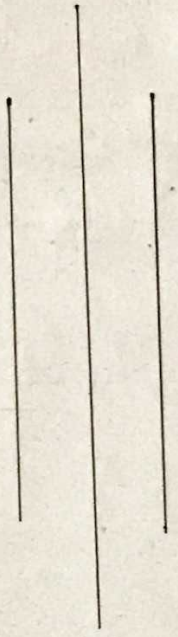
KATHMANDU UNIVERSITY

Dhulikhel, Kavre



Subject:- COMP 342

LAB no. 5

Submitted by,

Name: Prajal Ghimire

Roll no:- 15

Group:- CS

Year:- 3

Submitted to,

Mr. Dhiraj Shrestha

(Department of Computer Science and engineering)

Pranjal Ghumare

1) Implement Cohen / Sutherland Line Clipping Algorithm.

⇒ Algorithm

Step1 : Calculate positions of both endpoints of the line.

Step2 : Perform OR operation on both of these 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 visible.

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/p

Pranjal Ghimire

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

$Y_3 = Y_1 + m(x - x_1)$, where $x = X_{wmin}$

X_{wmin} is the minimum value of x-coordinate of the window.

b) If bit 2 is "1" line intersects with right boundary

$x_3 = x_1 + (y - y_1)/m$, where $y = y_{wmax}$

y_{wmax} is the maximum value of Y-coordinate of the window.

Code Screenshots

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>CohenSutherland Line clipping algorithm</title>
        <script src="sketch.js"></script>
        <style>
            body {
                margin: 0;
                padding: 0;
            }
            canvas {
                margin: auto;
            }
        </style>
    </head>
    <body>
        <h1><b><u>Output</u></b></h1>
        <canvas width="800" height="800" id= "canvas"></canvas>
        <script type="text/javascript">
            var start=[];
            var end=[];

            //Stack that keep tracks of the end points.
            var stack=[];

            //viewport is a square of vertices a, b, c and d
            var ax = 200;
            var ay = 200;
            var bx = 500;
            var by = 200;
            var cx = 500;
            var cy = 500;
            var dx = 200;
            var dy = 500;

            //min max values required to create the outcodes
            var xmin = ax
            var xmax = cx
            var ymin = ay;
            var ymax = cy;

            //Boilerplate code. Required to access the html5 canvas
            var canvas = document.getElementById("canvas");
            var context = canvas.getContext("2d");

```

```
47
48          //Google "html5 compositing"
49          context.globalCompositeOperation = 'source-over';
50
51          //The next 4 blocks draw our viewport
52          //drawing line x=200, of length 300 units (pixels)
53          context.beginPath();
54          context.moveTo(ax, ay);
55          context.lineTo(bx, by);
56          context.stroke();
57
58          //drawing line y=200, length 300 pixels
59          context.beginPath();
60          context.moveTo(bx, by);
61          context.lineTo(cx,cy);
62          context.stroke();
63
64          //drawing line x=500
65          context.beginPath();
66          context.moveTo(cx, cy);
67          context.lineTo(dx, dy);
68          context.stroke();
69
70          //drawing line y=500
71          context.beginPath();
72          context.moveTo(dx, dy);
73          context.lineTo(ax, ay);
74          context.stroke();
75
76          draw_line1();
77          draw_line2();
78          draw_line3();
79          draw_line4();
80
81          //This is an event listener
82          //The function inside fires only when the mouse is clicked
83          canvas.addEventListener('mousedown', function(evt){
84
85              if(stack.length > 0)
86                  end_points = stack.pop();
87              console.log('end points of line : ' + end_points[0] + " to " + end_points[1]);
88              clip(end_points);
89
90          });
91
92
```

```
function clip(end_points)
{
    //New variables to hold end points. No relation
    //to global 'start' and 'end'

    start_ = end_points[0];
    end_ = end_points[1];

    o1 = set_outcode(start_);
    o2 = set_outcode(end_);

    console.log('outcodes are : ' + o1 + ' and ' + o2);

    //Botht the outcodes are 0. This means both end points are inside the viewport
    if(o1 == '0000' && o2 == '0000')
        console.log('accept');

    //both the outcodes have the same bit set when both end points are outside viewport
    else if( (o1 & o2) != 0)
    {
        console.log('reject');
        delete_line(start_, end_);
    }

    //One end point inside viewport and one outside viewport
    else if( (o1 & o2) == 0 && o1 == '0000' || o2 == '0000')
    {
        intersections = find_intersection(o1, end_points);

        //Assuming there is only one intersection. TO DO : HANDLE MULTIPLE INTERSECTIONS
        console.log("Intersections are : " + intersections[0]);

        if(o1 != '0000')
        {
        delete_line(start_, intersections[0]);
        }
        else if(o2 != '0000')
        {
        delete_line(end_, intersect[0]);
        }
    }

```

```
            //When both end points are outside viewport but portion of line is inside
            else if( (o1 & o2) == 0)
            {
                intersections = find_intersection(o1, end_points);
                console.log("Intersections of start point : " + intersections[0]);
                delete_line(start_, intersections[0]);

                intersections = find_intersection(o2, end_points);
                console.log("Intersections of end point : " + intersections[0]);
                delete_line(end_, intersections[0]);
            }
        }

        function set_outcode(point)
        {
            outcode = '';

            x = point[0];
            y = point[1];

            if(y > ymax)
                outcode = outcode + '1';
            else
                outcode = outcode + '0';

            if(y < ymin)
                outcode = outcode + '1';
            else
                outcode = outcode + '0';

            if(x > xmax)
                outcode = outcode + '1';
            else
                outcode = outcode + '0';

            if(x < xmin)
                outcode = outcode + '1';
            else
                outcode = outcode + '0';

            return outcode;
        }

```

```
178         function delete_line(start_, end_)
179         {
180         //To delete a line, we draw a white line
181         //over the original line.
182
183         context.beginPath();
184         context.moveTo(start_[0], start_[1]);
185         context.lineTo(end_[0], end_[1]);
186
187         context.strokeStyle = '#ffffff';
188
189         //We need a larger linewidth due to some issues in html5 canvas.
190         //white line of same width will only dim the original line.
191         //No issues with same width when drawing over another color than white
192         context.lineWidth = 2;
193         context.stroke();
194         }
195
196         function find_intersection(outcode, end_points)
197         {
198
199         start_ = end_points[0];
200         end_ = end_points[1];
201
202         //All lines are of the form y = mx + c
203         //m = (y2-y1)/(x2-x1)
204         x1 = start_[0];
205         x2 = end_[0];
206         y1 = start_[1];
207         y2 = end_[1];
208
209         //Keeps track of all intersections
210         //We use a list because if the outcode contains 2 '1's, we need to calculate 2 intersections
211         intersections_list = []
212         //To temporarily store an intersection point
213         intersect=[0, 0];
214
215         //find slope
216         m = (y2-y1)/(x2-x1);
217
218         //find constant 'c' by substituting one of the end points for x and y
219         c = y1 - m*x1;
220
```

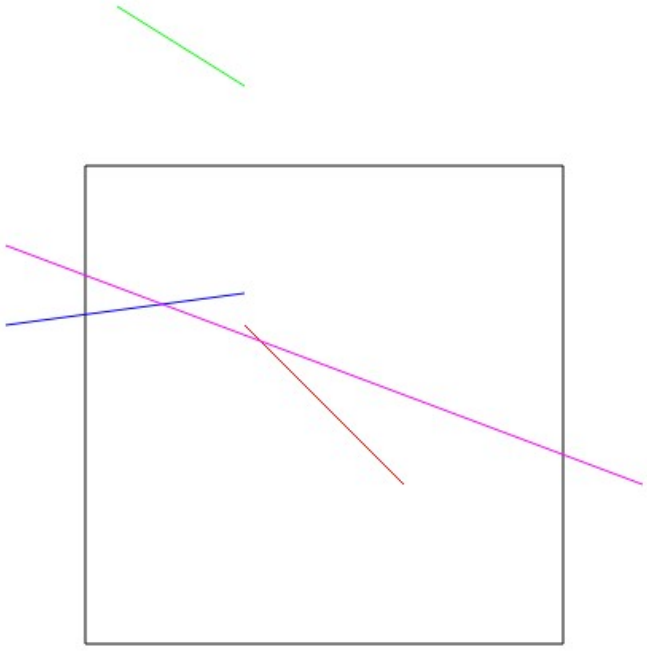
```
            if(outcode.charAt(0) == '1')
            {
                //intersection is on the line x=ymax
                //The abscissa of intersect is to be calculated as (x, ymax)
                //Ordinate of intersect is same as ymax
                intersect[0] = (ymax - c)/m;
                intersect[1] = ymax;

                intersections_list.push(intersect);
            }

            if(outcode.charAt(1) == '1')
            {
                //Intersection is on the line x=ymin

                intersect[0] = (ymin - c)/m;
                intersect[1] = ymin;

                intersections_list.push(intersect)
            }

            if(outcode.charAt(2) == '1')
            {
                //Intersection on line y=xmax
                //The abscissa of intersect is same as xmax
                //The ordinate of intersect is to be calculated
                intersect[0] = xmax;
                intersect[1] = (m * xmax + c);

                intersections_list.push(intersect);
            }

            if(outcode.charAt(3) == '1')
            {
                //Intersection is on the line y=xmin
                intersect[0] = xmin;
                intersect[1] = (m * xmin + c);

                intersections_list.push(intersect);
            }

            return intersections_list;
            }

```

```
265         //Draw a line that is completely inside the viewport
266         function draw_line1()
267         {
268             start = [300, 300];
269             end = [400, 400];
270 
271             stack.push([start, end]);
272 
273             //drawing the line
274             context.beginPath();
275             context.moveTo(start[0], start[1]);
276             context.lineTo(end[0], end[1]);
277 
278             //setting stroke color in RGB. Here R=ff, G=0, B=0 to get full red
279             context.strokeStyle = "#ff0000";
280             context.lineWdith = 1;
281             context.stroke();
282         }
283 
284         //Draw a line completely outside the viewport
285         function draw_line2()
286         {
287             start = [220, 100];
288             end = [300, 150];
289 
290             stack.push([start, end]);
291 
292             context.beginPath();
293             context.moveTo(start[0], start[1]);
294             context.lineTo(end[0], end[1]);
295 
296             context.strokeStyle = "#00ff00";
297             context.lineWidth = 1;
298             context.stroke();
299         }
300 
```

```
            //Draw a line with one end point outside and one endpoint inside the viewport
            function draw_line3()
            {
                start = [150, 300];
                end = [300, 280];

                stack.push([start, end]);

                context.beginPath();
                context.moveTo(start[0], start[1]);
                context.lineTo(end[0], end[1]);

                context.strokeStyle = "#0000ff";
                context.lineWidth = 1;
                context.stroke();
            }

            //Draw a line that cuts the viewport at 2 locations, with both endpoints outside
            function draw_line4()
            {
                start = [150, 250];
                end = [550, 400];

                stack.push([start, end]);

                context.beginPath();
                context.moveTo(start[0], start[1]);
                context.lineTo(end[0], end[1]);

                context.strokeStyle = "#ff00ff";
                context.lineWidth = 1;
                context.stroke();
            }

        </script>
    </body>
</html>

```

Output

Output



Pranjal Chimile

2) Implement Liang Barsky line Clipping Algorithm.

⇒ Algorithm

Step1 :- Set the endpoints of the line (x_1, y_1) and (x_2, y_2).

Step2 :- Calculate the value of P_1, P_2, P_3, P_4 & q_1, q_2, q_3 and q_4.

Step3 :- Now, we calculate the value of t.

$t_1 = 0$ (for initial point)

$t_2 = 1$ (for final point)

Step4 :- Now, we have to calculate the value of P_K and q_K.

If $P_K = 0$ then, (the line is parallel to the window)

If $q_K < 0$ then, {the line is completely outside the window}

Step5 :- If we have non zero value of P_K.

If $P_K < 0$ then $t_1 = \max(0, q_K/P_K)$

If $p_k > 0$ then $t_2 = \min(1, q_k/p_k)$

Now,

If $t_1 < t_2$ (if value of t_1 is changed then, the first point is outside the window.

If t_2 value is changed.

Then, the second point is outside the window.

Else,

$t_1 > t_2$ then, (the line is completely outside of the window)

Step 6: Stop

Code Screenshots

```
//LiangBarsky
function setup(){
    createCanvas(screen.width, screen.height);
}

let xWmin=400;
let yWmin=250;
let xWmax= 800;
let yWmax=500;

function draw(){
    stroke(0);
    strokeWeight(0.5);
    fill('black');
    textSize(30);
    text('Legend',45,30);
    fill('blue');
    textSize(20);
    text('RECTANGULAR WINDOW',20,60);
    fill('orange');
    text('CLIPPED LINE',20,90);

    fill('white');
    stroke(0,0,255);
    strokeWeight(2);
    rect(xWmin,yWmin,(xWmax-xWmin),(yWmax-yWmin));


    clip(500,60,600,500); //rejected
    clip(700,600,800,660);
    clip(490,300,600,350);
}

function clip(x1,y1,x2,y2){
    let a = [];
    let b = [];
    let r1 = [];
    let r2 = [];
    let flag = true;

```

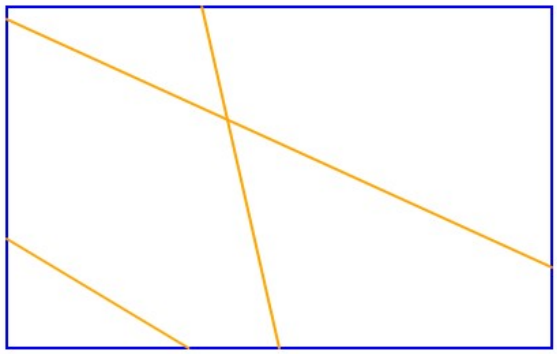
```
    dx = (x2 - x1);
    dy = (y2 - y1);
    a[0] = -dx;
    a[1] = dx;
    a[2] = -dy;
    a[3] = dy;

    b[0] = x1 - xWmin;
    b[1] = xWmax - x1;
    b[2] = y1 - yWmin;
    b[3] = yWmax - y1;

    for (i = 0; i < 4; i++) {
        if (a[i] === 0) {
            if (b[i] < 0)
                flag = false;
        }
    }

    if (flag == true) {
        for (i = 0; i < 4; i++) {
          if (a[i] < 0)
            {
                r1.push((b[i] / a[i])); //Append r1 array
            }
            else if (a[i] > 0)
            {
                r2.push((b[i] / a[i])); //Append r2 array
            }
        }
    }

    u1 = max(r1);
    u2 = min(r2);

    if(u1<u2){
        xp = x1 + u1*dx;
        yp = y1 + u1*dy;
        xq = x1 + u2*dx;
        yq = y1 + u2*dy;
        stroke(255,165,0);
        strokeWeight(2);
        line(xp,yp,xq,yq); //required Clipped Line
    }
}
```

Output

Legend

RECTANGULAR WINDOW

CLIPPED LINE



Pranjal Ghumre

Conclusion

In this lab 5, I learnt to implement the Cohen-sutherland and Liang Barsky Line Clipping algorithm using HTML with javascript & p5.js graphics library.

The source code can be found at following git-hub link:

https://github.com/pranjal667/graphics-lab.git