Batch: D-2 **Roll No.:** 16010122151

Experiment No. 05

TITLE: Write a program to demonstrate the LINE CLIPPING algorithm

AIM:

Visit Vlab and Explore it
https://cse18-iiith.vlabs.ac.in/exp/clipping-line/

Write a program to demonstrate the LINE CLIPPING algorithm

a. Cohen-Sutherland-algorithm
b. Mid-Point Subdivision Line Clipping Algorithm
c. Liang-Barsky Line Clipping Algorithm

Expected OUTCOME of Experiment:

Implement Clipping, 3D Geometric Transformations and 3D viewing.

Books/ Journals/ Websites referred:

Algorithm 1, 2 and 3

Algorithm - Cohen-Sutherland

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

Algorithm (Mid-Point Subdivision):

Step1: Calculate the position of both endpoints of the line

Step2: Perform OR operation on both of these endpoints

Step3: If the OR operation gives 0000
then
Line is guaranteed to be visible
else
Perform AND operation on both endpoints.
If AND ≠ 0000
then the line is invisible
else

AND=6000
then the line is clipped case.

Step4: For the line to be clipped. Find midpoint
Xm=(
x1+x2
)/2
Ym=(
y1+y2
)/2
Xmis midpoint of X
coordinate.Ymis
midpoint of Y
coordinate.

Step5: Check each midpoint, whether it nearest to the boundary of a

window or not. Step6: If the line is totally visible or totally rejected not

found then repeat step 1 to 5.Step7: Stop algorithm.

Algorithm (Liang-Barsky):

Step1: Set tmin=0 and tmax=1

Step2: Calculate the values tL,tR,tT and tB(tvalues).
If tmin or tmax? ignore it and go to the next edge
Otherwise classify the tvalue as entering or exiting value (using inner product toclassify)
If t is entering value set tmin=t if t is exiting value set tmax=t

Step3: If tmin< tmax? then draw a line from (x1 + dx*tmin, y1 + dy*tmin) to (x1 + dx*tmax?,y1 + dy*tmax?)

Step4: If the line crosses over the window, you will see (x1 + dx*tmin, y1 + dy*tmin) and (x1
+ dx*tmax? , y1 + dy*tmax?) are intersection between line and edge.

Implementation details:

a. Cohen-Sutherland

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const int width = 1500, height = 1000;

vector<vector<pair<float, float>>> linePoints;
vector<vector<int>> regionCodes;
vector<vector<pair<float, float>>> renderPoints;
int masks[] = {1, 2, 4, 8};
int clippingBounds[] = {200, width - 200, 200, height - 200};

void iterate() {
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, width, 0.0, height, 0.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void draw() {
    glColor3f(0.0, 1.0, 1.0);
    glLineWidth(1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(200, 200);
    glVertex2f(200, height - 200);
    glVertex2f(width - 200, height - 200);
    glVertex2f(width - 200, 200);
    glEnd();

    glColor3f(1.0, 1.0, 1.0);
    glLineWidth(2.0);
    glBegin(GL_LINES);
```

```
    for (const auto& line : linePoints) {
        glVertex2f(line[0].first, line[0].second);
        glVertex2f(line[1].first, line[1].second);
    }
    glEnd();

    glColor3f(1.0, 1.0, 0.0);
    glLineWidth(4.0);
    glBegin(GL_LINES);
    for (const auto& line : renderPoints) {
        glVertex2f(line[0].first, line[0].second);
        glVertex2f(line[1].first, line[1].second);
    }
    glEnd();
}

void showScreen() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    iterate();
    draw();
    glutSwapBuffers();
}

int calcRegionCode(int i, int j) {
    int code = 0;
    if (linePoints[i][j].first < clippingBounds[0]) code |= masks[0];
    else if (linePoints[i][j].first > clippingBounds[1]) code |=
masks[1];
    if (linePoints[i][j].second < clippingBounds[2]) code |= masks[2];
    else if (linePoints[i][j].second > clippingBounds[3]) code |=
masks[3];
    return code;
}

int main(int argc, char** argv) {
    int n;
    cout << "Enter number of lines to render: ";
    cin >> n;
    cout << "\nClipping Boundaries\nx: 200 - 1300\ny: 200 - 800\n";
    cout << "\nEnter the coordinates separated by a space\n";

    for (int i = 0; i < n; ++i) {
        float startX, startY, endX, endY;
        cout << "Enter point " << i + 1 << " start: ";
        cin >> startX >> startY;
```

```
        cout << "Enter point " << i + 1 << " end: ";
        cin >> endX >> endY;
        linePoints.push_back({{startX, startY}, {endX, endY}});
    }

    for (size_t i = 0; i < linePoints.size(); ++i) {
        vector<int> lineCode;
        for (size_t j = 0; j < linePoints[i].size(); ++j) {
            int code = calcRegionCode(i, j);
            lineCode.push_back(code);
        }
        regionCodes.push_back(lineCode);

        while (true) {
            if ((regionCodes[i][0] | regionCodes[i][1]) == 0) {
                renderPoints.push_back(linePoints[i]);
                break;
            } else if ((regionCodes[i][0] & regionCodes[i][1]) != 0) {
                break;
            } else {
                int outsideInd = regionCodes[i][0] != 0 ? 0 : 1;
                float x1 = linePoints[i][outsideInd].first, y1 =
linePoints[i][outsideInd].second;
                float x2 = linePoints[i][outsideInd ^ 1].first, y2 =
linePoints[i][outsideInd ^ 1].second;
                float newX, newY;

                if (regionCodes[i][outsideInd] & masks[0]) {
                    newX = clippingBounds[0];
                    newY = (((newX - x1) / (x2 - x1)) * (y2 - y1)) + y1;
                } else if (regionCodes[i][outsideInd] & masks[1]) {
                    newX = clippingBounds[1];
                    newY = (((newX - x1) / (x2 - x1)) * (y2 - y1)) + y1;
                } else if (regionCodes[i][outsideInd] & masks[2]) {
                    newY = clippingBounds[2];
                    newX = (((newY - y1) / (y2 - y1)) * (x2 - x1)) + x1;
                } else if (regionCodes[i][outsideInd] & masks[3]) {
                    newY = clippingBounds[3];
                    newX = (((newY - y1) / (y2 - y1)) * (x2 - x1)) + x1;
                }

                linePoints[i][outsideInd] = {newX, newY};
                lineCode.clear();
                for (size_t j = 0; j < linePoints[i].size(); ++j) {
                    int code = calcRegionCode(i, j);
                    lineCode.push_back(code);
```

```
                }
                regionCodes[i] = lineCode;
            }
        }
    }

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Cohen-Sutherland Clipping");
    glutDisplayFunc(showScreen);
    glutIdleFunc(showScreen);
    glutMainLoop();

    return 0;
}
```

b. MidPoint Subdivision

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

const int INSIDE = 0;
const int LEFT = 1;
const int RIGHT = 2;
const int BOTTOM = 4;
const int TOP = 8;

const float x_max = 400.0;
const float y_max = 350.0;
const float x_min = 100.0;
const float y_min = 200.0;

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;
```

```
}

void midpointSubdivision(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 {
            float x = 1.0, y = 1.0;
            int code_out = code1 ? code1 : code2;

            if (code_out & TOP) {
                x = (x1 + x2) / 2;
                y = (y1 + y2) / 2;
            } else if (code_out & BOTTOM) {
                x = (x1 + x2) / 2;
                y = (y1 + y2) / 2;
            } else if (code_out & RIGHT) {
                x = (x1 + x2) / 2;
                y = (y1 + y2) / 2;
            } else if (code_out & LEFT) {
                x = (x1 + x2) / 2;
                y = (y1 + y2) / 2;
            }

            if (code_out == code1) {
                x1 = x;
                y1 = y;
                code1 = computeCode(x1, y1);
            } else {
                x2 = x;
                y2 = y;
                code2 = computeCode(x2, y2);
            }
        }
    }

    if (accept) {
        std::cout << "Line accepted from " << x1 << ", " << y1 << " to "
<< x2 << ", " << y2 << std::endl;
```

```
        glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        glEnd();
    } else {
        std::cout << "Line rejected" << std::endl;
    }
}

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

void iterate() {
    glViewport(0, 0, 500, 500);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void showScreen() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    iterate();
    glColor3f(30.0 / 255.0, 33.0 / 255.0, 3.0 / 255.0);

    draw(x_min, y_min, x_max, y_min);
    draw(x_max, y_min, x_max, y_max);
    draw(x_max, y_max, x_min, y_max);
    draw(x_min, y_max, x_min, y_min);

    midpointSubdivision(50, 200, 300, 400);
    midpointSubdivision(350, 150, 450, 250);
    midpointSubdivision(300, 250, 350, 300);
    midpointSubdivision(300, 150, 250, 250);
    midpointSubdivision(50, 240, 50, 300);

    glutSwapBuffers();
}

int main(int argc, char** argv) {
```

```
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Mid-Point Subdivision Line Clipping");
    glutDisplayFunc(showScreen);
    glutIdleFunc(showScreen);
    glutMainLoop();

    return 0;
}
```

c. Liang-Barsky

```
#include <iostream>
#include <algorithm>

void liangbarsky(float xmax, float xmin, float ymax, float ymin, float
x1, float y1, float x2, float y2) {
    float p[4], q[4], t[4];
    p[0] = -(x2 - x1); p[1] = x2 - x1;
    p[2] = -(y2 - y1); p[3] = y2 - y1;
    q[0] = x1 - xmin;  q[1] = xmax - x1;
    q[2] = y1 - ymin;  q[3] = ymax - y1;

    float t1 = 0, t2 = 1;
    int count_parallel = 0;

    for (int i = 0; i < 4; i++) {
        if (p[i] > 0) {
            t[i] = q[i] / p[i];
            t2 = std::min(t2, t[i]);
        } else if (p[i] < 0) {
            t[i] = q[i] / p[i];
            t1 = std::max(t1, t[i]);
        } else if (q[i] < 0) {
            std::cout << "Line completely outside the rectangle/window"
<< std::endl;
            return;
        } else {
            count_parallel++;
            std::cout << "Line parallel to: " << count_parallel << "
Edges" << std::endl;
```

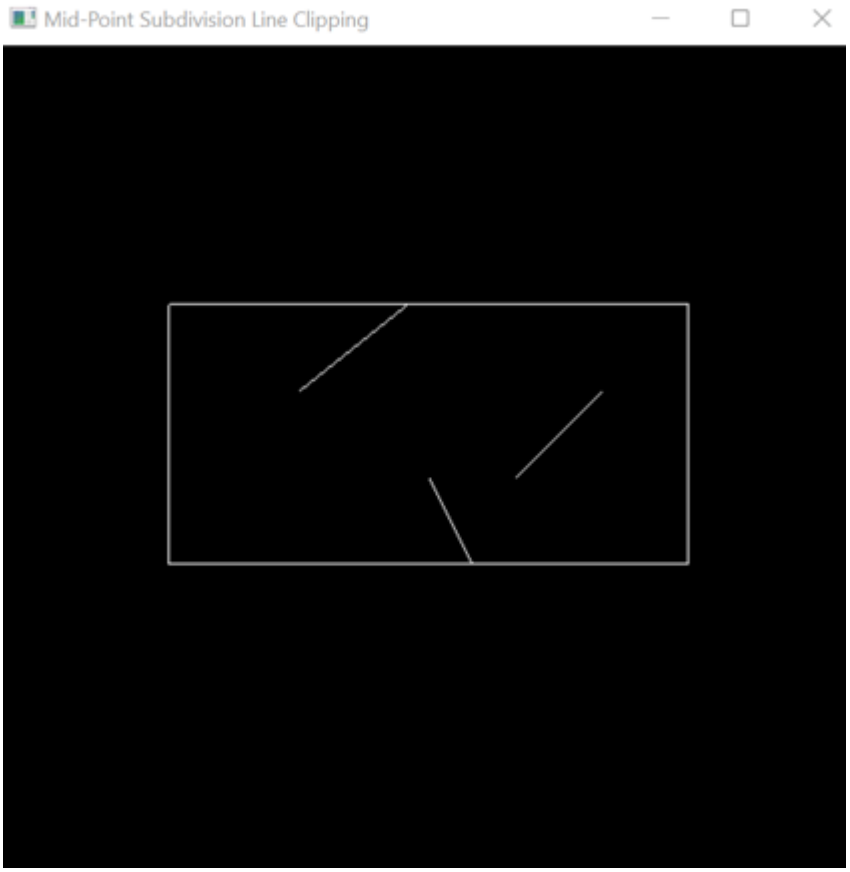
```
        }
    }

    if (t1 == 0 && t2 == 0) {
        std::cout << "Line completely inside or one of the points of the
line lies on the edge" << std::endl;
        return;
    }

    if (t1 < t2) {
        float xx1 = x1 + t1 * p[1];
        float xx2 = x1 + t2 * p[1];
        float yy1 = y1 + t1 * p[3];
        float yy2 = y1 + t2 * p[3];
        std::cout << xx1 << ", " << yy1 << ", " << xx2 << ", " << yy2 <<
std::endl;
    } else {
        std::cout << "Line outside the rectangle/window" << std::endl;
    }
}

int main() {
    liangbarsky(2, 1, 2, 1, 1.4, 1.2, 1.4, 1.3);
    return 0;
}
```

Output(s) (final edited screen shot):


```
Enter number of lines to render: 3

Clipping Boundaries
x: 200 - 1300
y: 200 - 800

Enter the coordinates separated by a space
Enter point 1 start: 100 50
Enter point 1 end: 600 400
Enter point 2 start: 100 400
Enter point 2 end: 1100 900
Enter point 3 start: 700 400
Enter point 3 end: 1200 750

Blue: clipping boundary
White: all lines
Yellow: rendered lines
```

MidPoint Subdivision

```
Line accepted from 175.00, 300.00 to 237.50, 350.00
Line accepted from 400.00, 200.00 to 400.00, 200.00
Line accepted from 300.00, 250.00 to 350.00, 300.00
Line accepted from 275.00, 200.00 to 250.00, 250.00
Line rejected
```



Liang-Bersky

```
ansal/Desktop/LiangBarsky.py"
{0: -0.0, 1: 0.0, 2: -0.10000000000000009, 3: 0.10000000000000009} {0: 0.3999999999999999, 1: 0.6000000000000001, 2: 0.19999999
999999996, 3: 0.8}
Line parallel to: 2 Edges
1.4 1.2 1.4 1.3
PS C:\Users\Kavya Bansal\Desktop>
```

Screenshots from VLab(if any):

Conclusion and discussion (Comparative):

Successfully learnt and implemented clipping algorithms like - Cohen Sutherland, Mid-Point Subdivision, Liang-Barsky line clipping algorithms.

- **Cohen-Sutherland** is simple and works well for many cases but can be less efficient with complex line clipping.
- **Mid-Point Subdivision** is conceptually simple but inefficient for practical use due to its recursive nature.
- **Liang-Barsky** is the most efficient for regular clipping windows, using fewer calculations and is better suited for performance-critical applications.

Date:

Signature of faculty in-charge

Post Lab
What is Turtle in CG, Demonstrate use of Turtle by implementing it?

Turtle in computer graphics is a simple and educational graphics module used to introduce programming concepts and basic computer graphics, especially for beginners. Turtle graphics are often used to teach programming concepts like loops, functions, and conditionals in a visual and engaging way. By controlling the turtle's movements and drawing shapes, beginners learn how programming logic translates into visual outcomes.

```
import turtle as t

t.penup()
t.setpos(-50,50)
t.pendown()
t.pensize(40)
t.pencolor("blue")

t.forward(100)
t.backward(100)
t.right(90)
t.forward(100)
t.left(90)
t.forward(100)
t.backward(100)
t.right(90)
t.forward(100)
```

Output: