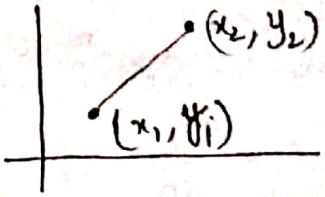
Unit-I
2D-Primitives

Output Primitives:

① Line-Drawing Algorithm:

Cartesian slope-intercept equation for a straight line.



$$y = mx + b \quad \text{—①}$$

$$\text{slope } m = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{—②}$$

$$y\text{-intercept } b = y_1 - mx_1 \quad \text{—③}$$

DDA (Digital Differential Analyzer) Algorithm:

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

[∵ m = slope b/w starting Point and end-Point]

Case-1: [∵ x ↑↑, y = constant] $m < 1$

x — unit intervals

$$x_{k+1} = x_k + 1$$

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

$$m = \frac{y_{k+1} - y_k}{x_k + 1 - x_k}$$

$$m = y_{k+1} - y_k$$

$$\boxed{y_{k+1} = y_k + m}$$

Case-2: $m > 1$ [∵ y ↑↑, x constant]

y - unit intervals

$$y_{k+1} = y_k + 1$$

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

$$m = \frac{y_k + 1 - y_k}{x_{k+1} - x_k}$$

$$x_{k+1} - x_k = \frac{1}{m}$$

$$\boxed{x_{k+1} = x_k + \frac{1}{m}}$$

Case-3: $m = 1$

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = y_k + 1$$

x, y — Unit intervals

Ex:

(10, 6) (15, 9)

slope $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{9-6}{15-10} = \frac{3}{5} = 0.6$

$m = 0.6 < 1$

x_k	y_k	(x_{k+1}, y_{k+1})
10	6	(10, 6)
10+1 = 11	6+0.6 = 6.6	(11, 7)
11+1 = 12	6.6+0.6 = 7.2	(12, 7)
12+1 = 13	7.2+0.6 = 7.8	(13, 8)
13+1 = 14	7.8+0.6 = 8.4	(14, 8)
14+1 = 15	8.4+0.6 = 9.0	(15, 9)

$m < 1$:

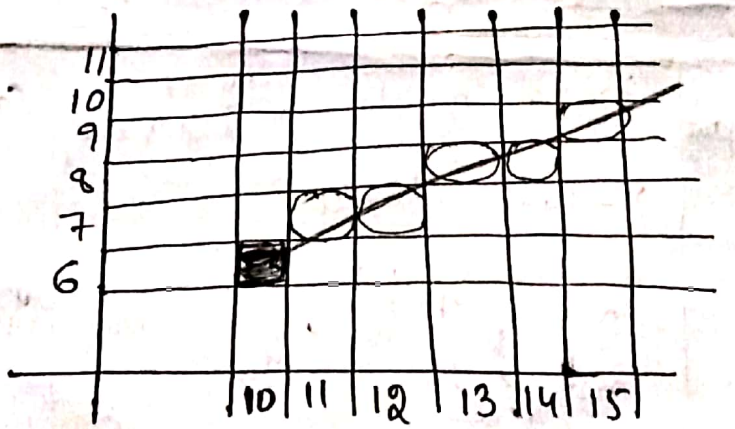
$(x_{k+1}, \text{round}(m + y_k))$

$m > 1$:

$(\text{round}(x_k + \frac{1}{m}), y_{k+1})$

[∵ Round operation rule.

1. when value > 0.5 Apply ceil func
2. when value < 0.5 Apply floor func]



Draw Back:

* Rounding operation

```
Algorithm DDA (x1, y1, x2, y2)
{
  dx = x2 - x1 ;
  dy = y2 - y1 ;                    [∵ abs = Absolute]
  if (abs(dx) > abs(dy))
    step = abs(dx)
  else
    step = abs(dy)
  xinc = dx/step
  yinc = dy/step
  for (i=1; i<step; i++)
  {
    Putpixel (x1, y1);
    x1 = x1 + xinc;
    y1 = y1 + yinc;
  }
}
```

Bresenham's Line Drawing Algorithm:

```
Algorithm Bresenham (x1, y1, x2, y2)
{
  x = x1;
  y = y1;
  dx = x2 - x1 ; dy = y2 - y1;
  P = 2dx - dy
  while (x <= x2)
  {
    Putpixel (x, y);
    x++;
    if (P<0)
    {
```

```
P = P+2Δy;
}
else
{
 P = P+2Δy-2Δx;
 y++;
}
}
}
```

Ex: End points $(20,10)$ and $(30,18)$ (x_1, y_1 and x_2, y_2)

slope $m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{18-10}{30-20} = \frac{8}{10} = 0.8 < 1$

$P_0 = 2\Delta y - \Delta x = 2\times 8 - 10 = 16 - 10 = 6$

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

$P_K < 0 \Rightarrow (x_K+1, y_K)$

$P_{K+1} = P_K + 2\Delta y$

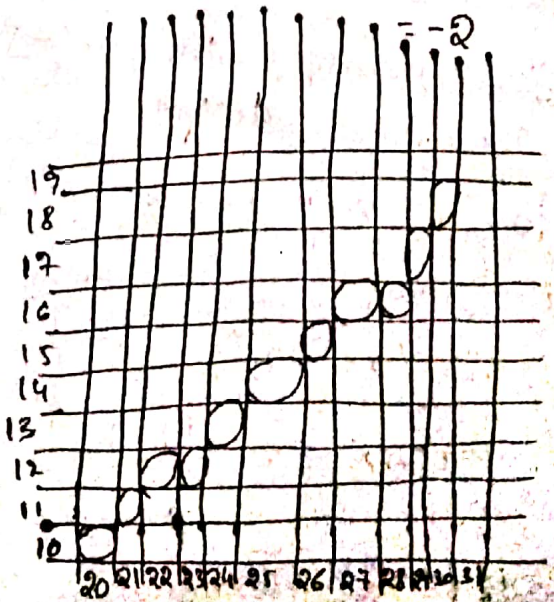
$P_K \geq 0 \Rightarrow (x_K+1, y_K+1)$

$P_{K+1} = P_K + 2\Delta y - 2\Delta x$

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

k=0,
$P_K = P_0 = 6 > 0$
(x_{K+1}, y_K+1)
$= (20+1, 10+1)$
$= (21, 11)$
$P_{K+1} = P_{0+1} = P_1$
$P_1 = P_K + 2\Delta y - 2\Delta x$
$= P_0 + 2\Delta y - 2\Delta x$
$= 6 + 16 - 20$
$= 2$

k=1, $P_K = P_1 = 2 > 0$
$(x_K + 1, y_K + 1)$
$= (22, 12)$
$P_{K+1} = P_{1+1} = P_2 = P_K + 2\Delta y - 2\Delta x$
$= 2 + 16 - 20$
$= -2$

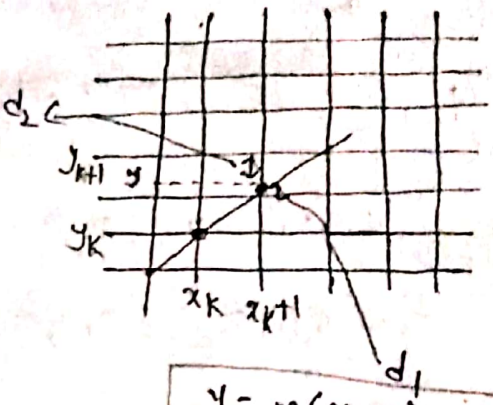


Note;

Δx times = 10 times process should repeat.

Bresenham's Line Algorithm: $m < 1$

An accurate and efficient raster-line generating algorithm, developed by Bresenhams.



$$y = mx + b$$

$$\boxed{y = m(x_k + 1) + b} \quad \text{—— (1)}$$

decision parameter $= \Delta x\,(d_1 - d_2)$

$$d_1 = y - y_k$$

$$d_1 = m(x_k + 1) + b - y_k$$

$$d_2 = y_{k+1} - y$$

$$d_2 = y_{k+1} - m(x_k + 1) - b$$

$$d_1 - d_2 = m(x_k + 1) + b - y_k - y_k - 1 + m(x_k + 1) + b$$

$$\boxed{d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1} \quad \text{— (2)} \qquad \left[\because m = \frac{\Delta y}{\Delta x}\right]$$

$$\Delta x\,(d_1 - d_2) = \Delta x\left[2 \cdot \frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1\right]$$

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

$$\boxed{P_k = 2\Delta y\, x_k + 2\Delta y - 2\Delta x y_k + \Delta x(2b - 1)} \quad \text{— (3)}$$

$$P_{k+1} = 2\Delta y\, x_{k+1} + 2\Delta y - 2\Delta y_{k+1} + \Delta x(2b - 1)$$

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

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

~~$P_{k+1} = P_k + 2\Delta y$~~ [$\because x_{k+1} = x_k + 1$ // circle - A circle at ...]

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

$$\boxed{P_{k+1} = P_k + 2\Delta y - 2\Delta x (y_{k+1} - y_k)} \quad -(4)$$

Initial (x_k, y_k) [$\because$ from $y = mx + b$, $b = y - mx$]

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

$$P_k = 2\Delta y x_k + 2\Delta y - 2\Delta x y_k + \Delta x (2(y - m(x)) - 1) \quad [\because m = \frac{\Delta y}{\Delta x}]$$

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

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

Replace y with y_k because initial point is y_k

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

$$\boxed{P_0 = 2\Delta y - \Delta x}$$

$m < 1$

If $P_k \geq 0$ then	$P_k < 0$
$x_{k+1} = x_k + 1$	$x_{k+1} = x_k + 1$
$y_{k+1} = y_{k+1}$	$y_{k+1} = y_k$
Next Point $= (x_{k+1}, y_{k+1})$	Next Point $= (x_{k+1}, y_k)$

$m \geq 1$

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

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

$P_k \geq 0 \Rightarrow x_{k+1} = x_k + 1$	$P_k < 0 \Rightarrow x_{k+1} = x_k$
$NC = (x_{k+1}, y_k + 1)$	$NC = (x_k, y_k + 1)$

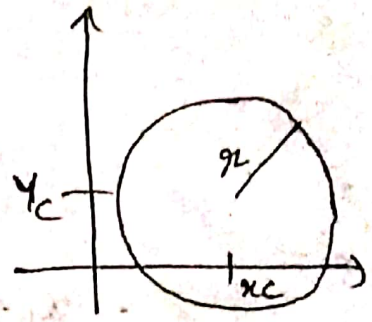
$(x^2 - 1+x^2)$

Circle-Generating Algorithm:

A circle is defined as the set of points that are all at a given distance 'r' from a center position (x_c, y_c)

This distance relationship is expressed by the Pythagorean theorem in cartesian coordinate as

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \quad \text{—(1)}$$

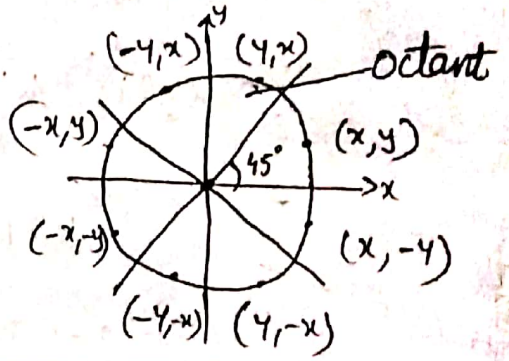


Position of Points on circle: $y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$

Mid Point

~~Bresenhams~~ circle drawing Algorithm:

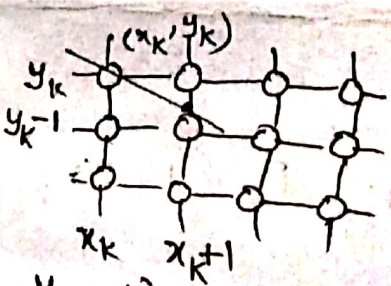
* Circle has 8-Symmetry Property



Function of a circle

$$f_{circle}(x,y) = x^2 + y^2 - r^2$$

$$f_{circle}(x,y) \begin{cases} <0, & (x,y) \text{ is inside circle boundary} \\ =0, & (x,y) \text{ on circle boundary} \\ >0, & (x,y) \text{ is outside circle boundary} \end{cases}$$



Mid Point of (x_k+1, y_k) and (x_k+1, y_k-1)

$$= \left(\frac{x_k+1 + x_k+1}{2}, \frac{y_k + y_k - 1}{2}\right) \quad \left[\because \frac{x+x_2}{2}, \frac{y_1+y_2}{2}\right]$$

$$= \left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$P_K = f_{circle}\left(x_k+1, y_k - \frac{1}{2}\right) = \left(x_k+1\right)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \quad \text{—(1)}$$

$P_k < 0 \Rightarrow f_{circle}\left(x_k+1, y_k - \frac{1}{2}\right) < 0 \Rightarrow$ mid Point inside circle boundary

$$(x_{k+1}, y_{k+1}) = (x_k+1, y_k)$$

$P_k > 0 \Rightarrow f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right) > 0 \Rightarrow$ mid point outside circle

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

Successive decision parameter [∵ Replace k with k+1]

$$P_{k+1} = (x_{k+1} + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2 \quad \text{—②}$$

② − ①

$$P_{k+1} - P_k = (x_{k+1} + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2 - \left[(x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2\right]$$

$$= (x_{k+1} + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2 - (x_k + 1)^2 - \left(y_k - \frac{1}{2}\right)^2 + r^2$$

$$= (x_k + 1)^2 + 1 + 2x_{k+1} + \left(y_{k+1} - \frac{1}{2}\right)^2 - (x_k + 1)^2 - \left(y_k - \frac{1}{2}\right)^2$$

$$= 1 + 2x_{k+1} + \left[y_{k+1}^2 + \frac{1}{4} - y_{k+1}\right] - \left[y_k^2 + \frac{1}{4} - y_k\right]$$

$$= 1 + 2x_{k+1} + y_{k+1}^2 + \frac{1}{4} - y_{k+1} - y_k^2 - \frac{1}{4} + y_k$$

$$= 1 + 2x_{k+1} + \left(y_{k+1}^2 - y_{k+1}\right) - \left(y_k^2 - y_k\right)$$

$$\boxed{P_{k+1} = P_k + 1 + 2x_{k+1} + \left(y_{k+1}^2 - y_{k+1}\right) - \left(y_k^2 - y_k\right)}$$

Initial decision parameter

$$P_0 = f_{circle}(0, r)\left(x_k + 1, y_k - \frac{1}{2}\right) \quad [\because (x_k, y_k) = (0, r)]$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

$$= (0 + 1)^2 + \left(r - \frac{1}{2}\right)^2 - r^2$$

$$= 1 + r^2 + \frac{1}{4} - r^2 - r = \frac{5}{4} - r$$

Apply round to P_0

$$\therefore \boxed{P_0 = 1 - r}$$

$P_{K+1} = P_K + 1 + 2(x_K + 1) + (y_{K+1}^2 - y_K^2) - (y_{K+1} - y_K)$

$P_K < 0 \Rightarrow y_{K+1} = y_K \quad [\because y_{K+1} = y_K]$

$\Rightarrow P_{K+1} = P_K + 1 + 2(x_K + 1) + (y_K^2 - y_K^2) - (y_K - y_K)$

$= P_K + 1 + 2(x_K + 1)$

$P_K > 0 \Rightarrow y_{K+1} = y_K - 1$

$P_{K+1} = P_K + 1 + 2(x_K + 1) + ((y_K - 1)^2 - y_K^2) - (y_K - 1 - y_K)$

$= P_K + 1 + 2(x_K + 1) + (y_{K+1}^2 - 2y_K + y_K^2) + 1$

$= P_K + 1 + 2(x_K + 1) + 1 - 2y_K + 1$

$= P_K + 1 + 2(x_K + 1) - 2y_K + 2$

$= P_K + 1 + 2(x_K + 1) - 2(y_K - 1) \quad [\because y_K - 1 = y_{K+1}]$

$P_{K+1} = P_K + 1 + 2(x_K + 1) - 2y_{K+1}$

Ex: $r = 10$ units, center $(0,0)$

$P_0 = 1 - r = 1 - 10 = -9 < 0$

K	P_K	(x_{K+1}, y_{K+1})
0	-9	$(x_K, y_K) = (0, r) = (0, 10)$
		$(x_{K+1}, y_{K+1}) = (x_K + 1, y_K)$
		$= (1, 10) \leftarrow (x_K, y_K)$
		$P_{K+1} = P_{0+1} = P_1 = P_K + 1 + 2(x_K + 1)$
		$= P_0 + 1 + 2(x_K + 1)$
		$= -9 + 1 + 2(0 + 1)$
		$= -9 + 3 = -6$
1	-6	(2, 10)
2	-1	(3, 10)
3	6	(4, 9)

$P_0 = 1 - r$ in integer

$\frac{5}{4} - r$ in fraction

$P_K = (x_K + 1)^2 + (y_K - \frac{1}{2})^2 - r^2$

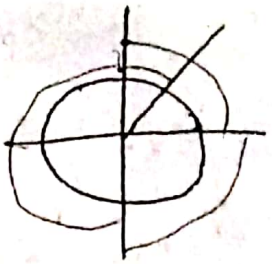
when $P_K < 0$ $(x_K + 1, y_K)$

$\Rightarrow P_{K+1} = P_K + 1 + 2(x_K + 1)$

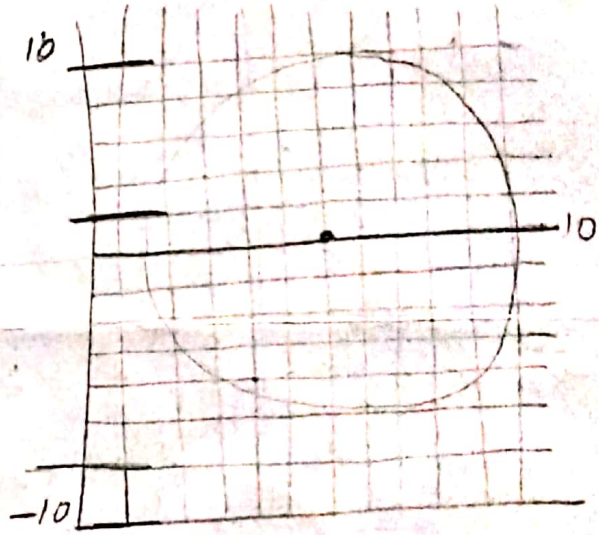
when $P_K > 0$ $(x_K + 1, y_K - 1)$

$\Rightarrow P_{K+1} = P_K + 1 + 2(x_K + 1) - 2y_{K+1}$

4	-3	(5,9)
5	8	(6,8)
6	5	(7,7)
7		(8,6)
8		(9,5)
9		(9,4)
10		(10,3)
11		(10,2)
12		(10,1)



continue this process untill $x \geq y$



Q2	Q3	Q4
(-1,10)	(-1,-10)	(1,-10)
(-2,10)	(-2,-10)	(2,-10)
(-3,10)	(-3,-10)	(3,-10)
(-4,9)	(-4,-9)	(4,-9)
(-5,9)	(-5,-9)	(5,-9)
(-6,8)	(-6,-8)	(6,-8)
(-7,7)	(-7,-7)	(7,-7)
(-8,6)	(-8,-6)	(8,-6)
(-9,5)	(-9,-5)	(9,-5)
(-9,4)	(-9,-4)	(9,-4)
(-10,3)	(-10,-3)	(10,-3)
(-10,2)	(-10,-2)	(10,-2)
(-10,1)	(-10,-1)	(10,-1)

Point circle Algorithm:

Input radius r and circle center (x_c, y_c) and obtain the first point on the circumference of a circle centered on the origin as $(x_0, y_0) = (0, r)$

2. Calculate the initial value of the decision parameter as $P_0 = \frac{5}{4} - r$

3. At each x_k position, starting at $k=0$, perform the following test: if $P_k < 0$, the next point along the circle centered on $(0,0)$ is (x_{k+1}, y_k) and

$$P_{k+1} = P_k + 2x_{k+1} + 1$$

Otherwise, the next point along the circle is $(x_{k+1}, y_k - 1)$ and $P_{k+1} = P_k + 2x_{k+1} + 1 - 2y_{k+1}$

where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$

4. Determine symmetry points in the other seven octants

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

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

Ellipse-Generating Algorithm:

Ellipse is defined as the set of points such that the sum of the distances from two fixed ~~Points~~ positions (foci) is the same for all points.

Point $P = (x, y)$

two distances d_1, d_2

$d_1 + d_2 = \text{constant}$

Express distances d_1, d_2 in terms of focal coordinates

$F_1 = (x_1, y_1)$ & $F_2 = (x_2, y_2)$

$$\sqrt{(x-x_1)^2+(y-y_1)^2} + \sqrt{(x-x_2)^2+(y-y_2)^2} = \text{constant}$$

General Ellipse eqn $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$

Function of ellipse $\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1$ center coordinate $= \left(\frac{x-x_c}{r_x}\right)^2 + \left(\frac{y-y_c}{r_y}\right)^2 = 1$

$f_{ellipse}(x,y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$

Anticlock wise ← (0, r_y), ($-x$, y), R_1, (x, y), r_y, R_2, slope = −1, (r_x), (0,0), ($-x$, $-y$), (x, $-y$) → clock wise

$f_{ellipse} \begin{cases} <0, & \text{if } (x,y) \text{ is inside the ellipse boundary} \\ =0, & \text{if } (x,y) \text{ is on the ellipse boundary} \\ >0, & \text{if } (x,y) \text{ is outside the ellipse boundary} \end{cases}$

slope is $m = \frac{dy}{dx} = \frac{-2r_y^2 x^2}{2r_x^2 y^2}$ [$\because$ slope = −1]

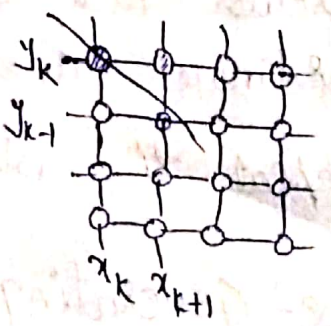
$= \frac{-2r_y^2 x^2}{2r_x^2 y^2} = -1$

$\Rightarrow 2r_y^2 x^2 = 2r_x^2 y^2$

↳ Moving $R_1 \rightarrow R_2$ $(2r_y^2 x^2 \geq 2r_x^2 y^2)$

Region 1: Present Pixel = (x_k, y_k)

choice next Pixel = (x_{k+1}, y_k) ① or (x_{k+1}, y_{k-1}) ②



Midpoint of ① & ② Pixels

$\Rightarrow \left(\frac{x_{k+1} + x_{k+1}}{2}, \frac{y_k + y_{k-1}}{2}\right)$

$\Rightarrow \left(x_{k+1}, y_k - \frac{1}{2}\right)$

Ordinary

$$f_{ellipse}(x,y)\left(x_K+1, y_K-\frac{1}{2}\right) = r_y^2\left(x_K+1\right)^2 + r_x^2\left(y_K-\frac{1}{2}\right)^2 - r_x^2 r_y^2$$

① $P_K^1 < 0 \Rightarrow (x_{K+1}, y_{K+1}) = (x_K+1, y_K)$

② $P_K^1 > 0 \Rightarrow (x_{K+1}, y_{K+1}) = (x_K+1, y_K-1)$

Substitute K with K+1

$$P_{K+1}^1 = r_y^2\left(x_{K+1}+1\right)^2 + r_x^2\left(y_{K+1}-1/2\right)^2 - r_x^2 r_y^2$$

$$\Rightarrow P_{K+1}^1 - P_K^1 = r_y^2\left[(x_{K+1}+1)^2 - (x_K+1)^2\right] + r_x^2\left[(y_{K+1}-1/2)^2 - \left(y_K-\frac{1}{2}\right)^2\right]$$

$$= r_y^2\left[((x_K+1)+1)^2 - (x_K+1)^2\right] + r_x^2\left[\left(y_{K+1}-\frac{1}{2}\right)^2 - \left(y_K-\frac{1}{2}\right)^2\right]$$

$$= r_y^2\left[(x_K+1)^2 + 1 + 2(x_K+1) - (x_K+1)^2\right] + r_x^2\left[\left(y_{K+1}-\frac{1}{2}\right)^2 - \left(y_K-\frac{1}{2}\right)^2\right]$$

Region 1: $P_{K+1}^1 - P_K^1 = r_y^2\left[1+2(x_K+1)\right] + r_x^2\left[y_{K+1}^2 + \frac{1}{4} - y_{K+1} - y_K^2 - \frac{1}{4} + y_K\right]$

$$\boxed{P_{K+1}^1 = P_K^1 + r_y^2\left(1+2(x_K+1)\right) + r_x^2\left(y_{K+1}^2 - y_K^2 - (y_{K+1}-y_K)\right)}$$

Initial Point $(0, r_y)$

$= f_{ellipse}\left(x_{K+1}, y_K-\frac{1}{2}\right)$ $\quad [\because x_K = 0, y_K = r_y]$

$= f_{ellipse}\left(1, r_y-\frac{1}{2}\right)$

$$\boxed{P_0^1 = r_y^2 + r_x^2\left(r_y-1/2\right)^2 - r_x^2 r_y^2}$$

$P_K^1 < 0 \Rightarrow y_{K+1} = y_K$

$$P_{K+1}^1 = P_K^1 + r_y^2\left(1+2(x_K+1)\right) + r_x^2\left(y_K^2 - y_K^2 - (y_K - y_K)\right)$$

$$\Rightarrow \boxed{P_{K+1}^1 = P_K^1 + r_y^2\left(1+2(x_K+1)\right)}$$

$P^1_k > 0 \Rightarrow y_{k+1} = y_k - 1$

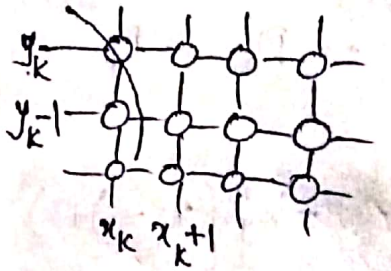
$$P^1_{k+1} = P^1_k + r_y^2\left(1 + 2(x_k+1)\right) + r_x^2\left[(y_k-1)^2 - y_k^2 - (y_k-1)\right]$$

$$= P^1_k + r_y^2\left(1 + 2(x_k+1)\right) + r_x^2\left[y_k^2 + 1 - 2y_k - y_k^2 - (y_k - 1 - y_k)\right]$$

$$\boxed{P^1_{k+1} = P^1_k + r_y^2\left(1 + 2(x_k+1)\right) + 2r_x^2(1 - y_k)}$$

Region 2:

$$(x_{k+1}, y_{k+1}) = \begin{cases} (x_k, y_k - 1) \\ (x_k + 1, y_k - 1) \end{cases}$$



$$\text{midpoint} = \left(\frac{x_k + x_k + 1}{2}, \frac{y_k + y_k - 1}{2}\right)$$

$$= \left(x_k + 1/2, y_k - 1\right)$$

$$P^2_k = f_{\text{ellipse}}\left(x_k + 1/2, y_k - 1\right)$$

$$= r_y^2\left(x_k + \tfrac{1}{2}\right)^2 + r_x^2\left(y_k - 1\right)^2 - r_x^2 r_y^2$$

$P^2_k < 0 \Rightarrow$ midpoint inside ellipse boundary

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

$P^2_k > 0 \Rightarrow$ midpoint outside ellipse boundary

$$(x_{k+1}, y_{k+1}) = (x_k, y_k - 1) \qquad [\because y_{k+1} = y_k - 1]$$

$$P^2_{k+1} = r_y^2\left(x_{k+1} + 1/2\right)^2 + r_x^2\left(y_{k+1} - 1\right)^2 - r_x^2 r_y^2$$

$$P^2_{k+1} - P^2_k = r_y^2\left[\left(x_{k+1} + \tfrac{1}{2}\right)^2 - \left(x_k + 1/2\right)^2\right] + r_x^2\left[(y_{k+1} - 1)^2 - (y_k - 1)^2\right]$$

$$= r_y^2\left[\left(x_{k+1} + \tfrac{1}{2}\right)^2 - \left(x_k + \tfrac{1}{2}\right)^2\right] + r_x^2\left[(y_k - 1 + 1)^2 - (y_k - 1)^2\right]$$

$$P^2_{k+1} = P^2_k + r_y^2\left[\left(x_{k+1} + \tfrac{1}{2}\right)^2 - \left(x_k + \tfrac{1}{2}\right)^2\right] + r_x^2\left(1 - 2(y_k - 1)\right)$$

Initial decision Parameter of Region 2

initial Point $= (x_0, y_0)$

P_0^2 = fellipse $\left(x_0 + \frac{1}{2}, y_0 - 1\right)$

$$P_0^2 = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2\left(y_0 - 1\right)^2 - r_x^2 r_y^2$$

$P_{k+1}^2 = P_k^2 \le 0 \Rightarrow (x_{k+1}, y_{k+1}) = (x_k + 1, y_k - 1) \Rightarrow P2_{k+1} = P2_k + 2x_{k+1} r_y^2 + r_x^2 - 2y_{k+1} r_x^2$

$P_{k+1}^2 = P_k^2 > 0 \Rightarrow (x_{k+1}, y_{k+1}) = (x_k, y_k - 1) \Rightarrow P_{2k+1} = P_{2k} - 2(y_k - 1) 2x_2^2 + r_x^2$

Ex: center $(0,0)$ $r_x = 8$, $r_y = 6$

Region 1: Initial Point $(0, r_y) = (0, 6)$

$$P1_0 = r_y^2 + \frac{1}{4} r_x^2 - r_x^2 y$$

$$= 6^2 + \frac{1}{4} 8^2 - 8^2 (6)$$

$$= 36 + \frac{1}{4} \times 64 - 64 \times 6$$

$$= -332$$

$2r_y^2 x \ge 2r_x^2 y$

$k = 0$; $P1_k = P1_0 = -332 < 0$

$(x_{k+1}, y_{k+1}) = (x_k + 1, y_k)$

$= (0+1, 6) = (1, 6)$

$2r_y^2 x = 2(6)^2 (8) = +72$

$2r_x^2 y = 2(8)^2 (6) = 768$

$P1_{k+1} \Rightarrow P1_{0+1} = P1_0 + r_y^2 (1 + 2(x_k + 1))$

$= -332 + 6^2 (1 + 2(0+1))$

$= -332 + 36(1+2)$

$= -224$

$K = 1$; $P1_1 = -224 < 0$

$(x_{k+1}, y_{k+1}) = (1+1, 6)$

$= (2, 6)$

$2r_y^2 x = 2(6)^2 (1) = 2(36) = 144$

$2r_x^2 y = 2(8)^2 (6) = 2(64)(6) = 768$

[∵ Region 1:

$P1_0 = r_y^2 + \frac{1}{4} r_x^2 - r_x^2 y$

$P1_k < 0 \Rightarrow (x_k + 1, y_k)$

$P1_{k+1} = P1_k + r_y^2 (1 + 2(x_k + 1))$

$P1_k > 0 \Rightarrow (x_k + 1, y_k - 1)$

$P1_{k+1} = P1_k + r_y^2 (1 + 2(x_k + 1)) + 2r_x^2 (1 - y_k)$]

K	$P1_k$	(x_{k+1}, y_{k+1})	$2r_y^2 x$	$2r_x^2 y$
0	-332	(1,6)	72	768
1	-224	(2,6)	144	768
2	-44	(3,6)	216	768
3	208	(4,5)	288	640
4	-108	(5,5)	360	640
5	288	(6,4)	432	512
6	244	(7,3)	504	382

Note: Continue the process until $2r_y^2 x \ge 2r_x^2 y$

Region 2: Initial Point = (7,3)

$P2_0 = r_y^2\left(x_0+\frac{1}{2}\right)^2 + r_x^2(y_0-1)^2 - r_x^2 r_y^2$

$= 6^2\left(7+\frac{1}{2}\right)^2 + 8^2(3-1)^2 - 8^2 \times 6^2$

$= -23$

$K=0 \Rightarrow P2_0 = -23 < 0$

$\Rightarrow (x_K+1, y_K-1)$

$= (7+1, 3-1) = (8,2)$

$P2_{K+1} = P2_1 + 2r_y^2(x_K+1) + r_x^2(1-2(y_K-1))$

$= -23 + 2(6)^2(7+3) + 8^2(1-2(3-1))$

$= 361$

Region 2: $P2_0 = r_y^2\left(x_0+\frac{1}{2}\right)^2 + r_x^2(y_0-1)^2 - r_x^2 r_y^2$

$P2_K < 0 \Rightarrow (x_K+1, y_K-1)$

$P2_{K+1} = P2_K + 2r_y^2(x_K+1) + r_x^2(1-2(y_K-1))$

$P2_K > 0 \rightarrow (x_K, y_K-1)$

$P2_{K+1} = P2_K + r_x^2(1-2(y_K-1))$

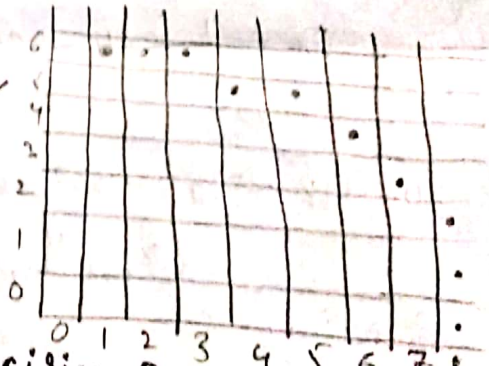
K	$P2_K$	(x_{K+1}, y_{K+1})
0	-23	(8,2)
1	361	(8,1)
2	336	(8,0)

Note: Continue process until Point $(0, r_x)$

Mid Point Ellipse Algorithm:

1. Input r_x, r_y, and ellipse center (x_c, y_c), and obtain the first Point on an ellipse centered on the origin as

$(x_0, y_0) = (0, r_y)$



2. Calculate the initial value of the decision Parameter in region 1 as

$P1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4}r_x^2$

3. At each x_K Position in region 1, starting at K=0, perform the following test: If $P1_K < 0$, the next Point along the ellipse centered on (0,0) is (x_{K+1}, y_K) and

$P1_{K+1} = P1_K + 2r_y^2 x_{K+1} + r_y^2$

Otherwise, the next Point along the circle is (x_K+1, y_K-1) and $P1_{K+1} = P1_K + 2r_y^2 x_{K+1} - 2r_x^2 y_{K+1} + r_y^2$

with

$$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2, \quad 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$

and continue until $2r_y^2 x \geq 2r_x^2 y$.

4. Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculate in region 1 as

$$P2_0 = r_y^2 \left(x_0 + \tfrac{1}{2}\right)^2 + r_x^2 \left(y_0 - 1\right)^2 - r_x^2 r_y^2$$

5. At each y_k position in region 2, starting at $k=0$, perform the following test: If $P2_k > 0$, the next point along the ellipse centered on $(0,0)$ is $(x_k, y_k - 1)$ and

$$P2_{k+1} = P2_k + 2r_x^2 - 2r_x^2 y_{k+1}$$

otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$P2_{k+1} = P2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

using the same incremental calculations for x & y as in region 1.

6. Determine symmetry points in the other three quadrants.

7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, \quad y = y + y_c$$

8. Repeat the steps for region 1 until $2r_y^2 x \geq 2r_x^2 y$.

Attributes of output Primitives:

Any Parameter that affects the way a primitive is to be displayed is referred to as an attribute Parameter

Ex: Attribute parameters are color, size etc.

* A line drawing function for example could contain Parameter to set color, width & other properties.

1. Line attributes
2. Curve attributes
3. Color & Grayscale Levels
4. Area Fill Attributes
5. Character Attributes
6. Bundled Attributes

[∵ PHIGS = Programmer's Hierarchical Interactive Graphics System]

Line Attributes:

Basic attributes of a straight Line segment are its type, its width, & its color. In some graphics packages, lines can be also be displayed using selected Pen or brush options.

- Line Type
- Line width
- Pen and Brush options
- Line color

Line Type: Possible selection of line type attribute includes solid lines, dashed lines & dotted lines, TO set line type attributes in a PHIGS application program, a user invokes the function.

Set Linetype (lt)

where parameter it is assigned a +ve integer value 1, 2, 3, & 4 to generate lines that are solid, dashed, dash dotted respectively.

* other values for line type parameter it could be used to display variations in dot-dash Patterns.

Line width: Implementation of line ~~with~~ width option depends on the capabilities of the output device to set the line width attributes.

setLinewidthScaleFactor (lw)

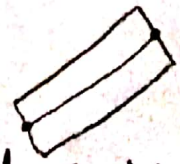
* Line width Parameter lw is assigned a positive number to indicate the relative width of line to be displayed.
* Value of 1 specifies a standard width line. A user could set lw to a value of 0.5 to plot a line whose width is half that of the standard line.
* values greater than 1 Produce lines thicker than the standard.

Line cap: we can adjust the shape of the line ends to give them a better appearance by adding ~~to~~ line caps.

There are three type of line cap. They are

- Butt cap
- Round cap
- Projecting square cap

Butt cap: It is obtained by adjusting the end positions of the component parallel lines. so, that the thick line is displayed with square ends that are perpendicular to the line path.



Round cap: It is obtained by adding a filled semicircle to each butt cap. The circular arcs are centered on the line endpoints and have a diameter equal to the line thickness.

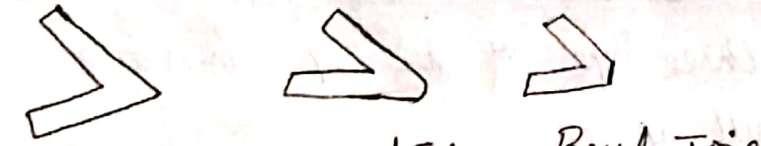
Projecting square cap: Extend the line and add butt caps that are positioned one-half of the line width beyond the specified endpoints.

Three possible methods for smoothly joining two line segments
- Miter Join
- Round Join
- Bevel Join

Miter Join: It is accomplished by extending the outer boundaries of each of the two lines until they meet.

Round Join: It is produced by capping the connection between the two segments with a circular boundary whose diameter is equal to the width.

Bevel Join: It is generated by displaying the line segment with but caps and filling in triangular gap where the segments meet.

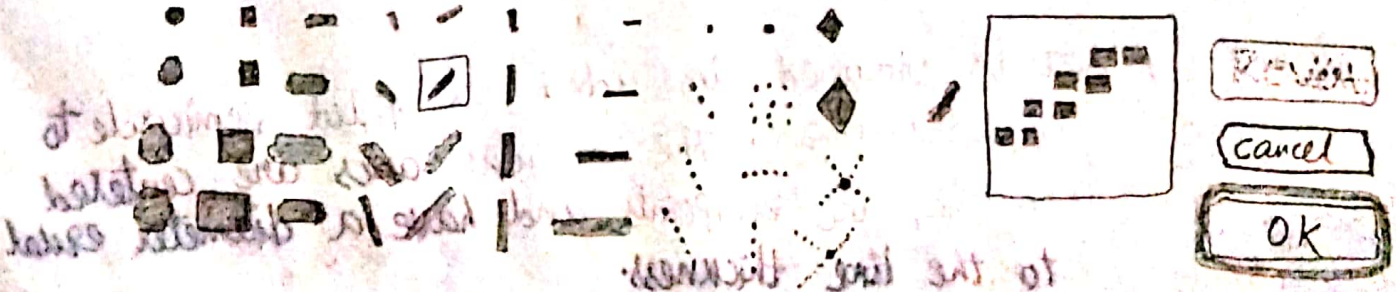


miter Join round Join Bevel Join

Pen and Brush options:

With some packages, lines can be displayed with pen or brush selections. Options in this category include shape, size and pattern. Some possible pen or brush shapes are given.

Custom Document Brushes



* These shapes can be stored in a pixel mask that identifies the array of pixel positions that are to be set along the line path.

* Lines generated with Pen or brush, shapes can be displayed in various widths by changing the size of the mask.

Ex: Rectangular Pen line could be narrowed with a 2×2 rectangular mask or widened with a 4×4 mask.

Line Color: A Poly line routine displays a line in the current color by setting this color value in the frame buffer at pixel locations along the line path using the set pixel procedure.

* Set the line color value in PHIGS with the function

```
setPolyLineColourIndex(lc)
```

* Nonnegative integer values, corresponding to allowed color choices, are assigned to the line color Parameter lc.

Ex: use of various line attribute commands in an applications program is given by the following sequence of statements.

```
setLinetype(2);
setLinewidthScaleFactor(2);
setPolylineColourIndex(5);
Polyline(n1, wc Points1);
setPolylineColorIndex(6);
Poly line(n2, wc Points2);
```

This Program segment would display two figures, drawn with double-wide dashed lines, the first is displayed in a color corresponding to code 5, and the second in color 6.

Curve attributes: Parameters for curve attribute are same as those for line segments. curves displayed with varying colors, widths, dot-dash patterns and available Pen or brush options.

curve slope <1 - vertical spans
>1 - horizontal

Color & Grayscale Levels: Various color and intensity-level options can be made available to a user, depending on the capabilities and design objectives of a particular system.

* In a color raster system, the number of color choices available depends on the amount of storage provided per Pixel in the frame buffer.
* Color-information can be stored in the frame buffer in two ways: - we can store color codes directly in the frame buffer.
  - we can put the color codes in a separate table and use Pixel values as an index into this table.
* with the direct storage scheme, whenever a particular color code is specified in an application program, the corresponding binary value is placed in the frame buffer for each-component Pixel in the output primitives to be displayed in that color.
* A minimum number of colors can be provided in this scheme with 3 bits of storage per Pixel.

THE EIGHT COLOR CODES FOR A THREE BIT PER PIXEL FRAME BUFFER

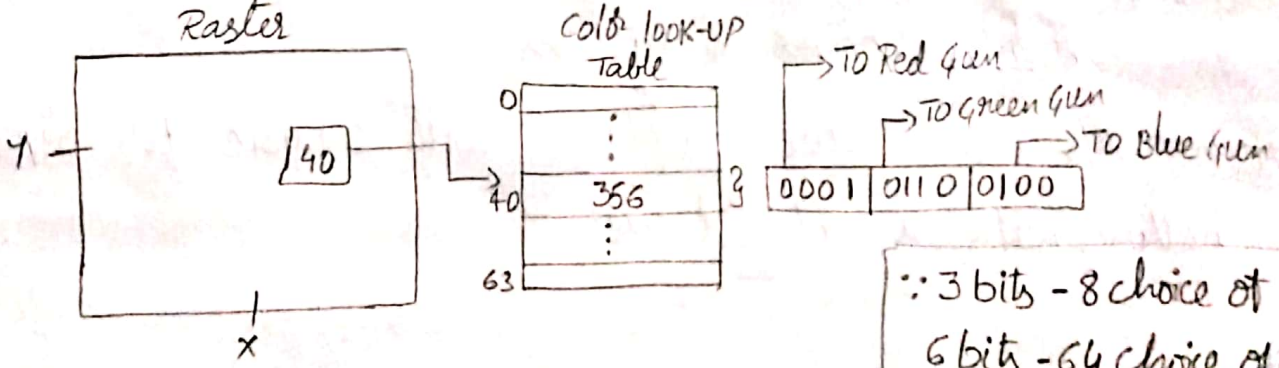
Color code	Stored color values in Frame Buffer RED	GREEN	BLUE	Displayed color
0	0	0	0	BLACK
1	0	0	1	Blue
2	0	1	0	Green
3	0	1	1	Cyan
4	1	0	0	Red
5	1	0	1	Magenta
6	1	1	0	Yellow
7	1	1	1	White

· Color Tables: This is an alternate means for providing extended color capabilities to a user without requiring large frame buffers.

* A user can set color-table entries in a PHIGS applications program with the function.

setColourRepresentation (ws, ci, colorptr)

Parameters ws - Identifies the workstation output device
ci - specifies the color index, which is the color-table Position number (0 to 255)
colorptr - Points to a trio of RGB color values (r, g, b) each specified in the range from 0 to 1.



∵ 3 bits - 8 choice of color
6 bits - 64 choice of color
8 bits - 256 choice of color

Grayscale:

Monitors that have no color capability, color functions can be used in an application program to set the shades of gray, or grayscale, for displayed primitives.

* Numeric values over the range from 0 to 1 can be used to specify grayscale levels, which are then converted to appropriate binary codes for storage in the raster.

Intensity codes for a four-level grayscale system:

Intensity Codes	Stored Intensity Values in the Frame Buffer (Binary code		Displayed Grayscale
0.0	0	(00)	Black
0.33	1	(01)	Dark gray
0.67	2	(10)	Light gray
1.0	3	(11)	White

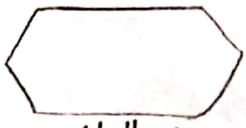
$$\text{Intensity} = 0.5\,[\min(r,g,b) + \max(r,g,b)]$$

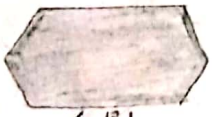
Area Fill Attributes:

Options for filling a defined region include a choice between a solid color or a pattern fill and choices for particular colors & patterns.

Fill styles: Areas are displayed with 3 basic fill styles.

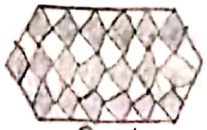
hollow with a color border





filled with a solid color

filled with a specified Pattern or design



* A basic fill style is selected in a PHIGS Program with the function. `setInteriorStyle(fs)`
* values for the fill-style parameter fs include hollow, solid & Pattern. Another value for fill style is hatch, which is used to fill an area with selected hatching Patterns. Parallel lines or crossed lines.

* The color for a solid interior or for a hollow area outline is chosen with where fill color Parameter fc is set to the desired color code.

`setInteriorColourIndex(fc)`

Diagonal Hatch Fill

Diagonal cross-Hatch fill

Pattern fill:

Select fill patterns with `setInteriorStyleIndex(Pi)` where Pattern index Parameter Pi specifies a table Position.

[∵ Pi = Pattern Index]

Ex: Set of statements would fill the area defined in the fillArea command with the second Pattern type stored in the pattern table.

```
setInteriorStyle(Pattern);
setInteriorStyleIndex(2);
fillArea(n, Points);
```

Pattern Table

Index (Pi)	Pattern (CP)
1	$\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$
2	$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$

* Separate tables are set up for hatch Patterns.
* If hatch fill is selected for the interior style in this Program segment, then the value assigned to Parameter Pi is an index to the stored Patterns in the hatch table.

`SetPatternRepresentation(WS, Pi, nx, ny, CP)`

WS - Workstation
Pi - Pattern Index
nx - Columns
ny - rows
CP - two-dimensional array of color code.

* Another is to allow repainting of a color area that was originally filled with a semitransparent brush, where the current color is then a mixture of the brush color & the background colors "behind" the area.
* In either case, if want to fill the new fill color to have the same variations over the area as the current fill color.

Ex: The linear soft-fill algorithm repaints an area that was originally Painted by merging a foreground color F with a single background color B, where $F \neq B$.

* F, B values are known, we can determine how these colors were originally combined by checking the current color contents of the frame buffer. The current RGB color P of each pixel within the area to be refilled is some linear combination of F and B.

$$P = tF + (1-t)B$$

where transparency factor 't' has a value between 0 & 1 for each pixel. vector eqn holds for each RGB component of the color. $P = (P_R, P_G, P_B)$, $F = (F_R, F_G, F_B)$, $B = (B_R, B_G, B_B)$

Calculate the value of Parameter 't' using one of the RGB color components as:

$$t = \frac{P_K - B_K}{F_K - B_K}$$

where $K = R, G, or B$ & $F_K \neq B_K$

* Similar soft-fill procedures can be applied to an area whose foreground color is to be merged with multiple background color areas, such as checkerboard Pattern.

when two background colors B_1 & B_2 are mixed with foreground color F, the resulting pixel color P is

$$P = t_0 F + t_1 B_1 + (1 - t_0 - t_1) B_2$$

Character Attributes: The appearance of displayed character is controlled by attributes such as font, size, color & orientation.

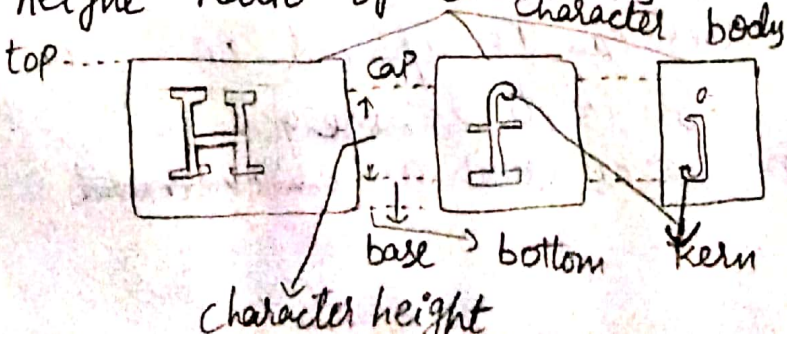
* Attributes can be set both for entire character strings (text) and for individual characters defined as marker symbols.

Text Attributes: The choice of font or typeface is set of characters with a particular design style as courier, Helvetica, times roman, and various symbol groups.

* The characters in a selected font also be displayed with styles. Solid : ——, dotted : ....., double : ═══ in boldface in italics, and in outline or shadow styles.
* A particular font & associated style is selected in a PHIGS program by setting an integer code for the text font parameter 'tf' in the function.

`setTextFont (tf)`

* Control of text color (or intensity) is managed from an application program with `setTextColourIndex (tc)` where text color parameter tc specifies an allowable color code.
* Text size can be adjusted without changing the width to height ratio of characters with `setCharacterHeight (ch)`



* parameter `ch` is assigned a real value greater than 0 to set the coordinate height of capital letters.

The width only of text can be set with function.

setCharacterExpansionFactor(cw)

Height 1
Height 2
Height 3

where the character-width parameter cw is set to a positive real value that scales the body width of character.

Spacing b/w characters is controlled separately with

setCharacterSpacing(cs)

width 0.5
width 1.0
width 2.0

where the character-spacing parameter `cs` can be assigned any real value.

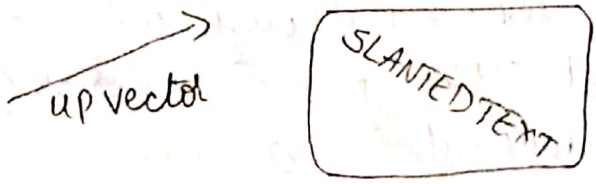
Spacing 0.0
Spacing 0.5
Spacing 1.0

* The orientation for a displayed character string is set according to the direction of the character up vector.

setCharacterUpVector(upvect)

* Parameter upvect in this function is assigned two values that specify the x & y vector components.

Ex: upvect = (1,1), direction = 45° & text would be displayed as



* TO arrange character strings vertically & horizontally

setTextPath(tp)

where the text path parameter `tp` can be assigned the value: right, left, up, or down

* Attribute for character strings is alignment, this attribute specifies how text is to be positioned with respect to the start coordinates.

gnirtS
string

Alignment attributes are set with

`setTextAlignment (h,V)`

'h' - horizontal alignment is set by value of left, center or right

'V' - vertical alignment is set by assigning value of top, cap, half, base or bottom.

* Precision specification for text display is given with

`setTextPrecision (tpr)`

tpr - one of values string, char or stroke.

Marker Attributes: A marker symbol is a single character that can be displayed in different colors & in different sizes. Marker attributes are implemented by procedures that load the chosen character into the raster at the defined positions with the specified color & size.

* To select a particular character to be the marker symbol with

`setMarkerType (mt)`

where 'mt' is set to an integer code. Typical codes for marked type are the integers 1 through 5.

a dot (·), a vertical cross (+), an asterisk (*), a circle (o), & a diagonal cross (x).

* Marker size can be set by using

`setMarkerSizeScaleFactor (ms)`

* Parameter marker size 'ms' assigned a positive number, scaling parameter is applied to the normal size for the particular marker symbol chosen.

Values >1 Produce: character enlargement.

<1 : Produce the marker size.

* Marker color specified with `setPolymarkerColourIndex(mc)`

A selected color code parameter 'mc' is stored in the current attribute list & used to display subsequently specified marker primitives.

Bundled Attributes: The Procedures considered so far each function reference a single attribute that specifies exactly how a primitive is to be displayed these specifications are called individual attributes.

* A Particular set of attributes values for a primitive on each output device is chosen by specifying appropriate table index.
* Attributes specified in this manner are called bundled attributes.
* The choice between a bundled & an unbundled specification is made by setting a switch called the aspect source flag for each of these attributes.

setIndividualASF (attributeptr, flagptr)

where Parameter attribute ptr Points to a list of attributes & Parameter flagptr Points to the corresponding list of aspect source flags, each aspect source flag can be assigned a value of individual or bundled.

Bundled line attributes: Entries in the bundle table for line attributes on a specified workstation are set with the function

setPolylineRepresentation (ws, li, lt, lw, lc)

Parameter ws - workstation identifier
li - line index define the bundle table position.
lt, lw, lc - assigned values to set the line type, line width, and line color specifications for designated table index.

Ex: setPolylineRepresentation (1, 3, 2, 0.5, 1)
setPolylineRepresentation (4, 3, 1, 1, 7)

A Polyline that is assigned a table index value of 3 would be displayed using dashed lines at half tickness in a blue color on workstation 1; while on workstation 4, this same index generates solid, standard-sized white lines.

Bundle area-fill attributes: Table entries for bundled-area-fill attributes are set with `setInteriorRepresentation(ws, fi, fs, pi, fc)`

* Defines the attributes list corresponding to fill index `fi` on workstation `ws`. Parameter fs, pi & fc are assigned values for the fill style Pattern index & fill color.

Bundled Text Attributes: `setTextRepresentation(ws, ti, tf, tp, te, ts, tc)`

~~setPolymarker~~

* Bundles values for text font, Precision expansion factor size an color in a table Position for work station `ws` that is specified by value assigned to text index Parameter `ti`.

Bundled marker Attributes: `setPolymarkerRepresentation(ws, mi, mt, ms, mc)`

That defines marker type marker scale factor marker color for index mi on workstation ws.

Inquiry Functions:

Current settings for attributes & other Parameters as workstations types & status in the system lists can be retrieved with inquiry functions.

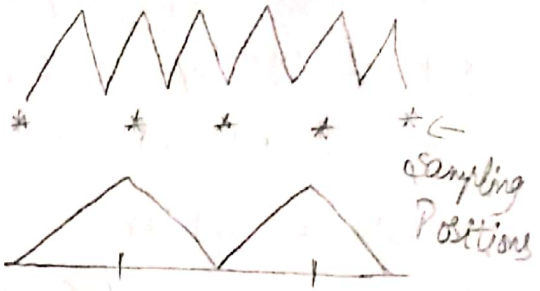
`inquirePolyLineIndex(lastli)` and

`inquireInteriorColourIndex(lastfc)`

⇒ copy the current values for line index & fill color into Parameter lastli & lastfc.

Antialiasing: Displayed Primitives generated by the raster algorithms have a jagged, or stairstep, appearance because the sampling process digitizes coordinate Points on an object to discrete integer Pixel Positions.

* This distortion of information due to low-frequency sampling (undersampling) is called aliasing.
* To improve the appearance of displayed raster lines by applying antialiasing methods that compensate for the undersampling process.
* To avoid losing information from such periodic objects, we need to set the sampling frequency to at least twice that of the highest frequency occuring in the object, referred to as the Nyquist sampling frequency (or Nyquist sampling rate) f_s:



$$f_s = 2f_{max}$$

* Another way to state this is that the sampling interval should be no larger than one-half the cycle interval (called the Nyquist sampling interval).

For x-interval sampling, the Nyquist sampling interval Δx_s is

$$\Delta x_s = \frac{\Delta x_{cycle}}{2} \quad \text{where } \Delta x_{cycle} = 1/f_{max}.$$

* one way to increase sampling rate with raster systems is simply to display objects at higher resolution.

⇒ A straightforward antialiasing method is to increase sampling rate by treating the screen as if it were covered with a finer grid than is actually available.

* Can use multiple sample Points across this finer grid to determine an appropriate intensity level for each screen Pixel.
* sampling object characteristics at a high resolution & displaying the results at a lower resolution is called supersampling (or Post filtering).

* An alternative to supersampling is to determine pixel intensity by calculating the areas of overlap of each pixel with the objects to be displayed.
* Antialiasing by computing overlap areas is referred to as area sampling (or prefiltering).
* Pixel overlap areas are obtained by determining where object boundaries intersect individual pixel boundaries.
* Raster objects can also be antialiased by shifting the display location of pixel areas. this technique is called Pixel Phasing. is applied by "micropositioning" the electron beam in relation to object geometry.
* Another advantage of supersampling with a finite-width line is that the total line intensity is distributed over more pixels.

Two-Dimensional Geometric Transformations

Changes in orientation, size and shape are accomplished with geometric transformations that alter the coordinate description of objects.

Basic transformations are
- Translation
- Scaling
- Rotation

Translation: A translation is applied to an object by representing it along a straight line path from one coordinate location to another adding translation distances tx, ty to original coordinate position (x, y) to move the point to a new position (x', y'):

* The translation distance point (tx, ty) is called translation vector or shift vector.

* Translation equation can be expressed as single matrix eqn by using column vectors to represent the coordinate position & the translation vector as

$P = (x, y)$ - Point before translation

$P'(x', y')$ - Point after translation

t_x, t_y

$$x' = x + t_x$$
$$y' = y + t_y$$

$$P' = P + T$$

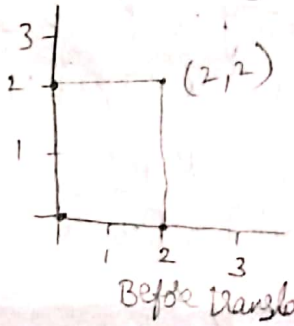
$$[x' \; y'] = [x \; y] + [t_x \; t_y]$$

or

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Ex: Square (0,0) (2,0) (0,2) (2,2)

$t_x = 2$, $t_y = 3$



Before translation

(0,0)

$x' = 0 + 2 = 2$

$y' = 0 + 3 = 3$

(0,0) → (2,3)

(2,0) $x' = 2 + 2 = 4$

$y' = 0 + 3 = 3$

(2,0) → (4,3)

(0,2) $x' = 0 + 2 = 2$

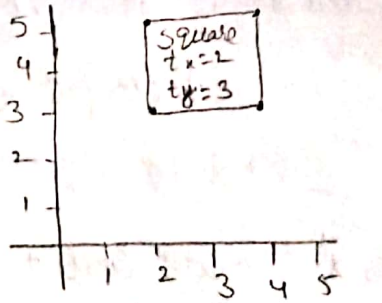
$y' = 2 + 3 = 5$

(0,2) → (2,5)

(2,2) $x' = 2 + 2 = 4$

$y' = 2 + 3 = 5$

(2,2) → (4,5)



After translation

Rotation:

A 2-D rotation is applies to an object by repositioning it along a circular path on xy plane. To generate a rotation, specify a rotation angle θ & the position (x_r, y_r) of the rotation point (Pivot Point) about which the object is to be rotated.

* Positive values for the rotation angle define counter clock wise rotation, Negative values of angle rotate objects in clockwise direction.

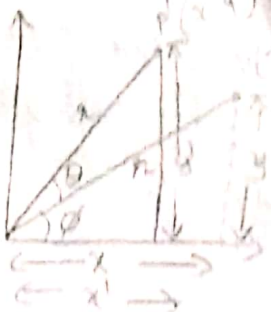
New angle after rotation P to P' = $(\phi+\theta)$

$\cos(\phi+\theta) = \frac{x'}{r}$

$\boxed{x' = r\cos(\phi+\theta)}$ [$\because \cos(A+B) = \cos A\cos B - \sin A \sin B$]

$\sin(\phi+\theta) = \frac{y'}{r}$

$\boxed{y' = r\sin(\phi+\theta)}$ [$\because \sin(A+B) = \sin A\cos B + \cos A \sin B$]



$\cos\phi = \frac{\text{Adj side}}{\text{hyp}}$

$x = r\cos\phi$

$\sin\phi = \frac{y}{r}$

$y = r\sin\phi$

Anticlockwise (+)

$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$

$y' = r\sin(\phi+\theta) = r\sin\phi\cos\theta + r\cos\phi\sin\theta$

$x' = x\cos\theta - y\sin\theta$ [$\because r\cos\phi = x'$, $r\sin\phi = y$]

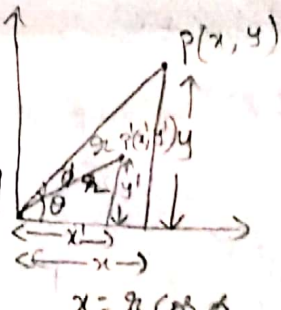
~~$y' = y\cos\theta + x\sin$~~

$y' = x\sin\theta + y\cos\theta$

$\boxed{P' = R \cdot P}$

$$[x' \; y'] = [x \; y] \begin{bmatrix} \cos\theta & +\sin\theta \\ +\sin\theta & \cos\theta \end{bmatrix}$$

clockwise (-)



$x = r\cos\phi$

$y = r\sin\phi$

New angle after Rotation P to P' = $(\phi-\theta)$

$x' = r\cos(\phi-\theta)$ [$\because \cos(A-B) = \cos A\cos B + \sin A\sin B$]

$y' = r\sin(\phi-\theta)$ [$\because \sin(A-B) = \sin A\cos B - \cos A\sin B$]

$x' = r\cos\phi\cos\theta + r\sin\phi\sin\theta$

$\boxed{x' = x\cos\theta + y\sin\theta}$

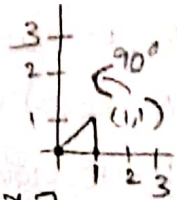
$y' = r\sin\phi\cos\theta - r\cos\phi\sin\theta$

$\boxed{y' = y\cos\theta - x\sin\theta}$

$$[x' \; y'] = [x \; y] \begin{bmatrix} \cos\theta & +\sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Ex: Triangle (0,0) (1,0) (1,1)

$\theta = 90°$ (anti-clock)



$\sin 90° = 1$

$\cos 90° = 0$

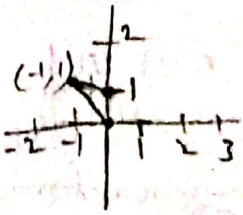
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

(0,0)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$\boxed{(0,0) \rightarrow (0,0)}$



(1,0)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(1,0) → (0,1)

(1,1)

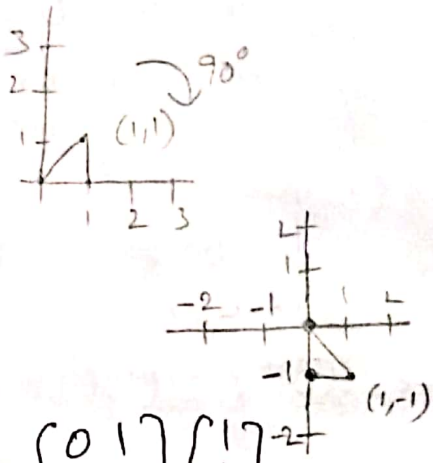
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

(1,1) → (-1,1)

Clockwise:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



(0,0)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(0,0) → (0,0)

(1,0)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

(1,0) → (0,-1)

(1,1)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

(1,1) → (1,-1)

Scaling: A scaling transformation alters the size of an object. This operation can be carried out for polygons by multiplying the coordinate values (x, y) to each vertex by scaling factors S_x & S_y to produce the transformed coordinates (x', y').

$S_x \neq S_y$ ⇒ change in shape

$S_x = S_y$ → No change in shape (uniform)

S_x & S_y < 1 → size decreases

S_x & S_y > 1 – size increases

If S_x & S_y in between 0 & 1 (Point is closer to origin)

$P = (x, y)$ → before scaling

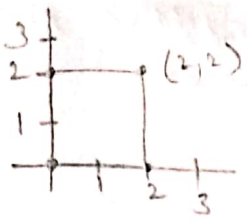
$P' = (x', y')$ → after scaling

$$x' = x \cdot S_x \quad , \quad y' = y \cdot S_y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$P' = S \cdot P$

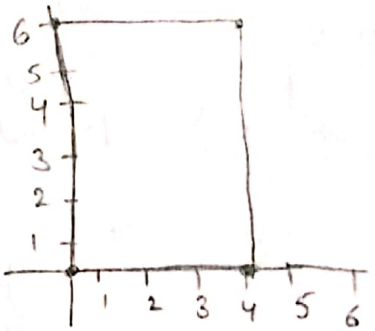
Ex: Square (0,0) (2,0) (0,2) (2,2)



① $S_x = 2$, $S_y = 3$

(0,0)
$x' = 0 \times 2 = 0$
$y' = 0 \times 3 = 0$
$(0,0) = (0,0)$

(2,0)
$x' = 2 \times 2 = 4$
$y' = 0 \times 3 = 0$
$(2,0) \rightarrow (4,0)$



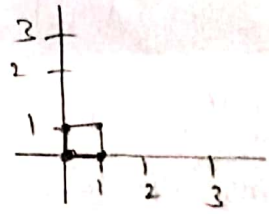
(0,2)
$x' = 0 \times 2 = 0$
$y' = 2 \times 3 = 6$
$(0,2) \rightarrow (0,6)$

(2,2)
$x' = 2 \times 2 = 4$
$y' = 2 \times 3 = 6$
$(2,2) \rightarrow (4,6)$

② $S_x = 0.5$, $S_y = 0.5$

(0,0) $x' = 0 \times 0.5 = 0$
$y' = 0 \times 0.5 = 0$
$(0,0) \rightarrow (0,0)$

(2,0)
$x' = 2 \times 0.5 = 1$
$y' = 0 \times 0.5 = 0$
$(2,0) \rightarrow (1,0)$



(0,2)
$x' = 0 \times 0.5 = 0$
$y' = 2 \times 0.5 = 1$
$(0,2) \rightarrow (0,1)$

(2,2) $x' = 2 \times 0.5 = 1$
$y = 2 \times 0.5 = 1$
$(2,2) \rightarrow (1,1)$

Matrix Representation & Homogeneous Coordinates:

Many graphics applications involve sequences of geometric transformations.

Ex: For object required a translate & rotate at each increment of the motion.

* The coordinate system is called as Homogeneous coordinate system and it allows to express transformation eq^n as matrix multiplication.

. cartesian coordinate position (x,y) is represented as homogeneous coordinate triple (x,y,h)

$$P' = M_1 \cdot P + M_2$$

[M_1 = Multiplicative Matrix
M_2 = Additive Matrix]

$(x,y) \rightarrow (x_h, y_h, h)$

$\Downarrow$ h → any non zero $\Downarrow$

$x_h = x \cdot h$

$y_h = y \cdot h$

$x = \frac{x_h}{h}$

$y = \frac{y_h}{h}$

Ex : (2,3)

h = 2

$(2\times2, 3\times2, 2)$

$\Downarrow$

$(4, 6, 2)$

$\frac{4}{2}, \frac{6}{2} = (2,3)$

Translation: $x' = t_x + x$

$y' = t_y + y$

$$\begin{bmatrix} t_x \\ t_y \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

Default [$\because h = 1$]

$\Downarrow$

$$\begin{bmatrix} x' \\ y' \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or $P' = T(t_x, t_y) \cdot P$

Rotation:

$$\begin{bmatrix} x' \\ y' \\ h \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or $P' = R(\theta) \cdot P$

Scaling:

$$\begin{bmatrix} x' \\ y' \\ h \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or $P' = S(S_x, S_y) \cdot P$

Composite Transformations: A composite transformation is a sequence of transformations, one followed by the other. To set up a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of the individual transformations.

Translation: If two successive translation vectors (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate position P.

$$P' = T(t_{x2}, t_{y2}) \cdot \{T(t_{x1}, t_{y1}) \cdot P\}$$
$$= \{T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1})\} \cdot P$$

where P & P' are represented as homogeneous-coordinate column vectors.

$$\begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x1}+t_{x2} \\ 0 & 1 & t_{y1}+t_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

(or)

$$T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1}) = T(t_{x1}+t_{x2}, t_{y1}+t_{y2})$$

Rotations: two successive translations are additive.

Two successive rotations applied to point P produce the transformed position.

$$P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}$$
$$= \{R(\theta_2) \cdot R(\theta_1)\} \cdot P$$

By multiplying two rotation matrices, we can verify that two successive rotation are additive.

$$R(\theta_2) \cdot R(\theta_1) = R(\theta_1 + \theta_2)$$

Final rotated coordinates can be calculated with the composite rotation matrix.

$$P' = R(\theta_1 + \theta_2) \cdot P$$

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_2 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_2+\theta_1) & -\sin(\theta_2+\theta_1) & 0 \\ \sin(\theta_2+\theta_1) & \cos(\theta_2+\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling: Concatenating transformation matrices for two successive scaling operations produces the following composite scaling matrix.

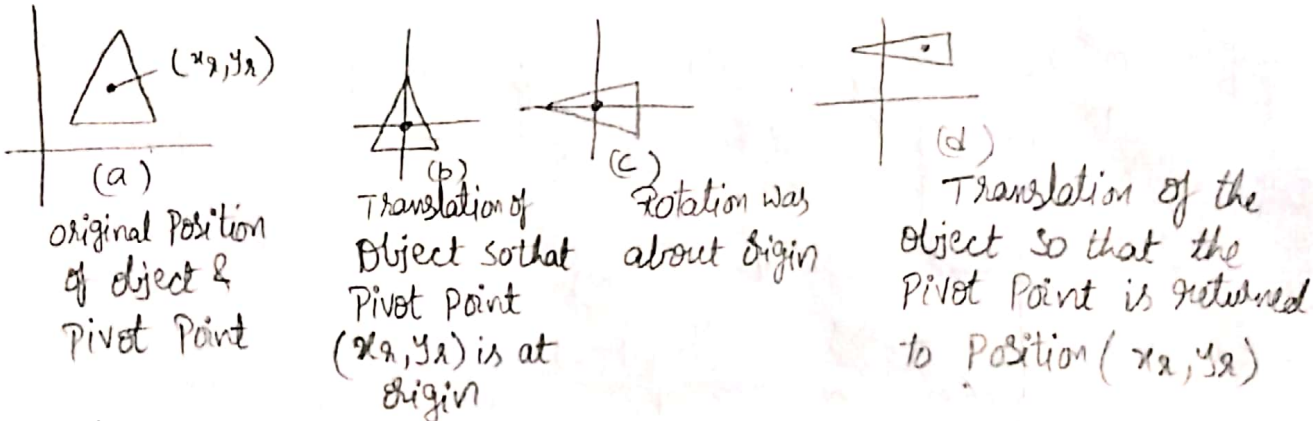
$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x2} \cdot s_{x1} & 0 & 0 \\ 0 & s_{y2} \cdot s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

or

$$S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1}) = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2})$$

General Pivot-Point Rotation:

1. Translate the object so that pivot-position is moved to the coordinate origin.
2. Rotate the object about the coordinate origin
3. Translate the object so that the Pivot Point is returned to its original Position.



The composite transformation matrix for this sequence is obtained with the concatenation

$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

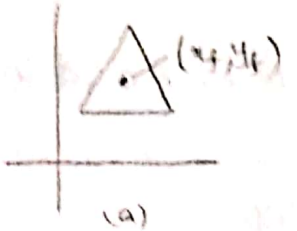
which can expressed as

$$T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) = R(x_r, y_r, \theta)$$

General fixed Point Scaling:

1. Translate object so that the fixed Point coincides with the coordinate origin.
2. Scale the object with respect to the coordinate origin.
3. Use the inverse translation of step 1 to return the object to its original position.

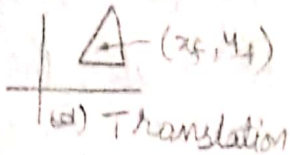
$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$



(a) original position of object & fixed point

(b) Translation of object so that fixed point (x_f, y_f) is at origin

(c) Scaling was about origin



(d) Translation of the object so that the fixed point is returned to position (x_f, y_f).

$$\begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_x & 0 & x_f(1-S_x) \\ 0 & S_y & y_f(1-S_y) \\ 0 & 0 & 1 \end{bmatrix}$$

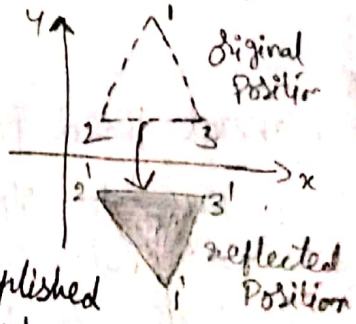
(or) $T(x_f, y_f) \cdot S(S_x, S_y) \cdot T(-x_f, -y_f) = S(x_f, y_f, S_x, S_y)$

Other Transformations:

1. Reflection
2. Shear

Reflection: A reflection is a transformation that produces a mirror image of an object. The mirror image for a two-dimensional reflection is generated relative to an axis of reflection by rotating the object 180° about the reflection axis.

* For reflection axis that are perpendicular to the xy plane or coordinate origin.

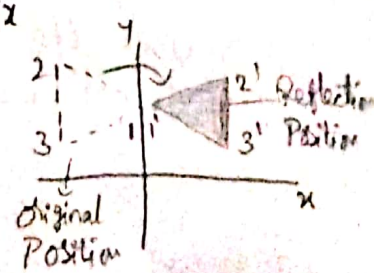


$$\begin{bmatrix} x' \\ y' \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reflection of x axis is accomplished with the transformation matrix

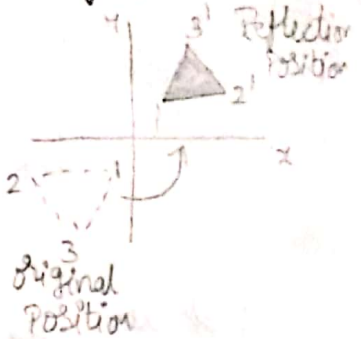
Reflection of an object about the y-axis

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

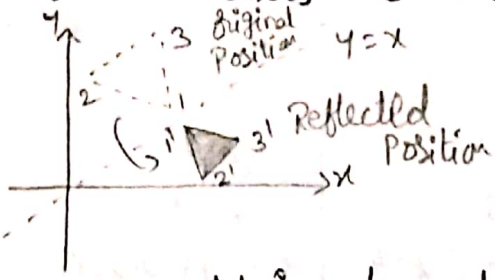


- Reflection of an object about the coordinate origin

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflection axis as the diagonal line $y=x$



To obtain transformation matrix for reflection about diagonal $y=x$ the transformation sequence is

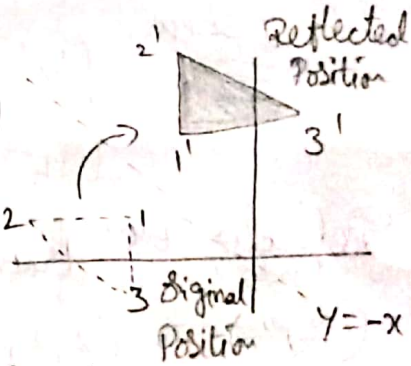
1. clock wise rotation by 45°
2. Reflection about x axis
3. counter clock wise by 45°

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflection axis as the diagonal line $y=-x$

1. clockwise rotation by 45°
2. Reflection about y-axis
3. Counter clock wise by 45°

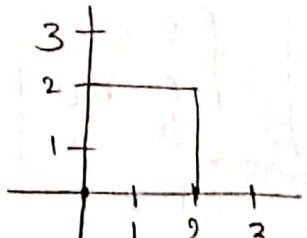
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Shear: A transformation that slants the shape of an object is called the shear transformation.

x-shear:
$y' = y$
$x' = x + Sh_x \cdot y$

y-shear:
$x' = x$
$y' = y + Sh_y \cdot x$



Ex:
(0,0) (0,2)
(2,0) (2,2)

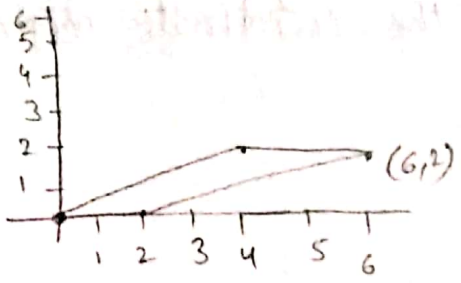
x-shear: Sh_x = 2units

(2,0)
$y' = 0$
$x' = 2 + 2 \cdot 0$
$= 2$
$(x', y') = (2,0)$

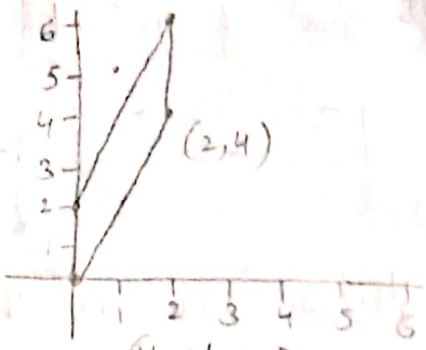
(2,2)
$y' = 2$
$x' = 2 + 2 \cdot 2$
$= 6$
$(x', y') = (6,2)$

(0,0) $y' = 0$
$x' = 0 + 2 \cdot 0$
$x' = 0$
$(x', y') = (0,0)$

(0,2)
$y' = 2$
$x' = 0 + 2 \cdot 2 = 4$
$(x', y') = (4,2)$



(x-shear)



(Y-shear)

y-shear: Shy = 2 units

(0,0)	(0,2)	(2,0)	(2,2)
$x'=0$	$x'=0$	$x'=2$	$x'=2$
$y'=0+2\cdot0=0$	$y'=2+2\cdot0=2$	$y'=0+2\cdot2=4$	$y'=2+2\cdot2=6$
$(x',y')=(0,0)$	$(x',y')=(0,2)$	$(x',y')=(2,4)$	$(x',y')=(2,6)$

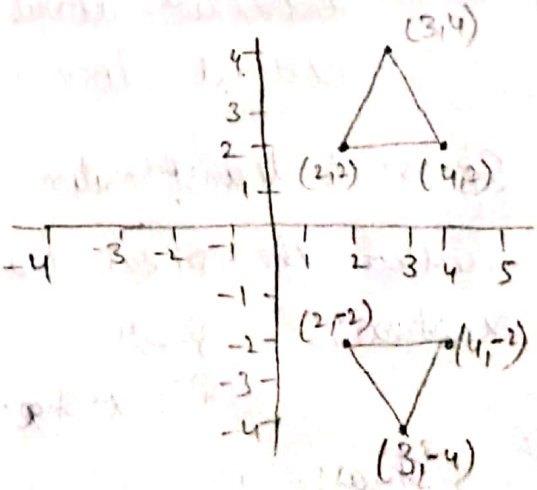
x-shear:

$$\begin{bmatrix} x' \\ y' \\ h \end{bmatrix} = \begin{bmatrix} 1 & Shx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

y-shear:

$$\begin{bmatrix} x' \\ y' \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ Shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reflection Ex: triangle (2,2), (4,2), (3,4)



(2,2) ⇒ (2,2,1)

x-axis

$$\begin{bmatrix} x' \\ y' \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix} \Rightarrow (2,-2)$$

(4,2) ⇒ (4,2,1)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix} \Rightarrow (4,-2)$$

(3,4) ⇒ (3,4,1)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \\ 1 \end{bmatrix}$$

$= (3,-4)$

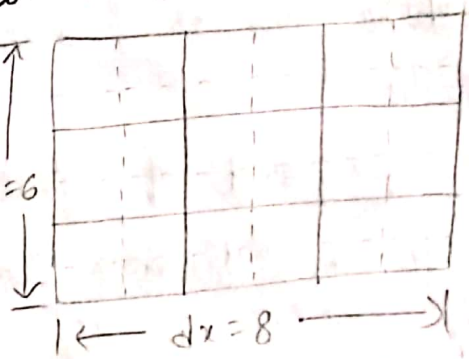
```
CP[1,1]:= 4;        CP[2,2]:= 4;
CP[1,2]:=0;         CP[2,1]:=0;
Set Pattern Representation (1, 1,2,2,CP);
```

* When a color array CP is to be applied to fill a region, we need to specify the size of the area that is to be covered by each element of the array.

Set Pattern Size (dx, dy)

* dx, dy are coordinate width & height of the array mapping. Each Element of the color array would be applied to a 2x2 screen grid containing four pixels.
* Reference position for starting a Pattern fill is assigned with

set Pattern Reference Point (Position)



Pattern array with 4 columns & 3 rows mapped to an 8 by 6 coordinate rectangle.

* Parameter Position is a pointer to coordinates (xP, yP) that fix the lower left corner of the rectangular Pattern.
* The process of filling an area with a rectangular Pattern is called tiling.

Soft fill: Modified boundary-fill & flood-fill Procedures that are applied to repaint areas so that the fill color is combined with the back ground colors are referred to as soft-fill & tint-fill algorithms.

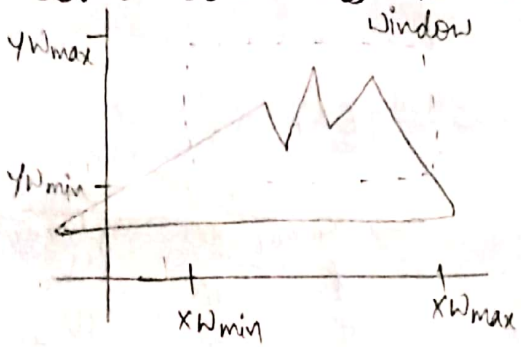
* one use for these fill methods is to soften the fill colors at object borders that have been blurred to antialias the edges.

Two-Dimensional Viewing.

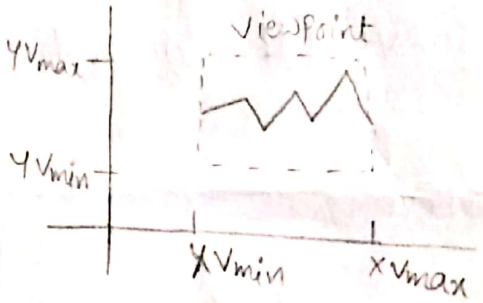
Viewing Pipeline:

A world-coordinate area selected for display is called a **window**.

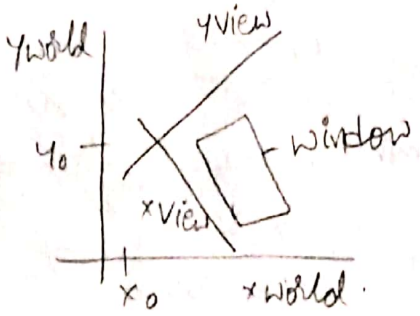
* An area on a display device to which a window is mapped is called a viewport.
  - The window defines what is to be viewed.
  - The viewport defines where it is to be displayed.
* Windows & viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes.
* The mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation.



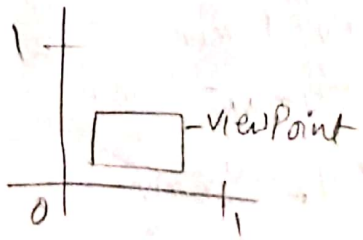
World coordinates



Device coordinates



World coordinates



Normalized Device coordinates

Viewing coordinate Reference frame: This coordinate system provides the reference frame for specifying the world coordinate window.

$P_0 = (x_0, y_0)$.

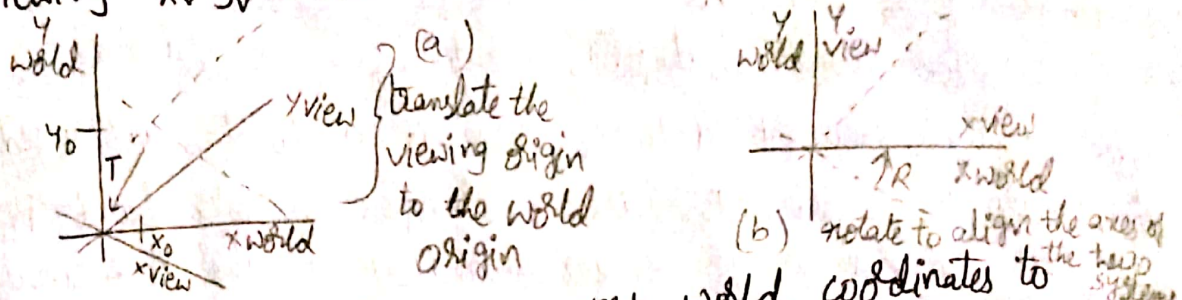
[V = view up vector]

Vector V defines the viewing y_0 direction

components of unit vectors $v = (v_x, v_y)$ and $u = (u_x, u_y)$ for the viewing y_v & x_v axes

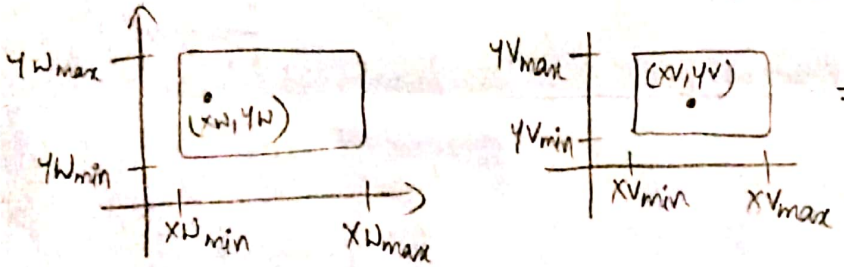
Unit vectors are used to form the first & second rows of the rotation matrix R.

Viewing $x_v y_v$ axes with the world $x_w y_w$ axes.



composite 2D transformation to convert world coordinates to viewing coordinate is $\boxed{M_{WC,VC} = R \cdot T}$

Window-TO-Viewport Coordinate transformation:



(x_w, y_w) & (x_v, y_v)

$$\frac{x_v - x_{V_{min}}}{x_{V_{max}} - x_{V_{min}}} = \frac{x_w - x_{W_{min}}}{x_{W_{max}} - x_{W_{min}}}$$

$$\frac{y_v - y_{V_{min}}}{y_{V_{max}} - y_{V_{min}}} = \frac{y_w - y_{W_{min}}}{y_{W_{max}} - y_{W_{min}}}$$

Viewport Position (x_v, y_v)

$$x_v = x_{V_{min}} + (x_w - x_{W_{min}})\, s_x$$

$$y_v = y_{V_{min}} + (y_w - y_{W_{min}})\, s_y$$

Scaling factors are

$$s_x = \frac{x_{V_{max}} - x_{V_{min}}}{x_{W_{max}} - x_{W_{min}}}$$

$$s_y = \frac{y_{V_{max}} - y_{V_{min}}}{y_{W_{max}} - y_{W_{min}}}$$

Ex:

$x_{W_{min}} = 80$, $x_{W_{max}} = 80$, $y_{W_{min}} = 40$, $y_{W_{max}} = 80$, $(x_w, y_w) = (30, 80)$

$x_{V_{min}} = 30$, $x_{V_{max}} = 60$, $y_{V_{min}} = 40$, $y_{V_{max}} = 60$, $(x_v, y_v) = ?$

$$\frac{x_v - 30}{60 - 30} = \frac{30 - 20}{80 - 20}$$

$$x_v - 30 = \left(\frac{+10}{60}\right) \times 30$$

$$x_v - 30 = 5$$

$$\boxed{x_v = 35}$$

$$\frac{y_v - 40}{60 - 40} = \frac{80 - 40}{80 - 40}$$

$$y_v - 40 = (60 - 40) \times 1$$

$$\boxed{y_v = 60}$$

$$\boxed{(x_v, y_v) = (35, 60)}$$

2-D clipping:

* The Procedure that identifies those Portions of a Picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm or clipping.
* The region against which an object is to clip is called a clip window.

Algorithms for clipping primitive types

1; Point clipping 2; Line clipping (Straight-line segments)
3; Area clipping (Polygons) 4; curve clipping 5; Text clipping.

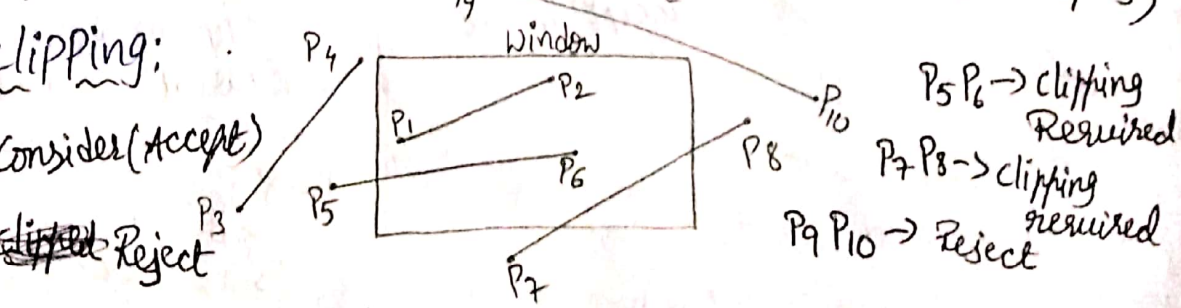
Point clipping: Assuming that the clip window is a rectangle in standard Position, we save a Point $P(x,y)$ for display if the following inequalities are satisfied.

$$xw_{min} \leq x \leq xw_{max}$$
$$yw_{min} \leq y \leq yw_{max}$$

where the edges of the clip window (xw_{min}, xw_{max}, yw_{min}, yw_{max}) can be either the world-coordinate window boundaries or viewport boundaries.

* If any one of these four inequalities is not satisfied, the Point is clipped (not saved for display).

Line clipping:



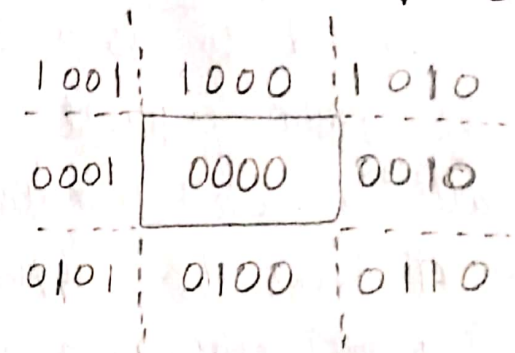
Cohen-Sutherland clipping: This is one of the oldest & most popular line-clipping procedures.

* This method speeds up the processing of line segments by performing initial tests that reduce the number of intersections that must be calculated.

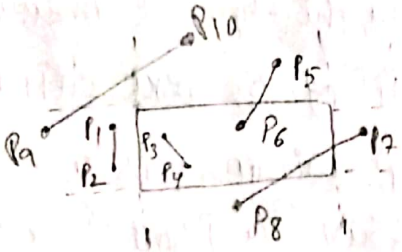
* Every line end-Point in a picture is assigned a four-digit binary code, called a region code, that identifies the location of the Point relative to the boundaries of the clipping rectangle.

bit1: left
bit2: right
bit3: below
bit4: above

or (ABRL code)



b_4	b_3	b_2	b_1
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0



P1 & P2:

P1 → 0001 (non-zero)
P2 → 0001 (non-zero)
AND → 0001 (non-zero)

↳ line is outside of window (Reject)

* Both the end-Points are non-zero apply "AND" operation

P3 & P4:

P3 → 0000 – (zero)
P4 → 0000 – (zero)
AND → 0000 (zero)

} line is inside the window, No clipping is applied. So, accept the line.

P7 & P8:

P7: 0100 (non-zero)
P8: 0010 (non-zero)
AND: 0000 (zero) (Partial)

P7' P7 – clipped

P7' & P8: P7' – 0000 → (zero)
P8 – 0010 → (Non-zero)
AND: 0000 – zero (Partial)

P5 & P6:

P5 → 1000 (non-zero)
P6 → 0000 (zero)
AND → 0000 (zero)

Clipping is required (Partial), to find intersection Point on window

P5' & P6:

P5' → 0000 (zero)
P6 → 0000 (zero)
AND → 0000 (inside window)

P5, P5' is clipped

P_8 P_8' - clipped

P_8' P_7' - P_8' - 0000 (zero)

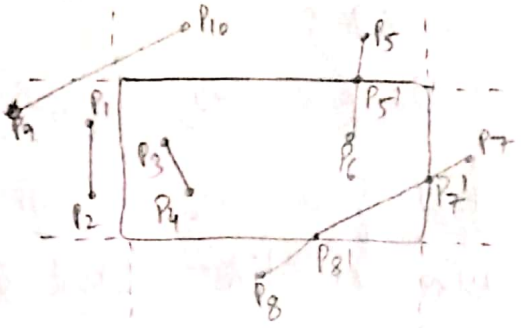
P_7' - 0000 (zero)

AND = 0000 (zero)

P_9 & P_{10}: P_9: 0001 - (non-zero)

P_{10}: 1000 - (non-zero)

AND: 0000 - (zero)



Finding intersecting Points:

(x_1, y_1) (x_2, y_2)

① y Value of vertical lines:

(x, y) intersecting Point

$$m = \frac{y - y_1}{x - x_1}$$

$$\Rightarrow y - y_1 = m(x - x_1)$$

$$\Rightarrow \boxed{y = y_1 + m(x - x_1)}$$

$x = x_{min}$ left boundary

$x = x_{max}$ right boundary

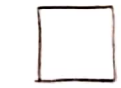
② x Value of horizontal lines:

$$m = \frac{y - y_1}{x - x_1}$$

$$\Rightarrow m(x - x_1) = y - y_1$$

$$\Rightarrow x - x_1 = \frac{y - y_1}{m}$$

$$\Rightarrow \boxed{x = x_1 + \frac{y - y_1}{m}}$$



$y = y_{min}$ for bottom boundary

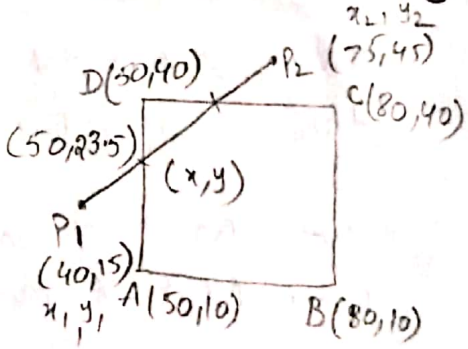
$y = y_{max}$ for upper boundary

Ex:

$P_1 = 0001$

$P_2 = 1000$

0000 (zero)



Intersecting Points: (x, y)

$xw_{min} = 50$, $xw_{max} = 80$

$yw_{min} = 10$, $yw_{max} = 40$

$y = y_1 + m(x - x_1)$

$= 15 + m(50 - 40)$

$= 23.563$

$(x, y) = (50, 23.563)$

$$\boxed{y = y_1 + m(x - x_1)}$$

$$\boxed{x = x_1 + \frac{1}{m}(y - y_1)}$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$= \frac{45 - 15}{75 - 40} = 0.85$$

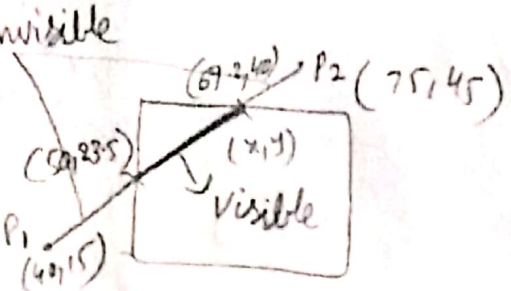
$(x,y) = (x,40)$

$x = x_1 + \frac{1}{m}(y - y_1)$

$= 75 + \frac{1}{m}(40 - 45)$

$= 69.2$

$(x,y) = (69.2, 40)$



Liang-Barsky Line clipping:

Based on time-interval can find the intersect Point.

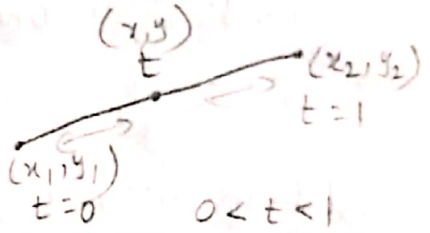
$x = t \cdot x_2 + (1-t)x_1$

$x = t \cdot x_2 + x_1 - x_1 \cdot t$

$= x_1 + t(x_2 - x_1)$

$\boxed{x = x_1 + t \cdot \Delta x}$ $0 < t < 1$

$\boxed{y = y_1 + t\,\Delta y}$ $0 < t < 1$



(x,y)

$x_{wmin} \le x \le x_{wmax}$

$y_{wmin} \le y \le y_{wmax}$

$x_{wmin} \le x_1 + t \cdot \Delta x \le x_{wmax}$

$y_{wmin} \le y_1 + t \cdot \Delta y \le y_{wmax}$

$x_1 + t \cdot \Delta x \ge x_{wmin}$

$x_1 + t \cdot \Delta x \le x_{wmax}$

$y_1 + t \cdot \Delta y \ge y_{wmin}$

$y_1 + t \cdot \Delta y \le y_{wmax}$

$\boxed{t \cdot P_K \le q_K}$ Derive inequality $K = 1,2,3,4$

$t \cdot \Delta x \ge x_{wmin} - x_1$ — (1)

$t \cdot \Delta x \le x_{wmax} - x_1$ — (2)

$t \cdot \Delta y \ge y_{wmin} - y_1$ — (3)

$t \cdot \Delta y \le y_{wmax} - y_1$ — (4)

Eq (1) & (3) → Multiplied with -1

$-t \cdot \Delta x \le x_1 - x_{wmin}$

$t \cdot \Delta x \le x_{wmax} - x_1$

$-t \cdot \Delta y \le y_1 - y_{wmin}$

$t \cdot \Delta y \le y_{wmax} - y_1$

$P_1 = -\Delta x$, $P_2 = \Delta x$, $P_3 = -\Delta y$

$P_4 = \Delta y$

$q_1 = x_1 - x_{wmin}$, $q_2 = x_{wmax} - x_1$

$q_3 = y_1 - y_{wmin}$, $q_4 = y_{wmax} - y_1$

If $\boxed{P_K = 0}$ (line is ||el to window)

$q_K < 0$ (line is outside)

$q_K > 0$ (line is inside/partial inside)

$q_K = 0$ (with in boundary/Partial)

If $\boxed{P_K < 0} \rightarrow t_1$

$t_1 = \max\left(0, \frac{q_K}{P_K}\right)$

$\boxed{x = x_1 + t_1 \cdot \Delta x}$

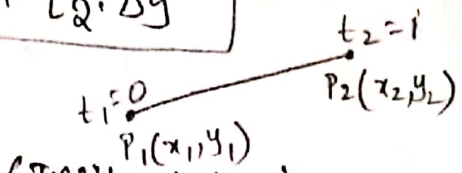
$\boxed{y = y_1 + t_1 \cdot \Delta y}$

If $P_K > 0 \rightarrow t_2$

$t_2 = \min(1, q_K/P_K)$

$x = x_1 + t_2 \cdot \Delta x$

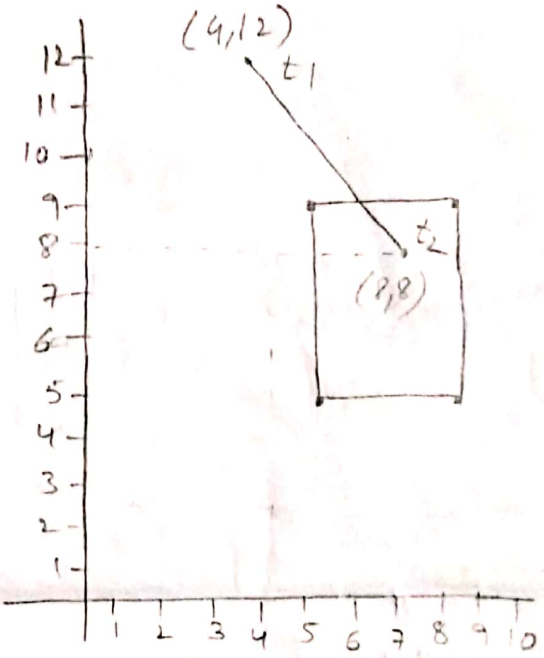
$y = y_1 + t_2 \cdot \Delta y$



t_1 changes

$t_1 \neq 0$ (Inside window)

t_2 changes $t_2 \neq 1$ (outside Window)



Ex: $xw_{min} = 5$, $xw_{max} = 9$, $yw_{min} = 5$

$yw_{max} = 9$

line $P_1(4, 12)$

$P_2(8, 8)$

$P_1 = -\Delta x$	$q_1 = x_1 - xw_{min}$
$P_2 = \Delta x$	$q_2 = xw_{max} - x_1$
$P_3 = -\Delta y$	$q_3 = y_1 - yw_{min}$
$P_4 = \Delta y$	$q_4 = yw_{max} - y_1$

$\Delta x = x_2 - x_1 = 8 - 4 = 4$, $\Delta y = y_2 - y_1 = 8 - 12 = -4$

$P_1 = -4$	$q_1 = 4 - 5 = -1$
$P_2 = 4$	$q_2 = 9 - 4 = 5$
$P_3 = 4$	$q_3 = 12 - 5 = 7$
$P_4 = -4$	$q_4 = 9 - 12 = -3$

$P_K < 0$ (P_1, P_4)

$t_1 = \max(0, \frac{q_K}{P_K})$

$= \max(0, \frac{q_1}{P_1}, \frac{q_4}{P_4})$

$= \max(0, \frac{-1}{-4}, \frac{-3}{-4})$

$t_1 = \frac{3}{4}$

$P_K > 0$ (P_2, P_3)

$t_2 = \min(1, \frac{q_2}{P_2}, \frac{q_3}{P_3})$

$t_2 = \min(1, \frac{5}{4}, \frac{7}{4})$

$t_2 = (1,$

$t_2 = 1$

$x = x_1 + t_1 \cdot \Delta x$

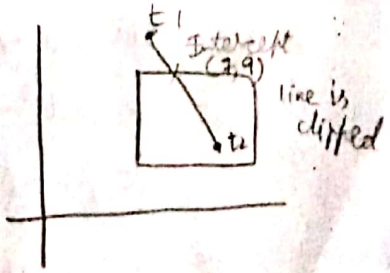
$= 4 + (\frac{3}{4}) \cdot 4 = 7$

$y = y_1 + t_1 \cdot \Delta y$

$= 12 + (\frac{3}{4})(-4)$

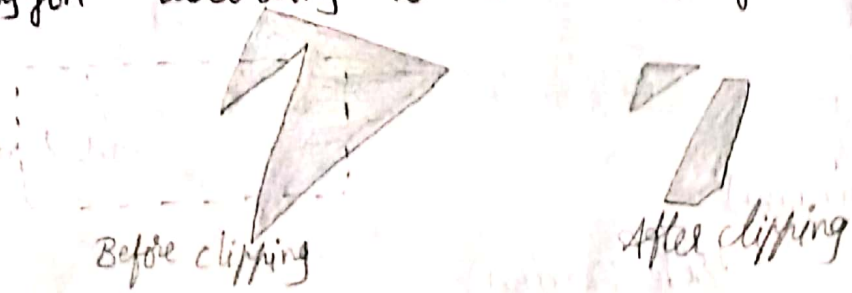
$= 9$

$t = \frac{3}{4} \Rightarrow (x, y) = (7, 9)$

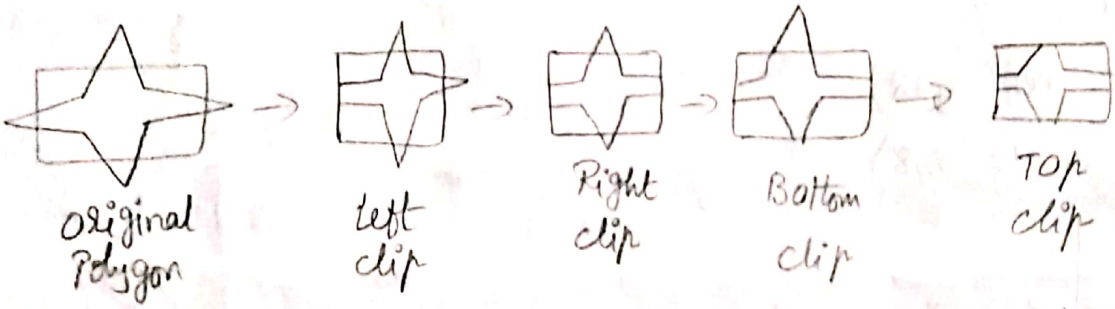


Polygon clipping:

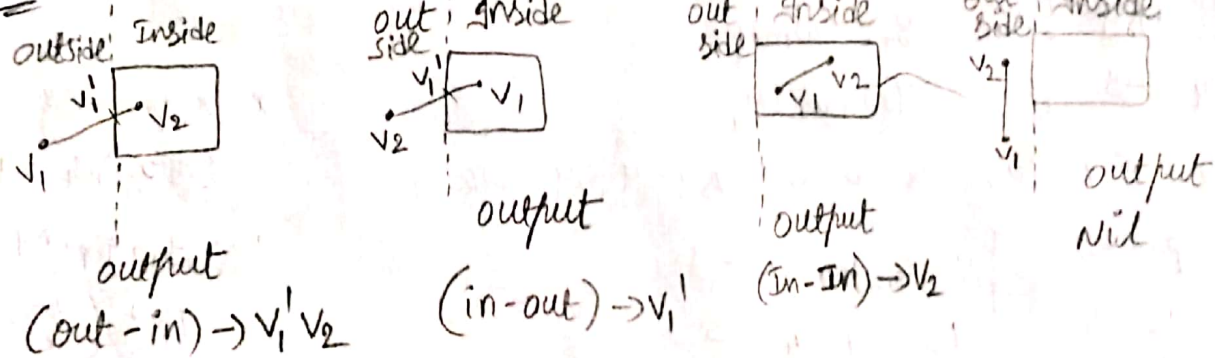
* To clip Polygon, we need to modify the line-clipping procedure.
* Polygon clipping is a process of clipping or cutting a Polygon according to our clipping window.



Sutherland-Hodgeman Polygon clipping:



4 Cases:



Ex:

Left clip:

$v_1 v_2$ – In-In – v_2

$v_2 v_3$ – In-out – v_2'

$v_3 v_1$ – out-In – $v_3' v_1$

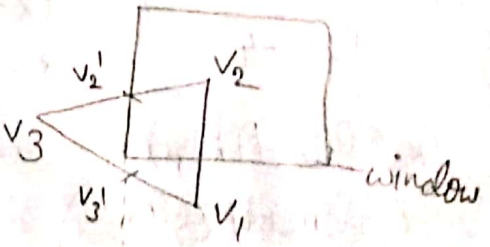
Right clip:

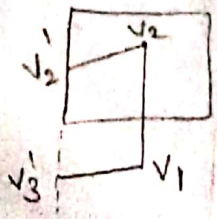
$v_1 v_2$ – v_2

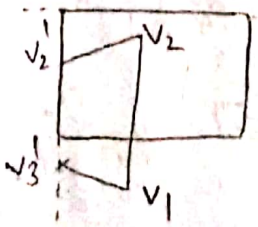
$v_2 v_2'$ – v_2'

$v_2' v_3'$ – v_3'

$v_3' v_1$ – v_1

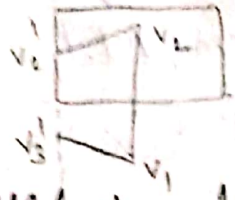






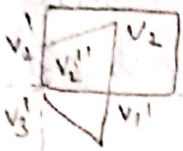
Top clip:

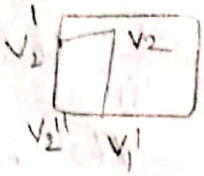
V_1V_2 – V_2
V_2V_2' – V_2'
$V_2'V_3'$ – V_3'
$V_3'V_1$ – V_1



Bottom-clip:

V_1V_2 – out-In – $V_1'V_2$
V_2V_2' – In-In – V_2'
$V_2'V_3'$ – In-out – V_2''
$V_3'V_1$ – out-out – null





Final output

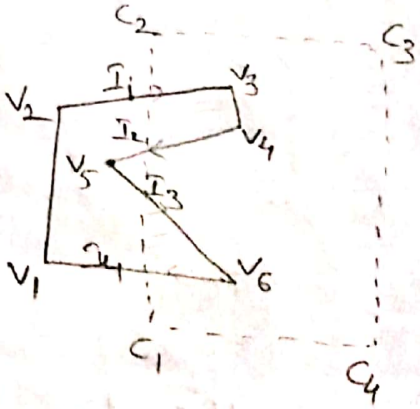
* used to clip only convex polygon correctly.

Weiler-Atherton Polygon clipping:

Convex Polygon
→ Interior angle < 180°

Concave Polygon
→ one or more interior angles > 180°.

* Vertex-processing procedures for window boundaries are modified so that concave polygons are displayed correctly.
* This clipping procedure was developed as a method for identifying visible surfaces.



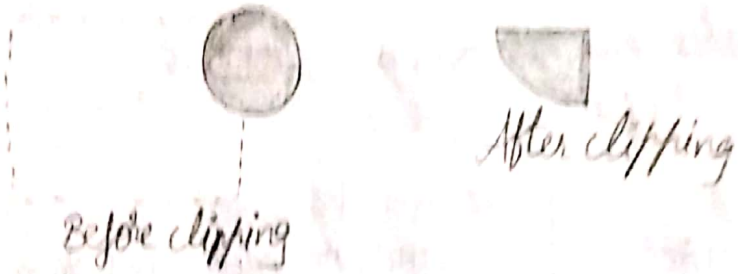
V - vertices
C - clip Polygon
I - Intersection

Subject	clip
V_1	C_1
V_2	I_4
Start I_1	I_3
V_3	I_2
V_4	I_1
I_2	C_2
V_5	C_3
I_3	C_4
V_6	
I_4	

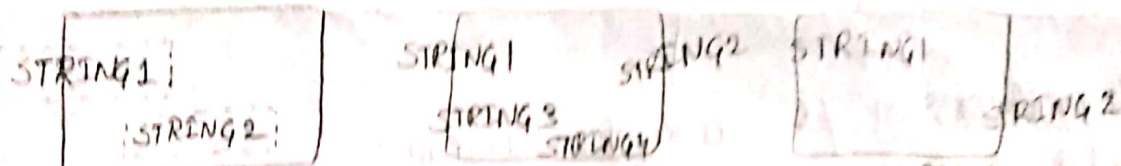
Curve clipping:

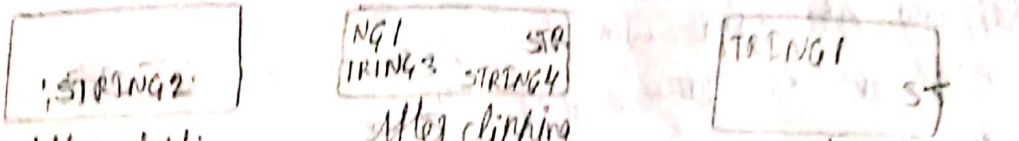
Curve clipping procedures will involve nonlinear equations, however, this requires more processing than for objects with linear boundaries.

* If the bounding rectangle for the object is completely inside the window, we save the object
* If the rectangle is determined to be completely outside the window, we discard the object.



Text clipping:





* An Alternative to rejecting an entire character string that overlaps a window boundary is to use all-or-none character-clipping strategy.
* Discarding the characters that are not completely inside the window.
* The boundary limits of individual characters are compared to the window. Any character that either overlaps or is outside a window boundary is clipped.