Computer Graphics Syllabus

01 Graphic systems

Display devices, Bit map and vector graphics resolution, Aspect ratio, physical input and output devices, Display processors graphics Software Coordinate representation graphics function and standards.

02 Scan conversion and output primitives

Scan converting the point, scan converting the straight line - Bresenham's line Algorithm, Scan converting a circle - Defining a circle, Bresenham's circle Algorithm Region filling - introduction, flood filling, boundary filling, side effects of Scan conversion Graphic primitives inc, Point plotting, line drawing algorithms - DDA algorithms, Bresenham's line drawing line algorithm, circle-generating algorithms.

03 Two-Dimensional Transformations

Basic transformation - translation, Scalling, rotation, matrix representations and homogeneous coordinates, composite transformation - scalling relative to a fixed pivot, rotation about a fixed pivot point, general transformation equations, other transformation - reflection and shearing

01 Algorithm in Computer Graphics

Line Drawing Algorithm :-

01 ~~DDA :-~~

Methods of line drawing :-

A line can be generated on the screen by turning on a row or column pixels. If we draw a vertical line the columns of the pixels are turned on and if we draw a horizontal line, the rows of the pixels are turned on. The line can be drawn with the help of following methods

(i) DDA (Digital differential Analyzer)

(ii) Bresenham's line algorithm.

DDA :- In computer graphics, a digital differential analyzer (DDA) is hardware or software used for interpolation of variables over an interval between start and end point.

DDA are used for rasterization of lines, ~~trian~~ triangles and polygons.

we have four characterstics for line drawing algorithms

- Line should be terminated
- The dencity of line should be constant
- The dencity of line should be independent of length and angle.
- Line and point should be specified.

The cartesian slope intercept equation for straight line is.

$$y = mx + c$$

where m = slope of the line $= (y_2 - y_1)/(x_2 - x_1)$

c is y-intercept and is calculated as

$$c = y_1 - mx_1$$

There are some case for line drawing algorithm.

Case 01 - If $m < 1$

x ↑↑ more than y

$$(x_{k+1}, y_{k+1}) = (x+1, \text{Round}(y+m))$$

$x_n = x_0 + 1$ $\quad (x, \text{Round}(y))$

$y_n = y_0 + m$

we repeat this process until $x_1 = x_2$

case-02 - If $m > 1$

y ↑↑ more than x

$$(x_{k+1}, y_{k+1}) = \left(\text{ROUND}\left(x_1 + \frac{1}{m}\right), y_1 + 1\right)$$

$x_n = x_0 + \frac{1}{m}$

$y_n = y_0 + 1$ $\quad (\text{Round}(x), y)$

we repeat this process until $y_1 = y$

Case-3 $m = 1$

$$x_n = x + 1$$

$$y_n = y + 1$$

DDA Algorithm Disadvantage

Key disadvantage :- It rolls on floating point operations to computer pixel positions.

Implications :-

(i) computationally inefficient because floating point operation are slow.

(ii) Round-off errors accumulate Producing incorrect drawing.

(iii) The algorithm is orientation dependent. Therefore the end point accuracy is poor.

Example of line drawing algorithm.

we have two quardinate

(10, 8) (15, 9)

$x_1 = 10$ $x_2 = 15$

$y_1 = 6$ $y_2 = 9$

Slope $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3}{5} = 0.6$

$m < 1$

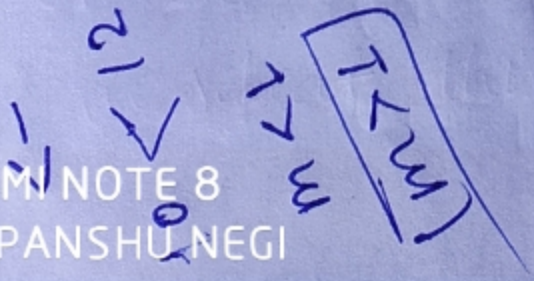
$x_n = x_1 + 1$

$y_n = y_\phi + m$

$(x_{K+1}, y_{K+1}) = (x_1 + 1, \text{Round}(y_1 + m))$

$X1 = X2$

x_K	y_K	(x_{K+1}, y_{K+1})
10	6	(10, 6)
10+1=11	6+0.6=6.6	(11, 6.7)
11+1=12	6.6+0.6=7.2	(12, 7.2)
12+1=13	7.2+0.6=7.8	(13, 8)
13+1=14	7.8+0.6=8.4	(14, 8.4)
14+1=15	8.4+0.6=9	(15, 9)



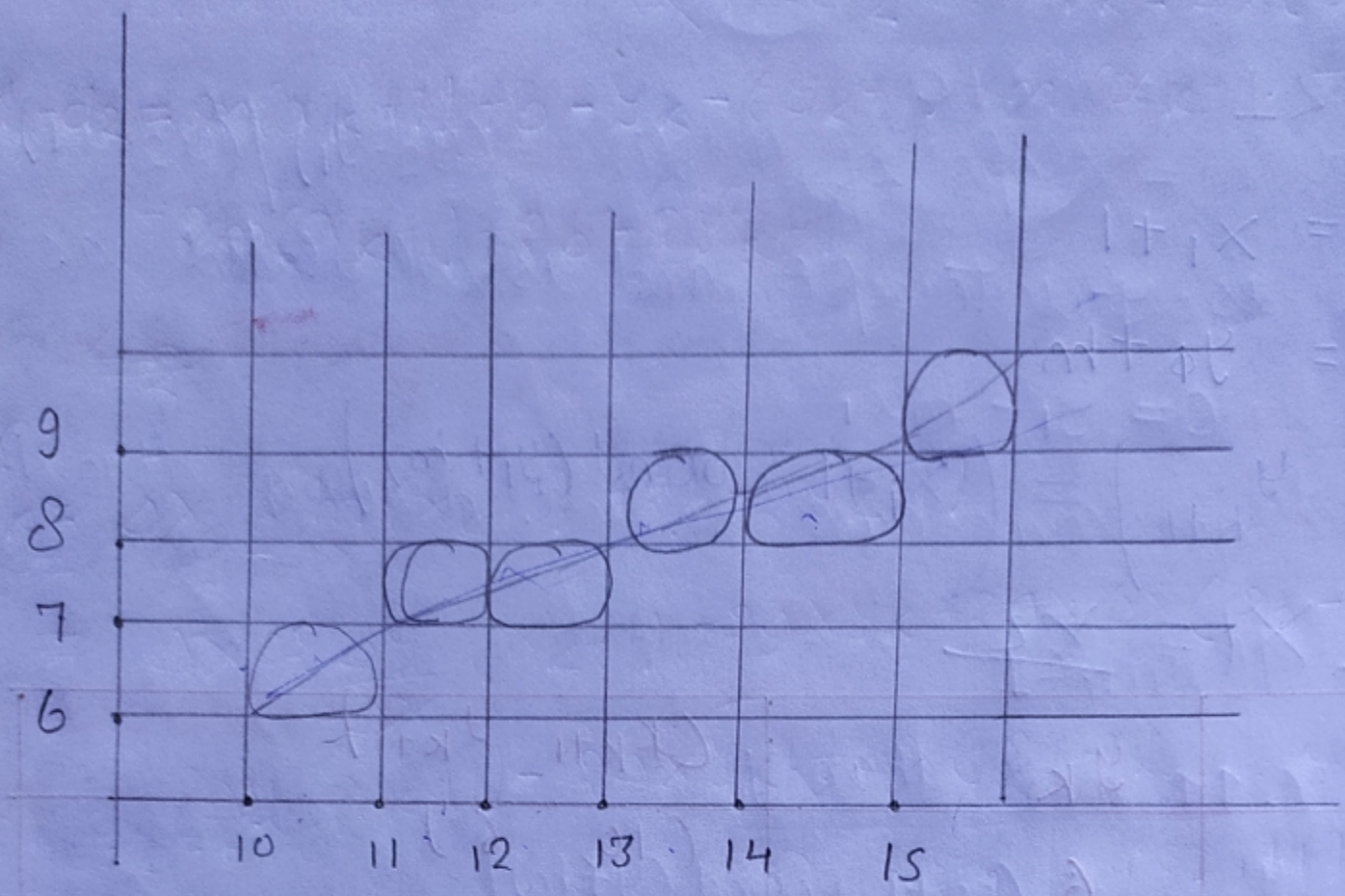


fig:- DDA Algorithm for line generation.

(ii) Breshenham's Algorithm for line drawing :-

we have four characterstics for line drawing algorithm.

(i) Line should be terminated.

(ii) The density of line should be constant.

(iii) The density of line should be independent of length and angle

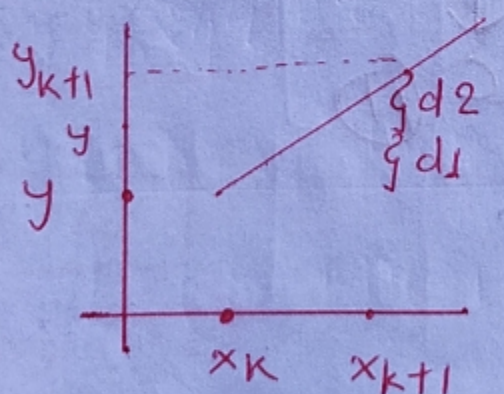
(iv) Line and point should be specified.

~~Bhc~~

Bresenham line drawing algorithm is more accurate and efficient method of line drawing. In this method, the floating point arithmetic is avoided to a great extent and only the incremental integer calculations are done. In this method, ~~for for~~ first objective is chosen to the closest pixel position of the line for its path of integer values. As integer arithmetic is much faster than floating point arithmetic so this algorithm is faster than DDA algorithm. Further more this algorithm doesn't require any multiplication or division.

This algorithm can be divided into two parts one for slope $|m|<1$ and another for slope $|m|>1$

decision parameter → It is directly proposanal to difference of sepration between two pixel value from actual line segment.



slope intercept formula

$$y = m(x_{k+1}) + b \quad \text{— (1)}$$

$$d_1 = y - y_k$$

$$d_1 = m(x_{k+1}) + b - y_k$$

$$d_1 = (y_{k+1}) - y$$

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

∴ decision parameter $\propto d_1 - d_2$ $\quad P_k = \Delta x(d_1 - d_2)$

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

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

$\Rightarrow$

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

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

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

$P_k < 0 \quad d_1 < d_2 \Rightarrow (x_{k+1}, y_k)$

$P_k > 0 \quad d_1 > d_2 \Rightarrow (x_{k+1}, y_{k+1})$

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

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

$P_k = 2\Delta y x_k - 2\Delta x y_k + c$ — (ii)

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

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

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

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

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

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

$P_{k+1} - P_k$

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

for Equation-01 $P_k = 2\Delta y - 2\Delta x y_k + 2\Delta y + \Delta x(2b-1)$

$$P_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x(2b-1)$$

$x_0 = 0$

$y_0 = 0$

$$P_0 = 0 - 0 + 2\Delta y + \Delta x(2b-1)$$

$$P_0 = 2\Delta y + \Delta x(2b-1)$$

$\because y = mx + b$

$b = y - mx$

for P_0

$x, y = 0$

$$P_0 = 2\Delta y + \Delta x(2(y - mx) - 1)$$

$$P_0 = 2\Delta y + \Delta x(2(0 - m \times 0) - 1)$$

$$P_0 = 2\Delta y + \Delta x(0 - 1)$$

$$P_0 = 2\Delta y - \Delta x \quad \text{— (IV)}$$

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

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

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

$P_k > 0 \Rightarrow (x_{k+1}, y_{k+1})$

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

Note 01 – If the slope is < 1 we need to ~~find~~ sample through x interval

If slope is > 1 we need to sample through y interval

Example 1

End points (20,10) and (30,18)

(x_1, y_1) (x_2, y_2)

$$\text{Slope} = \frac{\Delta y}{\Delta x} = \frac{8}{10} = 0.8$$

$m < 1$

$0.8 < 1$

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

$= 16 - 10$

$P_0 = 6$

$K=0, P_k = P_0 = 6 > 0$

$(x_{k+1}, y_{k+1}) = (20+1, 10+1)$

$K=1, P_k = P_1 = P_k + 2\Delta y - 2\Delta x$

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

$= 6 + 16 - 20$

$= 2$

$K=1, P_k = P_1 = 2 > 0$

$(x_{k+1}, y_{k+1}) = (21+1, 11+1)$

$K_2 \quad P_k = P_2 = P_k + 2\Delta y - 2\Delta x$

$= P_1 + 2\Delta y - 2\Delta x$

$= 2 + 16 - 20$

$= -2$

$K=2, P_k = P_2 = -2 < 0$

$(x_{k+1}, y_{k+1}) = (22+1, 12)$

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)

we need to follow this step Δx time because we sampling to x axis.

~~K 3 P~~

$K=3,\ P_k = P_3 = P_k + 2\Delta y$

$-2+16$

$= 14$

$K=3,\ P_k = P_3 = 14 > 0$

$(x_{k+1}, y_{k+1}) = (24, +13)$

$K=4,\ P_k = P_4 = P_k + 2\Delta y - 2\Delta x$

$= 14 + 16 - 20$

$K=4,\ P_k = P_3 = 10 > 0$

$(x_{k+1}, y_{k+1}) = (24+1,\ 13+1)$

$= (25, 14)$

$K=5,\ P_k = P_5 = Pk + 2\Delta y - 2\Delta x$

$10 + 16 - 20$

$P_5 = 6 > 0$

$(x_{k+1}, y_{k+1}) = (26, 15)$

$K=6,\ P_k = 6 = P_k + 2\Delta y - 2\Delta x$

$6 + 16 - 20$

$P_6 = 2 > 0$

$(x_{k+1}, y_{k+1}) = (26+1,\ 15+1)$

$K=7,\ P_k = P_7 = P_k + 2\Delta y - 2\Delta x$

$2 + 16 - 20$

$P_7 = -2 < 0$

$(x_{k+1}, y_{k+1}) = (27+1, 16)$ $(28, 16)$

$K=8,\ P_k = 8$

$= Pk + 2\Delta y$

$P_8 = 14 > 0$

$(x_{k+1}, y_{k+1}) = (29, 17)$

$K=9,\ P_k = P_9$

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

$14 + 16 - 20$

$P_9 = 10 > 0$

$(x_{k+1}, y_{k+1}) = (30, 18)$

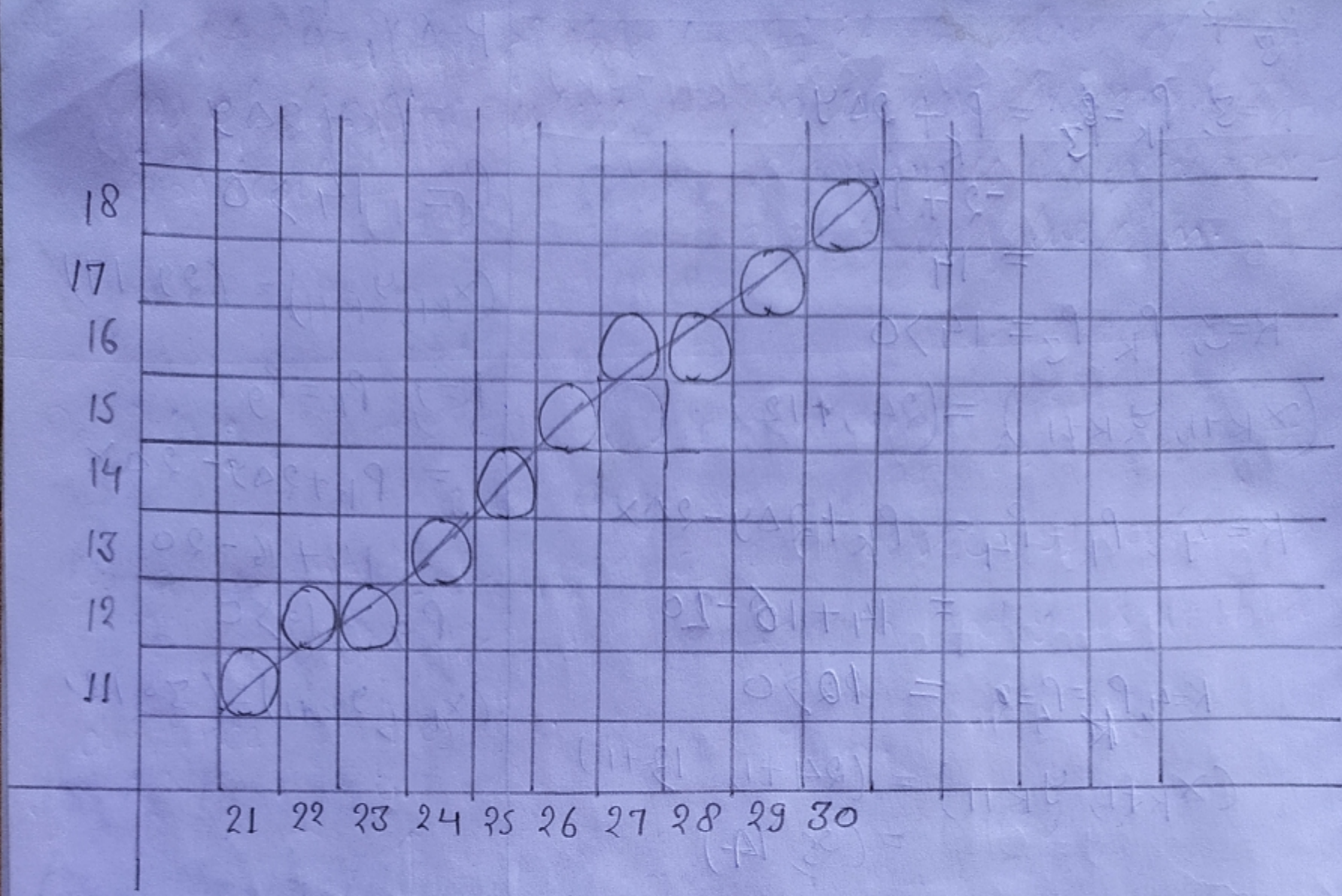


fig- Bresenham's algo for line drawing.

What are the advantages of Bresenham's algorithm over DDA?

or

Compare Bresenham's and DDA line drawing algorithm.

DDA and Bresenham algorithm both are efficient line drawing algorithm.

Bresenham's algorithm has following advantages on DDA.

01 DDA uses float number and uses operators such as division and multiplication in its calculation, Bresenham's algorithm uses ints and only uses addition and subtraction.

02. Due to the use of only addition, subtraction and bit shifting Bresenhams algorithm is faster than DDA in producing the line.

03. Fixed point DDA algorithms are generally superior to Bresenhams algorithm on modern computers. The reason is that Bresenhams algorithm uses a condition branch in the 3 loop and this result in frequent branch mis predictions in the CPU.

04- Fixed point DDA does not require conditional JUmp we can compute several lines in parallel with SIMD (Single instruction multiple data ~~bbbb~~ techniques)..

The main differences between the DDA and Bresenham's are as under

S-NO	DDA	Bresenhams
01	It is less accurate and less efficient line drawing algorithm that is because we have to round of values	It is more accurate and more efficient line drawing algo that is because there is no need to round off the values.
	It is only applicable for lines	It is applicable for lines as well as curves.

03 In it the value of slope (m) of the line is less important	In it the value of slope m is more important. The algorithm is designed according to the value of m.

Point :- A position in a plane is known as point and any point can be represented by any ordered pair of numbers (x, y), where x is horizontal distance from origin and y is vertical distance from origin.

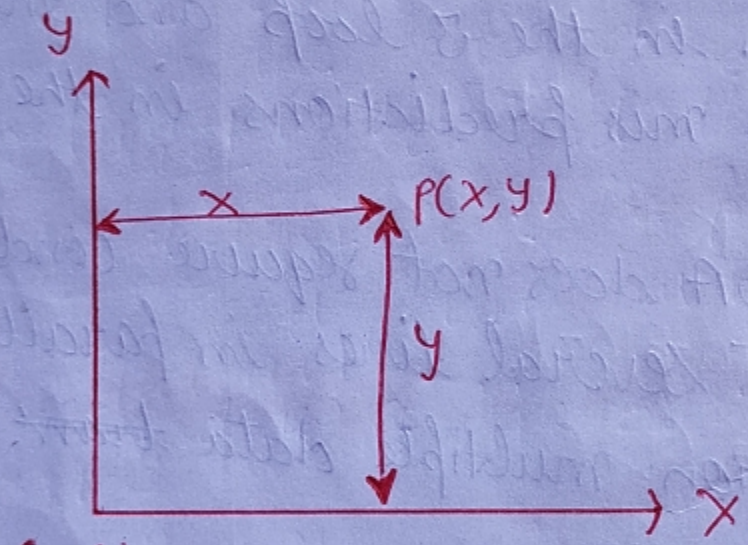
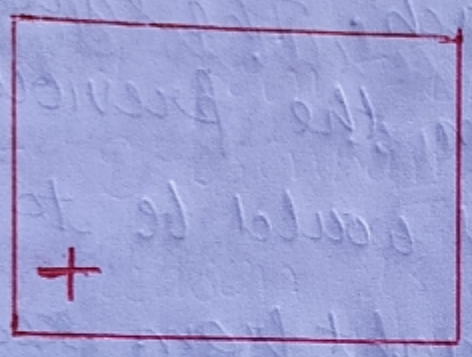


Fig 1 Point.

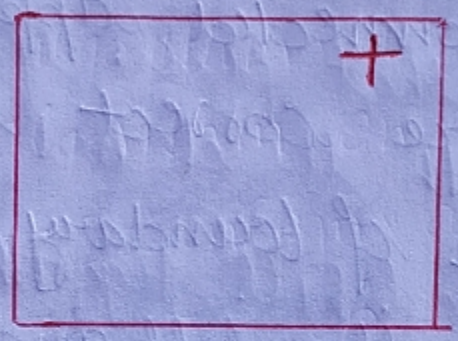
Point plotting :- Point plotting gives the user the capability of selecting a particular point on the screen. This is usually done by a combination of a locator and a button. The locator is used to tell which point the user selects. The button indicates when the locator is correctly positioned. The algorithm to perform point

plotting must await the button event; as soon as it occurs, the locator may be read. The selection of points can be used in many ways.

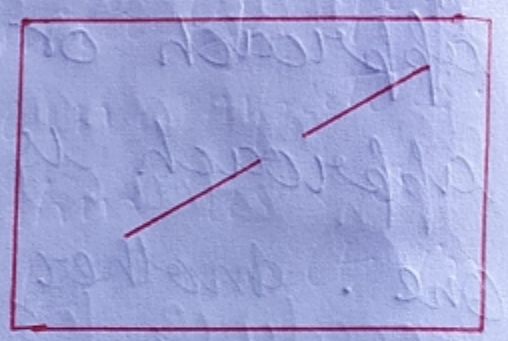
for example :- It can give end points of line segments, positions for text or translations of picture segments.



(i) Plot a point



(ii) plot a second point



Connect the points with a line segment.

Boundary filling :- Boundary filling is the process to fill the area that starts at a point inside a region and points the interior of the boundary. If the boundary is specified in a single colour, the fill algorithm proceeds outward pixel by pixel until the boundary colour is encountered. This method is called the boundary fill method. This method is particularly useful in interactive painting packages, where interior point are easily selected.

Boundary fill algorithm :-

This algorithm is similar to flood fill algorithm. In this, the adjacent pixels are checked for boundary colour has reached or not. If pixel colours are not that of boundary colour, the pixel is coloured. Here also we make use of seed point. we may use either 4-connected approach or 8-connected approach. The latter approach is more correct than the previous one. Another way of boundary fill would be to check the pixels to the left and right from seed point for boundary colour and then draw a line across these. Next look for pixels from top to bottom from seed point having boundary colour and then draw a line or run of pixels.

4-Connected

8-Connected

Procedure for Boundary fill :-

```
Procedure boundaryfill 4(x, y, fcolour, bcolour :
                                        integer)
int current;
current = get pixels (x, y);
if (current != bcolour) && (current != fcolour)
{
set pixel (x, y, fcolour);
boundaryfill 4 (x+1, fcolour, bcolour);
boundaryfill 4 (x-1, fcolour, bcolour);
boundaryfill 4 (x, y+1, fcolour, bcolour);
boundaryfill 4 (x, y-1, fcolour, bcolour);
}
```

Flood Filling :- Flood filling is the process of filling the area of the region. The area can be total number of pixels by boundary pixels outlining the region. In the flood fill algorithm, we start with the initial pixel, from this pixel the algorithm ~~insef~~ inspects all the surrounding eight pixels and fill them. This process is repeated till all the pixels inside the region are inspected and filled.

Flood-fill algorithm :-

In this algorithm, we take a point say P(x, y) which is interior to the region. This point is known as a seed point. Then, using a 4-connected approach or 8-connected approach the entire area can be filled by a specified colour. we start with seed point, check all 4-connected or 8-connected pixels if they have a colour which is that of boundary then do not consider that pixel. if the colour is different then colour it with the desired colour.

Procedure for Flood Fill Algorithm.

```
Procedure floodfill4 (x, y, fcolour, ocolour);
If (getpixel (x, y) = ocolour)
{
setpixel (x, y, fcolour);
floodfill4(x+1, y, fcolour, ocolour);
floodfill4(x-1, y, " , " );
floodfill4 (x, y+1, " , " );
floodfill4 (x, y-1, " , " );
}
```

SIDE EFFECTS OF SCAN CONVERSION

Scan conversion is essentially a systematic approach of mapping objects that are defined in continuous space to their discrete approximation. The various forms of distortion that results from this operation are collectively referred to as the aliasing effects of scan conversion.

o| Staircase :- A common example of aliasing effects is the staircase or jagged appearance we can see when scan-converting a primitive such as a line or a circle. we also see the stair steps or "jaggies" along the border of a filled region.

03. TWO DIMENSIONAL TRANSFORMATION

Two Dimensional transformation is the process by which we can change the shape, position, and direction of any object with respect to any co-ordinate system on background by translation, rotation, scaling and ~~reflection~~ reflection. Basically transformation is described in two categories.

01 Geometric Transformation

02 Co-ordinate Transformation

When object itself is moved relative to a stationary Co-ordinate system or background, then it is referred to as Geometric Transformation. and applied to each point ~~to object~~. of the object.

And while the Co-ordinate system is moved relative to the object and object is held stationary then this process is termed as Co-ordinate transformation

Geometric Transformation :-

Every object is assumed as a set of points. Every object point P has co-ordinates (x, y) and so the object is the sum of total of all-cordinates points. If any object is transformed to a new location then co-ordinates of new location can be obtained by the application of geometric transformation.

01+ Translation :-

In Translation, an object is displaced at a given distance and direction from its original position i.e from one co-ordinate location to another without changing its shape. Let P(x, y) be the original co-ordinate location and P'(x', y') be the new co-ordinate location and t_x and t_y are the translation distance along x-axis and y-axis, then the equation become :

$$x' = x + t_x$$

$$y' = y + t_y$$

The equation can be represented in matrix form as:

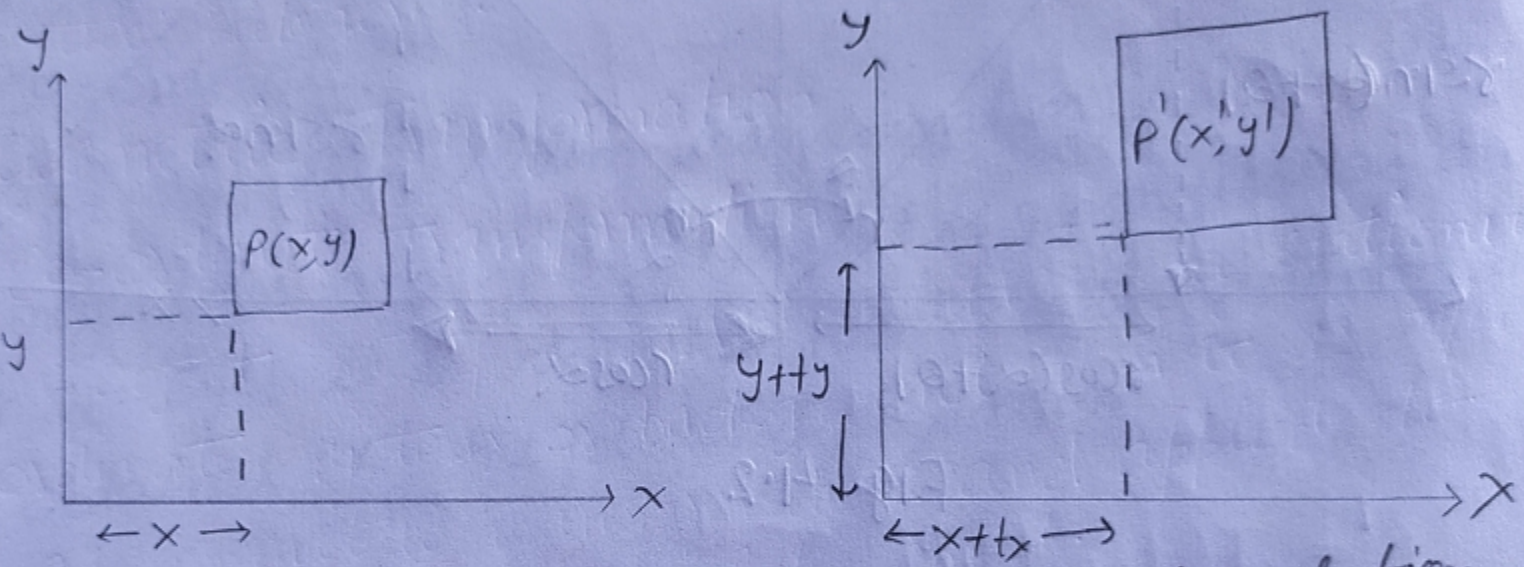
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$P' = P + T$$

where $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$

and $P = \begin{bmatrix} x \\ y \end{bmatrix}$

and $T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$



(a) Before translation (b) After translation

Fig :- Translation

Rotation :- In this transformation, object is rotated θ° about origin. we take θ positive for clockwise, otherwise negative and co-ordinates of new point is given by the following equation.

$$P' = R_\theta(P) \quad \text{——} (1)$$

Let co-ordinates before rotation of point P are (x, y) and after clockwise rotation by amount of θ° are $P'(x', y')$. If initially P is at an angle θ° from x-axis, then.

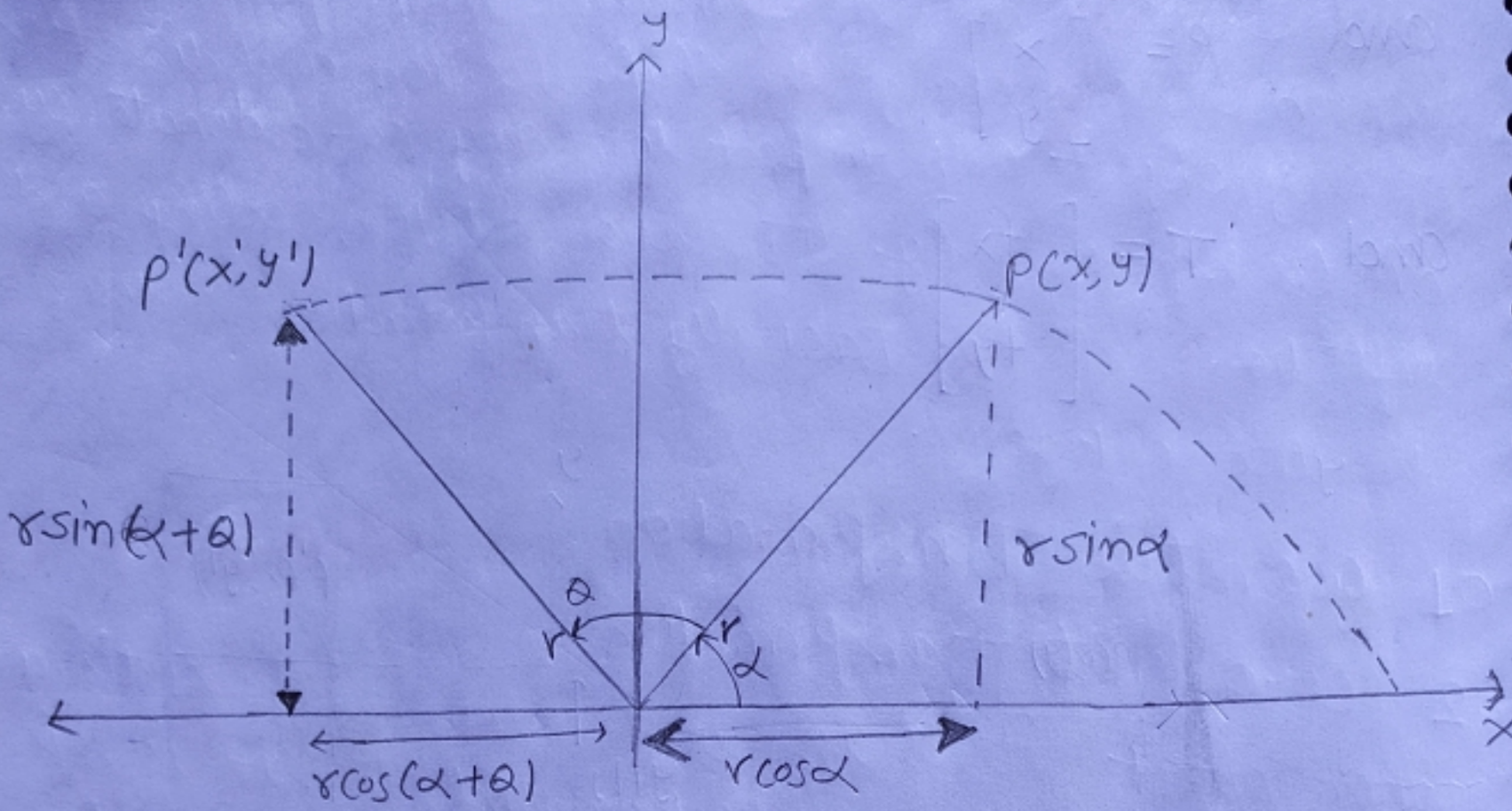


Fig 4.2

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

since $x = r\cos\alpha$ and $y = r\sin\alpha$

therefore $x' = x\cos\theta - y\sin\theta$

and $y' = r\sin\alpha\cos\theta + r\sin\theta\cos\alpha$

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

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Scaling Transformation :-

Scaling is the process of changing the size and properties of the image. Alternatively, it is the process of expanding or compressing the dimensions of an object. There are two factors used in scaling transformation i.e S_x and S_y, where S_x is scale factor for the x co-ordinate and S_y is scale factor for the y co-ordinate.

If $S_x = S_y$, then scaling transformation is said to be homogeneous and if $S_x = S_y > 1$, it is magnification and for $S_x = S_y < 1$ it is reduction. Points after rotation can be obtained by the following equation :

$$x' = S_x \cdot x$$

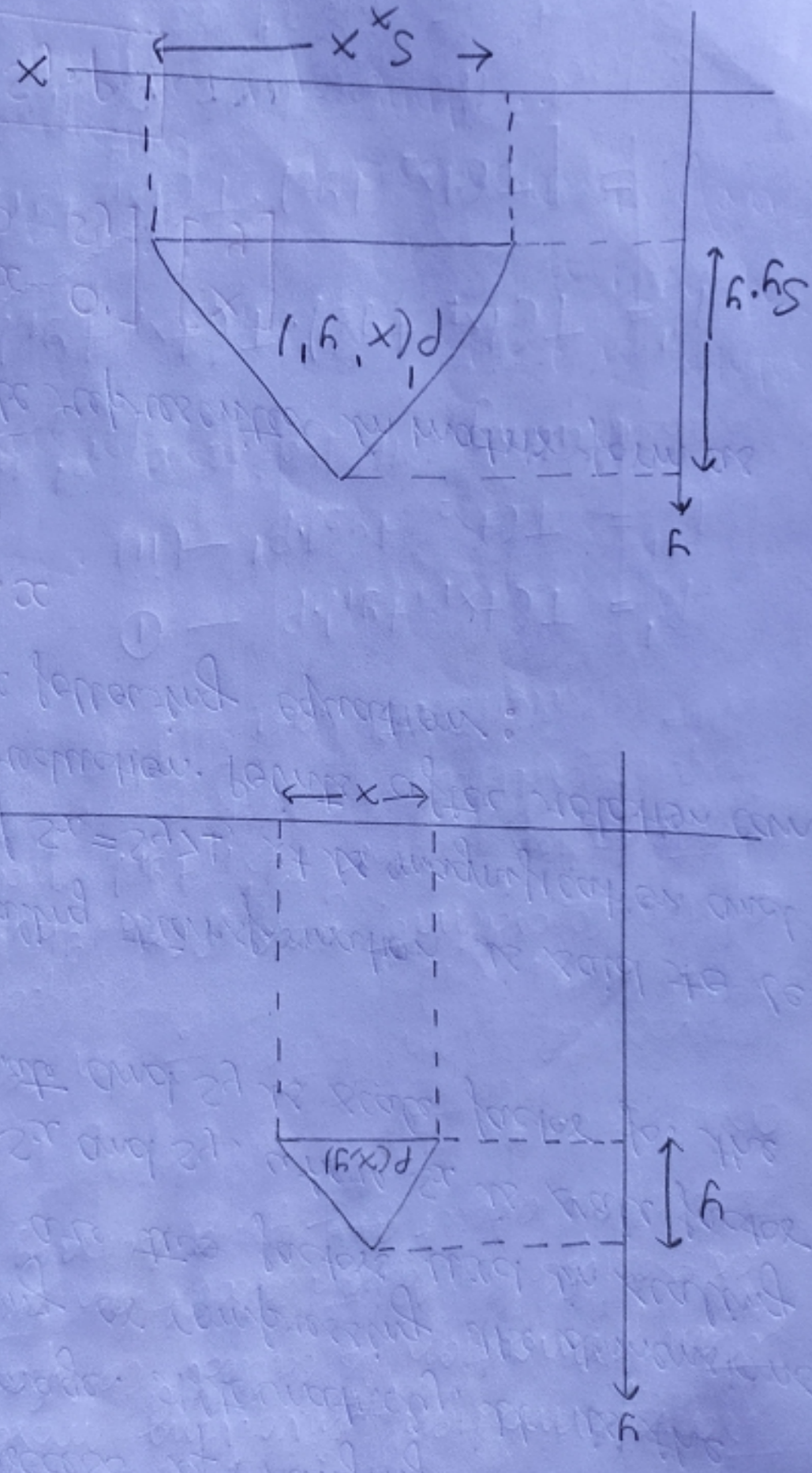
$$y' = S_y \cdot y$$

The equation can be represented in matrix form as

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

$$\boxed{P' = S_x \, S_y \cdot P}$$

Fig :- Scaling Transformation



Mirror Reflection about any axis :-

Reflection produces a mirror image of an object. If we assume any axis as a mirror then object has a mirror image or reflection. Because the reflection of object P is obtained at the same distance from the axis as P. Co-ordinates of P' are given by the following expression if axis is x-axis.

Now, $P' = M_x(P)$

where $x' = x$

and $y' = -y$

or $M_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

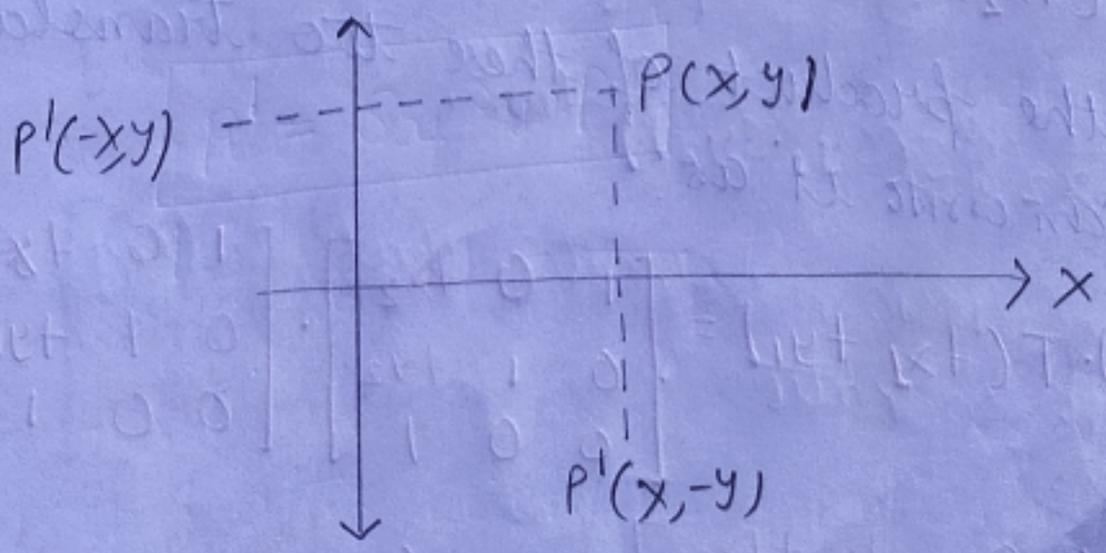
Similarly, reflection about an y-axis

$P' = M_y(P)$

where $x' = -x$

and $y' = y$

and $M_y = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$



mirror reflection

COMPOSITE TRANSFORMATION

Transformation involving more than one basic transformation is called composite transformation. For example relation about an arbitary point, mirror reflection about ~~any~~ an arbitrary line rather than any axis etc.

Composite Translation :- In composite translation, suppose the original position of the co-ordinate was $P(x_h, y_h, h)$. If two successive translation vectors (tx_1, ty_1) and (tx_2, ty_2) are applied to a co-ordinate position P, the final transformed location P'' is calculated as:-

$$P' = T(tx_1, ty_1) \cdot P \quad \text{— (i)}$$

$$P'' = T(tx_2, ty_2) \cdot P' \quad \text{— (ii)}$$

Substituting the value of P' in equation (ii) we get.

$$P'' = T(tx_2, ty_2) \cdot T(tx_1, ty_1) \cdot P$$

$$= [T(tx_2, ty_2) \cdot T(tx_1, ty_1)] \cdot P$$

If we find the product of these two translation vector we can write it as!

$$T(tx_2, ty_2) \cdot T(tx_1, ty_1) = \begin{bmatrix} 1 & 0 & tx_2 \\ 0 & 1 & ty_2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & tx_1 \\ 0 & 1 & ty_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & tx_1 + tx_2 \\ 0 & 1 & ty_1 + ty_2 \\ 0 & 0 & 1 \end{bmatrix} = T(tx_1 + tx_2, ty_1 + ty_2)$$

Or we can also write it as

$$T(+x_2, +y_2) \cdot T(+x_1, +y_1) = T(+x_1 + +x_2,\ +y_1 + +y_2)$$

The above expression indicates that two successive translations are additive in nature.

Composite Rotation :- Whenever the two successive rotation $R(\alpha)$ and $R(\beta)$ are applied to a point P, Produce the ~~transformation~~ transformed position.

$$P' = R(\alpha) \cdot P \quad \text{---(i)}$$

$$P'' = R(\beta) \cdot P' \quad \text{---(ii)}$$

Substitute the value of P' in eq (ii) we get

$$P'' = R(\beta) \cdot R(\alpha) \cdot P$$

$$= \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\alpha\cos\beta - \sin\alpha\sin\beta & -\cos\beta\sin\alpha - \sin\beta\cos\alpha & 0 \\ \sin\beta\cos\alpha + \cos\beta\sin\alpha & -\sin\alpha\sin\beta + \cos\alpha\cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$\therefore R(\beta) \cdot R(\alpha) = R(\alpha+\beta)$$

The above expression represents that for two successive rotation, the effect will be the sum of the angles of two rotational ~~matrix~~ matrices.

Composite scaling :- In composite scaling, if two successive scalings are applied to a coordinate position P, then the final position P' is calculated as;

$$P' = S(S_{x_1}, S_{y_1}) \cdot P$$

$$P'' = S(S_{x_2} \cdot S_{y_2}) \cdot P'$$

Substituting the value of P' in equation (2) we get

$$P' = S(S_{x_2}, S_{y_2}) \cdot S(S_{x_1} S_{y_1}) \cdot P$$

$$= \begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_{x_1} \cdot S_{x_2} & 0 & 0 \\ 0 & S_{y_1} \cdot S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

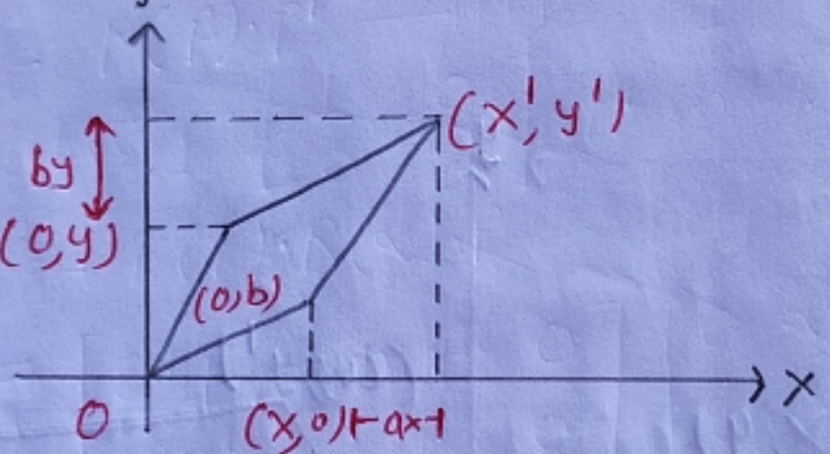
$$P'' = S(S_{x_1} \cdot S_{x_2}, S(S_{y_1} \cdot S_{y_2}))$$

The above expression indicates that scalings are multiplicative in nature.

Other Transformation :-

Shearing Transformation :-

When tangential force is applied to any object it distorts the shape of the object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called a Shear and such transformation is called Shearing transformation.



Shearing Transformation

When Shearing is applied in both x and y directions Simultaneously, it is termed as Simultaneous Shearing, represented by following matrix

$$= \begin{bmatrix} 1 & a \\ b & 1 \end{bmatrix}$$

where a is shearing in x-direction and b is Shearing in y direction.

When $b=0$, an x-direction shearing relative to the x-axis is produced with the transformation matrix.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{----(1)}$$

Where x', and y' can be represented in the form of following equations:

$$x' = x + ay$$
$$y' = y$$

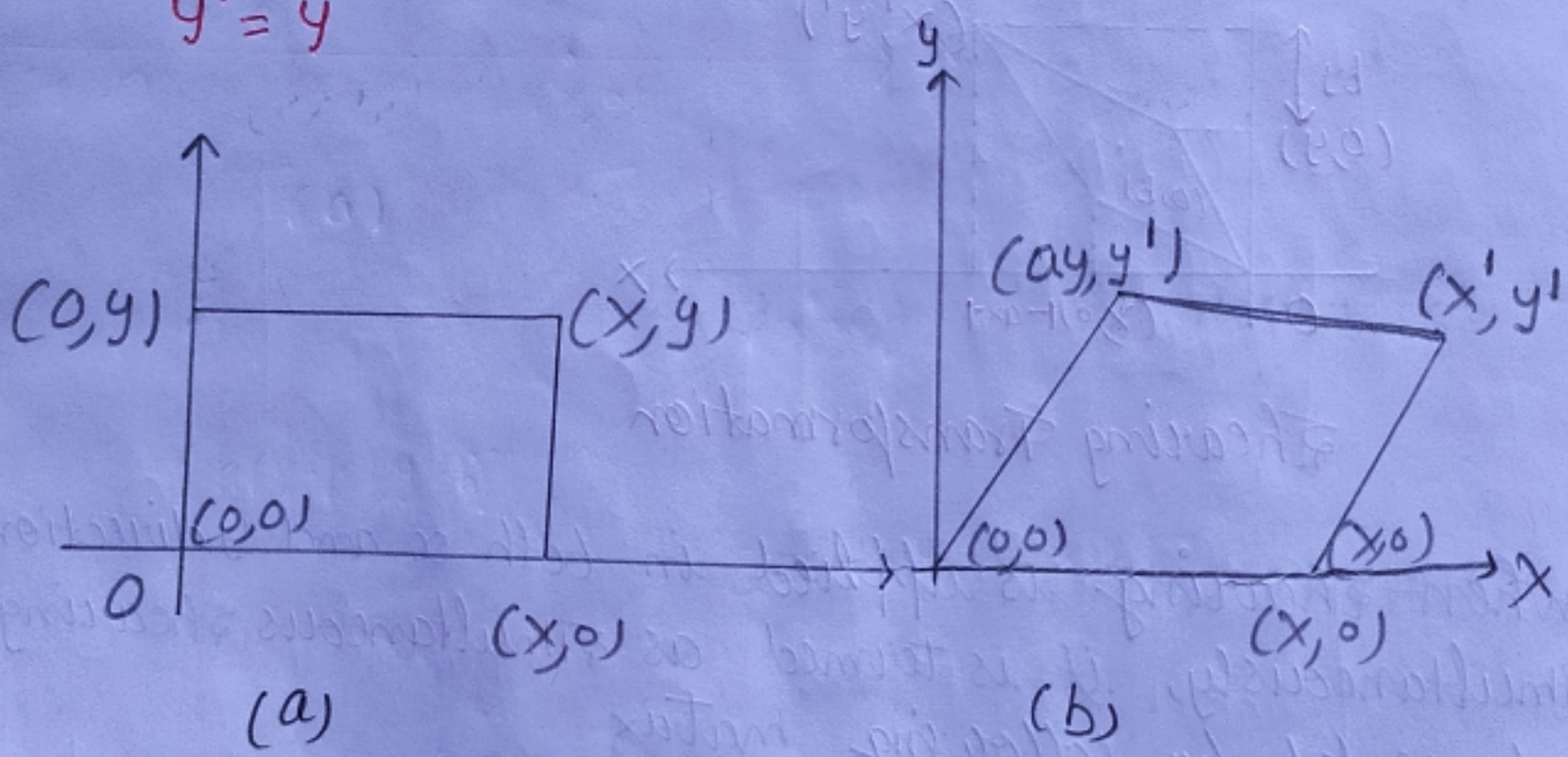


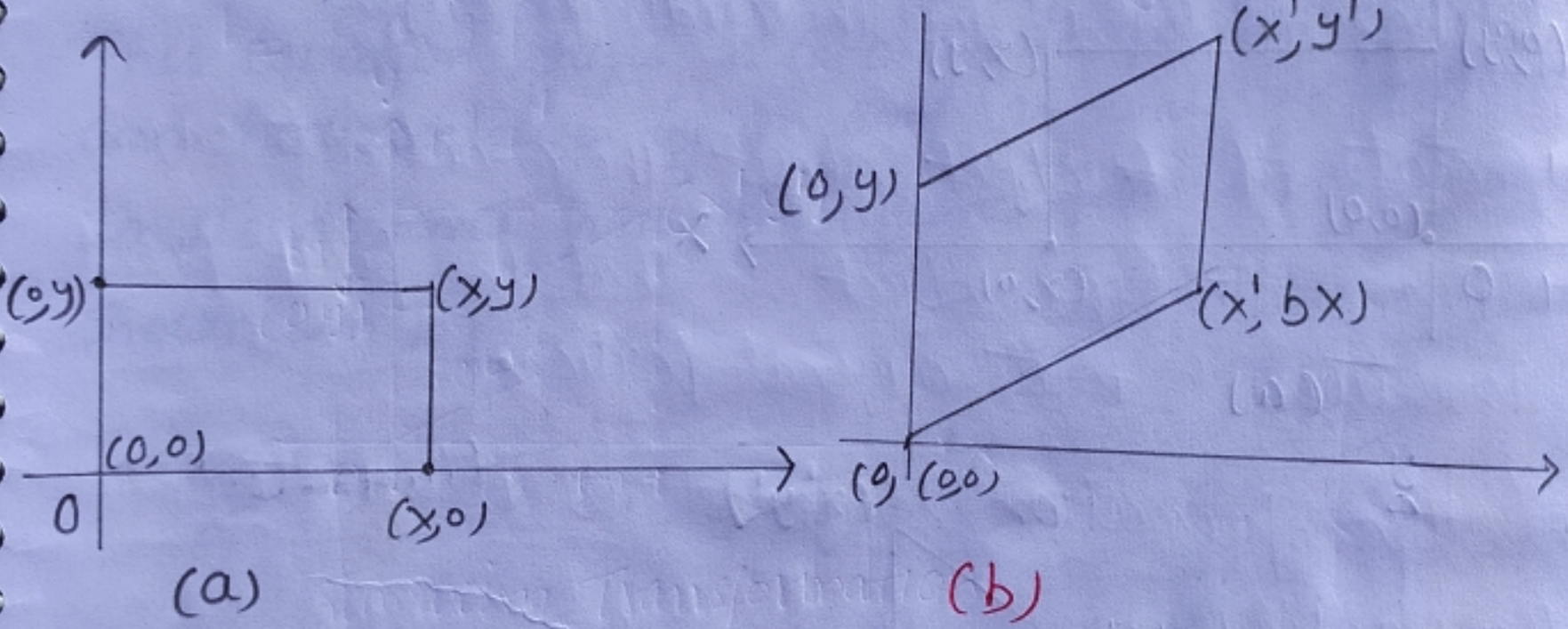
Fig: Shearing in x-direction.

Similarly, when $a=0$ an y-direction, shearing relative to the y-axis is given by the following ~~equation~~ matrix.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{---(2)}$$

Where x' and y' can be represented in the form of following equation.

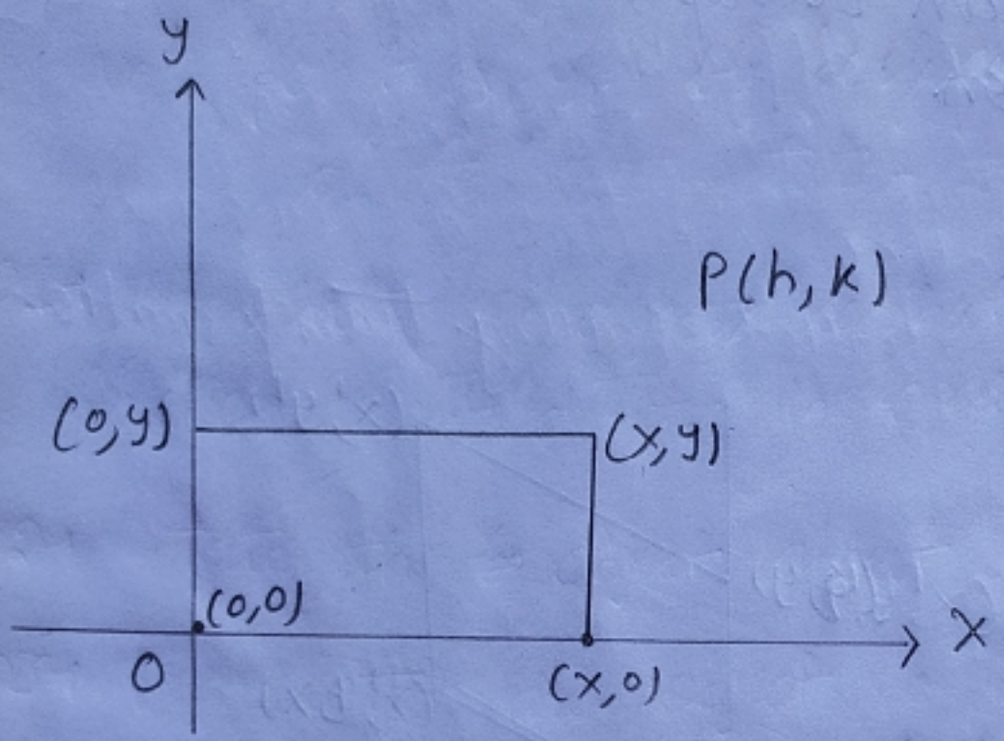
$$x' = x$$
$$y' = bx + y$$



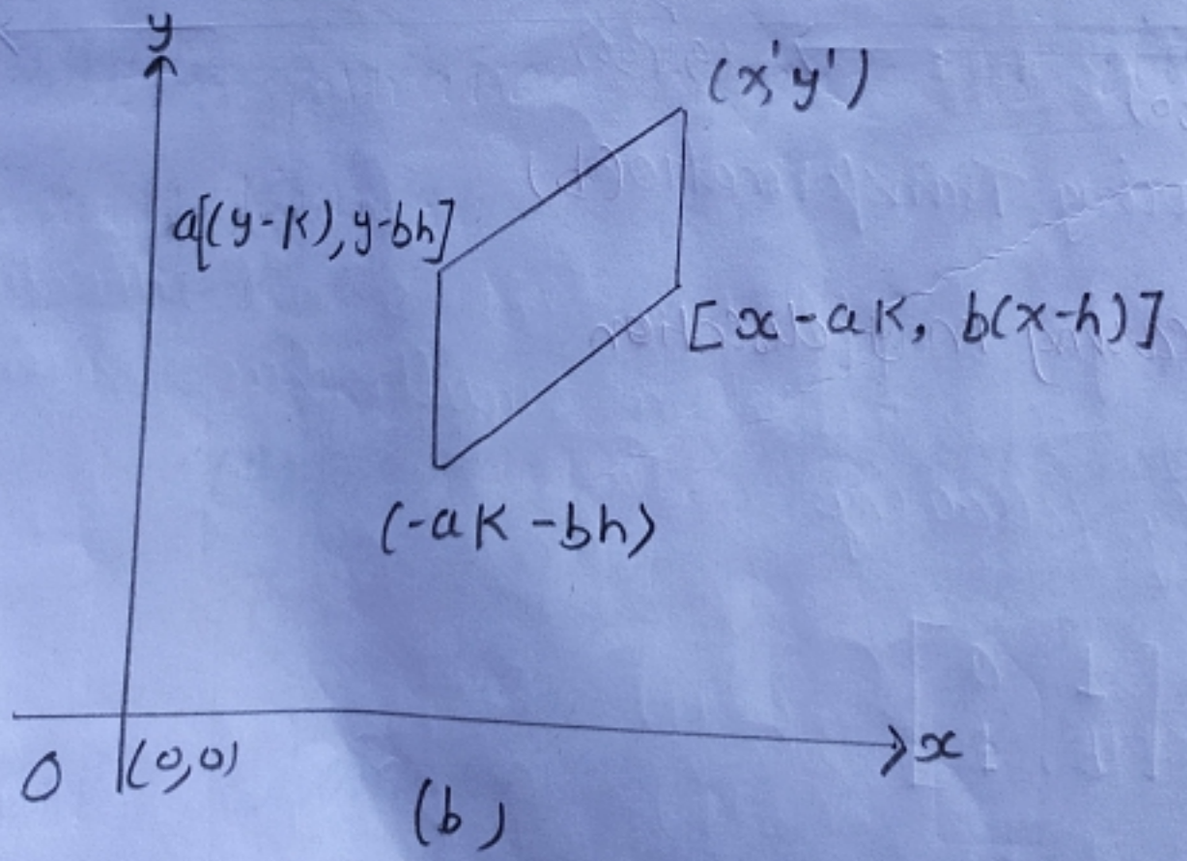
Shearing in y-direction

Shearing Transformation with Respect to any Arbitrary point :-

Simultaneous shearing with respect to an arbitrary point can be calculated as follows:



(a)



(b)

figure

Let P(h, k) is an arbitrary point then x-direction and y-direction shearing are as follows:

$$x' = x + a(y - k)$$

and $$y' = b(x - h) + y$$

Then, transformation matrix will be given by the following

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & -ak \\ b & 1 & -bh \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This equation shifts a co-ordinate position vertically and horizontally by an ~~and~~ amount proportional to its distance from the reference point (h, k) as shown in Fig.

Rotation about a Fixed Pivot Point :-

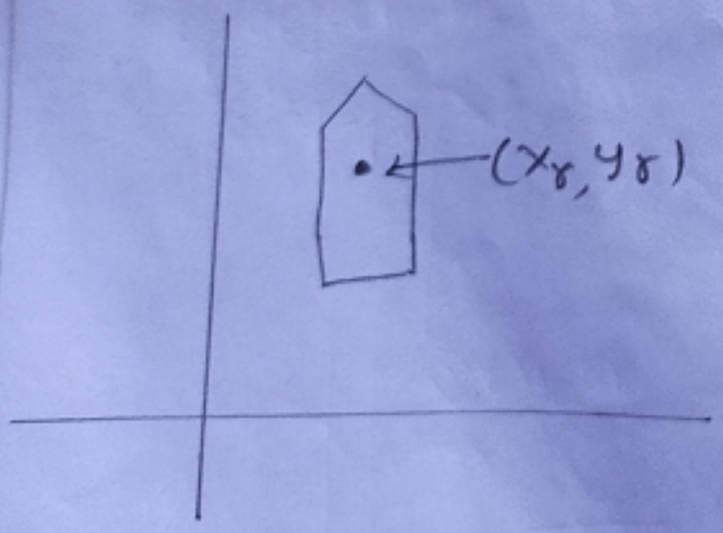
With a graphics package that only provides a rotate function for revolving objects about the co-ordinate origin, we can generate rotations about any selected pivot point (x_r, y_r) by performing the following sequence of translate-rotate-translate operations.

01- Translate the objects so that the pivot-point position is moved to the cordinate origin.

02- Rotate the object about the co-ordinate origine.

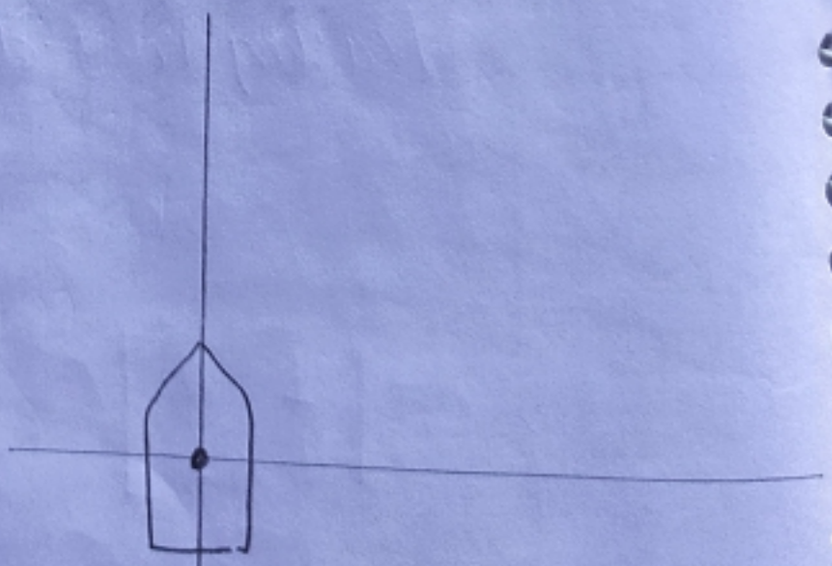
03- Translate the object so that the pivot-point is returned to its original position.

This transformation sequence is illustrated in following figure.



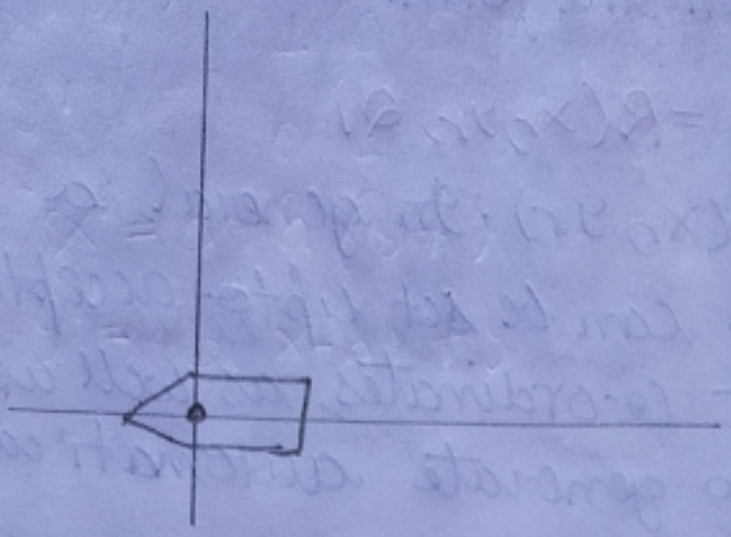
(a)

original position of object and pivot-point



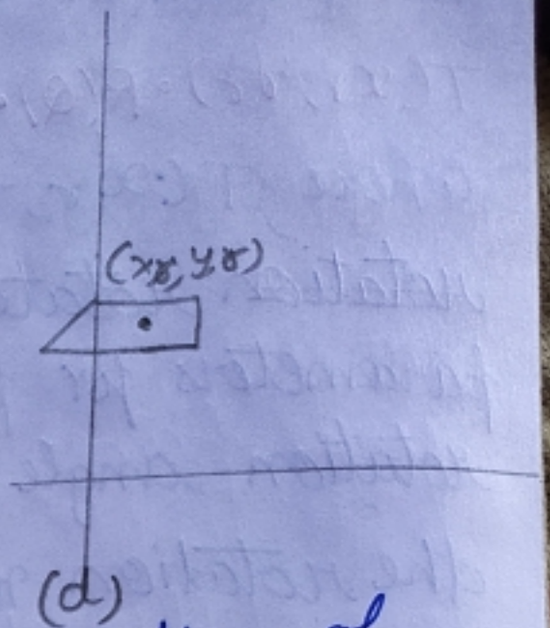
(b)

Translate the object so that the pivot-point (x_r, y_r) is at origin.



(c)

Rotation about origin



(d)

Translation of object so that the pivot-point is returned to position (x_r, y_r)

diag- A transformation sequence for rotating an object about a specified pivot-point using the rotation matrix R(Q) of transformation.

The composite transformation matrix for thissequence is obtained with the concatenation.

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

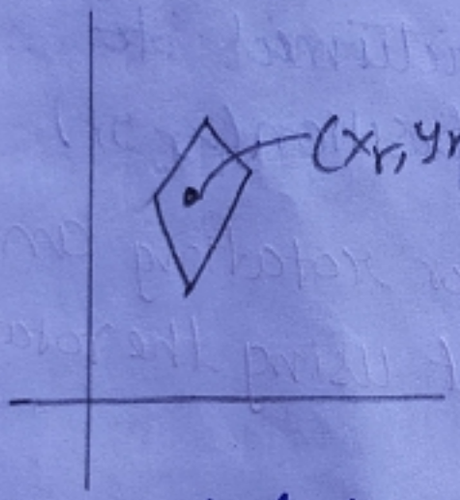
$$= \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos Q & -\sin Q & 0 \\ \sin Q & \cos Q & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

which can be expressed in the form.

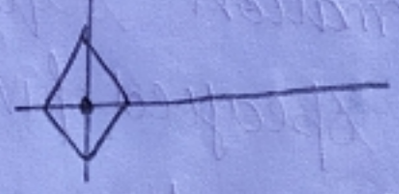
$T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) = R(x_r, y_r, \theta)$

where, $T(-x_r, -y_r) = T^{-1}(x_r, y_r)$. In general, a ~~rotation~~ rotate function can be set upto acceptable parameters for pivot-point co-ordinates, as well as the rotation angle ~~to~~ and to generate automatically the rotation matrix of equation.

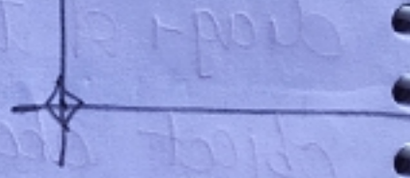
Scalling relative to a fixed pivot-point :



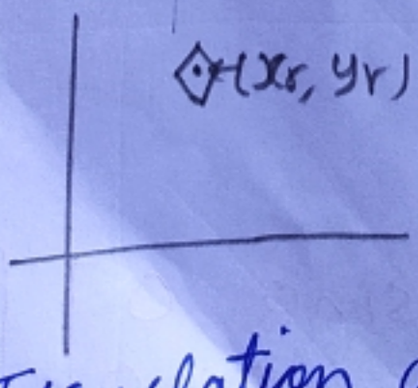
(a) original position of object and pivot-point



(b) Translation of object so that the for pivot-point (x, y) is at origin



(c) Scale object with respect to origin



(d) Translation object so that the fixed point is Returned to position (x_r, y_r)