Scan Conversion of primitive objects.

Q. What is scan conversion?

→ To output or draw an object on a display system we need to represent continuous primitive objects like line, curve etc. as a collection of discrete pixels. The process of doing so, is called scan conversion or rasterization.

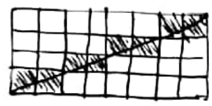
Q. What is the objective of developing a good scan conversion algorithm?

→ A good scan conversion algorithm should have the following qualities:

1. discrete approximation of an continuous object should be visually satisfactory.
2. the time required for the rasterization procedure should be minimum/less.

Q. What is staircase effect?

→ When a tilted line or a curve is approximated by a sequence of pixels the smooth line or curve has an stair step like appearance as shown in the figure. This jaggered appearence is called staircase effect or jaggies.



This effect is easy to observe in low resolution displays.
In modern systems this effect is countered by a process called anti-aliasing.

Q. Describe the outline of scan conversion of a line segment.

→ A line segment is described by its two end points (x_1, y_1) and (x_2, y_2).
Lets assume $x_2 \geq x_1$
For this, we can obtain the line equation $y = mx + c$

where $m = \frac{y_2 - y_1}{x_2 - x_1}$ and $c = y_2 - \frac{x_2}{x_2 - x_1}$, and we have:

```
naive_line_drawing_algorithm (x1, y1, x2, y2) {
    calculate m and c
    for x = x1 to x2 with step 1 do {
        calculate y by y = mx + c
    }   ~~putpixel~~ putpixel ( round(x), round(y) )
}
```

Q. What are draw backs of the above algorithm?

→ • The above algorithm iterates over the x-coordinates and computes the y-coordinate for a choosen x using the line equation.

If x_1 and x_2 are very close while y_1 and y_2 are far apart. the produced line might not be visually satisfactory. The situation is worst when $x_2 - x_1 \leq 1$, only one or two poins are displayed which may not be even connected.

• The second draw back is that m and c can be floating point numbers Thus using $y = mx + c$ in each iteration requires floating point multiplication and addition which are very time consuming compared to their integer counterparts.

Q. Describe how DDA algorithm overcomes these drawbacks.

→ The Digital Differential Analyzer (DDA) algorithm avoids the costly floating point multiplications by computing the constant differene between succesive points along the line. And to produce a visually pleasing line it chooses between x and y- coordinates ~~which~~ to loop on whichever produces maximum number of pixels.

Q. Describe the DDA algorithm.

→ [write the prev. ans.]

Let us define, $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$

Let us assume the case when $\Delta x \geq 0$, $\Delta y \geq 0$ and $\Delta x \geq \Delta y$ $\left[0 \leq \Delta Y \leq \Delta x\right]$

$\therefore \quad m = \dfrac{\Delta Y}{\Delta x} \leq 1$

Here iterating on the x-coordinates would produce maximal number of points

(P.T.O)

if (x_i, y_i) be the current point

the next point will be (x_{i+1}, y_{i+1}) where $x_{i+1} = x_i + 1$ [$\because$ loop over x-coordinate]

and $y_{i+1} = m x_{i+1} + c$

$$y_i = mx_i + c$$

$$\therefore \quad y_{i+1} - y_i = m(x_{i+1} - x_i) = m$$

There fore, starting at $x = x_1, y = y_1$ we can compute next points by these updation expressions:

$$x = x + 1$$
$$y = y + m$$

Similarly when, $0 \le \Delta x \le \Delta y$, we have $m \ge 1$

we iterate over y-coordinates. thus $y_{i+1} = y_i + 1$

And we have $y_{i+1} = m x_{i+1} + c$

and $y_i = m x_i + c$

Thus, $y_{i+1} - y_i = m(x_{i+1} - x_i)$ $\quad \therefore \quad x_{i+1} - x_i = \frac{y_{i+1} - y_i}{m} = \frac{1}{m}$

or $x_{i+1} = x_i + \frac{1}{m}$

The case when Δx (or Δy) is negetive we write $x_{i+1} = x_i - 1$ (or $y_{i+1} = y_i - 1$)

~~i.e. loo~~ i.e. loop in the opposite direction with step = -1

Following in similar fashion, we can compute the updation equations. for all cases.
The following table summerizes this.

	(1st quadrant) $\Delta x \ge 0$, $\Delta y \ge 0$, m +ve	(2nd ...) $\Delta x < 0$, $\Delta y \ge 0$, m -ve	(3rd ...) $\Delta x < 0$, $\Delta y < 0$, m +ve	(4th ...) $\Delta x \ge 0$, $\Delta y < 0$, m -ve
$\lvert\Delta x\rvert \ge \lvert\Delta y\rvert$ i.e. $\lvert m\rvert \le 1$	$x_{i+1} = x_i + 1$ $y_{i+1} = y_i + m$	$x_{i+1} = x_i - 1$ $y_{i+1} = y_i - m$	$x_{i+1} = x_i - 1$ $y_{i+1} = y_i - m$	$x_{i+1} = x_i + 1$ $y_{i+1} = y_i + m$
$\lvert\Delta x\rvert < \lvert\Delta y\rvert$ $\lvert m\rvert > 1$	$y_{i+1} = y_i + 1$ $x_{i+1} = x_i + 1/m$	$y_{i+1} = y_i + 1$ $x_{i+1} = x_i + 1/m$	$y_{i+1} = y_i - 1$ $x_{i+1} = x_i - 1/m$	$y_{i+1} = y_i - 1$ $x_{i+1} = x_i - 1/m$

Based on this table we write a general equation for updation/increments where.

$$x_{i+1} = x_i + x_{inc} \qquad \text{where} \qquad x_{inc} = \frac{\Delta x}{steps}$$

$$y_{i+1} = y_i + y_{inc} \qquad\qquad y_{inc} = \frac{\Delta y}{steps}$$

where $step = \begin{cases} |\Delta x| & \text{when } |\Delta x| \geq |\Delta y| \\ |\Delta y| & \text{otherwise.} \end{cases}$ ~~and steps = { |x₂ − x₁| when~~

// (you may skip this part in exam)

Therefore the algorithm is as follows:

```
DDA_line_drawing_algorithm( x1, y1, x2, y2 ) {
    dx = x2 - x1
    dy = y2 - y1
    if abs(dx) >= abs(dy) then {
            steps = abs(dx)
    } else {
            steps = abs(dy)
    }
    xinc = dx / steps
    yinc = dy / steps

    x = x1
    y = y1
    for ( i = 0; i <= steps ; i++ ) {
            putpixel( round(x) , round(y) )

            x = x + xinc
            y = y + yinc
    }
}
```

Q. Scan convert the line segment from (1, 2) to (3, 6) using DDA algorithm.

→ Here $\Delta x = 3-1 = 2$
$\Delta y = 6-2 = 4$

// since both are positive the line segment is in first quadrant //

steps $= |\Delta y| = 4$

$x_{inc} = \Delta x / steps = 2/4 = \frac{1}{2} = 0.5$

$y_{inc} = \Delta y / steps = 4/4 = 1$

Step	current x, y	pixel round(x), round(y)	next x, y ($x = x + x_{inc}$, $y = y + y_{inc}$)
iter 0	1, 2	1, 2	1 + 0.5 = 1.5, 2 + 1 = 3
iter 1	1.5, 3	2, 3	1.5 + 0.5 = 2, 3 + 1 = 4
iter 2	2, 4	2, 4	2 + 0.5 = 2.5, 4 + 1 = 5
iter 3	2.5, 5	3, 5	2.5 + 0.5 = 3, 5 + 1 = 6
iter 4	3, 6	3, 6	3 + 0.5 = 3.5, 6 + 1 = 7

Q. Scan convert the line segment from (1, 6) to (3, 2) using DDA Algorithm

→ Here $\Delta x = 3-1 = 2$, steps $= \max(|\Delta x|, |\Delta y|) = |\Delta y| = 4$
$\Delta y = 2-6 = -4$

$x_{inc} = \Delta x / steps = 0.5$
$y_{inc} = \Delta y / steps = -1$

steps	current x, y	pixel round(x), round(y)	next x, y ($x = x + x_{inc}$, $y = y + y_{inc}$)
step 0	1, 6	1, 6	1 + 0.5 = 1.5, 6 + (-1) = 5
1	1.5, 5	2, 5	1.5 + 0.5 = 2, 5 - 1 = 4
2	2, 4	2, 4	2 + 0.5 = 2.5, 4 - 1 = 3
3	2.5, 3	3, 3	2.5 + 0.5 = 3, 3 - 1 = 2
4	3, 2	3, 2	3 + 0.5 = 3.5, 2 - 1 = 1

Q. Scan convert line segment (3,6) to (1,2) using DDA

Q. Scan convert line segment (3,2) to (1,6) using DDA

Q. ... (10,10) to ~~(3~~,1) ... [H/W: DYI]

Q. ... (10,10) to (1,7) ...

Q. ... (5,5) to (1,1) ...

Q. What are the drawbacks of DDA line drawing algorithm?

→ Although DDA manages to avoid time consuming floating point multiplications inside the loop, but it still does floating point addition. Not only these floating point operation is time consuming, but also addition of floating point numbers causes accumulation round-off errors due to fixed amount of ~~pre~~ precision available in the variables of any programming language. And thus the generated pixels eventually drifts away from the original line.

Q. How Bresenham algorithm ~~improve~~ overcomes the drawbacks of DDA?

→ The algorithm devised by Bresenham efficiently produces more accurate scan converted line compared to DDA by cleverly avoiding floating point arithmetics and round-offs by only using incremental integer calculations.

Q. Describe the Bresenham line drawing algorithm [for the case $0 \le \Delta y \le \Delta x$]

→ The algorithm raterizes/scan converts a line segment by incrementing by one unit either in x or y depending on the slope of the line and then selects the pixels lying at least distance from the true line path at each position.

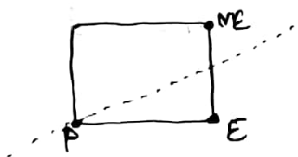
(P.T.O.)

For simplicity let us first consider the case of 1st octant

i.e. $\Delta x \geq 0$, $\Delta y \geq 0$ and $\Delta x \geq \Delta y$ or $m \leq 1$

If $P(x,y)$ is a point on the line segment the next pixel point must be either the point E, the <u>east</u> point w.r.t P, located at $(x+1, y)$ or the point NE, the <u>north-east</u> - - - - - , located at $(x+1, y+1)$

Since the line must pass in between E and NE



The choice between E and NE is ~~not~~ resolved by picking the point which is closest to the original line

Let us consider the line equation as $ax + by + c = 0$

where $a = \Delta y$, $b = -\Delta x$, and c is also a fn. of x_1, y_1, x_2, y_2

Let $f(x,y)$ be $ax + by + c$.

The decision between E and NE can be resolved systematically as follows:

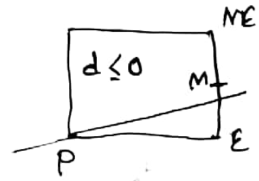
Consider the point M located at $(x+1, y+1/2)$, the <u>mid</u> point between E and NE.

We evaluate $f(M)$, let's call this value d, which will help us take the <u>decision</u>.

If we find $d \leq 0$

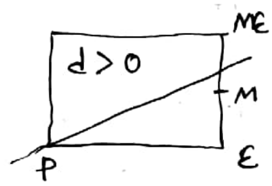
The situation must be that the line is below M

i.e. E is more closer to the line.



And if $d > 0$

The line passes above M, i.e. NE is more closer to the line.

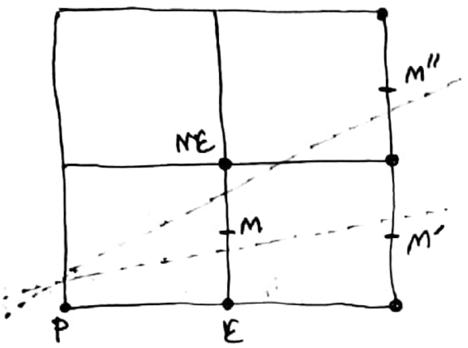


Notice that this analysis does not require the point P to be exactly on the line.

As long as, the line does pass ~~the~~ between E and NE we are fine.

So, now depending on our choice, we move to either E or NE

We consider this ~~new~~ point as new P and repeat the analysis.



Depending on our choice E or NE the next decision point must be

$M'(x+2, y+1/2)$ or $M''(x+2, y+3/2)$ respectively.

Knowing the value of $f(M)$ we can easily obtain the values of $f(M')$ and $f(M'')$ as shown below.

We have, $f(M') = a(x+2) + b(y+1/2) + c$ ---- (1)

$f(M'') = a(x+2) + b(y + 3/2) + c$ ---- (2)

we also have, $f(M) = a(x+1) + b(y+1/2) + c$ -- (3)

(1) − (3) yields, $f(M') - f(M) = a = \Delta y$, lets call this Δd_E

(2) − (3) yields $f(M'') - f(M) = a + b = \Delta y - \Delta x$, lets call this Δd_{NE}

case East: So, if at point P we observe that $d = f(M) \leq 0$
we move to point E and compute d_{new} as $f(M') = d + \Delta d_E$

case North East: if we have $d > 0$
we move to point NE and compute d_{new} as $f(M'') = d + \Delta d_{NE}$

Now all we need that the starting value of the decision variable.
Since we start at the point (x_1, y_1)
the initial decision value will be

$$
\begin{aligned}
d_{init} &= f(x_1+1, y_1+1/2) \\
&= a(x_1+1) + b(y_1+1/2) + c \\
&= ax_1 + by_1 + c + a + b/2 \\
&= a + b/2 \qquad \left[\text{Since } (x_1, y_1) \text{ on the line we have } f(x_1, y_1) = 0\right] \\
&= \Delta y - \Delta x/2
\end{aligned}
$$

Now, we have a situation where in only the initial expression of the decision variable we have division which might result into a floating point value. This value will be carried all the way to the last point generated.

To avoid, floating point numbers altogether, we perform a simple trick. We multiply the expressions involving decision variables by a factor of two.

This results into.

$$d_{init} = \cancel{2a+b} \quad 2\Delta y - \Delta x$$

$$\Delta d_E = 2\Delta y$$

$$\Delta d_{NE} = 2(\Delta y - \Delta x)$$

Since, the decision variable is used only for the decision $d \gtrless 0$ and nowhere else, and a uniform multiplication by a positive scalar value does not change the decision at any point, our analysis still remains valid.

Thus the algorithm for 1st octant ($0 \leq \Delta y \leq \Delta x$) is as follows:

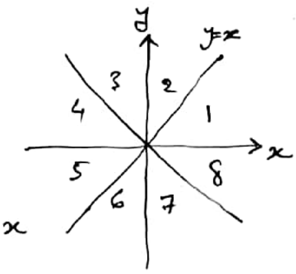
```
Bresenham_line_drawing_algorithm(x1, y1, x2, y2) {
        dx = x2 - x1
        dy = y2 - y1

        ddE = 2 * dy
        ddNE = 2 * (dy - dx)

        d = 2 * dy - dx

        x = x1
        y = y1
        putpixel(x,y)
        while ( x < x2 ) {
                if (d <= 0) {   // case East
                        x = x + 1
                        d = d + ddE
                } else {        // case North East
                        x = x + 1
                        y = y + 1
                        d = d + ddNE
                }
                putpixel(x, y)
        }
}
```

Q. Obtain the generalized Bresenhan line drawing algorithm.

[prev. ans in brief]

→ One might repeat the same analysis for the remaining 7 octants. and then comeup with a generalization.

Instead, we exploit the symmetry of the octants and reuse our analysis for first octant to obtain a generalized algorithm.



Using symmetry we make the following observations :

- octant 2 is reflection of octant 1 about the line $y = x$
  this basically exchanges x and y with each other.
- octant 3 and 4 are reflitions of octant 2 and 1 respectively about y-axis
  this basically ~~makes x n~~ inverts the sign of x-coordinates / increments
- octant 7 and 8 are - - . . . . . - - - - - 2 and 1 respectively - . . x-axis
  this baically inverts the sign of y-coordinates / increments.
- octant 5 and 6 . . - - - - - - . . 1 and 2 respectively about the origin
  this ~~exchanges x and y and also~~ inverts both the signs of x and y-coordinates / increments.

Thus we can write the following table:

	$\Delta x \geq 0$, $\Delta y \geq 0$	$\Delta x < 0$, $\Delta y > 0$	$\Delta x < 0$, $\Delta y < 0$	$\Delta x > 0$, $\Delta y < 0$
$\lvert\Delta x\rvert \geq \lvert\Delta y\rvert$, $\lvert m\rvert \leq 1$	(1st octant)	(4th octant)	(5th octant)	(8 th octant)
	$d_{init} = 2\lvert\Delta y\rvert - \lvert\Delta x\rvert$	✓	✓	✓
	$\Delta d_E = 2\lvert\Delta y\rvert$	✓	✓	✓
	$\Delta d_{NE} = 2(\lvert\Delta y\rvert - \lvert\Delta x\rvert)$	✓	✓	✓
	$E = x+1, y$	$x-1, y$	$x-1, y$	$x+1, y$
	$NE = x+1, y+1$	$x-1, y+1$	$x-1, y-1$	$x+1, y-1$
$\lvert\Delta x\rvert < \lvert\Delta y\rvert$, $\lvert m\rvert > 1$	(2nd octant)	(3rd octant)	(6 th octant)	(7 th octant)
	$d_{init} = 2\lvert\Delta x\rvert - \lvert\Delta y\rvert$	✓	✓	✓
	$\Delta d_E = 2\lvert\Delta x\rvert$	✓	✓	✓
	$\Delta d_{NE} = 2(\lvert\Delta x\rvert - \lvert\Delta y\rvert)$	✓	✓	✓
	$E = x, y+1$	$x, y+1$	$x, y-1$	$x, y-1$
	$NE = x+1, y+1$	$x-1, y+1$	$x-1, y-1$	$x+1, y-1$

if $d \leq 0$ then East / if $d > 0$ then North East

The above table makes use of the fact that

$$|a| = \begin{cases} a & \text{when } a \geq 0 \\ -a & \text{when } a < 0 \end{cases}$$

Thus the generalized algorithm is :

```
general_Bresenham_line_drawing_algorithm(x1, y1, x2, y2){
    dx=abs(x2 - x1)
    dy=abs(y2 - y1)
    xinc = sign(x2 - x1)
    yinc = sign(y2 - y1)

    if (dy > dx){   // |m| > 1
        swap(dy, dx)
    }
   d = 2 * dy - dx   // dinit
    ddE = 2 * dy
    ddNE = 2 * (dy - dx)

    x = x1
    y = y1
    putpixel(x, y)
    for(i=1; i <= dx; i++){
        if (d <= 0) {   // case East
            if ( dy > dx) { // |m| > 1
                y = y + yinc
            } else {   // |m| ≤ 1
                x = x + xinc
            }
            d = d + ddE
        } else {   // case North East
            x = x + xinc
            y = y + yinc
            d = d + ddNE
        }
        putpixel(x, y)
} }
```

Here sign( ) is a helper function defined as:

$$\text{sign}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \end{cases}$$

and swap(x, y) exchanges the values of the two variables.

(P.T.O)

NB The multiplication with 2 can be done by a left shift.

NB the above analysis is called midpoint analysis method.

The original Bresenham analysis was based on the drift error incurred by choosing pixel point. It computes a error parameter P_k at k-th step and updates its value depending on the next point. The next point is choosen based on the sign of the error parameter. This basically does the same thing as midpoint method from a different perspective and the end result turns out to be ~~the~~ exact same.

Q. Scan convert the line segment from (0,0) to (6,7) using Bresenham method.

→ $\Delta x = |6-0| = 6$ $x_{inc} = +1$

$\Delta y = |7-0| = 7$ $y_{inc} = +1$

$\Delta y > \Delta x$ so we swap: $\Delta x = 7$, $\Delta y = 6$

$d_{init} = 2\Delta y - \Delta x = 2\times 6 - 7 = 5$

$\Delta d_E = 2\Delta y = 2\times 6 = 12$

$\Delta d_{NE} = 2(\Delta y - \Delta x) = 2\times(6-7) = -2$

i	~~current~~ current x, y	current d d>0 → NE d≤0 → E	~~choice~~ next x, y E = (x, y+1) NE = (x+1, y+1)	next d	plot x, y
0					0, 0
1	0, 0	5 > 0 → NE	0+1 = 1, 0+1 = 1	5 + (-2) = 3	1, 1
2	1, 1	3 > 0 → NE	1+1=2, 1+1=2	3+(-2) = 1	2, 2
3	2, 2	1 > 0 → NE	2+1 =3, 2+1, 3	1+(-2) = -1	3, 3
4	3, 3	-1 < 0 → E	3, 3+1 =4	-1 + 12 = 11	3, 4
5	3, 4	11 > 0 → NE	3+1=4, 4+1=5	11 +(-2) = 9	4, 5
6	4, 5	9 > 0 → NE	4+1 = 5, 5+1=6	9 +(-2) = 7	5, 6
7	5, 6	7 > 0 → NE	5+1=6, 6+1=7	7 + (-2)=5	6, 7 (stop)