Experiment No: 5

Experiment Name: **Line & Polygon Clipping using Cohen–Sutherland Algorithm**

Objective / Aim

- To implement line clipping using the Cohen–Sutherland algorithm.
- To display only the visible portion of a line inside a clipping window.
- To understand the basic concept of polygon clipping.

Theory

Clipping is the process of removing the parts of graphics objects that lie outside the viewing window.The Cohen–Sutherland algorithm is a line clipping algorithm that divides the screen into 9 regions using a rectangular clipping window. Each line endpoint is assigned a 4-bit region code based on its position relative to the window.

Region Code Bits

- Left (0001)
- Right (0010)
- Bottom (0100)
- Top (1000)

Cases

1. Trivial Accept: Both endpoints lie inside the window.
2. Trivial Reject: Both endpoints lie outside in the same region.
3. Partial Visibility: Line intersects the window and needs clipping.

Polygon clipping is achieved by clipping each edge of the polygon individually using line clipping techniques.

Requirements

- **Software:** Code::Blocks / Dev-C++ / Visual Studio
- **Libraries:** OpenGL, GLUT, Basic C++

Description of Code

1. Initialize OpenGL and set a 2D orthographic projection using `gluOrtho2D`.
2. Define the clipping window boundaries (`xmin`, `ymin`, `xmax`, `ymax`).
3. Assign region codes to each line endpoint.
4. Apply Cohen–Sutherland rules:
    - Accept the line if both endpoints are inside.
    - Reject if both endpoints share a common outside region.
    - Otherwise, calculate intersection points and clip the line.
5. Draw:
    - Clipping window
    - Original line
    - Clipped line inside the window
6. Display the final output.

Source Code with Input / Output:

```
#include <windows.h>

#include <GL/gl.h>

#include <GL/glut.h>

const int LEFT = 1;

const int RIGHT = 2;

const int BOTTOM = 4;

const int TOP = 8;

int xmin = 100, ymin = 100, xmax = 400, ymax = 400;

float lx1 = 50, ly1 = 150;

float lx2 = 450, ly2 = 350;
```

```
int computeCode(float x, float y)
{
   int code = 0;
   if (x < xmin) code |= LEFT;
   else if (x > xmax) code |= RIGHT;
   if (y < ymin) code |= BOTTOM;
   else if (y > ymax) code |= TOP;
   return code;
}
void cohenSutherland()
{
   float x, y;
   int code1 = computeCode(lx1, ly1);
   int code2 = computeCode(lx2, ly2);
   bool accept = false;

   while (true)
   {
      if (code1 == 0 && code2 == 0)
      {
```

```
    accept = true;
    break;
}
else if (code1 & code2)
{
    break;
}
else
{
    int outcode = code1 ? code1 : code2;

    if (outcode & TOP)
    {
        x = lx1 + (lx2 - lx1) * (ymax - ly1) / (ly2 - ly1);
        y = ymax;
    }
    else if (outcode & BOTTOM)
    {
        x = lx1 + (lx2 - lx1) * (ymin - ly1) / (ly2 - ly1);
        y = ymin;
```

```
}
else if (outcode & RIGHT)
{
    y = ly1 + (ly2 - ly1) * (xmax - lx1) / (lx2 - lx1);
    x = xmax;
}
else if (outcode & LEFT)
{
    y = ly1 + (ly2 - ly1) * (xmin - lx1) / (lx2 - lx1);
    x = xmin;
}

if (outcode == code1)
{
    lx1 = x;
    ly1 = y;
    code1 = computeCode(lx1, ly1);
}
else
{
```

```
            lx2 = x;
            ly2 = y;
            code2 = computeCode(lx2, ly2);
        }
    }
}

if (accept)
{
    glColor3f(1, 0, 0); // Clipped line
    glBegin(GL_LINES);
    glVertex2f(lx1, ly1);
    glVertex2f(lx2, ly2);
    glEnd();
}
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
```

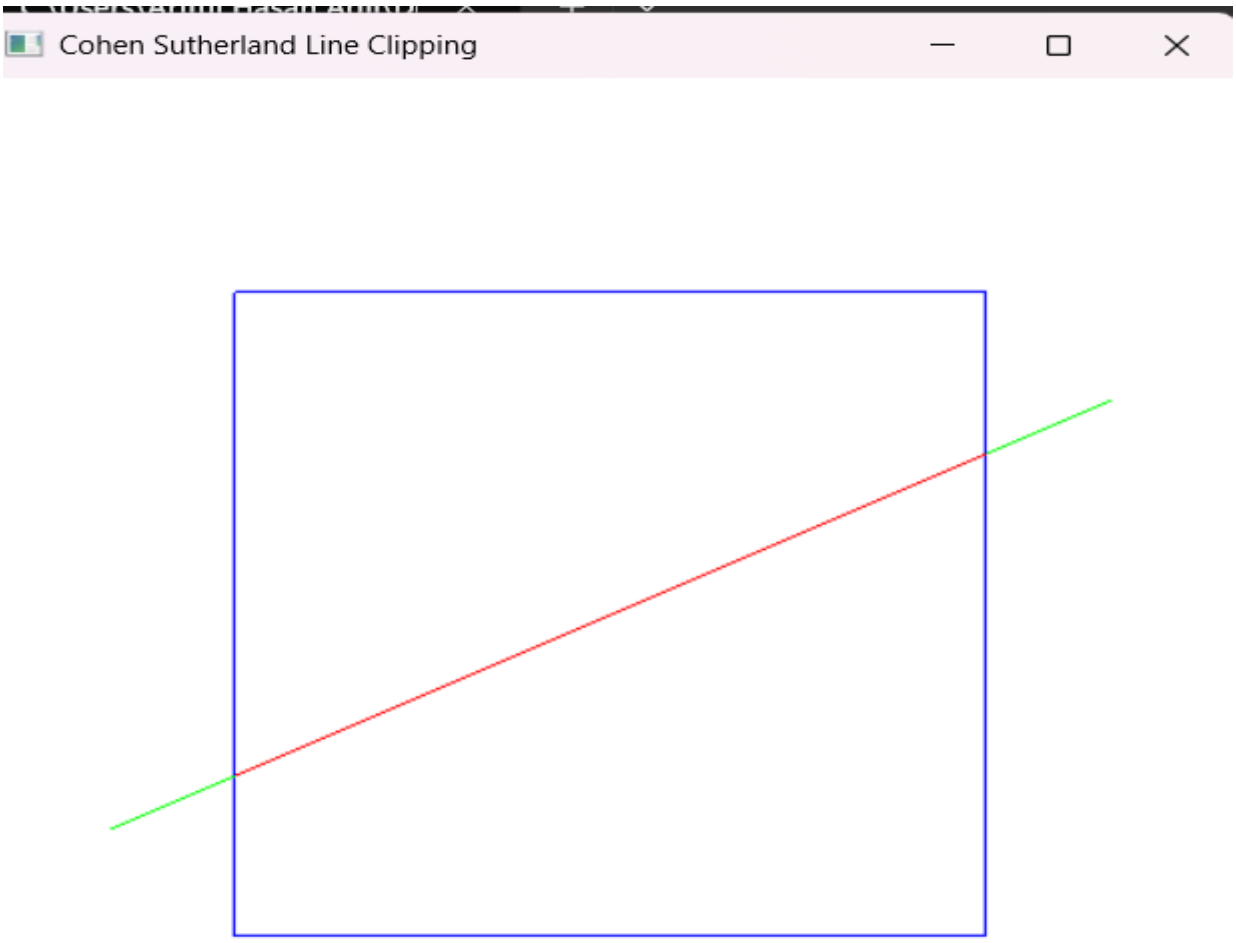
```
    // Clipping window
    glColor3f(0, 0, 1);
    glBegin(GL_LINE_LOOP);
    glVertex2i(xmin, ymin);
    glVertex2i(xmax, ymin);
    glVertex2i(xmax, ymax);
    glVertex2i(xmin, ymax);
    glEnd();

    // Original line
    glColor3f(0, 1, 0);
    glBegin(GL_LINES);
    glVertex2f(50, 150);
    glVertex2f(450, 350);
    glEnd();

    cohenSutherland();

    glFlush();
}
```

```
void init()
{
    glClearColor(1,1,1,1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0,500,0,500);
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutCreateWindow("Cohen Sutherland Line Clipping");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Conclusion

- The Cohen–Sutherland algorithm efficiently clips lines using **region codes**.
- It reduces unnecessary calculations through **trivial acceptance and rejection**.
- The algorithm is simple, fast, and widely used in **computer graphics systems**.
- Polygon clipping can be performed by applying this algorithm to each edge of the polygon.