a. Then, the line is considered to be visible. TERMINATE.
    b. Else, go to step 4
4. Perform Bitwise-AND operation on both endpoints
5. If And ≠ 000,
    a. Then, the line is invisible. TERMINATE.
    b. Else, the line is considered the clipped case.
6. If line is clipped, find an intersection with boundaries of the window

    Let the slope be, $m = (y_2-y_1)(x_2-x_1)$

    a. If *bit 1 is "1"* line intersects with left boundary of rectangle window

    $y_3 = y_1 + m(x - x_1)$

    where x is the minimum value of X coordinate of window

    b. If *bit 2 is "1"* line intersect with right boundary

    $y_3 = y_1 + m (x - x_1)$

    where x more is the maximum value of X coordinate of the window

c. If *bit 3 is "1"* line intersects with bottom boundary

$x_3 = x_1 + (y - y_1) / m$

where y is the minimum value of Y coordinate of the window

d. If *bit 4 is "1"* line intersects with the top boundary

$x_3 = x_1 + (y - y_1) / m$

y is the maximum value of Y coordinate of the window

Implementation using C++ Program Code

1. main.cpp - Driver and handler to render the clipped line, window and original line

```
#include <GL/glut.h>
#include <stdio.h>


void renderSpacedBitmapString(float x, float y, void *font, char
*string) {
   char *c;
   int x1 = x;
   for (c = string; *c != '\0'; c++) {
       glRasterPos2f(x1, y);
       glutBitmapCharacter(font, *c);
       x1 = x1 + glutBitmapWidth(font, *c);
   }
}

void markString(char *string, int x, int y, int x_offset, int y_offset)
{
   glColor3f(255.0, 0, 0.0); // red color
   renderSpacedBitmapString(x+x_offset, y+y_offset,
GLUT_BITMAP_HELVETICA_12, string);
   glFlush();
}


typedef short* RegionCode;
```

```
short trivial_accept(RegionCode code_1, RegionCode code_2)  {
   // all zeros in result implies accept --> return 1
   short result = 0;
   for(int i=0; i<4; i++)  {
       result = result || (code_1[i] || code_2[i]);
   }
   return !result;
}

short trivial_reject(RegionCode code_1, RegionCode code_2)  {
   // all zeros in result implies reject the reject! --> return 1
   short result = 0;
   for(int i=0; i<4; i++)  {
       result = result || (code_1[i] && code_2[i]);
   }
   return !result;
}


struct window_constraints   {
   int x_min;
   int x_max;
   int y_min;
   int y_max;
};
typedef struct window_constraints WindowConstraints;

struct point    {
   float x;
   float y;
};
typedef struct point Point;

void display_point(Point pt)    {
   printf("\n(%f, %f)", pt.x, pt.y);
}
```

```
RegionCode get_region_code(Point pt, WindowConstraints window)    {
   // TBRL
   RegionCode code = (short*)malloc(sizeof(short)*(4));

   if(pt.x < window.x_min) {
       code[3] = 1;
       code[2] = 0;
   }
   else if(pt.x > window.x_max)    {
       code[3] = 0;
       code[2] = 1;
   }
   else    {
       code[3] = 0;
       code[2] = 0;
   }

   if(pt.y < window.y_min) {
       code[1] = 1;
       code[0] = 0;
   }
   else if(pt.y > window.y_max)    {
       code[1] = 0;
       code[0] = 1;
   }
   else    {
       code[1] = 0;
       code[0] = 0;
   }
   // printf("\n() %d %d %d %d", code[0], code[1], code[2], code[3]);

   return code;
}

short is_outside(RegionCode code)   {
   // if any code is 1 -> lies outside -> return 1
   short result = 0;
   for(int i=0; i<4; i++)  {
```

```
        result = result || code[i];
    }
    return result;
}


void plotLine(Point p1, Point p2)   {
    glBegin(GL_LINES);
    glVertex2f(p1.x, p1.y);
    glVertex2f(p2.x, p2.y);
    glEnd();

    char *string = (char*)malloc(sizeof(char)*100);
    sprintf(string, "(%.2f, %.2f)", p1.x, p1.y);
    markString(string, p1.x, p1.y, 0, 0);

    sprintf(string, "(%.2f, %.2f)", p2.x, p2.y);
    markString(string, p2.x, p2.y, 0, 0);
}

void plotWindow(WindowConstraints window)   {
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(window.x_min, window.y_min);
    glVertex2f(window.x_min, window.y_max);
    glVertex2f(window.x_max, window.y_max);
    glVertex2f(window.x_max, window.y_min);
    glEnd();

    char *string = (char*)malloc(sizeof(char)*100);
    sprintf(string, "(%d, %d)", window.x_min, window.y_min);
    markString(string, window.x_min, window.y_min, 0, 0);

    sprintf(string, "(%d, %d)", window.x_min, window.y_max);
    markString(string, window.x_min, window.y_max, 0, 0);

    sprintf(string, "(%d, %d)", window.x_max, window.y_max);
    markString(string, window.x_max, window.y_max, 0, 0);
```

```
   sprintf(string, "(%d, %d)", window.x_max, window.y_min);
   markString(string, window.x_max, window.y_min, 0, 0);
}


void display_line_clipping()    {

   WindowConstraints view_window = {10, 400, 10, 300};
   Point start_pt = {30, 400};
   Point end_pt = {500, 25};
   // Point start_pt = {30, 200};
   // Point end_pt = {500, 25};
   // Point start_pt = {30, 200};
   // Point end_pt = {350, 25};

   float slope = (float) (end_pt.y - start_pt.y)/(end_pt.x -
start_pt.x);
   Point _start_pt = start_pt;
   Point _end_pt = end_pt;
   RegionCode start_pt_code = get_region_code(_start_pt, view_window);
   RegionCode end_pt_code = get_region_code(_end_pt, view_window);
   short x_bound, y_bound;

   Point *outside_pt = (Point*)malloc(sizeof(Point));
   RegionCode outside_pt_code;
   short is_clipped= trivial_accept(start_pt_code, end_pt_code) || (
       !trivial_accept(start_pt_code, end_pt_code) &&
trivial_reject(start_pt_code, end_pt_code)
   );
   while(is_clipped) {

       if(is_outside(start_pt_code))  {
           outside_pt = &_start_pt;
           outside_pt_code = start_pt_code;
       }
       else if(is_outside(end_pt_code))   {
           outside_pt = &_end_pt;
```

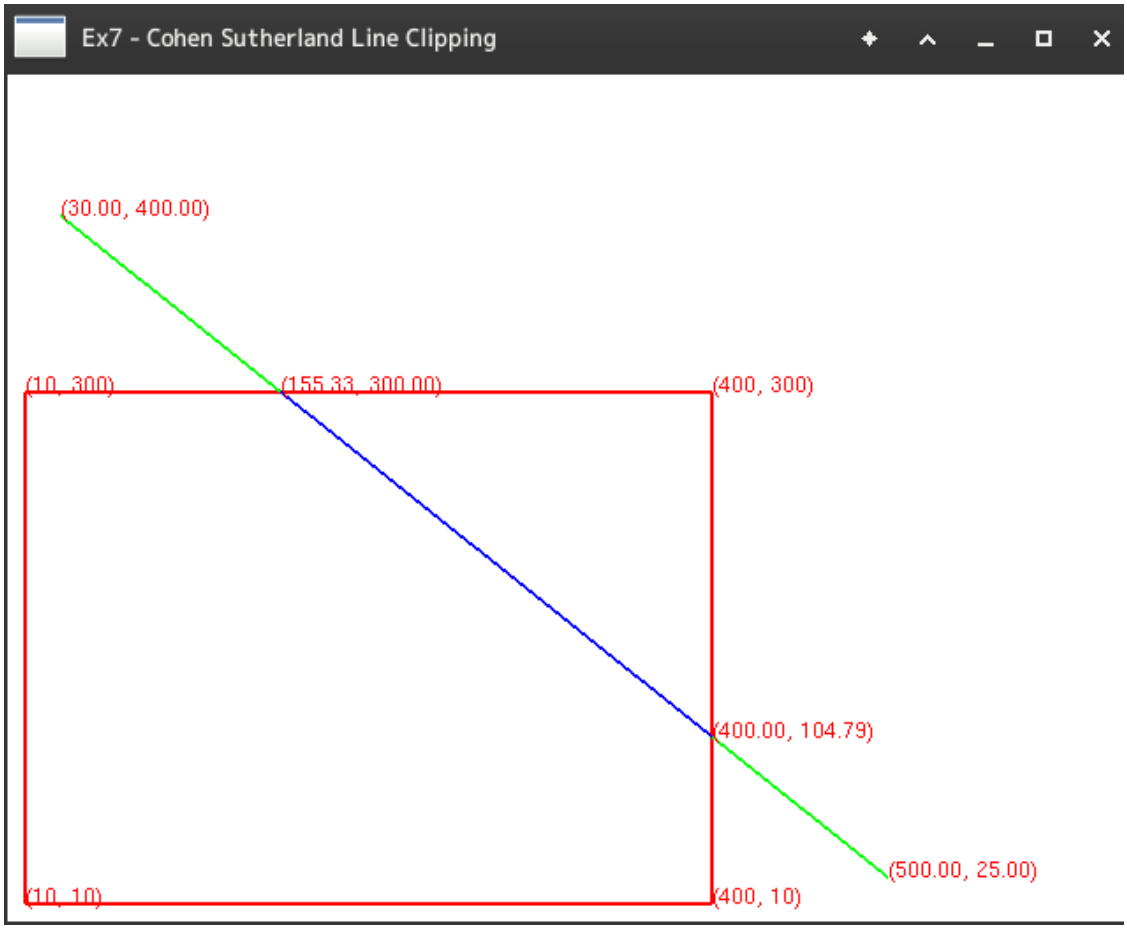
```
            outside_pt_code = end_pt_code;
        }

        // find intersection
        x_bound = -1;
        x_bound = outside_pt_code[2] - outside_pt_code[3];
        y_bound = outside_pt_code[0] - outside_pt_code[1];
        if(x_bound!=0)  {
            // solve y
            if(x_bound==-1) {
                // LEFT intersection
                outside_pt->y = (float) outside_pt->y +
slope*(view_window.x_min - outside_pt->x);
                outside_pt->x = view_window.x_min;
            }
            else    {
                // RIGHT intersection
                outside_pt->y = (float) outside_pt->y +
slope*(view_window.x_max - outside_pt->x);
                outside_pt->x = view_window.x_max;
            }
        }
        else{
            // solve x
            if(y_bound==-1) {
                // BOTTOM intersection
                outside_pt->x = (float) outside_pt->x +
(1/slope)*(view_window.y_min - outside_pt->y);
                outside_pt->y = view_window.y_min;
            }
            else if(y_bound==1)  {
                // TOP intersection
                outside_pt->x = (float) outside_pt->x +
(1/slope)*(view_window.y_max - outside_pt->y);
                outside_pt->y = view_window.y_max;
            }
            else{
                printf("\nUnexpected error\n");
```

```
            }
        }

        start_pt_code = get_region_code(_start_pt, view_window);
        end_pt_code = get_region_code(_end_pt, view_window);
        is_clipped = !trivial_accept(start_pt_code, end_pt_code) &&
trivial_reject(start_pt_code, end_pt_code);
    }

    glClear(GL_COLOR_BUFFER_BIT);
    plotWindow(view_window);
    glColor3f(0.0, 1.0, 0.0);
    plotLine(start_pt, end_pt);
    glColor3f(0.0, 0.0, 1.0);
    plotLine(_start_pt, _end_pt);
    glFlush();
}


void init() {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(2);
    glLineWidth(2);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 640.0, 0, 480.0);
}


int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);

    glutCreateWindow("Ex7 - Cohen Sutherland Line Clipping");
    glutDisplayFunc(display_line_clipping);
```

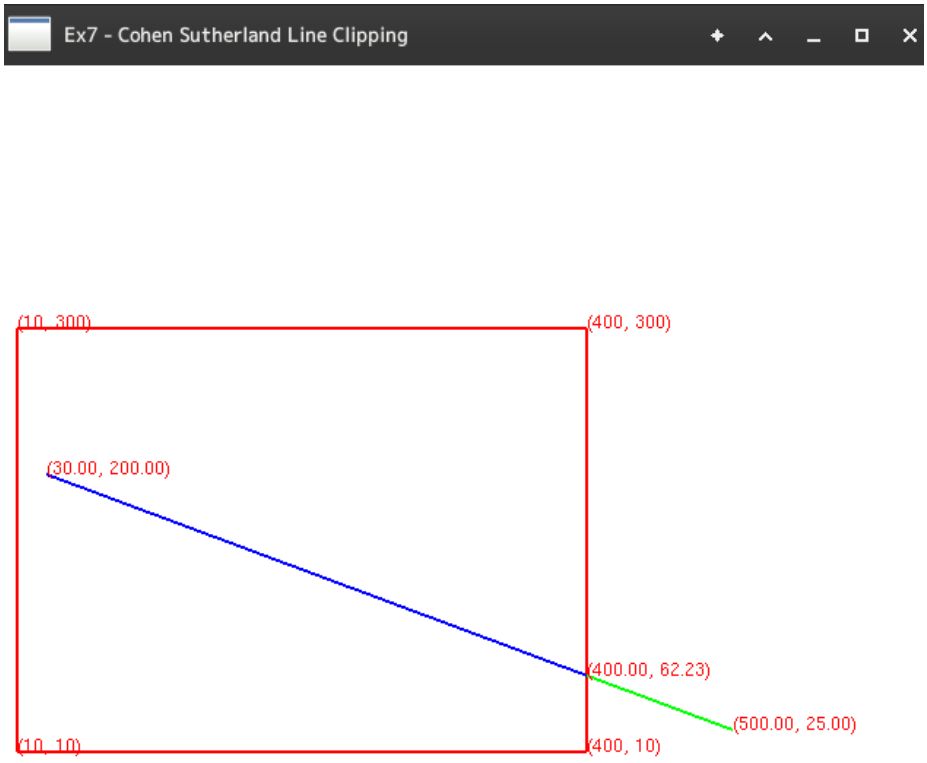
```
    init();
    glutMainLoop();
    return 0;
}
```

Sample Output

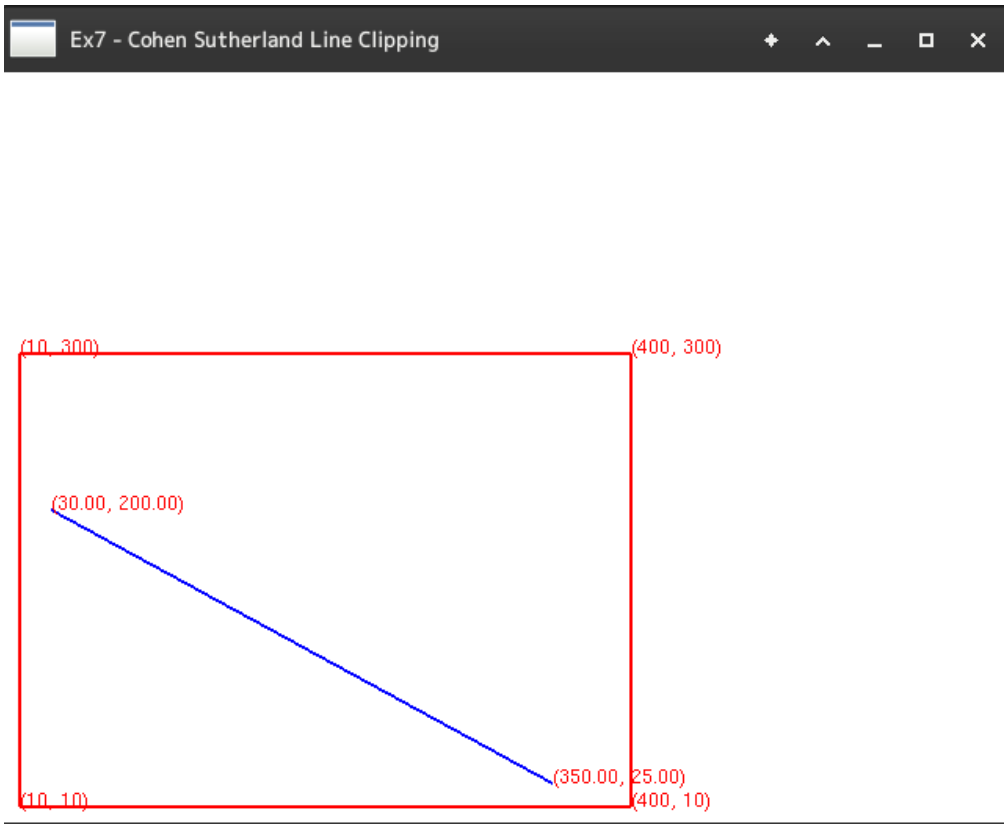
- **Both sides clipped**



- **One side clipped**



- **Line lies completely inside the window**



Learning Outcomes

Through this implementation of the Cohen Sutherland line clipping algorithm using the OpenGL framework and C++ programming language, the following concepts were learnt:

1. The working details of Cohen Sutherland line clipping algorithm.
2. The use and application of line clipping when windowing is performed in graphical rendering.
3. General understanding of the OpenGL framework and its APIs.