# DDA (Digital Differential Analyzer) PSEUDOCODE

Get the input of two end points (X1,Y1) and (X2,Y2).

calculate dx, dy

dx = X2 - X1;

dy = Y2 - Y1;

Depending upon absolute value of dx & dy

choose number of steps to put pixel as

steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy)

steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

Calculate the increment in x coordinate and y coordinate for each step

xIncrement = dx / (float) step;

yIncrement = dy / (float) step;

set the pixel by successfully incrementing x and y coordinates

//accordingly and complete the line drawing.

X = X1;

Y = Y1;

//for first starting pixel of line

setpixel (X,Y);

for (int i = 0; i <= step; i++)

{

setpixel (X,Y);

X += xIncrement;

Y += yIncrement;

}

Exit.

What is the limit

# **BRESENHAM'S LINE PSEUCODE**

Step1: Start Algorithm

Step2: Declare variable x1,x2,y1,y2,dx,dy

Step3: Enter value of x1,y1,x2,y2  
                Where x1,y1are coordinates of starting point  
                And x2,y2 are coordinates of Ending point

Step4: Calculate dx = x2-x1  
                Calculate dy = y2-y1  
                Calculate sx (the greater between X1 and X2)  
                Calculate sy(the greater between y1 and y2)

Step5: Consider (x, y) as starting point and (x2,y2)as maximum possible value of x and y.

Printand plot the starting points(x,y)

If (dx > dy) Then

p = 2 \* dy - dx;

Given that (x != x2) then

x += sx;

if (p < 0) Then

p += 2 \* dy

otherwise if (p>0) then

y += sy;

p += 2 \* (dy - dx)

Print and plot theconsecutive points(x,y)

OTHERWISE IF (dy >dx)

p = 2 \* dx - dy;

AS LONG AS (y != y2) then

y += sy;

if (p < 0)

p += 2 \* dx;

otherwise if(p>0) then

x += sx;

p += 2 \* (dx - dy)

Print and plot theconsecutive points(x,y)

END OF ALGORITHM

**MIDPOINT CIRCLE**

function draw\_circle(center\_x, center\_y, radius):

x = radius

y = 0

p = 1.25 - radius # Updated decision parameter

// Initialize a list to store the coordinates

coordinates = []

// Plot the initial points in all octants

while x >= y:

// Add coordinates to the list for the current octant

coordinates.append((x + center\_x, y + center\_y))

coordinates.append((y + center\_x, x + center\_y))

coordinates.append((-x + center\_x, y + center\_y))

coordinates.append((-y + center\_x, x + center\_y))

coordinates.append((-x + center\_x, -y + center\_y))

coordinates.append((-y + center\_x, -x + center\_y))

coordinates.append((x + center\_x, -y + center\_y))

coordinates.append((y + center\_x, -x + center\_y))

y += 1

if p <= 0:

p = p + 2 \* y + 1

else:

x -= 1

p = p + 2 \* y - 2 \* x + 1

// Plot the circle using the list of coordinates

// Set the aspect ratio to 'equal' to ensure it appears as a complete circle

// Display the circle coordinates in a table

// Input: center\_x, center\_y, radius

// Assuming center at (0, 0)

// Call draw\_circle function with the provided inputs