UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA,INFORMÁTICA Y MECÁNICA

INGENIERÍA INFORMÁTICA Y DE SISTEMAS

Guía de Laboratorio 3 - Line Clipping

Alumno:
Ian Logan Will Quispe Ventura
211359

Docente:
Hector Eduardo Ugarte Rojas

Curso:
Computación Gráfica

Cusco - Perú
2023 - II

Algoritmo de Cohen-Sutherland en python

```cpp
#include <iostream>
#include <thread>
#include <vector>

// Función para sumar los elementos de un rango del arreglo
void sumRange(const std::vector<int>& arr, int start, int end, int& result) {
    int partialSum = 0;
    for (int i = start; i < end; ++i) {
        partialSum += arr[i];
    }
    result = partialSum;
}

int main() {
    // Arreglos de ejemplo
    std::vector<int> arr1 = {1, 2, 3, 4, 5};
    std::vector<int> arr2 = {6, 7, 8, 9, 10};

    // Número de elementos en cada arreglo
    int N = arr1.size();

    // Dividir los arreglos en dos partes iguales
    int mid = N / 2;

    // Variables para almacenar los resultados parciales
    int result1, result2;

    // Crear dos hilos para sumar las partes de los arreglos de forma paralela
    std::thread thread1(sumRange, std::ref(arr1), 0, mid, std::ref(result1));
    std::thread thread2(sumRange, std::ref(arr2), 0, mid, std::ref(result2));

    // Esperar a que ambos hilos terminen
    thread1.join();
    thread2.join();

    // Sumar los resultados parciales para obtener la suma total
```

```
    if y0 > y1:
        dy = -1
    else:
        dy = 1
    delta_x = x1 - x0
    delta_y = abs(y1 - y0)
    y = y0
    error = 0

    for x in range(x0, x1 + 1):
        if steep:
            Plot(y, x)
        else:
            Plot(x, y)
        error = error + delta_y
        if 2 * error >= delta_x:
            y = y + dy
            error = error - delta_x

def dibujaRectangulo(xmin, ymin, xmax, ymax):
    dibujaLinea(xmin, ymin, xmin, ymax)
    dibujaLinea(xmin, ymax, xmax, ymax)
    dibujaLinea(xmax, ymax, xmax, ymin)
    dibujaLinea(xmax, ymin, xmin, ymin)

def calculo_outcode(x, y, xmin, ymin, xmax, ymax):
    oc = 0
    if y > ymax:
        oc |= TOP
    elif y < ymin:
        oc |= BOTTOM
    if x > xmax:
        oc |= RIGHT
    elif x < xmin:
        oc |= LEFT
    return oc
```

```
def cohen_sutherland(x0, y0, x1, y1, xmin, ymin,
  xmax, ymax):
   global k
   in_, done = False, False
   outcode0 = calculo_outcode(x0, y0, xmin, ymin,
     xmax, ymax)
   outcode1 = calculo_outcode(x1, y1, xmin, ymin,
     xmax, ymax)
   while not done:
       if (outcode0 | outcode1) == 0:
           done = True
           in_ = True
       elif (outcode0 & outcode1) != 0:
           done = True
       else:
           m = (y1 - y0) / (x1 - x0)
           x, y = 0, 0
           outcode = 0

           if outcode0 != 0:
               outcode = outcode0
           else:
               outcode = outcode1
           if (outcode0 & TOP) != 0:
               x = x0 + (ymax - y0) / m
               y = ymax
           elif (outcode & BOTTOM) != 0:
               x = x0 + (ymin - y0) / m
               y = ymin
           elif (outcode & RIGHT) != 0:
               y = y0 + m * (xmax - x0)
               x = xmax
           else:
               y = y0 + m * (xmin - x0)
               x = xmin
           if outcode == outcode0:
               x0, y0 = x, y
               outcode0 = calculo_outcode(x0, y0,
                 xmin, ymin, xmax, ymax)
```

```
            else:
                x1, y1 = x, y
                outcode1 = calculo_outcode(x1, y1,
                  xmin, ymin, xmax, ymax)

    print(f"{x0},{y0}---{x1},{y1}")
    if in_:
        lnclip[k][0], lnclip[k][1], lnclip[k][2],
          lnclip[k][3] = x0, y0, x1, y1
        k += 1
def display2():
    glClear(GL_COLOR_BUFFER_BIT)
    dibujaRectangulo(xmin, ymin, xmax, ymax)
    for i in range(k):
        dibujaLinea(lnclip[i][0], lnclip[i][1],
          lnclip[i][2], lnclip[i][3])
    glFlush()

def display1():
    glClear(GL_COLOR_BUFFER_BIT)
    dibujaRectangulo(xmin, ymin, xmax, ymax)
    for i in range(n):
        dibujaLinea(ln[i][0], ln[i][1], ln[i][2], ln
          [i][3])
    glFlush()

def myinit():
    glClearColor(1.0, 1.0, 1.0, 1.0)
    glColor3f(1.0, 0.0, 0.0)
    glPointSize(1.0)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluOrtho2D(0.0, 499.0, 0.0, 499.0)
```

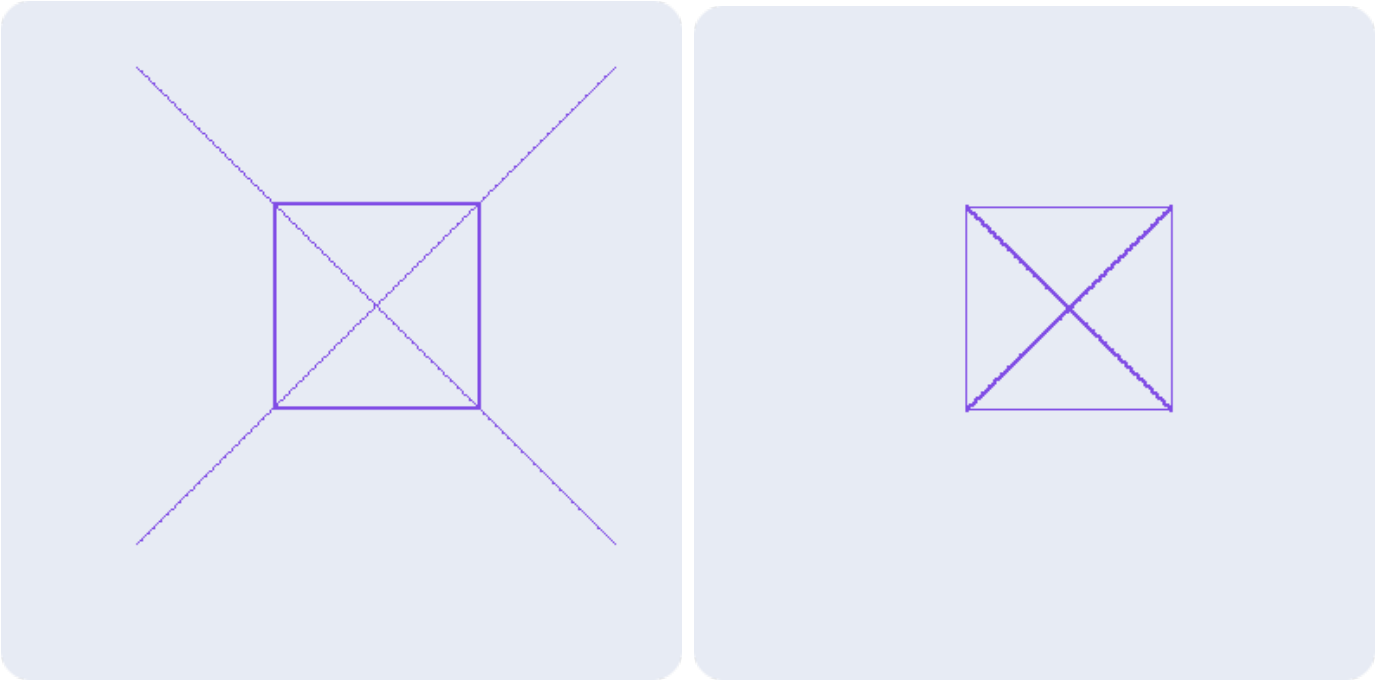
```
def main():
    global n, xmin, ymin, xmax, ymax
    k = 0
    n = int(input("Ingrese el número de lineas a
      recortar: "))
    print("Ingrese las coordenadas x - y de los
      puntos extremos de las lineas:")
    for i in range(n):
        print(f"INGRESE PUNTOS DE LA LINEA {i + 1}:"
          )
        for j in range(4):
            coord_type = "Coordenada X:" if j % 2 ==
               0 else "Coordenada Y:"
            ln[i][j] = int(input(coord_type))
    print("INGRESE LAS COORDENADAS X-Y DEL
      RECTANGULO DE RECORTE:")
    xmin = int(input("Ingrese Xmin: "))
    ymin = int(input("Ingrese Ymin: "))
    xmax = int(input("Ingrese Xmax: "))
    ymax = int(input("Ingrese Ymax: "))
    for i in range(n):
        cohen_sutherland(ln[i][0], ln[i][1], ln[i
          ][2], ln[i][3], xmin, ymin, xmax, ymax)
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(500, 500)
    glutInitWindowPosition(0, 0)
    glutCreateWindow("LINEAS ANTES DEL RECORTE")
    glutDisplayFunc(display1)
    myinit()

    glutInitWindowSize(500, 500)
    glutInitWindowPosition(550, 0)
    glutCreateWindow("LINEAS DESPUES DEL RECORTE")
    glutDisplayFunc(display2)
    myinit()
    glutMainLoop()
if __name__ == "__main__":
    main()
```

Obtención de datos

```
Terminal
Ingrese el número de lineas a recortar: 2
Ingrese las coordenadas x - y de los puntos extremos de
 las lineas:
INGRESE PUNTOS DE LA LINEA 1:
Coordenada X:100
Coordenada Y:100
Coordenada X:450
Coordenada Y:450
INGRESE PUNTOS DE LA LINEA 2:
Coordenada X:100
Coordenada Y:450
Coordenada X:450
Coordenada Y:100
INGRESE LAS COORDENADAS X-Y DEL RECTANGULO DE RECORTE:
Ingrese Xmin: 200
Ingrese Ymin: 200
Ingrese Xmax: 350
Ingrese Ymax: 350
200.0,200---350,350.0
200.0,350---350.0,200
```

Gráfico generado antes y despues del recorte



Algoritmo paramétrico estudiado en clases

```
import sys
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import numpy as np

Xmin, Ymin, Xmax, Ymax = 90, 90, 450,360

def comparacion(x0, y0, x1, y1, Xmin, Ymin, Xmax,
  Ymax):
    u = 0
    if Xmin <= x0 <= Xmax:
        u = u + 1
    if Ymin <= y0 <= Ymax:
        u = u + 1
    if Xmin <= x1 <= Xmax:
        u = u + 1
    if Ymin <= y1 <= Ymax:
        u = u + 1

    if u == 4:
        line_clipping1(x0, y0, x1, y1, Xmin, Ymin,
          Xmax, Ymax)
    elif u == 3:
        line_clipping2(x0, y0, x1, y1, Xmin, Ymin,
          Xmax, Ymax)
    elif u <= 2:
        line_clipping3(x0, y0, x1, y1, Xmin, Ymin,
          Xmax, Ymax)

def line_clipping1(x0, y0, x1, y1, Xmin, Ymin, Xmax
  , Ymax):
    dibujaLinea(x0, y0, x1, y1)
```

```
def line_clipping2(x0, y0, x1, y1, Xmin, Ymin, Xmax,
  Ymax):
    # Resolviendo el sistema de ecuaciones
    A = np.array([[(x1-x0), -(Xmax - Xmin)], [(y1-y0)
      , -(Ymin - Ymin)]])
    b = np.array([(Xmin-x0), (Ymin-y0)])
    solucion = np.linalg.solve(A, b)
    # Calcular los puntos de corte
    x = x0 + solucion[0] * (x1-x0)
    y = y0 + solucion[0] * (y1-y0)
    dibujaLinea(x0, y0, x, y)

puntos = []
def line_clipping3(x0, y0, x1, y1, Xmin, Ymin, Xmax,
  Ymax):
    ##########################################
    #    Intersecta con el borde inferior    #
    ##########################################
    # Resolviendo el sistema de ecuaciones
    A = np.array([[(x1-x0), -(Xmax - Xmin)], [(y1-y0)
      , -(Ymin - Ymin)]])
    b = np.array([(Xmin-x0), (Ymin-y0)])
    solucion = np.linalg.solve(A, b)

    # Si se cumple esto entonces la linea intersecta
      por este borde
    if 0 <= solucion[0] <= 1:
        X = x0 + solucion[0] * (x1-x0)
        Y = y0 + solucion[0] * (y1-y0)
        puntos.append(X)
        puntos.append(Y)

    ##########################################
    #    Intersecta con el borde superior    #
    ##########################################
    # Resolviendo el sistema de ecuaciones
    A = np.array([[(x1-x0), -(Xmax - Xmin)], [(y1-y0)
      , -(Ymax - Ymax)]])
    b = np.array([(Xmin-x0), (Ymax-y0)])
    solucion = np.linalg.solve(A, b)
```

```
# Si se cumple esto entonces la linea intersecta
  por este borde
if 0 <= solucion[0] <= 1:
    X = x0 + solucion[0] * (x1-x0)
    Y = y0 + solucion[0] * (y1-y0)
    puntos.append(X)
    puntos.append(Y)

##############################################
#    Intersecta con el borde izquierdo    #
##############################################
# Resolviendo el sistema de ecuaciones
A = np.array([[(x1-x0), -(Xmin - Xmin)], [(y1-y0)
  , -(Ymax - Ymin)]])
b = np.array([(Xmin-x0), (Ymin-y0)])
solucion = np.linalg.solve(A, b)

# Si se cumple esto entonces la linea intersecta
  por este borde
if 0 <= solucion[0] <= 1:
    X = x0 + solucion[0] * (x1-x0)
    Y = y0 + solucion[0] * (y1-y0)
    puntos.append(X)
    puntos.append(Y)

##############################################
#    Intersecta con el borde derecho    #
##############################################
# Resolviendo el sistema de ecuaciones
A = np.array([[(x1-x0), -(Xmax - Xmax)], [(y1-y0)
  , -(Ymax - Ymin)]])
b = np.array([(Xmax-x0), (Ymin-y0)])
solucion = np.linalg.solve(A, b)

# Si se cumple esto entonces la linea intersecta
  por este borde
if 0 <= solucion[0] <= 1:
    X = x0 + solucion[0] * (x1-x0)
    Y = y0 + solucion[0] * (y1-y0)
    puntos.append(X)
    puntos.append(Y)
```

```
    # Dibujar la linea recortada con los puntos de
      intersección
    dibujaLinea(puntos[0], puntos[1], puntos[2],
      puntos[3])
    glFlush()
# Necesario para dibujar las lineas
def Plot(ix, iy):
   ix = int(ix)
   iy = int(iy)
   glBegin(GL_POINTS)
   glVertex2i(ix, iy)
   glEnd()
def swap(x, y):
   return y, x
# Algortimo de Bresenham
def dibujaLinea(x0, y0, x1, y1):
   x0 = int(x0)
   y0 = int(y0)
   x1 = int(x1)
   y1 = int(y1)
   dy, x, y, error = 0, 0, 0, 0
   delta_x, delta_y = 0, 0
   steep = abs(y1 - y0) > abs(x1 - x0)
   if steep:
       x0, y0 = swap(x0, y0)
       x1, y1 = swap(x1, y1)
   if x0 > x1:
       x0, x1 = swap(x0, x1)
       y0, y1 = swap(y0, y1)
   if y0 > y1:
       dy = -1
   else:
       dy = 1
   delta_x = x1 - x0
   delta_y = abs(y1 - y0)
   y = y0
   error = 0
   for x in range(x0, x1 + 1):
       if steep:
           Plot(y, x)
```

```
        else:
            Plot(x, y)
        error = error + delta_y
        if 2 * error >= delta_x:
            y = y + dy
            error = error - delta_x

def dibujaRectangulo(xmin, ymin, xmax, ymax):
    dibujaLinea(xmin, ymin, xmin, ymax)
    dibujaLinea(xmin, ymax, xmax, ymax)
    dibujaLinea(xmax, ymax, xmax, ymin)
    dibujaLinea(xmax, ymin, xmin, ymin)

# Primera venta
def display1():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(0.5, 0.3, 0.9)
    glPointSize(2.0)
    dibujaRectangulo(Xmin, Ymin, Xmax, Ymax)
    glPointSize(1.0)
    glColor3f(0.8431372549019608,
      0.3686274509803922, 1.0)
    dibujaLinea(180, 120, 360, 300)
    dibujaLinea(120, 120, 210, 60)
    dibujaLinea(30, 180, 300, 450)
    glFlush()

def display2():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(0.5, 0.3, 0.9)
    glPointSize(1.0)
    dibujaRectangulo(Xmin, Ymin, Xmax, Ymax)
    glPointSize(2.0)
    glColor3f(0.8156862745098039,
      0.26666666666666666, 1.0)
    comparacion(180, 120, 360, 300, 90, 90, 450,360)
    comparacion(120, 120, 210, 60, 90, 90, 450,360)
    comparacion(30, 180, 300, 450, 90, 90, 450,360)
    glFlush()
```

```
def myinit():
    glClearColor(0.9058823529411765,
      0.9215686274509803, 0.9568627450980393, 1.0)
    glColor3f(1.0, 0.0, 0.0)
    glPointSize(1.0)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluOrtho2D(0.0, 499.0, 0.0, 499.0)

glutInit(sys.argv)
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
# Primera ventana
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)
glutCreateWindow("Lineas antes del recorte")
glutDisplayFunc(display1)
myinit()

glutInitWindowSize(500, 500)
glutInitWindowPosition(550, 0)
# Segunda ventana
glutCreateWindow("Lineas despues del recorte")
glutDisplayFunc(display2)
myinit()
glutMainLoop()
```

Gráfico generado antes y despues del recorte