Chapter 3
Scan Conversion

☐ **Pixel:** (Pel, Picture element)

The smallest addressable and controllable element in a display device is pixel.

The address of a pixel corresponds to its physical co-ordinates which is a physical point in an image.

each pixel is a sample of an original image; more samples typically provide more accurate representations of the original. So, picture quality is directly proportional to picture resolution.

☐ **Scan Conversion:**

other name: Rastenisation / Rasterization.

Defination: Task of taking an image described in a vecton graphics format (shapes) and converting it into a raster image (pixels or dots) for output on an video display or printer, or for storage in a bitmap file format is called scan conversion.

Mathmutically:

(x, y) is a mathmetical point and x and y are real numbers.

Now, (x, y) is to be scan converted to a pixel location at (x', y') where x' is the integer part of x and y' is the integer part of y.

Here, $x' = floor(x)$ and

$y' = floor(y)$.

function floor returns the largest integer that is less than or equal to argument.

Chapter 3
Scan Conversion

that is

all point satisfies $x' \leq x < x'+1$ and $y' \leq y < y'+1$ are mapped to pixel (x', y').

$P_1(1.7, 0.8) \longrightarrow (1, 0)$

$P_2(2.2, 1.3) \longrightarrow (2, 1)$

$P_3(2.8, 1.9) \longrightarrow (2, 1)$

Another approach is, $x' = \text{floor}(x + 0.5)$ and $y' = \text{floor}(y + 0.5)$.
That is, all points that satisfy $x' - 0.5 \leq x < x' + 0.5$ and $y' - 0.5 \leq y < y' + 0.5$ are mapped to pixel (x', y'). Then,

$P_1(1.7, 0.8) \longrightarrow (2, 1)$

$P_2(2.2, 1.3) \longrightarrow (2, 1)$

$P_3(2.8, 1.9) \longrightarrow (3, 2)$

☐ Line drawing Algorithms:

3.2 Scan-converting a line:

- Line segment :→ Line : portion of a straight line that extends indefinitely in opposite direction.

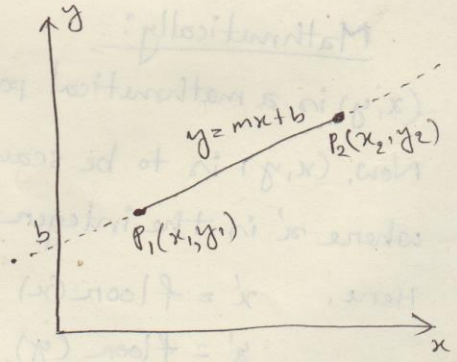
⇒ defined by two end points and the line equation $y = mx + b$

$m \rightarrow$ slope, $b \rightarrow$ y intercept of the line.

Limitation:

→ not suitable for vertical lines.

→ cannot provide faster response in case of interactive graphics.



DDA algorithms

Describe DDA algorithm with merits and demerits.

DDA : Digital Differential Algorithm.

DDA is an incremental scan conversion method. It is characterised by performing calculations at each step using result from the preceeding step. as follows:

Let, (x_i, y_i) is the initial point.

We determine slope, $m = \frac{dy}{dx}$. Now,

$$y_{i+1} = y_i + m\Delta x$$
$$x_{i+1} = x_i + \Delta y/m$$

if $|m| \leq 1$,

then increment, $\Delta x = 1$, compute and round y

$$y_{i+1} = y_i + m\Delta x = y_i + m$$

if $|m| > 1$,

then, by unit incrementing in y direction, set, $\Delta y = 1$,

compute and round x.

$$x_{i+1} = x_i + \Delta y/m = x_i + 1/m$$

This process continues until x reaches x_2' (for $|m| \leq 1$) or y reaches y_2' (for $|m| > 1$) and all points found are scan converted to pixel co-ordinates.

It is simplest line drawing algorithm. (merit)

demerits:

→ faster than direct use of line equation since it calculates points on the line without floating-point multiplication.

☐ Demerits:

→ Not very efficient

→ floating point operations (addition) and rounding operations are expensive.

→ Round-off causes calculated points to drift away from their true position when the line is relatively long.

☒ Pseudo code:

- Compute m;
- if $m < 1$:

```
{   float y = y0;   // initial value
    for (int x = x0; x <= x1; x++, y += m)
        ~~set pixel~~ setPixel (x, round (y));
}
else // m > 1
{
    float x = x0;   // initial value
    for (int y = y0; y <= y1; y++, x += 1/m)
        setPixel (round (x), y);
}
```

(*) Suppose we want to draw a line, starting at pixel (2,3) and ending at pixel (12,8). What are the values of the variable x and y at each time step? What are the pixel coloured, according to the DDA algorithm?

$x = x_0 + 1$

Here, $m = \frac{8-3}{12-2} = 0.5 < 1$, then, $y_{i+1} = y_i + m$. [$\because \Delta x = 1$]

t	x	$y_{i+1} = y_i + m$	y	R(x)	R(y)
0	2	—	3	2	3
1	(2+1) 3	$y_1 = 3 + 0.5$	3.5	3	4
2	(3+1) 4	$y_2 = 3.5 + 0.5$	4	4	4
3	(4+1) 5	$y_3 = 4 + 0.5$	4.5	5	5
4	6	$y_4 = 4.5 + 0.5$	5	6	5
5	7	$y_5 = 5 + 0.5$	5.5	7	6
6	8	$y_6 = 5.5 + 0.5$	6	8	6
7	9	$y_7 = 6 + 0.5$	6.5	9	7
8	10	$y_8 = 6.5 + 0.5$	7	10	7
9	11	$y_9 = 7 + 0.5$	7.5	11	8
10	12	$y_{10} = 7.5 + 0.5$	8	12	8

⊛ Bresenham's line algorithm:

⊛ What is Bresenham line algorithm? Derive parameter equation of it:

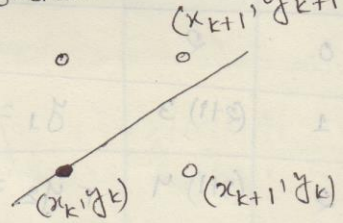
☐ Bresenham's line algorithm:

A highly efficient incremental method for scan converting lines.
It uses integer addition, subtraction and multiplication by 2 with simple arithmetic shift operation.
- move across the x axis in unit intervals and at each step choose between two different y co-ordinates.

☐ Derivation:

Let our current pixel position (x_k, y_k).

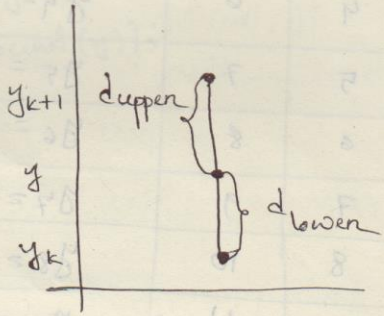
Now, in Bresenham line algorithm, our next choice will be (x_{k+1}, y_k) or (x_{k+1}, y_{k+1}).



original line equation at x_{k+1} is,

$$y = m(x_{k+1}) + b = m(x_k + 1) + b$$

Then, let, d_{upper} is at (x_{k+1}, y_{k+1}) and d_{lower} is at (x_{k+1}, y_k). Then,



$$d_{upper} = (y_{k+1}) - y$$
$$= (y_k + 1) - y$$
$$= y_k + 1 - m(x_k + 1) - b$$

and $$d_{lower} = y - y_k$$
$$= m(x_k + 1) + b - y_k$$

Therefore, $d_{lower} - d_{upper} = m(x_k+1) + b - y_k - y_k - 1 + m(x_k+1) + b$

$= 2m(x_k+1) - 2y_k + 2b - 1$

substituting m by $\Delta y/\Delta x$ & multiplying with Δx, we get,

$(d_{lower} - d_{upper})\Delta x = \Delta x \left\{ 2\frac{\Delta y}{\Delta x}(x_k+1) - 2y_k + 2b - 1 \right\}$

$= 2\Delta y(x_k+1) - 2\Delta x y_k + 2\Delta x b - \Delta x$

$= 2\Delta y x_k + 2\Delta y - 2\Delta x y_k + 2\Delta x b - \Delta x$

$= 2\Delta y x_k - 2\Delta x y_k + 2\Delta x(2b-1) + 2\Delta y$

Taking $\Delta x(2b-1) + 2\Delta y = c$, we get,

$(d_{lower} - d_{upper})\Delta x = 2\Delta y x_k - 2\Delta x y_k + c$.

Now, decision parameter, $P_k = (d_{lower} - d_{upper})\Delta x$

$= 2\Delta y x_k - 2\Delta x y_k + c$

Then, If P_k is negative, we choose lower pixel and

If P_k is positive, we choose upper pixel.

—o—

$P_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c$

$\therefore P_{k+1} - P_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$

$= 2\Delta y - 2\Delta x(y_{k+1} - y_k)$ [$\because x_{k+1} = x_k + 1$]

where $y_{k+1} - y_k$ is either 0 or 1 depending on sign of P_k.

[$y_{k+1} = (y_k+1)$ or (y_k)]

(*) Derive first decision parameter $P_0 = 2\Delta y - \Delta x$ at (x_0, y_0).

let our current pixel position is (x_k, y_k). Now in Bresenham line algorithm, our next choice will be (x_{k+1}, y_k) or (x_{k+1}, y_{k+1})

Line equation at $x_k + 1$ is,

$$y = m(x_{k+1}) + b = m(x_k + 1) + b$$

Now, $d_{upper} = y_{k+1} - y$

$$= (y_k + 1) - y$$

$$= y_k + 1 - m(x_k + 1) - b$$

and $d_{lower} = y - y_k$

$$= m(x_k + 1) + b - y_k$$

Therefore, $d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$

(Substituting m by $\frac{\Delta y}{\Delta x}$) and multiplying by Δx, we get,

$$\Delta x(d_{lower} - d_{upper}) = \Delta x\{2m(x_k + 1) - 2y_k + 2b - 1\}$$

$\Delta x(d_{lower} - d_{upper})$ is P_k. Then,

$$P_k = \Delta x\{2m(x_k + 1) - 2y_k + 2b - 1\}$$

Now, at (x_0, y_0),

$$P_0 = \Delta x\{2m(x_0 + 1) - 2y_0 + 2b - 1\}$$

$$= \Delta x\{2mx_0 + 2m - 2y_0 + 2b - 1\}$$

$$= \Delta x\{2(mx_0 - y_0 + b - y_0) + (2m - 1)\}$$

$$= \Delta x\{2(y_0 - y_0) + (2m - 1)\} \quad [\because y_0 = mx_0 + b]$$

$P_0 = \Delta x\,(2m - 1)$

substituting m by $\Delta y / \Delta x$, we get,

$$P_0 = \Delta x \cdot \left(2\frac{\Delta y}{\Delta x} - 1\right)$$

$$= 2\Delta y - \Delta x$$

Therefore, $P_0 = 2\Delta y - \Delta x$.

Write Pseudo code for Bresenham's line algorithm.

1. Input two end points.
2. Plot (x_0, y_0)
3. Calculate Δx, Δy, $2\Delta y$, and evaluate

$$P_0 = 2\Delta y - \Delta x$$

4. at each x_k along the line, starting at $k = 0$,

if $P_k < 0$, next point to plot is $(x_k + 1, y_k)$ and

$$P_{k+1} = P_k + 2\Delta y$$

otherwise, next point to plot is $(x_k + 1, y_k + 1)$ and

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x.$$

5. Repeat step 4 for $(\Delta x - 1)$ times.

Using Bresenham's line-drawing algorithm, digitalize the line with endpoints (20, 10) and (30, 18).

Here, $\Delta y = 18 - 10 = 8$

$\Delta x = 30 - 20 = 10$

$2\Delta y = 2 \times 8 = 16$

$2\Delta y - 2\Delta x = (2 \times 8) - (2 \times 10) = -4$

first point is $(x_0, y_0) = (20, 10)$. Now,

$P_0 = 2\Delta y - \Delta x = 16 - 10 = 6$ $P > 0$, then, next point is $(21, 11)$

and $P_1 = P_0 + 2\Delta y - 2\Delta x = 6 - 4 = 2$

k	P_k	(x_{k+1}, y_{k+1})	P_{k+1}
0	6	(21, 11)	$P_1 = P_0 + (2\Delta y - 2\Delta x) = 6 - 4 = 2$
1	2	(22, 12)	$P_2 = 2 - 4 = -2$
2	-2	(23, 12)	$P_3 = P_2 + 2\Delta y = -2 + 16 = 14$
3	14	(24, 13)	$P_4 = 14 - 4 = 10$
4	10	(25, 14)	$P_5 = 10 - 4 = 6$
5	6	(26, 15)	$P_6 = 6 - 4 = 2$
6	2	(27, 16)	$P_7 = 2 - 4 = -2$
7	-2	(28, 16)	$P_8 = -2 + 16 = 14$
8	14	(29, 17)	$P_9 = 14 - 4 = 10$
9	10	(30, 18)	

(*) Specify the region where Bresenham's line algorithm works.

Bresenham's line algorithm works in the region $0 < m < 1$ where m stands for slope of the line.

(*) How can we apply Bresenham's line algorithm in a region out of $0 < m < 1$?

→ By mirroring them ~~into~~ either horizontally, vertically or diagonally into the $0 < m < 1$ or 0° to 45° angle range.

→ For diagonal mirroring, ~~exchange~~ interchange the roles of x and y directions.

(*) Is it possible to apply Bresenham's line algorithm when $-1 < m < 0$ (m stands for its traditional meaning)? How?

A line with values of m other than $0 < m < 1$ can be used by mirroring them either horizontally, vertically or diagonally.

~~Here a line with $-1 < m < 0$ has a horizontally mirrored counterpart. Let.~~

Let a line from (x_1', y_1') to (x_2', y_2') with $-1 < m < 0$ has a horizontally mirrored counterpart from $(x_1', -y_1')$ to $(x_2', -y_2')$ with $0 < m < 1$.

So, we can simply use the algorithm to scan convert this counterpart, but negate the y co-ordinate at the end of each iteration to set the right pixel for the line.

⊛ For line with slope in between 45° to 90°

We can obtain mirrored counterpart by exchanging the x and y co-ordinates of its counterpart endpoints. We can then scan-convent this counterpart but we must exchange x and y in the call to setPixel.

⊛ Apply Bresenham's line algorithm for $y = 1.2x + 30$.

We know, Bresenham's line algorithm works on $0 < m < 1$. Here $m = 1.2$. So, we got m between 45° to 90°.

Then, we need to swap roles of (x, y) if $m > 1$

⊛ Bresenham's Line Algorithm's Advantage:

- Fast incremental algorithm.
- Uses only integer calculations.

⊛ Comparison to DDA algorithm:

⇒ Though DDA algorithm is an incremental algorithm, rounding operations and floating point arithmetic involved are time consuming.

other hand, Bresenham's line algorithm is faster in these cases as it's calculation is simple.

⇒ Accumulation of round-off errors can make the pixelated line drift away from what was intended.

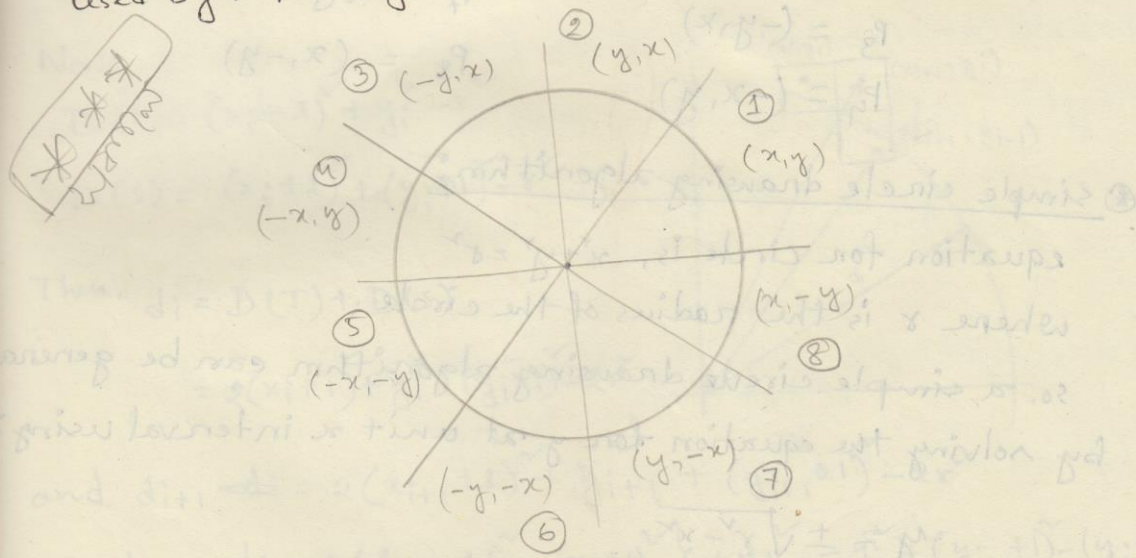
On the other hand, Bresenham uses only integer calculations and as round-off is not required, pixelated line drifts away less from what was intended.

3.3. Scan-converting a circle:

⑧ What is 8-way symmetry? Write short note on its application.

8-way symmetry:

8-way symmetry is the circle's symmetry to plot eight points for each value that the algorithm generates/calculates. It is used by reflecting each calculated point around each 45° axis.



☐ Application:

Let, value of (x, y) of the figure is calculated. Then, then, points at plot 2 is calculated by reversing x, y co-ordinates. Reversing x, y co-ordinates and reflecting about y-axis, we found point at plot 3. Reflecting about y-axis as in point of plot 4. Switching the signs of x and y as in point of plot 5, reversing x, y co-ordinates and reflecting about y-axis and reflecting about x-axis as in point of plot 6, reversing x, y co-ordinates and reflecting about y axis as in point of plot 7, reflecting about x-axis as in point of plot 8.

That is,

$P_1 = (x, y)$

$P_2 = (y, x)$

$P_3 = (-y, x)$

$P_4 = (-x, y)$

$P_5 = (-x, -y)$

$P_6 = (-y, -x)$

$P_7 = (y, -x)$

$P_8 = (x, -y)$

✱ Simple circle drawing algorithm:

equation for circle is, $x^2 + y^2 = r^2$

where r is the radius of the circle.

So, a simple circle drawing algorithm can be generated by solving the equation for y at unit x interval using:

$$y = \pm\sqrt{r^2 - x^2}$$

☐ Disadvantage:

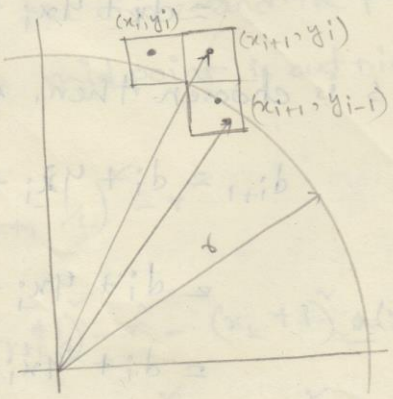
- large gaps where the slope approaches the vertical line.
- Calculations are not very efficient.
-

⊛ Bresenham's circle algorithm:

Scan-converted points of a circle is calculated with only integer addition, subtraction and multiplication by power of 2.

⊛ Derive Bresenham circle algorithm's formula:

Let, the current co-ordinate is (x_i, y_i). Then, our next point of choices are (x_{i+1}, y_i) or (x_{i+1}, y_{i-1})



Now,

$$D(T) = (x_i+1)^2 + y_i^2 - r^2$$

$$D(S) = (x_i+1)^2 + (y_i-1)^2 - r^2$$

Then, $d_i = D(T) + D(S)$

$$= 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$$

and d_{i+1} ~~$-d_i$~~ $= 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2$

so, $d_{i+1} - d_i = 2\{(x_{i+1}+1)^2 - (x_i+1)^2\} + y_{i+1}^2 - y_i^2 + (y_{i+1}-1)^2 - (y_i-1)^2$

Assuming $x_{i+1} = x_i + 1$, we get,

$$d_{i+1} - d_i = 2\{(x_i+2)^2 - (x_i+1)^2\} + y_{i+1}^2 - y_i^2 + (y_{i+1}-1)^2 - (y_i+1)^2$$

$$= 2\{x_i^2 + 4x_i + 4 - x_i^2 - 2x_i - 1\} + y_{i+1}^2 - y_i^2 + y_{i+1}^2 + 2y_{i+1} + 1 - y_i^2 - 2y_i - 1$$

$$= 4x_i + 6 + 2y_{i+1}^2 - 2y_i^2 + 2y_{i+1} - 2y_i$$

$$= 4x_i + 2(y_{i+1}^2 - y_i^2) + 2(y_{i+1} - y_i) + 6$$

or, $d_{i+1} = d_i + 4x_i + 2(y_{i+1}^2 - y_i^2) + 2(y_{i+1} - y_i) + 6$

Now, if T is choosen, then, $y_{i+1} = y_i$, so, $(d<0)$

$$d_{i+1} = d_i + 4x_i + 2(y_i^2 - y_i^2) + 2(y_i - y_i) + 6$$

$$= d_i + 4x_i + 6$$

if S is choosen then, $y_{i+1} = y_{i-1}$, then, $(d_i \geq 0)$

$$d_{i+1} = d_i + 4x_i + 2(y_{i-1}^2 - y_i^2) + 2(y_i - 1 - y_i) + 6$$

$$= d_i + 4x_i + 2(y_i^2 - 2y_i + 1 - y_i^2) + -2 + 6$$

$$= d_i + 4x_i + -4y_i + 2 - 2 + 6$$

$$= d_i + 4(x_i - y_i) + 10$$??

⊛ Midpoint circle algorithm:

Assume that we have current point is (x_k, y_k). Then, our choices for the next points are (x_k+1, y_k) and (x_k+1, y_k-1). The next point is choosen by testing the spatial relationship between an arbitrary point (x, y) and a circle of radius r centered at the origin as follows:

$$f(x,y) = x^2+y^2-r^2 = \begin{cases} <0, & \text{if } (x,y) \text{ is inside circle boundary} \\ =0 & \text{if } (x,y) \text{ is on circle boundary} \\ >0 & \text{if } (x,y) \text{ is outside circle boundary.} \end{cases}$$

Our decision variable can be defined as

$$P_k = f(x_k+1, y_k-\tfrac{1}{2})$$
$$= (x_k+1)^2 + (y_k-\tfrac{1}{2})^2 - r^2$$

If $P_k<0$, the midpoint is inside the circle and the pixel at y_k is closen to the circle. Otherwise, mid point is outside and y_k-1 is chosen.

Now, $P_{k+1} = (x_{k+1}+1)^2 + (y_{k+1}-\tfrac{1}{2})^2 - r^2$

$=$

Then. $P_{k+1} - P_k = ((x_k+1)+1)^2 + (y_{k+1}-\tfrac{1}{2})^2 - r^2 - (x_k+1)^2 - (y_k-\tfrac{1}{2})^2 + r^2$

$$= (x_k+1)^2 + 2(x_k+1) + 1 + (y_{k+1}-\tfrac{1}{2})^2 - (x_k+1)^2 - (y_k-\tfrac{1}{2})^2$$

$$= 2(x_k+1) + y_{k+1}^2 - y_{k+1} + \tfrac{1}{4} - y_k^2 + y_k - \tfrac{1}{4} + 1$$

$$= 2(x_k+1) + 1 + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)$$

Where y_{k+1} is either y_k or y_k-1 depending on sign of P_k.

Here, first decision variable is,

$$P_0 = f(1, r - \tfrac{1}{2})$$
$$= 1^2 + (r - \tfrac{1}{2})^2 - r^2$$
$$= 1 + r^2 - r + \tfrac{1}{4} - r^2$$
$$= \tfrac{5}{4} - r$$

If pixel T is choosen ($P_k < 0$) then, $y_{k+1} = y_k$, then,

$$P_{k+1} = P_k + 2(x_k + 1) + 1$$

If pixel S is choosen ($P_k \geq 0$) then, $y_{k+1} = y_k - 1$, so,

$$P_{k+1} = P_k + 2(x_k + 1) + 1 - 2(y_k - 1)$$

or,
$$P_{k+1} = \begin{cases} P_k + 2x_{k+1} + 1 & \text{if } P_k < 0 \\ P_k + 2(x_{k+1} - y_{k+1}) + 1 & \text{if } P_k \geq 0 \end{cases}$$

$$P_{k+1} = \begin{cases} P_k + 2x_k + 3 & ; \text{if } P_k < 0 \\ P_k + 2(x_{k+1} - y_{k+1}) + 1 & ; \text{if } P_k \geq 0 \end{cases}$$

⊛ Algorithm:

1. Input radius r and circle center (x_c, y_c), set first poi
$(x_0, y_0) = (0, r)$.
2. Calculate initial value of decision parameter, $P_0 = 1 - r$
$(\tfrac{5}{4} - r \cong 1 - r)$

3. If $P_k < 0$,

plot (x_k+1, y_k) and $P_{k+1} = P_k + 2x_{k+1} + 1$

otherwise,

plot (x_k+1, y_k-1) and $P_{k+1} = P_k + 2(x_{k+1} - y_{k+1}) + 1$

4. Determine symmetric point on other seven octants.

5. Move each calculated pixel position (x, y) on to the circular path centered on (x_c, y_c) and plot co-ordinate values : $x = x + x_c$, $y = y + y_c$

6. Repeat step 3 through 5 until $x \geq y$.

7. For all points, add center point (x_c, y_c)

(*) Given circle radius $r = 10$, demonstrate mid point circle algorithm by determining positions along the circle octant in first quadrant from $x = 0$ to $x = y$.

$P_0 = 1 - 10 = -9$

and initial point $(x_0, y_0) = (0, 10)$. Now,

k	P_k	P_{k+1}	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	-9	$-9+2+1 = -6$	(1, 10)	2	20
1	-6	$-6+4+1 = -1$	(2, 10)	4	20
2	-1	$-1+6+1 = 6$	(3, 10)	6	20
3	6	$6+(8-18)+1 = -3$	(4, 9)	8	18
4	-3	$-3+10+1 = 8$	(5, 9)	10	18
5	8	$8+(12-16)+1$	(6, 8)	12	16
6	5	—	(7, 7)	14	14

⊛ Given a circle radius $r = 15$, demonstrate midpoint circle algo. by determining position along the circle octant in first quadrant from $x = 0$ to ~~$y =$~~ $x = y$.

$P_0 = 1 - 15 = -14$

and initial point $(x_0, y_0) = (0, 15)$, then.

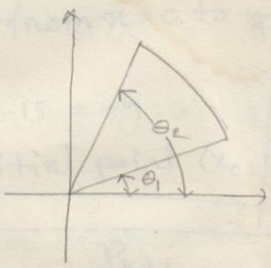
k	P_k	P_{k+1}	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	−14	$-14+2+1 = -11$	(1,15)	2	30
1	−11	$-11+4+1 = -6$	(2,15)	4	30
2	−6	$-6+6+1 = 1$	(3,15)	6	30
3	1	$1+(8-28)+1 = -18$	(4,14)	8	28
4	−18	$-18+10+1 = -7$	(5,14)	10	28
5	−7	$-7+12+1 = 6$	(6,14)	12	28
6	6	$6+(14-26)+1 = -5$	(7,13)	14	26
7	−5	$-5+16+1 = 12$	(8,13)	16	26
8	12	$12+(18-24)+1 = 7$	(9,12)	18	24
9	7	$7+(20-22)+1 = 6$	(10,11)	20	22
10	6	-	(11,10)	22	20

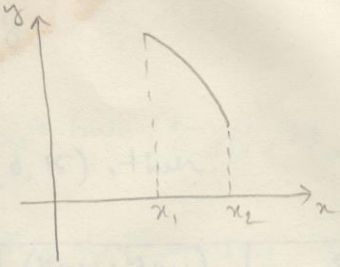
* Midpoint Ellipse Algorithm:

In case of ellipse, we have 4-way rather than 8-way symmetry.

3.5 Scan-converting arcs and sectors:

⊛ Arcs:





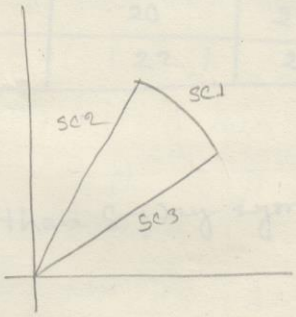
An arc may be generated by using either polynomial or trigonometric method.

In trigonometric method, we set initial value to θ_1 & ending value to θ_2. The rest of the steps are similar to those used when scan converting a circle, but symmetry is not used.

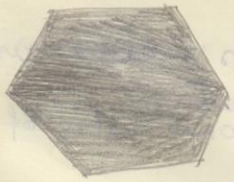
In polynomial method, x varies from x_1 to x_2 and y varies as $\sqrt{r^2 - x^2}$.

⊛ sector:

A sector is scan converted by scan converting an arc, then scan converting two lines from center to the endpoints of arc.



3.7 Region filling:

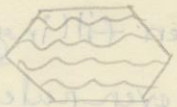


⊛ Polygon filling:

☐ Types of filling:

☒ Solid fill:
All the pixels inside the polygon's boundary are illuminated.

☒ Pattern fill:
Polygon is filled with an arbitrary predefined pattern.



⊛ Polygon representation:

· Polygon can be represented by listing its n vertices in an ordered list. $P = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\ldots (x_n, y_n)\}$

Polygon can be displayed by joining two consecutive points through out the ordered list and finally joining (x_n, y_n) and (x_1, y_1) to form the polygon.

☐ Limitation:

→ to translate the polygon, we need to apply translation transformation to each vertex in order to obtain translated polygon.

for objects described by many polygon with many vertices, this can be a time consuming process.

☐ Solution:

To reduce computational time, we can represent the polygon by the (absolute) location of its first vertex, and subsequent vertices as relative positions from previous vertex.

This enables us to translate polygon simply by changing co-ordinates of the first vertex.

⑧ Inside - outside Test:

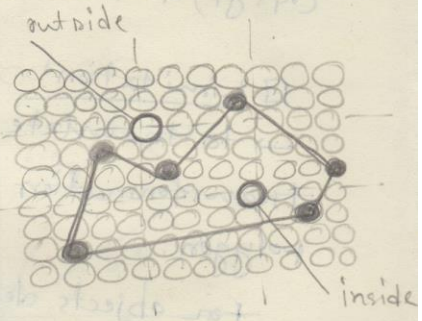
⑧ How can we determine whether or not a given point is inside the polygon using Even-odd method? Explain with example.

When filling polygons, a rule called odd-parity (or the odd-even rule) is applied to test whether a point is p interior or exterior to a polygon.

To apply this rule, we conceptually draw a line starting from the particular point and extending to a distance point outside the co-ordinate extends of the object in any direction such that no polygon vertex intersects with the line.

Now, the point is considered to be interior if the number of intersections between the line and the polygon edges is odd.

Otherwise, point is exterior point.



⑧ Scan line polygon filling algorithm:

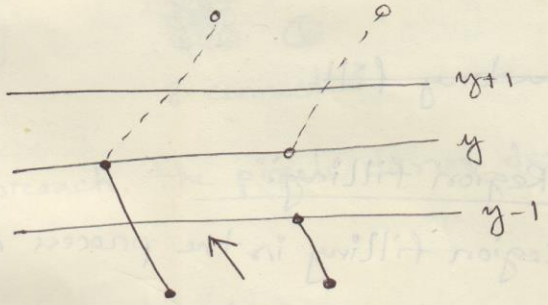
Scan-line polygon-filling algorithm involves -

- horizontal scanning of the polygon from its lowermost to its topmost vertex.

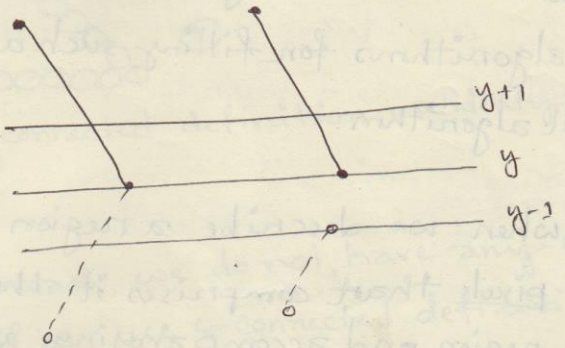
- identifying which edges intersects the scan-line,
- and finally, drawing the ~~intersection~~ interior horizontal lines with the special fill color process.

☐ Dealing with vertices:

- When the endpoints y co-ordinates of the two edges are increasing, the y value of the upper endpoint for the current edge is decreased by one.



and when endpoint y values are decreasing, the y value of the next edge is decreased by one.

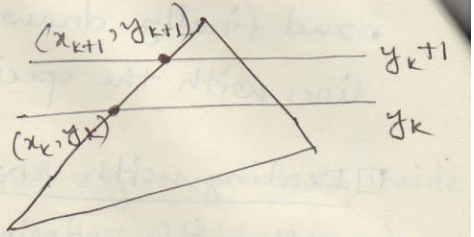


* Determining Edge intersections:

$$m = (y_{k+1} - y_k)/(x_{k+1} - x_k)$$

Then, $y_{k+1} - y_k = 1$

and $x_{k+1} - x_k = 1/m$



—o—

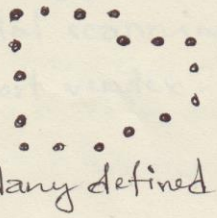
~~* Boundary fill~~

3.7 Region filling:

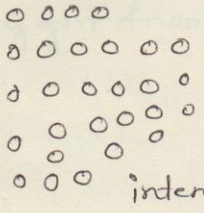
⇒ Region filling is the process of "coloring in" a definite image area or region.

⇒ When we describe a region ~~it~~ in terms of bounding pixels that outline, the region is called boundary defined and collection of algorithms for filling such a region are called boundary fill algorithm.

⇒ when we describe a region ~~in terms of~~ as the totality of pixels thart comprises it then it is called interior-defined region and accompanying algorithm are flood-fill algorithms.



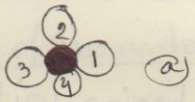
boundary defined regim



interior-defined region.

* Distinguish between 4-connected and 8-connected pixel and state their applications.

→ In 4-connected method, a pixel may have upto four neighbours where 8-connected method, defines that a pixel may have upto 8-neighbours.



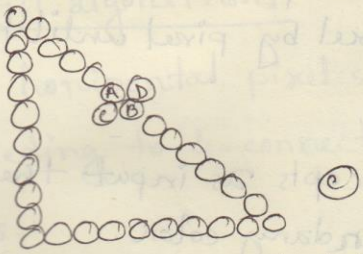
(a)

4-connected

(b)

8-connected.

→ Using 4-connected approach, the pixels do not define a region for such case -

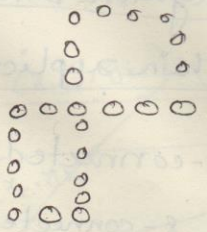


(c)

but however using 8-connected definition we identy a triang -ular region.

→ Using 8-connected approach, we do not have any enclosed region in fig(c) but if we use 4-connected definition we have a triangular region for this case.

→ 4-connected approach cannot be effective in region like this. but 8-connected can.



⊛ application:

⊛ Boundary fill algorithm:

A recursive algorithm that begins with a starting pixel within the region, and paint the interior outward towards the boundary.

If the boundary is specified in a single color, fill algorithm processed outward pixel by pixel until the boundary color is encountered.

This procedure accepts as input the co-ordinate of the seed a fill color and boundary color.

□ Algorithm:

(1) start from an interior point.

(2) If the pu current pixel is not already filled and if it is not an edge point, then, set the pixel with fill color and store the neighbouring pixels in the stack for processing store only neighbouring pixels that are not already ~~colored~~ filled and is not an edge point.

(3) select the next pixel from stack and continue with step 2.

⊗ Order of pixels:

☐ 4-connected: above, below, left and right

☐ 8-connected: above, below, left, right, above left, above-right, below-left and below-right.

⊛ Span flood fill algorithm:

☐ Limitation of boundary fill:

→ Boundary has to a single colour

→ requires considerable stacking of neighbouring pixels.

→ Cannot define properly regions like these.

☐ Span flood-fill algorithm:

Process fills horizontal pixel spans across scanlines, instead of proceeding to 4-connected or 8-connected neighbouring pixels.

☒ Advantage:

→ only begining position for each horizontal pixel spans, is stacked instead of stacking all of unprocessed neighbouring positions around the current position.

[K] Algorithm:

① Starting from the ~~interior~~ initial interior pixel, fill th contiguous span of pixels on this starting scan line.

② locate and stack starting positions for spans on the adjucent scan lines, where spans are defined as the contiguous horizontal string of positions bounded by pixels displayed in the area borden color.

③ At ~~the~~ each subsequent step, unstack the next start position and repeat the process.

[*] onden of pixel : below, above.

(*) flood-fill algorithm:

(*) Describe flood-fill algorithm. Also write down its applications

□ flood-fill algorithm:

When we are ~~try~~ trying to fill in (recolon) an area that is not defined within a single colon boundary, we paint such areas by replacing a specified interion colon instead of searching for a boundary colon values.

This approach is called flood-fill algorithm.

☐ Procedure:

We start at a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.

If the area has more than one interior color, we can first reassign pixel values so that all interior pixel have the same color.

Using either 4-connected or 8-connected approach, we then step through pixel positions until all interior pixels have been repainted.

☐ application:

3.8 Scan-converting a character:

⊛ Bitmap font:

- Character is represented by the on pixels in the a bilevel pixel grid pattern called a bitmap.
- Characters are already in scan converted form.
- We may overlay the bit map onto itself with a horizontal offset of one pixel to produce bold and shift rows of pixel to produce italic.

Thus variation in appearance and size from one font, the overall results tends to be less than satisfactory.

- Size of a bitmap font is dependent on image resolution.

□ outline font:

⇒ graphical primitives such as lines and arcs are used to define the outline of each character.

⇒ Requires scan-conversion operations.

⇒ It can be used to produce characters of varying size, appearance and orientation.

⇒ It can be resized by scaling transformation, made into italic through a shearing transformation and turned around with respect to a reference point through a rotation transformation.

3.9 Anti-aliasing:

⊛ Alising:

Various forms of distortion that result from scan-conversion are collectively referred to as the alising effects of scan conversion.

⊛ Effect of alising:

① staincase/jagged effect in raster graphics:
② unequal Brightness/poor representation of fine detail:
③ Picket fence problem

① Staircase:

when scan converting a primitive such as a line on a circle, we observed a staircase on jagged appearance. This is the staircase effect of alising. Stair steps on "jaggies" along the borden of filled regim is also observed.

② Unequal Brightness:

A slanted line appears dimmer than a horizontal on ventical line, although all are presented at the same intensity level.

This is because in horizontal on ventical line pixels are less distant than in diagonal line.

③ Pickel-fence Problem:

Anti-aliasing:

The techniques that can greatly reduce the aliasing artifacts and improve the appearance of images without increasing their resolution are collectively called anti-aliasing techniques.

Pre-filtering techniques:

Pre-filtering techniques works on the true signal in the continuous space to derive proper values for individual pixels (filter before sampling)

Post-filtering techniques:

Post-filtering technique takes discrete samples of continuous signal and uses the sample to compute pixel values (sampling before filtering).

Un-weighted filtering:

When the amount of coverage given to each pixel is determined irrespective of the distance from the center of the line, is called unweighted filtering.

The filter used to calculate the % coverage of each pixel and used to shade that pixel.

- Unweighted filtering is computationally simple, but better results can be achieved by using a weighted filter.

⊛ Weighted filtering:

When amount of covenage given to each pixel is determined with respect to distance from the center of the line is called weighted filtering.

- It gives a more accurate result by taking into account the pixels around the one currently being investigated.
- Use of a cone means that prionity is given to line covenage closest to the center of pixel in question.
- Possible to give different prionities to the outlying pixels by adjusting the height.

⊛ Area-Sampling:

Area-sampling is a pre-filtering technique in which we superimpose a pixel grid pattern onto the continuous object definition.

For each pixel area that intersects the object, we calculate the pententage of overlap by the object.

⊛ Super sampling : (nx antialiacing) (post-filtering)

Greater accuracy and more graphically impressive results can be obtained by a technique known as supersampling.

Here we subdivide each pixel into subpixels and check the position of each subpixel in relation to the object to be scan-converted. Object's contribution to a pixel's overall intensity value is proportional to the number of subpixels that are inside the area occupied by the object.

⇒ It performs calculations on a virtual image "n" times the resolution of the desired output.

⇒ This produces several pixels for each pixel in final output. Then filtering is applied to each of these virtual pixels and the resultant average value is used to shade the corresponding pixel in the output image.

→ Though it is expensive, modern ve video games strive for smooth, realistic edge and make extensive use of this technique.

→ Many games provide options to user to ~~choose~~ select level of supersampling to balance desired graphics level with computing power level.

→ Many high-end video cards provide on-board functionali-ty to speed the process up.

Low-pass filter:

A post-filtering technique in which we reassign each pixel a new value that is a weighted average of its original value and the original value of its neighbours.

Box filter:

Lowpass filter with equal weights and computes neighbourhood averaging is Box filter.

Gaussian filter:

A filter with its weight values conforming to a two-dimensional Gaussian distribution is called Gaussian filter.

Pixel-Phasing: