EXPERIMENT 6 DATED 23 NOV 2020

CLIPPING

AIM: Write a menu driven program that clips a line and a polygon based on user inputs.

Option 1: Line clipping using Cohen-Sutherland line clipping algorithm.

Option 2: Polygon clipping using Sutherland-Hodgeman polygon clipping algorithm.

PROGRAM:

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *

import sys
import math


def init():
	glClearColor(0.0, 0.0, 0.0, 0.0)
	gluOrtho2D(-50.0, 50.0, -50.0, 50.0)


def glutFunct():
	glutInit(sys.argv)
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
	glutInitWindowSize(500, 500)
	glutInitWindowPosition(0, 0)
	glutCreateWindow("Clipping Algorithms")
```

```
        init()

def drawClippingWindow(xWin0, xWinMax, yWin0, yWinMax):
        edges = [
        [0, 1],
        [1, 2],
        [2, 3],
        [3, 0]
        ]
        points = [
        [xWin0, yWin0],
        [xWinMax, yWin0],
        [xWinMax, yWinMax],
        [xWin0, yWinMax]
        ]
        rgb = (1.0, 1.0, 1.0)

        drawLines(edges, points, rgb)


def getClippingWindowSize():
        print("Enter the clipping window size")
        xWin0 = float(input("Enter the minimum window value of x : "))
        xWinMax = float(input("Enter the maximum window value of x : "))
        yWin0 = float(input("Enter the minimum window value of y : "))
        yWinMax = float(input("Enter the maximum window value of y : "))
        return xWin0, xWinMax, yWin0, yWinMax

def drawLines(edges, points, rgb):
        glColor3f(rgb[0], rgb[1], rgb[2])
        for e in edges:
        for v in e:
        glVertex2fv(points[v])



def getLine():
        x1 = float(input("Enter the initial x coordinate value : "))
        x2 = float(input('Enter the final x coordinate value : '))
        y1 = float(input("Enter the initial y coordinate value : "))
```

```
    y2 = float(input("Enter the final y coordinate value : "))

    return x1, x2, y1, y2


def drawGivenLine(x1, x2, y1, y2):
    edges = [
    [0, 1]
    ]
    points = [
    [x1, y1],
    [x2, y2]
    ]
    rgb = [0.0, 0.0, 1.0]

    drawLines(edges, points, rgb)

INSIDE = 0
LEFT = 1
RIGHT = 2
DOWN = 4
TOP = 8


def computeCode(x, y, xWin0, xWinMax, yWin0, yWinMax):
    code = INSIDE

    if x < xWin0:
    code |= LEFT

    elif x > xWinMax:
    code |= RIGHT

    if y < yWin0:
    code |= DOWN

    elif y > yWinMax:
    code |= TOP

    return code
```

```
def cohenSutherland(x1, x2, y1, y2, xWin0, xWinMax, yWin0, yWinMax):
	drawGivenLine(x1, x2, y1, y2)

	code1 = computeCode(x1, y1, xWin0, xWinMax, yWin0, yWinMax)
	code2 = computeCode(x2, y2, xWin0, xWinMax, yWin0, yWinMax)
	accept = False

	while True:
	if code1 == 0 and code2 == 0:
	accept = True
	break

	elif code1 & code2 != 0:
	break

	else:
	x = float()
	y = float()

	if code1 != 0:
		code_out = code1
	else:
		code_out = code2

	if code_out & TOP:
		y = yWinMax
		x = x1 + (x2 - x1) * (y - y1) / (y2 - y1)

	elif code_out & DOWN:
		y = yWin0
		x = x1 + (x2 - x1) * (y - y1) / (y2 - y1)

	elif code_out & LEFT:
		x = xWin0
		y = y1 + (x - x1) * (y2 - y1) / (x2 - x1)

	elif code_out & RIGHT:
		x = xWinMax
```

```
            y = y1 + (x - x1) * (y2 - y1) / (x2 - x1)

        if code_out == code1:
            x1, y1 = x, y
            code1 = computeCode(x, y, xWin0, xWinMax, yWin0, yWinMax)

        else:
            x2, y2 = x, y
            code2 = computeCode(x, y, xWin0, xWinMax, yWin0, yWinMax)

        if accept:
        edges = [[0, 1]]
        points = [
        [x1, y1],
        [x2, y2]
        ]
        rgb = [1.0, 0.0, 0.0]
        drawLines(edges, points, rgb)
        drawClippingWindow(xWin0, xWinMax, yWin0, yWinMax)

        else:
        print("The given line cannot be clipped!")


def clipLine(x1, x2, y1, y2, xWin0, xWinMax, yWin0, yWinMax):

        glClear(GL_COLOR_BUFFER_BIT)
        glPointSize(5.0)
        glBegin(GL_LINES)

        cohenSutherland(x1, x2, y1, y2, xWin0, xWinMax, yWin0, yWinMax)

        glEnd()
        glFlush()


def drawPolygons(edges, points, rgb):
        glColor3f(rgb[0], rgb[1], rgb[2])
        for e in edges:
        for v in e:
```

```
        glVertex2fv(points[v])



def getPolygon():
        n = int(input("Enter the number of edges : "))
        edges = list(list())
        points = list(list())

        for i in range(n):
        edges += [[i, (i+1) % n]]

        for i in range(n):
        x = float(input("Enter the x-coordinate value of point " + str(i+1) + ": "))
        y = float(input("Enter the y-coordinate value of point " + str(i+1) + ": "))
        points += [[x, y]]


        return edges, points


def drawGivenPolygon(edges, points):
        rgb = [1.0, 0.0, 0.0]
        drawPolygons(edges, points, rgb)


def SHC(points, xWin0, xWinMax, yWin0, yWinMax):
        global x_new, y_new, x_newr, y_newr, x_newb, y_newb, x_newt, y_newt
        x_new = []
        y_new = []
        x_newr = []
        y_newr = []
        x_newb = []
        y_newb = []
        x_newt = []
        y_newt = []
        n = len(points)
        for i in range(n-1):
        clipl(points[i][0], points[i][1],
        points[i+1][0], points[i+1][1], xWin0)
```

```
        clipl(points[n-1][0], points[n-1][1], points[0][0], points[0][1], xWin0)


        n = len(x_new)
        for i in range(n-1):
        clipr(x_new[i], y_new[i], x_new[i+1], y_new[i+1], xWinMax)
        clipr(x_new[n-1], y_new[n-1], x_new[0], y_new[0], xWinMax)


        n = len(x_newr)
        for i in range(n-1):
        clipb(x_newr[i], y_newr[i], x_newr[i+1], y_newr[i+1], yWin0)
        clipb(x_newr[n-1], y_newr[n-1], x_newr[0], y_newr[0], yWin0)


        n = len(x_newb)
        for i in range(n-1):
        clipt(x_newb[i], y_newb[i], x_newb[i+1], y_newb[i+1], yWinMax)
        clipt(x_newb[n-1], y_newb[n-1], x_newb[0], y_newb[0], yWinMax)


        n = len(x_newt)
        newEdges = list(list())
        for i in range(n):
        newEdges += [[i, (i+1) % n]]

        newPoints = list(list())
        for i in range(len(x_newt)):
        newPoints += [[x_newt[i], y_newt[i]]]

        rgb = [0.0, 0.0, 1.0]

        drawPolygons(newEdges, newPoints, rgb)


def clipl(x1, y1, x2, y2, xWin0):

        if x2 - x1 != 0:
        m = (y2 - y1)/(x2 - x1)
        else:
```

```
        m = 4000
        if x1 >= xWin0 and x2 >= xWin0:

        x_new.append(x2)
        y_new.append(y2)

        elif x1 < xWin0 and x2 >= xWin0:

        x_new.append(xWin0)
        y_new.append(y1 + m*(xWin0 - x1))
        x_new.append(x2)
        y_new.append(y2)

        elif x1 >= xWin0 and x2 < xWin0:

        x_new.append(xWin0)
        y_new.append(y1 + m*(xWin0 - x1))


def clipr(x1, y1, x2, y2, xWinMax):
        if x2 - x1 != 0:
        m = (y2 - y1)/(x2 - x1)
        else:
        m = 4000

        if x1 <= xWinMax and x2 <= xWinMax:

        x_newr.append(x2)
        y_newr.append(y2)

        elif x1 > xWinMax and x2 <= xWinMax:

        x_newr.append(xWinMax)
        y_newr.append(y1 + m*(xWinMax - x1))
        x_newr.append(x2)
        y_newr.append(y2)

        elif x1 <= xWinMax and x2 > xWinMax:
        x_newr.append(xWinMax)
        y_newr.append(y1 + m*(xWinMax - x1))
```

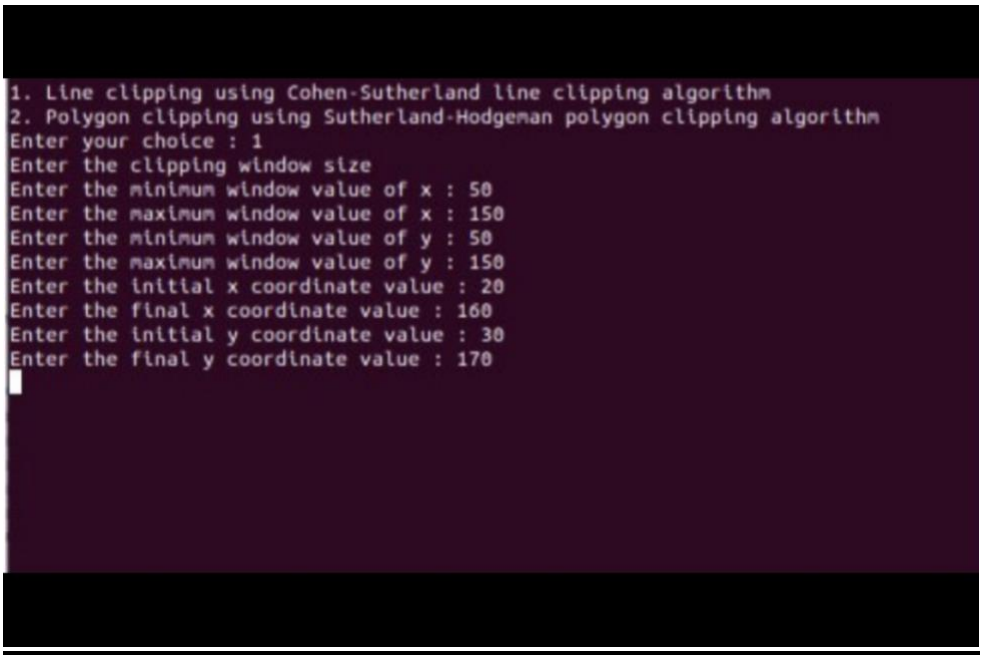
```
def clipt(x1, y1, x2, y2, yWinMax):
        if (y2-y1) != 0:
        m = (x2-x1)/(y2-y1)
        else:
        m = 4000
        if y1 <= yWinMax and y2 <= yWinMax:
        x_newt.append(x2)
        y_newt.append(y2)
        elif y1 > yWinMax and y2 <= yWinMax:
        x_newt.append(x1+m*(yWinMax-y1))
        y_newt.append(yWinMax)
        x_newt.append(x2)
        y_newt.append(y2)

        elif y1 <= yWinMax and y2 > yWinMax:
        x_newt.append(x1+m*(yWinMax - y1))
        y_newt.append(yWinMax)


def clipb(x1, y1, x2, y2, yWin0):
        if (y2-y1) != 0:
        m = (x2-x1)/(y2-y1)
        else:
        m = 4000
        if y1 >= yWin0 and y2 >= yWin0:
        x_newb.append(x2)
        y_newb.append(y2)
        elif y1 < yWin0 and y2 >= yWin0:
        x_newb.append(x1+m*(yWin0-y1))
        y_newb.append(yWin0)
        x_newb.append(x2)
        y_newb.append(y2)

        elif y1 >= yWin0 and y2 < yWin0:
        x_newb.append(x1+m*(yWin0 - y1))
        y_newb.append(yWin0)
```

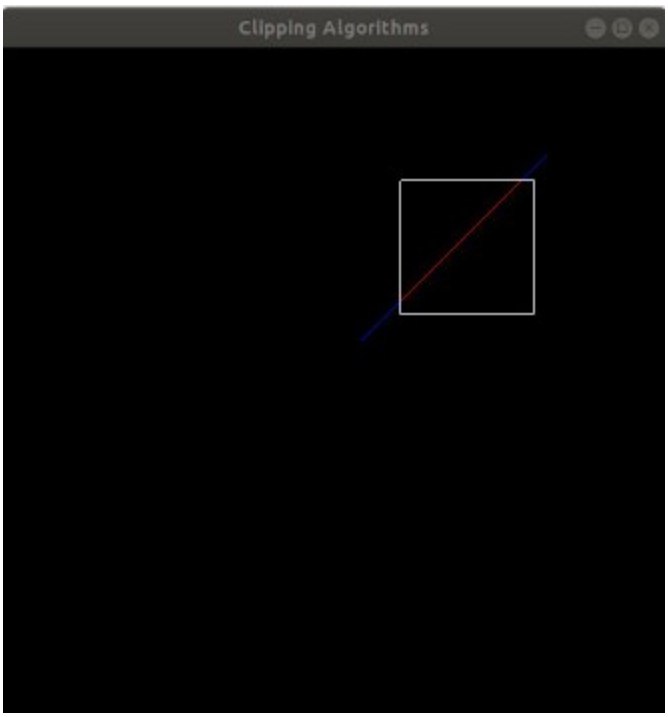
```
def clipPolygon(edges, points, xWin0, xWinMax, yWin0, yWinMax):

        glClear(GL_COLOR_BUFFER_BIT)
        glPointSize(5.0)
        glBegin(GL_LINES)

        drawClippingWindow(xWin0, xWinMax, yWin0, yWinMax)

        drawGivenPolygon(edges, points)

        SHC(points, xWin0, xWinMax, yWin0, yWinMax)

        glEnd()
        glFlush()



def menu():
        print("\t\tMENU")
        print("1. Line clipping using Cohen-Sutherland line clipping algorithm")
        print("2. Polygon clipping using Sutherland-Hodgeman polygon clipping algorithm")

def main():
        menu()
        ch = int(input("Enter your choice : "))

        if ch != 1 and ch != 2:
        print("Invalid choice! \nExiting...")
        exit()

        xWin0, xWinMax, yWin0, yWinMax = getClippingWindowSize()

        if ch == 1:
        x1, x2, y1, y2 = getLine()

        glutFunct()
        glutDisplayFunc(lambda: clipLine(
        x1, x2, y1, y2, xWin0, xWinMax, yWin0, yWinMax))
```

```
    if ch == 2:
    edges, points = getPolygon()

    glutFunct()
    glutDisplayFunc(lambda: clipPolygon(
    edges, points, xWin0, xWinMax, yWin0, yWinMax))

    glutMainLoop()


main()
```

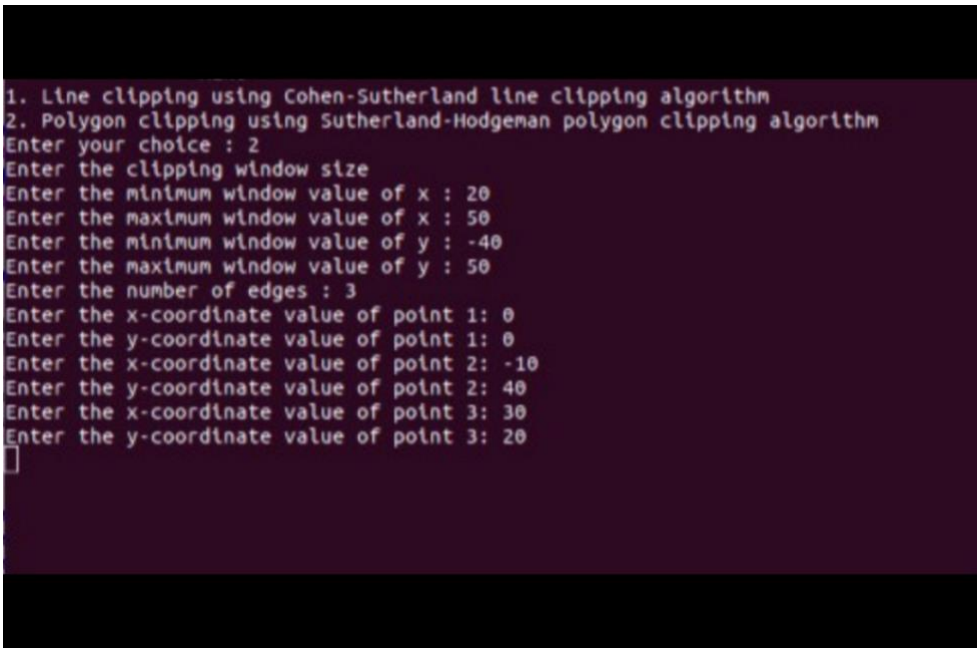
INPUT/OUTPUT:

CASE 1: Line clipping using Cohen-Sutherland line clipping algorithm

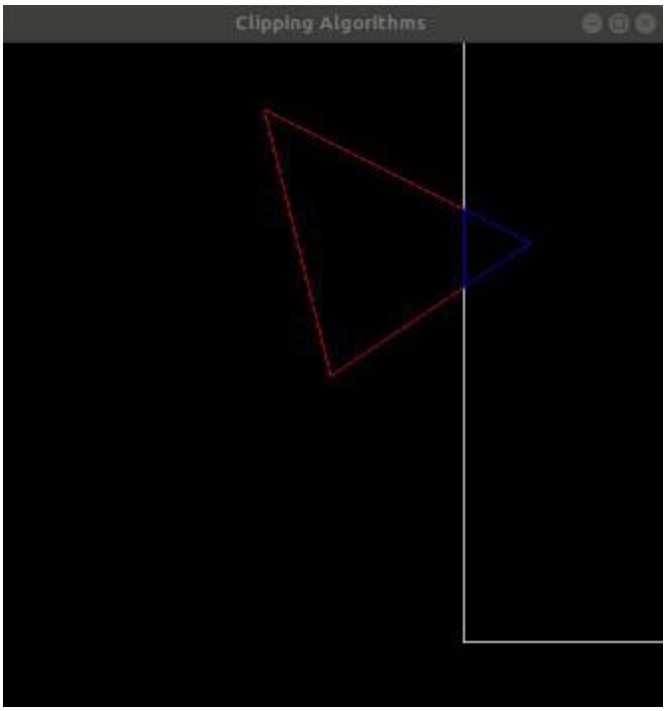




CASE 2: Polygon clipping using Sutherland-Hodge polygon clipping algorithm



```
1. Line clipping using Cohen-Sutherland line clipping algorithm
2. Polygon clipping using Sutherland-Hodgeman polygon clipping algorithm
Enter your choice : 2
Enter the clipping window size
Enter the minimum window value of x : 20
Enter the maximum window value of x : 50
Enter the minimum window value of y : -40
Enter the maximum window value of y : 50
Enter the number of edges : 3
Enter the x-coordinate value of point 1: 0
Enter the y-coordinate value of point 1: 0
Enter the x-coordinate value of point 2: -10
Enter the y-coordinate value of point 2: 40
Enter the x-coordinate value of point 3: 30
Enter the y-coordinate value of point 3: 20
```



RESULT: Successfully made a menu driven program that clips a line and a polygon based on user inputs.