UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERIAS
DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

MATERIA:
SIMULACION POR COMPUTADORA

MAESTRO:
DAVID ALEJANDRO GOMEZ ANAYA

TITULO DE INVESTIGACIÓN:
ACTIVIDAD POSTERIOR 2: LÍNEA RECTA

FECHA ENTREGA:
DOMINGO 24 DE FEBRERO 2019

ALUMNO:
FELIPE DE JESUS RUIZ GARCIA

CODIGO:
214522077

CARRERA: INGENIERIA INFORMATICA (INNI)

SECCION: D03

CALIFICACIÓN Y OBSERVACIONES:

Indice:

Objetivo. 2
Marco teórico. 2
Desarrollo.....2
Pruebas y resultados.....4
Conclusiones.....6
Apéndice(s).....6

Objetivo

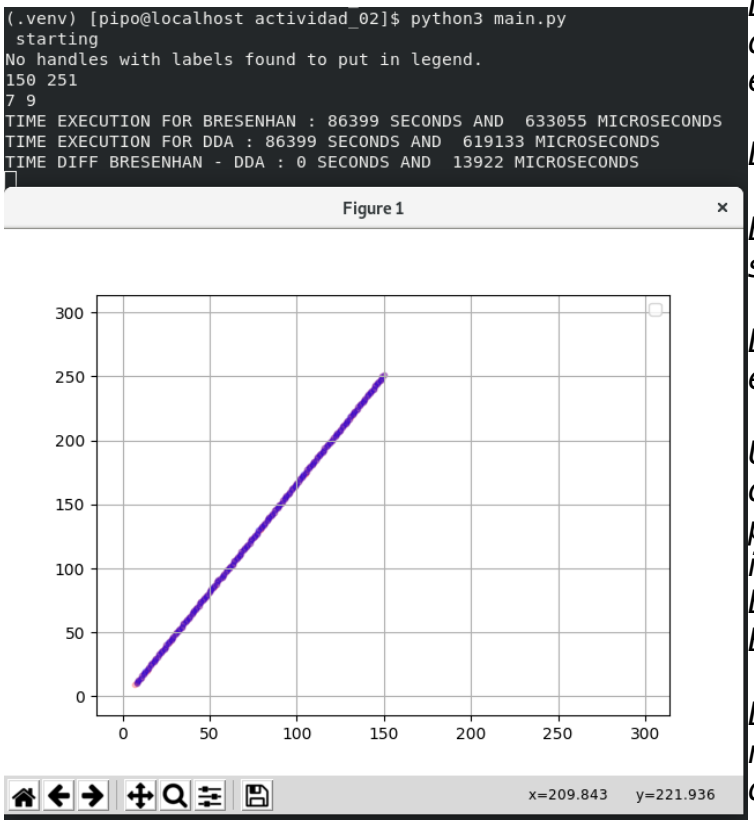
Desarrollar un programa capaz de trazar lineas rectas mediante dos puntos con los algoritmos de lineas DDA y Bresenham, mostrando el tiempo de ejecucion por cada algoritmo y la diferencia entre estos, en un sistema de coordenadas 2D.

Marco Teorico

Se realizo una investigacion de librerias, herramientas y plataformas capaces de manipular en tiempo real sistemas 2D, capaz de detectar y manipular la interaccion eventos como pulsacion de teclas y pulsacion de mouse/touchpad.
Se opto por usar matplotlib, una biblioteca de trazado 2D de Python.
Una vez definida la libreria principal y el lenguaje de programacion, se investigo acerca de los algoritmos DDA y Bresenham su implementacion y la manera de tomar el tiempo en eventos en el lenguaje seleccionado.

Desarrollo

Se desarrollo una solucion en el lenguaje python en 191 lineas de codigo que inicia mostrando una interfaz grafica :



Las coordenadas de la ubicacion del puntero dentro del sistema 2D se muestran en la esquina inferior derecha como X y Y.

La interfaz espera al evento click derecho.

El primer click derecho en el plano cartesiano se define como el punto inicial para la recta.

Las coordenadas del punto inicial se muestran en la terminal.

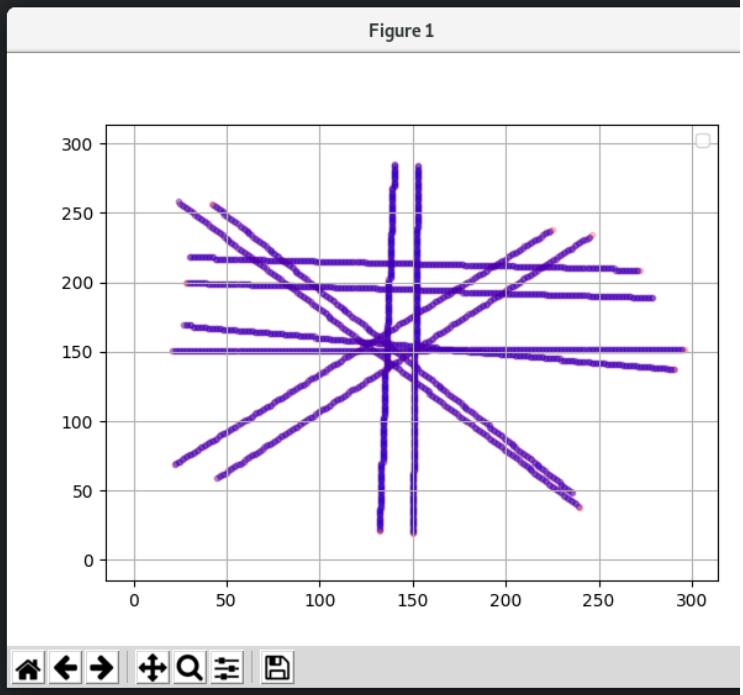
Una vez dado el segundo click derecho dentro del plano cartesiano, el cual se define como el punto final, se dibujan dos rectas, del punto de inicio al punto final.
En rojo la recta es dibujada con el algoritmo Bresenham. En azul la recta con DDA.

Despues de dibujar la linea, en la terminal se muestras de ejecucion de cada Algoritmo y la diferencia en tiempo entre ellos.

Pruebas y resultados

Se realizaron pruebas para comprobar la correcta captura de los tiempos de ejecucion y garantizar la correcta representacion de rectas en el plano en diferentes angulos y cuadrantes, especialmente para el algoritmo DDA que depende de estas cuestiones para su comportamiento: linea vertical, linea inclinada, horizontal segun el punto de inicio y fin e invirtiendo los puntos de inicio y fin.

```
(.venv) [pipo@localhost actividad_02]$ python3 main.py
 starting
No handles with labels found to put in legend.
140 285
132 21
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  599628 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  588835 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  10793 MICROSECONDS
153 284
150 19
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  518035 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  491205 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  26830 MICROSECONDS
21 151
296 152
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  417717 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  369368 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  48349 MICROSECONDS
27 169
291 137
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  301925 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  329829 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  -27904 MICROSECONDS
279 189
28 200
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  274593 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  262311 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  12282 MICROSECONDS
30 218
272 209
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  213494 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  143816 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  69678 MICROSECONDS
24 258
240 38
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  193388 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  180019 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  13369 MICROSECONDS
236 48
42 257
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  147296 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  144911 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  2385 MICROSECONDS
22 69
225 238
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  97381 MICROSECONDS
TIME EXECUTION FOR DDA : 86399 SECONDS AND  64634 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 0 SECONDS AND  32747 MICROSECONDS
45 59
246 234
TIME EXECUTION FOR BRESENHAN : 86399 SECONDS AND  17064 MICROSECONDS
TIME EXECUTION FOR DDA : 86398 SECONDS AND  967279 MICROSECONDS
TIME DIFF BRESENHAN - DDA : 1 SECONDS AND  -950215 MICROSECONDS
```



Conclusiones
Se desarrollo un sistema capaz de graficar lineas rectas y capturar sus tiempos de ejecucion en tiempo real a traves de los algoritmos DDA y Bresenham en la tecnolgia python con el uso de librerias graficas, el cual permite mostrar los puntos de inicio y fin de la recta definidos por el usuario mediante eventos de click derecho.

Apendice
LOS CODIGOS ESTAN DISPONIBLES EN EL REPOSITORIO:
https://github.com/CUCEI-TAREAS/SIMULACION-POR-COMPUTADORA-2019A

main.py
gui_cartesian.py

main.py

```
#!/usr/bin/env python

import sys

import gui_cartesian as plane

if __name__ == "__main__":

    print(" starting ")
    plane.setupPlane()
```

gui_cartesian.py

```
#!/usr/bin/env python

import datetime
import matplotlib.pyplot as plt
import numpy as np

from matplotlib.image import AxesImage

fig, ax = plt.subplots()

final_points = list()
dist = list()

def onclick(event):

    if event.button == 3:
        print(int(round(event.xdata)), int(round(event.ydata)))

        if final_points == []:
            final_points.append([int(round(event.xdata)), int(round(event.ydata))])
        else:
            x = final_points[0][0]
            y = final_points[0][1]

            bresenham(x, y, int(round(event.xdata)), int(round(event.ydata)))
            dda(x, y, int(round(event.xdata)), int(round(event.ydata)))

def setupPlane():

    plt.plot(range(0, 300), linewidth=0)
    #ax.axis=([0, 300, 0, 300])

    plt.grid()

    fig.canvas.mpl_connect('button_press_event', onclick)

    ax.legend()

    plt.show()

def drawPoint(points=[], col=0):
    color=['red', 'green', 'blue']
    scale = 10
    for point in points:
        x = point[0]
        y = point[1]
        #print(x, " valor ", y, " con respecto a ", point )
```

```
        ax.scatter(int(x), int(y), c=color[col], s=scale, alpha=0.2, edgecolors='face')
        #im1 = ax.imshow(rand(10, 5), extent=(1, 2, 1, 2), picker=True)



def dda(xi, yi, xf, yf):

    dda_time_inicio = datetime.datetime.now()

    ordenada = False
    absisa = False
    t = 0

    inc = 1

    dy = yf - yi
    dx = xf - xi

    m = 0

    if dy == 0:
        t += 1
        absisa = True

    if dx == 0:
        t += 1
        ordenada = True

    if t == 2:
        print("error, same point")
        final_points.clear()
        return -1

    if t != 1:
        m = dy / dx

    b = yi - (m * xi)

    if ordenada:

        if yi > yf :
            inc = -1
            b = yf

        for i in range(yi, yf, inc):
            yact = b
            drawPoint([[round(yact), i]], 2)
            #print("from ordenada") elif absisa:
```

```
        if xi > xf :
            inc = -1
            b = yf

        for i in range(xi, xf, inc):
            yact = b
            drawPoint([i, [round(yact)]], 2)
            #print("from absisa")

    elif m >= 1:

        if yi > yf :
            inc = -1

        for i in range(yi, yf, inc):
            yact = (i - b) / m
            drawPoint([[round(yact), i]], 2)
            #print("from m >= 1")

    elif m > -1 :
        if xi > xf :
            inc = -1

        for i in range(xi, xf, inc):
            yact = ( m * i) + b
            drawPoint([[i, round(yact)]], 2)
            #print("from m > 0")

    elif m <= -1 :
        if yi > yf :
            inc = -1

        for i in range(yi, yf, inc):
            yact = (i - b) / m
            drawPoint([[round(yact),i]], 2)
            #print("from m >= -1")


    #print(m)
    final_points.clear()
    dda_time_end = datetime.datetime.now()
    dda_time = dda_time_inicio - dda_time_end
    print("TIME EXECUTION FOR DDA :",dda_time.seconds, "SECONDS AND ",
dda_time.microseconds, "MICROSECONDS")
    print("TIME DIFF BRESENHAN - DDA :", dist[0].seconds - dda_time.seconds, "SECONDS AND
", dist[0].microseconds - dda_time.microseconds, "MICROSECONDS")
    dist.clear()
    plt.show()
```

```
def bresenham(xi, yi, xf, yf):

    bres_time_inicio = datetime.datetime.now()

    dx = abs(xf - xi)
    dy = abs(yf - yi)
    x, y = xi, yi
    sx = -1 if xi > xf else 1
    sy = -1 if yi > yf else 1
    if dx > dy:
        err = dx / 2.0
        while x != xf:
            drawPoint([[x,y]],0)
            err -= dy
            if err < 0:
                y += sy
                err += dx
            x += sx
    else:
        err = dy / 2.0
        while y != yf:
            drawPoint([[x,y]],0)
            err -= dx
            if err < 0:
                x += sx
                err += dy
            y += sy

    drawPoint([[x,y]],0)

    bres_time_end = datetime.datetime.now()
    bres_time = bres_time_inicio - bres_time_end
    dist.append(bres_time)

    print("TIME EXECUTION FOR BRESENHAN :", bres_time.seconds, "SECONDS AND ",
bres_time.microseconds, "MICROSECONDS")
```