

UNIT - II

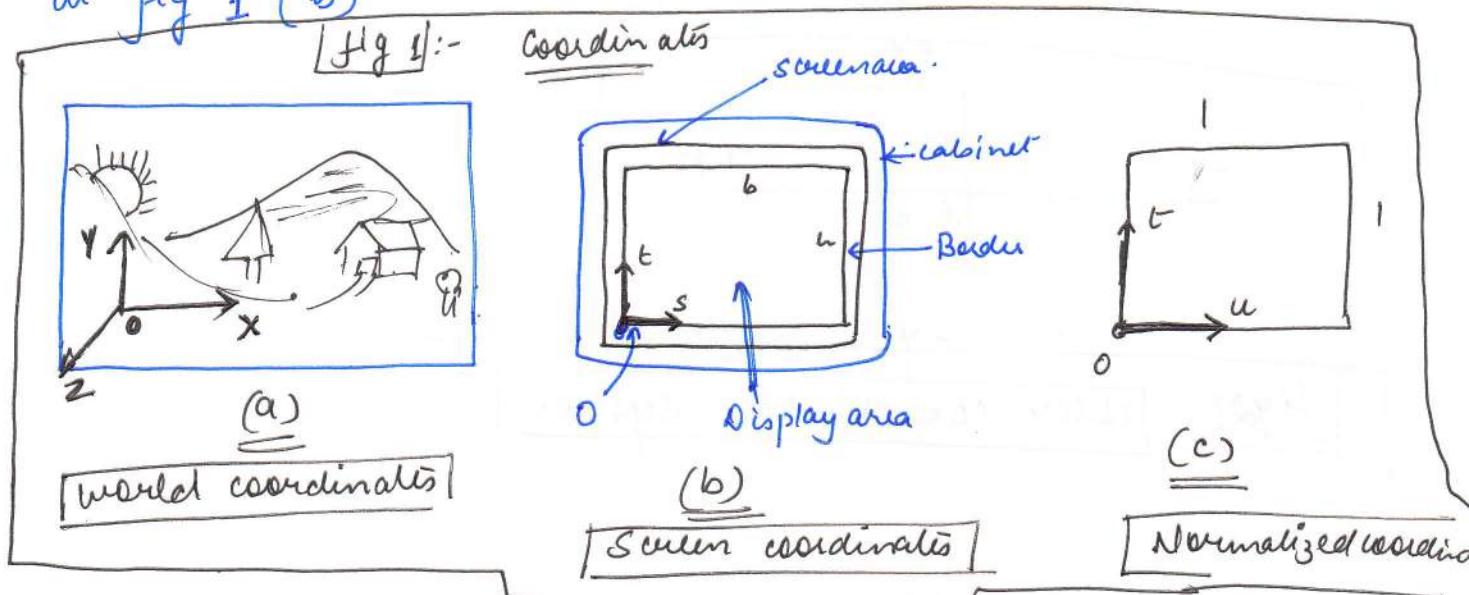
DISPLAY DESCRIPTION

SCREEN CO-ORDINATES :-

By screen, we actually mean the area of the glass tube that is framed within the plastic shell of the VDU.

In home TVs, the picture fills the entire screen but in computer monitors, a narrow border area is left on purpose, to define the rectangular area properly & to accommodate the quantitative nature of the images.

The actual Display area in computer monitors is thus usually a rectangular region, say of width [b] & height [h] length units, within the screen area, the intermediate strip b/w pictures & the screen edges is known as [Border] as marked in fig 1 (b)



NOTE :- we shall however use the two terms Screen area & Display area interchangeably, ~~not~~.

Now,

we attribute to the monitor screen, planar (s, t) coordinates with s horizontal & t vertical, measured in length units from origin at the bottom left corner of the screen, as in fig 1 (b)

To this we call, Screen coordinates obviously s & t
can range from zero to b &
from zero to h respectively.

USER CO-ORDINATES*:

In this system, we pick any point & single it out as a reference point called origin.

Through the origin, we construct 2 perpendicular no. lines called axes.

These are labeled the X axis & Y axis.

If my point in two dimensions in this X-Y plane can be specified by a pair of numbers.

the first no. is for X axis

second no. is for Y axis.

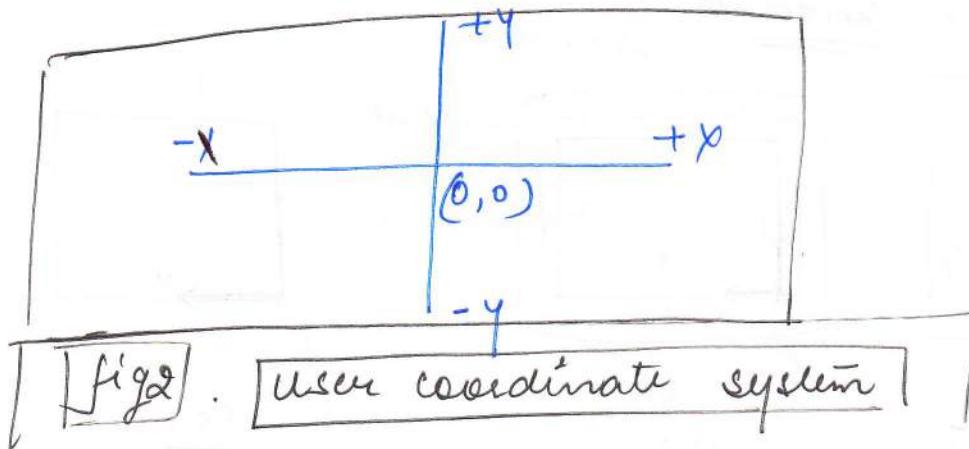


Fig. User coordinate system

Books Referred

* ~~http://~~ www.nettechnology.edu.in/
dep-a/subjects/3graph.pdf

+ Introduction to Computer Graphics by
—N. Krishnamurthy.

USE OF HOMOGENEOUS COORDINATES → ①

- To express any two-dimensional transformation as a matrix multiplication, we represent each Cartesian coordinate posⁿ (x, y) with the Homogeneous coordinate triple (x_h, y_h, h) where

$$\left[\begin{array}{l} x = \frac{x_h}{h} \\ y = \frac{y_h}{h} \end{array} \right] \quad - \quad (1)$$

- Thus, a general homogeneous coordinate representation can also be written as

$$\left[\begin{array}{l} (h \cdot x, h \cdot y, h) \end{array} \right]$$

- For a 2-D geometric transformations, we can choose the homogeneous parameter h to be any non-zero value.

⇒ Thus, there is an infinite no. of equivalent homogeneous representations for each coordinate point (x, y) .

* A convenient choice is simply to set $h=1$.

Each 2-D posⁿ is then represented with homogeneous coordinates $\left[\begin{array}{l} (x, y, 1) \end{array} \right]$

Other

- The term homogeneous coordinates is used in mathematics to refer to the effect of this representation on Cartesian eqⁿs.

- When a Cartesian point (x, y) is converted to a homogeneous representation $\left[\begin{array}{l} (x_h, y_h, h) \end{array} \right]$

eqⁿs containing x & y , such as $[f(x, y)=0]$, become homogeneous eqⁿs in the three parameters $\left[\begin{array}{l} x_h, y_h, h \end{array} \right]$

⇒ This just means that if each of the 3 parameters is replaced by any value v times that parameter, the value v can be factored out of the eqⁿ.

Advantages of using Homogeneous coordinates

o. (P)

- (1) The two - D transformations can be represented on 3×3 matrices so that usage of the opⁿ becomes relatively easy.
- (2) The floating arithmetic can be avoided sometimes by transforming them into integer arithmetic.

For Translation, we have

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

we can write it as

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

$T(t_x, t_y)$ → is 3×3 translation matrix
inverse of translation matrix is obtained by $[t_x]$ & $[-t_y]$

Many rotation transformation eqⁿ about the coordinate origin are now written as:-

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

$$\text{or } P' = R(\theta) \cdot P$$

notation to transformation matr operator $R(\theta)$ is true 3×3 matrix with rotation parameter θ .

Inverse rotation matrix is obtained by replacing θ with $-\theta$

Finally, scaling transformation is now expressed as the matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where $S(s_x, s_y)$ is the 3×3 matrix

Inverse scaling matrix is obtained by replacing parameters with their multiplicative inverse $(\frac{1}{s_x}, \frac{1}{s_y})$ yields the

$$\text{or } P' = S(s_x, s_y) \cdot P$$

Note: - Matrix multiplication are std. methods for implementing transformations in graphics system -

X

Book Refered

computer graphics by D. Meun & Baker

THE VIEW ALGORITHM

Any convenient cartesian co-ordinate system, referred to as the world coordinate reference frame, can be used to define a picture.

For a 2-D picture, a view is selected by specifying a subarea of the total picture area.

The picture parts within the selected areas are then mapped onto the specified areas of the device coordinates.

Terms Related to Algorithm :- [THE VIEWING PIPELINING]

→ window :- A world - coordinate area selected for display
[what is to be viewed]

→ viewport :- An area on display device to which a window is mapped.
[where it is to displayed]

→ viewing transformation or window to viewport transformation or windowing transformation :- The mapping of a part of a world co-ordinate scene to device co-ordinates is referred to as a viewing transformation etc.

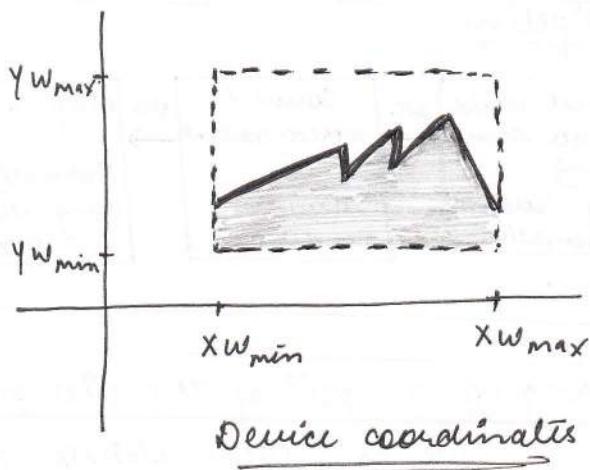
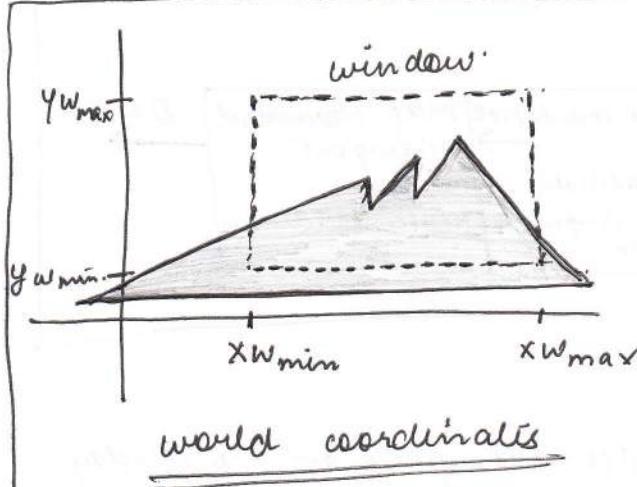


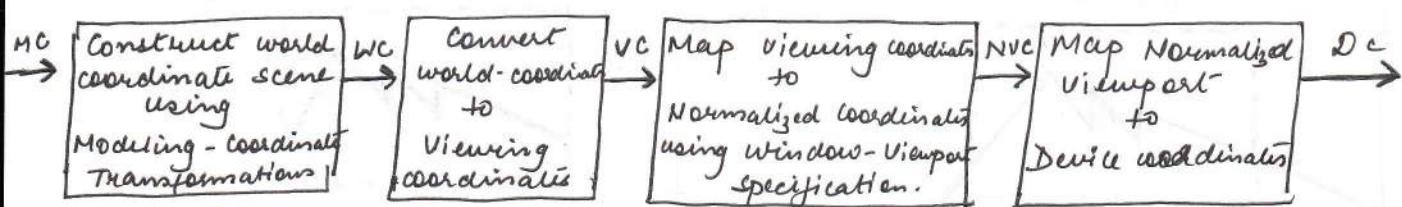
fig 1.

A viewing transformation using standard rectangles for the window and viewport

Viewing transformation in Several steps :-

- First, we construct the scene in world coordinates using the O/P primitives and attributes.
- To obtain a particular orientation for the window, we can set up a 2-D viewing coordinate system in the window coordinate plane and define window in viewing coordinate system
- Once viewing frame is established, we then transform description in world coordinates to viewing coordinates
- Then, we define viewport in normalized coordinates (range from 0 to 1) & map the viewing coordinates description of the scene to normalized coordinates
- At the final step, all parts of the picture that lie outside the viewport are clipped, and the contents are transferred to device coordinates.

[Fig 2.] The two-dimensional viewing - transformation pipeline



* By changing the posⁿ of the viewport, we can view objects at different posⁿs on the display area of an output device. as shown in Fig 3

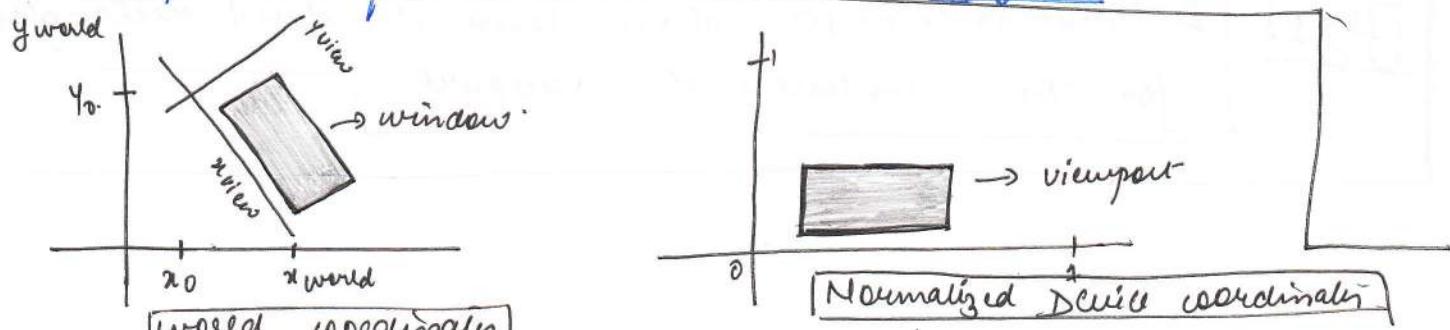
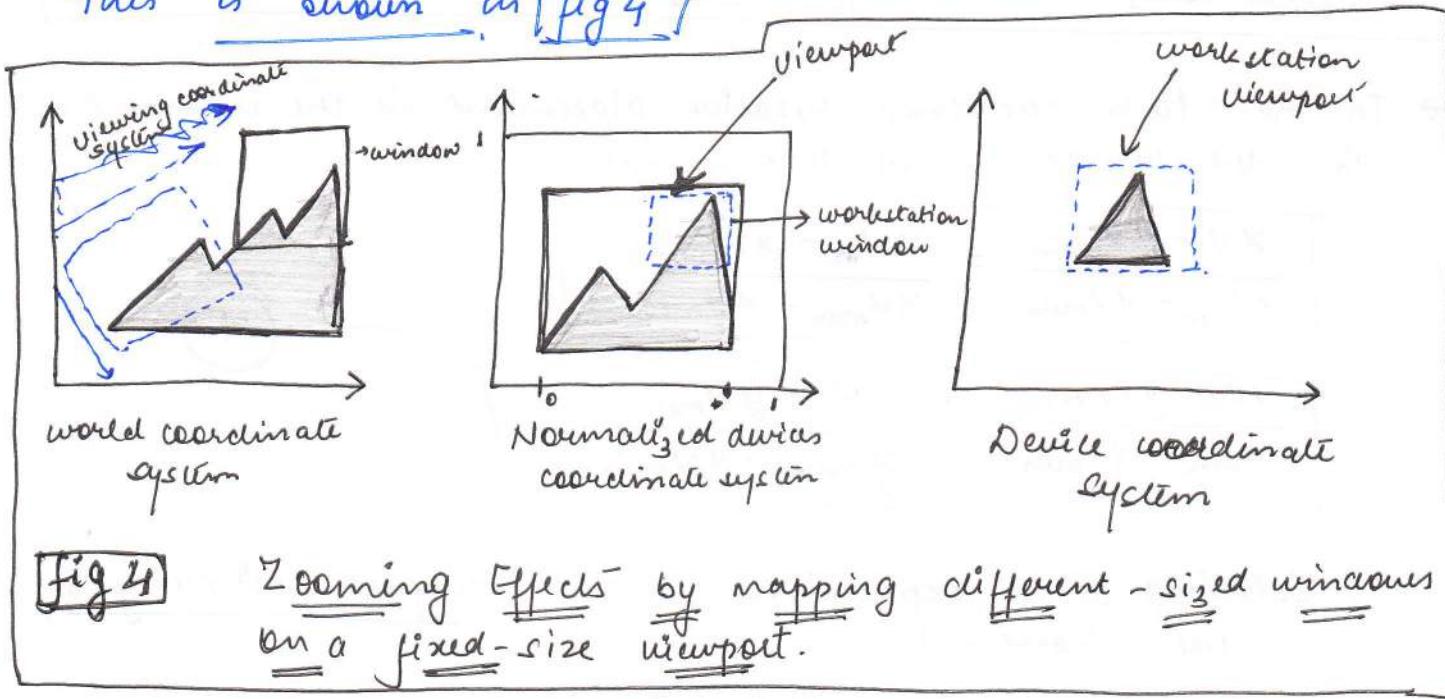


Fig 3. Setting up a rotated world window in corresponding normalized coordinate viewport

- * By varying the size of viewports
 we can change the size and proportions of displayed objects.
 → we can achieve zooming effects by successively mapping different-sized windows on a fixed-size viewport.
 As the windows are made smaller, we zoom in on some part of a scene to view details that are not shown with larger windows.

This is shown in fig 4



WINDOW TO VIEWPORT CO-ORDINATE TRANSFORMATION :-

- Once object descriptions have been transferred to the viewing reference frame, we choose the window extents in viewing coordinates and select the viewport limits in normalized coordinates.
- Object descriptions are then transferred to normalized device coordinates:-
 We do this thing using a transformation that maintains the same relative placement of objects in normalized space as they had in viewing coordinates.
- If a co-ordinate pair is at the center of the viewing window
 ⇒ It will be displayed at the center of the viewport.

→ fig 5. illustrates the window-to-viewport mapping

→ A pt. at $\text{pos}^n(x_w, y_w)$ in window $\xrightarrow{\text{mapped into}}$ $\text{pos}^n(x_v, y_v)$ in associated viewport

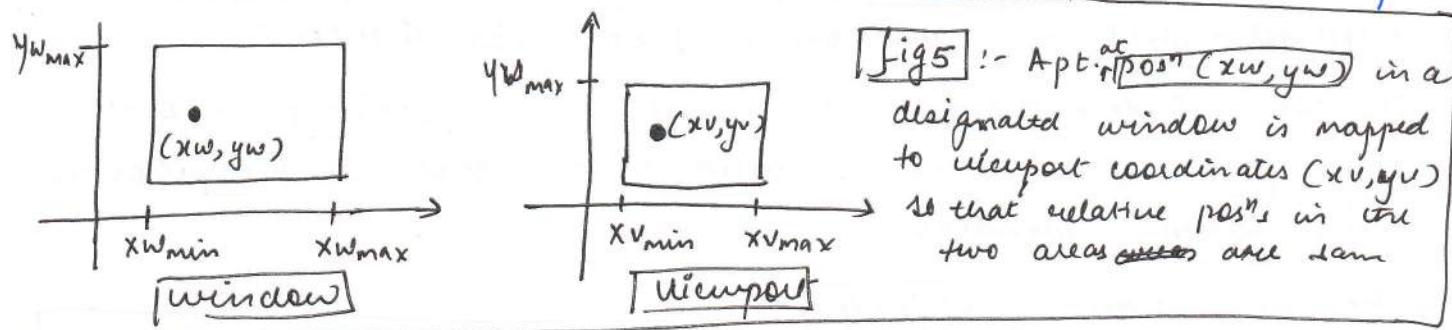


Fig 5 :- Apt. at $\text{pos}^n(x_w, y_w)$ in a designated window is mapped to viewport coordinates (x_v, y_v) so that relative pos's in the two areas ~~areas~~ are same

→ To maintain the same relative placement in the viewport as in the window, we require that

$$\frac{x_v - x_{v\min}}{x_{v\max} - x_{v\min}} = \frac{x_w - x_{w\min}}{x_{w\max} - x_{w\min}}$$

$$\frac{y_v - y_{v\min}}{y_{v\max} - y_{v\min}} = \frac{y_w - y_{w\min}}{y_{w\max} - y_{w\min}}$$

— (1)

Solving these expressions for the viewport $\text{pos}^n(x_v, y_v)$, we have.

$$\begin{cases} x_v = x_{v\min} + (x_w - x_{w\min})sx \\ y_v = y_{v\min} + (y_w - y_{w\min})sy \end{cases}$$

— (2)

where scaling factors are

$$\begin{cases} sx = \frac{x_{v\max} - x_{v\min}}{x_{w\max} - x_{w\min}}, \\ sy = \frac{y_{v\max} - y_{v\min}}{y_{w\max} - y_{w\min}}. \end{cases}$$

The conversion of window area to viewport area is carried out in following sequence

- 1) Perform a scaling transformation using a fixed pt.-posn. $(x_{w\min}, y_{w\min})$ that scales the window area to the size of viewport.
- 2) Translate the scaled window area to the posn of the viewport.

→ Relative proportions of objects are maintained if the scaling factors are same ($s_x = s_y$).

→ From normalized coordinates, object descriptions are mapped to the various display devices.

Any no. of o/p devices can be open in a particular app & another window-to-viewport transformation can be performed for each open o/p device.

This mapping called, Workstation Transformation



It is accomplished by selecting a window area in normalized space & a viewport area in the coordinates of the display devic.

→ As in Fig 6 we can use workstation transformations to partition a view so that diff parts of normalized space can be displayed on diff o/p devices.

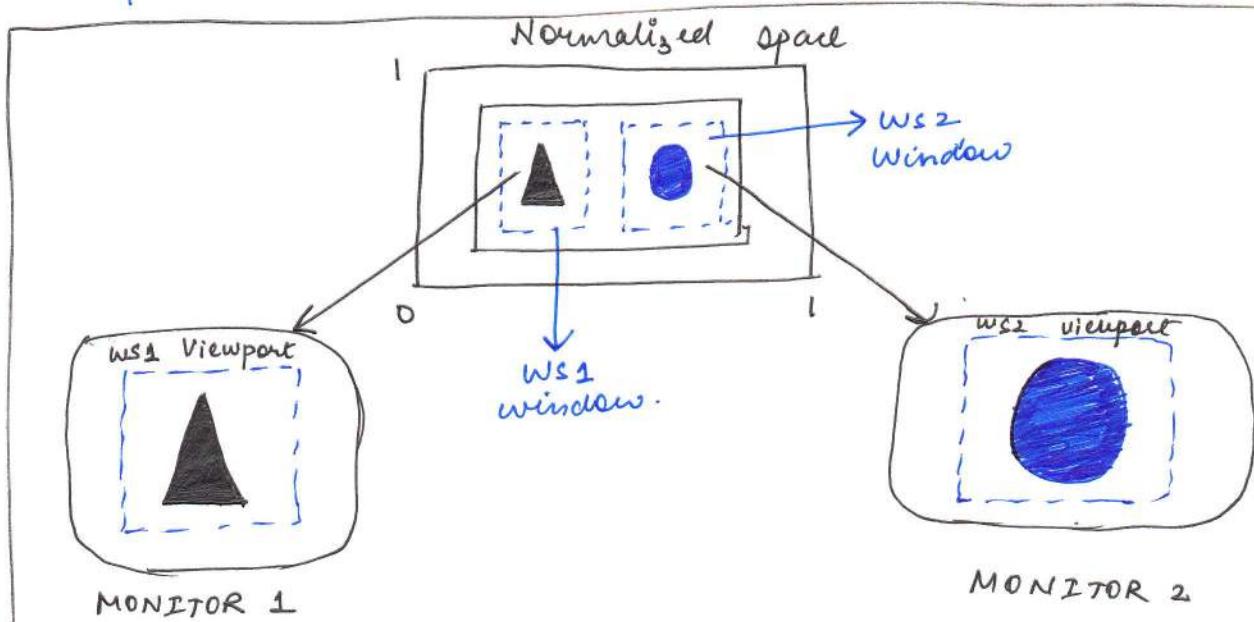


Fig 6 Mapping selected parts of a scene in normalized coordinates to different video monitors with workstation transformation.



Books Referred

Computer Graphics - by D. Hearn & Baker

TWO-DIMENSIONAL TRANSFORMATION

All computer graphics system has the ability to simulate the manipulation of objects in space.

This simulated spatial manipulation is referred to as transformation

Need for transformation :-

The need for transformation arises ~~when several objects, each of which is independent~~

When several objects, each of which is independently defined in its own coordinate system,

it needs to be properly positioned into a common scene in a master coordinate system.

Transformation is also useful in other areas of the image synthesis process. e.g. viewing transformation

There are two complementary points of view for describing object transformation.

(1) GEOMETRIC TRANSFORMATIONS :-

The object itself is transformed relative to stationary coordinate system or background

The mathematical statement of this viewpoint is described by geometric transformations applied to each point of the object.

(2) COORDINATE TRANSFORMATIONS :-

The object is held stationary while the coordinate system is transformed, relative to the object

This effect is attained through the application of coordinate transformations.

Example that helps to distinguish these two concepts

The movement of an automobile against a scenic background
we can simulate this by

- Moving the automobile while keep the background fixed
 - (geometric transformation)
- we can keep the car fixed while moving the backdrop scenery
 - (coordinate transformation)

Basic Transformations

:-

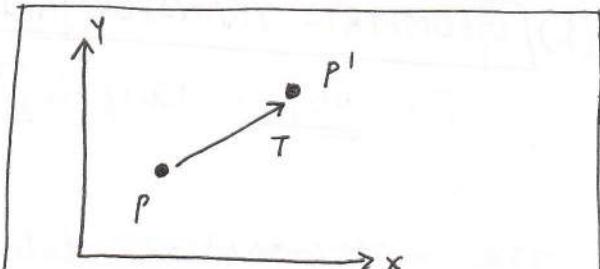
Here we discuss general procedures for applying translation, rotation, and scaling transformation to reposition and resize two dimensional objects.

TRANSLATION :-

- A translation is applied to an object by repositioning it along a straight line path from one coordinate location to another.
- We translate a two-dimensional point by adding translation distance, t_x & t_y , to the original coordinate position (x, y) to move the point to a new position (x', y') .

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

— (1)



The translation vector distance pair (t_x, t_y) is called translation vector or shift vector.

fig 1 Translating a point from posⁿ[P] to posⁿ[P'] with translation vector T.

- Eq. 1 can be written using column vectors to represent coordinate posⁿ's and the translation vector :-

$$P = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad P' = \begin{bmatrix} u'_1 \\ u'_2 \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad — (2)$$

- we can write the two-dimensional translation equation in the matrix form :-

$$P' = P + T \quad - (3)$$

def

Translation is a rigid-body transformation that moves objects without deformation \rightarrow every point on the object is translated by the same amount.

- A straight line is translated by applying the transformation eqⁿ (3) to each of the line end pts. and redrawing the line b/w the new end point posⁿs.
- Fig 2 illustrates the appⁿ of a specified translation vector to move an object from one posⁿ to another.

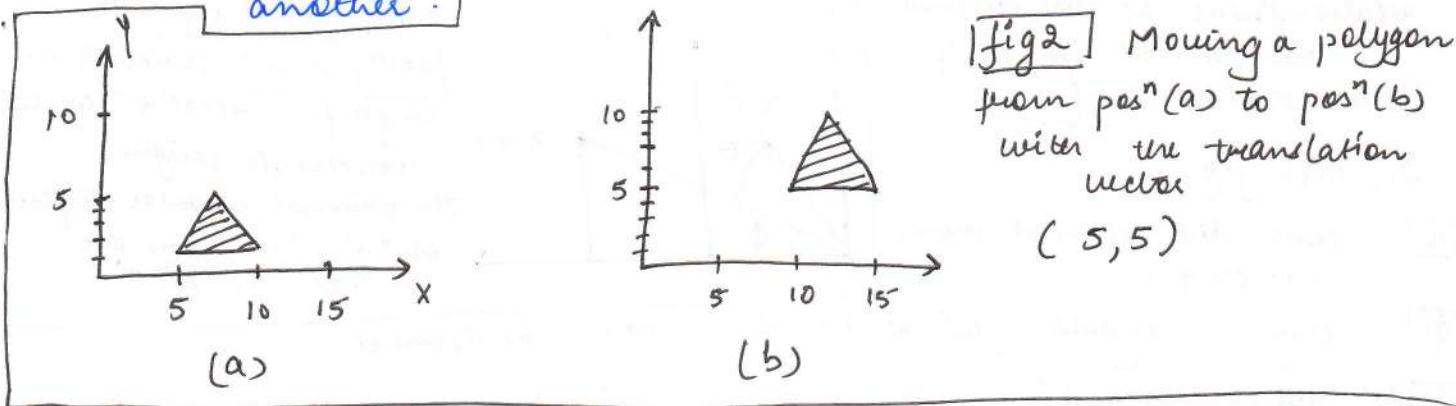


Fig 2] Moving a polygon from posⁿ(a) to posⁿ(b) with the translation vector (5, 5)

- Similar methods are used to translate curved objects. To change the posⁿ of a circle, we translate the center coordinates & redraw the figure in the new circle.

ROTATION :-

- A rotation & D-notation is applied to an object by repositioning it along a circular path in the xy plane.
- To generate a rotation we specify a rotation angle θ & posⁿ(x_n, y_n) of the rotation point (or pivot point) about which the object is to be rotated.

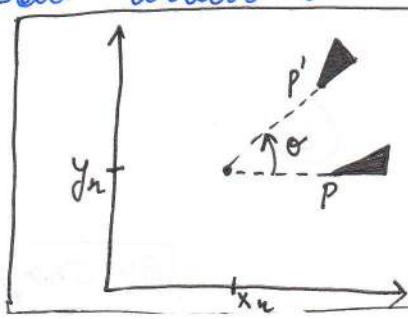


Fig 3] Rotation of an object through angle θ about the pivot point (x_n, y_n)

• +ve values for the rotation angle define counterclockwise rotations about the pivot point

-ve values rotate objects in the clockwise direction shown in fig. 3

This transformation can be also be described as a rotation about a rotation axis that is perpendicular to the xy plane and passes through the pivot point.

* We first determine the transformation equations for rotation of a point posⁿ P, when the pivot point is at the coordinate origin

- The angular & coordinate relationships of the original & transformed pt. posⁿ are shown in fig 4

- In this fig.

- const. dist. of the pt. from the origin

- original angular posⁿ of the point from horizontal

- rotation angle.

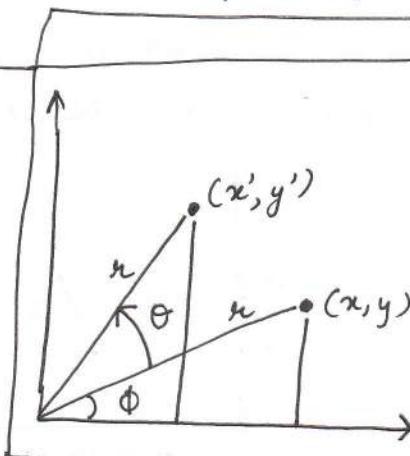


Fig 4 Rotation of a pt. from posⁿ (x, y) to posⁿ (x', y') through an angle θ relative to the coordinate origin.

The original angular displacement of the pt. from the x-axis is ϕ

→ The transformed coordinates in terms of angles θ & ϕ are as follows

$$\begin{cases} x' = r \cos(\phi + \theta) = r \cos\phi \cos\theta - r \sin\phi \sin\theta \\ y' = r \sin(\phi + \theta) = r \cos\phi \sin\theta + r \sin\phi \cos\theta \end{cases}$$

— ④

→ The original coordinates of the point in polar coordinates are:-

$$x = r \cos\phi, y = r \sin\phi \quad — ⑤$$

→ Substituting eq ⑤ into ④, we get eqn ⑥

We obtain the transformation eqⁿs for rotating a point at posⁿ (x, y) through an angle θ about the origin.

$$\begin{cases} x' = x \cos\theta - y \sin\theta \\ y' = x \sin\theta + y \cos\theta \end{cases} \quad — ⑥$$

→ we can write the [notation equations in the matrix form] - 3

$$P' = R \cdot P$$

- 7

where rotation matrix is

$$R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} - 8$$

→ eq 7 is transposed so that transformed new coordinate vector $[x' \ y']$ is calculated as.

$$\begin{aligned} P'^T &= (R \cdot P)^T \\ &= P^T \cdot R^T \end{aligned}$$

Rotation of point about a pivot pt posⁿ is shown in fig 5

→ The transformation eq's for rotation of a pt about any specified rotation posⁿ(x_n, y_n) :-

$$\begin{aligned} x' &= x_n + (x - x_n) \cos \theta - (y - y_n) \sin \theta \\ y' &= y_n + (x - x_n) \sin \theta + (y - y_n) \cos \theta \end{aligned}$$

- 9

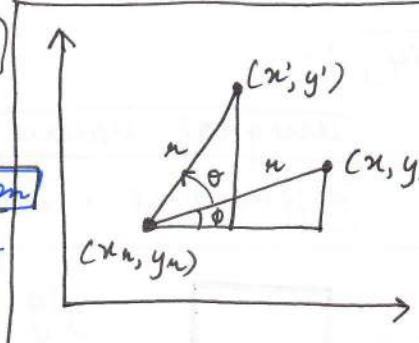


fig 5 Rotating a pt. from $[pos^n(x, y)]$ to $[pos^n(x', y')]$ through an angle θ about rotation $[pt(x_n, y_n)]$.

SCALING : → (Alters the size of an obj)

* A scaling transformation alters the size of an object.

This operⁿ can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by

Scaling factors s_x & s_y to produce the transformed coordinates (x', y') !

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

- 10

s_x

scales objects in the x direction

s_y

scales in the y direction

- Eq ⑩ can also be written in the matrix form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad - \quad ⑪$$

or

$$P' = S \cdot P \quad - \quad ⑫$$

S is the 2 by 2 scaling matrix.

- Any two numeric values can be assigned to the scaling factors s_x & s_y .

- Values less than ≤ 1 \rightarrow reduces the size of objects
- Values greater than ≥ 1 \rightarrow produce enlargement
- Specifying a value of ≤ 1 for both s_x & s_y \rightarrow leaves the size of objects unchanged.

- UNIFORM SCALING :-

when s_x & s_y are assigned same value,
uniform scaling is produced

- DIFFERENTIAL SCALING :-

unequal values for s_x & s_y result in
differential scaling.

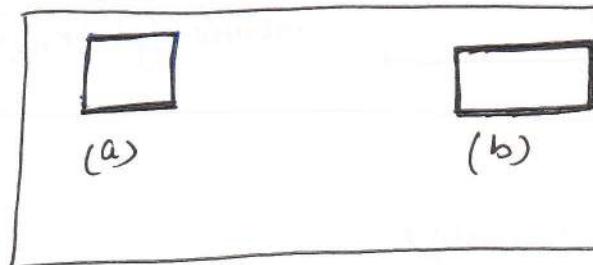


fig. 6 Turning a square (a) into rectangle (b) with scaling factors $s_x=2$ & $s_y=1$

- Note :- Scaling factors with values ~~less than 1~~
 - less than 1 \rightarrow move objects closer to coordinate origin
 - greater than 1 \rightarrow move coordinate pos'n further from the origin

- The location of a scaled object can be controlled by choosing a pos'n called fixed point.

coordinates for the fixed point (x_f, y_f) can \rightarrow be chosen as one of the vertices. (fig 8)

Poly gon $\xrightarrow{\text{scaled relative to fixed pt.}}$ By scaling - the distance from each vertex to fixed point.

* The scaled coordinates (x', y') are calculated as :- (4)

$$\boxed{\begin{aligned} x' &= x_f + (x - x_f) S_x \\ y' &= y_f + (y - y_f) S_y \end{aligned}} \quad \rightarrow (13)$$

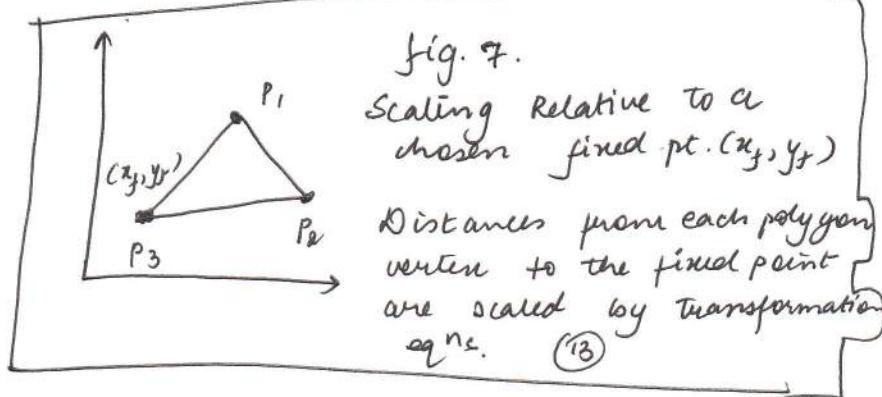
Eq (13) can be rewritten to separate the multiplicative & additive terms :

$$\Rightarrow \boxed{\begin{aligned} x' &= x \cdot S_x + x_f (1-S_x) \\ y' &= y \cdot S_y + y_f (1-S_y) \end{aligned}} \quad \rightarrow (14)$$

I App's

Polygons are scaled by applying transformations (14) to each vertex &

then regenerating the polygon using the transformed vertices.



~~→ Books Referred~~

- Computer Graphics - Schramm's series
- Computer Graphics - D. Hearn & Baker.

Books Referred

LINE DRAWING ALGORITHM

①

- Straight-line segments are used a great deal in computer generated picture.
- Following are the things a good line drawing algo should do well :-

- * Line should appear straight :- we must approximate the line by choosing addressable points close to it. If we choose well, the ~~big~~ line will appear straight; if not, we shall produce crooked lines. (fig 1)

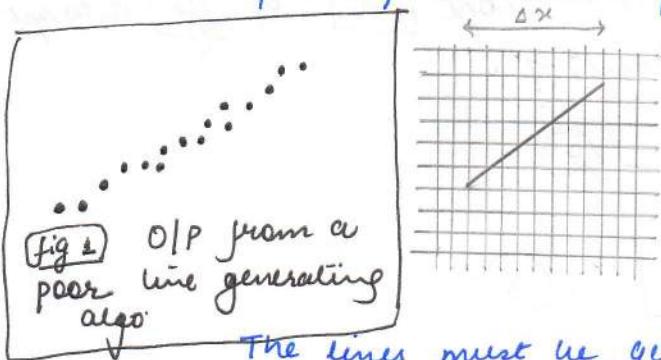


Fig 2 :- A straight line segment connecting 2 grid intersections may fail to pass through any other grid intersections.

The lines must be generated parallel or at 45° to the x & y axis. Other lines causes a problem; - a line segment though it starts & finishes at addressable points, may happen to pass through no other addressable pts. (in b/w). Fig 2)

- * Lines should terminate accurately :- unless lines are plotted accurately, they may terminate at the wrong place.

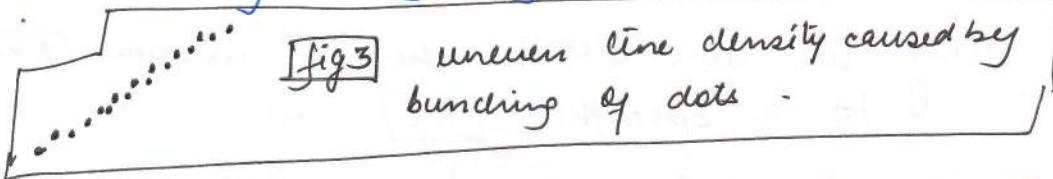


Fig 3 uneven line density caused by bundling of dots

- * Lines should have constant density :- line density is proportional to the no. of dots displayed divided by the length of the line. To maintain constant density, dots should be equally spaced.

- * Line density should be independent of line length & angle. This can be done by computing an approximating line-length estimate & to use a line-generation algo that keeps line density constant to within the accuracy of this estimate.

- * Line should be drawn rapidly :- This computation should be performed by special-purpose hardware.

Dine Drawing Algorithms

* The cartesian [slope intercept eqn] for a st. line is

$$y = m \cdot x + b \quad \text{--- (1)}$$

m

→ slope of line

b

→ ~~y~~ intercept

* Given → 2 end pts. of line segment are specified at pos
 (x_1, y_1) & (x_2, y_2)

we can determine values for the slope m & y intercept
 b with following calculations →

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{--- (2)}$$

$$\therefore b = y_1 - m \cdot x_1 \quad \text{--- (3)}$$

[Note] :- Algos for displaying st. lines are based on line eqn
(1), (2), (3).

* For any given x interval Δx along a line,
we can compute the corresponding y interval Δy
from eq (2)

$$\Delta y = m \Delta x \quad \text{--- (4)}$$

Only we can obtain the x interval Δx corresponding
to a specified Δy as

$$\Delta x = \frac{\Delta y}{m} \quad \text{--- (5)}$$

These eq's form the basis for determining deflection voltages
in analog devices.

Books Referred :-

- Principles of ICB by Newmann
- Computer Graphics by D. Hearn
& Baker.

(a) DDA Algorithm

→ The digital differential analyzer (DDA) is a scan conversion line algorithm based on the calculating either Δy or Δx using eq ④ or ⑤

→ we sample the line at unit intervals in one coordinate & determine corresponding integer values nearest the line path for the other coordinate.

* Consider a line with the slope as shown in fig 4

+ If slope is less than or equal to 1

we sample at unit n intervals ($\Delta x=1$) & compute each successive y value as

$$y_{k+1} = y_k + m \quad - ⑥$$

⑧ - integer value starting from 1 → for the first point increases by 1 until the final endpoint is reached.

⑨ → can be any real no. b/w 0 & 1

⑩ → values must be rounded to the nearest integer.

+ If slope is greater than 1

we sample at unit $1/y$ intervals ($\Delta y=1$) & calculate each succeeding x value as

$$x_{k+1} = x_k + \frac{1}{m} \quad - ⑦$$

Eq ⑥ & ⑦ are based on the assumption that lines are to be processed from the left endpoint to the right endpoint.

If this processing is reversed, so that the starting endpoint is at right, then we have $\Delta x = -1$

$$y_{k+1} = y_k - m \quad - ⑧$$

+ (when slope is greater than 1) → we have $\Delta y = -1$ with

$$x_{k+1} = x_k - \frac{1}{m} \quad - ⑨$$

~~Note~~ → Eq ⑥ to ⑨ can be used in

Eq ⑥ through ⑨ can also be used to calculate pixel posns along a line with -ve slope.

$\frac{dy}{dx}$

→ the start endpt is at [left], we set $[\Delta x=1]$ & calculate $[y]$ values with eq ⑥.

→ the start endpt is at [right], we set $[\Delta x=-1]$ obtain $[y]$ values from eq ⑧

→ when absolute value of a -ve slope > 1 , we use $[\Delta y=-1]$ from eq ⑨ or we use $[\Delta y=1]$ from eq ⑦

Advantages :-

① The DDA algorithm is a fast method for calculating pixel posns than the direct use of eqⁿ $[y = mx + b]$

② It eliminates the multiplication in eq ① by making use of raster characteristics.
so that appropriate increments are applied in the x or y direction to step to pixel posn along the line path.

Disadvantage :-

The rounding operⁿ & floating point arithmetic procedures are still time consuming.

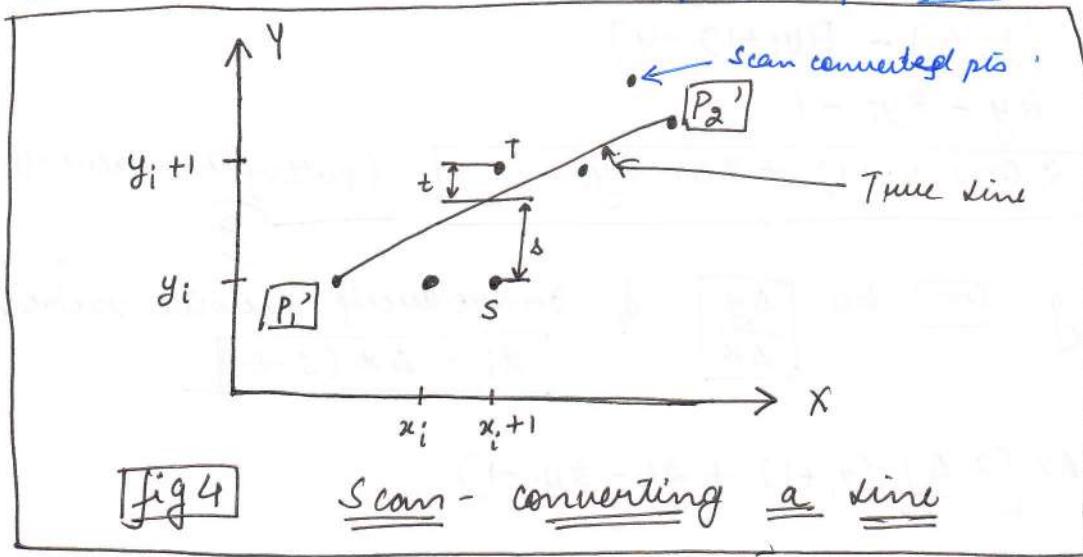
Book reference

- Computer Graphics by
Donald Hearn & Baker.

(3)

(b) BRESENHAM'S LINE DRAWING :-

- It is a highly efficient incremental method for scan converting lines.
- The best approximation of the true line → is described by those pixels in the raster that fall the least distance from the true line.
- The method works as follows :-
- Assume a pixel $P_i'(x_i', y_i')$, then select subsequent pixels as we work our way to the right, one pixel \rightarrow at a time in the horizontal direction toward $P_2'(x_2', y_2')$
- Once a pixel is chosen at any step
 - the next pixel is
 - either the one to its right [lower bound for the line]
 - one to its right & up [upper-bound for the line]
- The line is best approximated by those pixels that fall in least distance from its true path b/w P_i' & P_2'



- Our task :- To choose one next one b/w the bottom pixel S & Top pixel T.

→ If S is chosen

$$\text{we have } x_{i+1} = x_i + 1 \quad \& \quad y_{i+1} = y_i$$

→ If T is chosen

$$\text{we have } x_{i+1} = x_i + 1 \quad \& \quad y_{i+1} = y_i + 1$$

- The actual y coordinate of the line at $x = x_{i+1}$ is

$$\begin{aligned} y &= mx_{i+1} + b \\ y &= m(x_{i+1}) + b \end{aligned} \quad \text{--- (1)}$$

- The distance from S to the actual ^{line} in y direction.

$$s = y - y_i$$

The distance from T to the actual line in y direction

$$t = (y_{i+1}) - y$$

→ Now consider the difference b/w these 2 distances values

$$s-t$$

- * when $(s-t) < 0 \Rightarrow s < t$
the closest pixel is S
- * when $(s-t) \geq 0 \Rightarrow s \geq t$
the closest pixel is T

This difference is:

$$\begin{aligned} s-t &= (y - y_i) - [(y_{i+1}) - y] \\ &= \Delta y - \Delta y_i - 1 \end{aligned}$$

$$s-t = \Delta m(x_{i+1}) + \Delta b - \Delta y_i - 1 \quad \text{[Putting the value of (1)]} \quad \text{--- (2)}$$

Substituting m by $\frac{\Delta y}{\Delta x}$ & introducing decision variable

$$d_i = \Delta x(s-t)$$

$$\begin{aligned} d_i &= \Delta x \left[2 \frac{\Delta y}{\Delta x} (x_{i+1}) + 2b - \Delta y_i - 1 \right] \\ &= 2 \Delta y x_i - 2 \Delta y + \cancel{2 \Delta x \cdot 2b} - 2 y_i \Delta x - \Delta x \\ d_i &= 2 \Delta y x_i - 2 \Delta x y_i + c \quad \text{--- (3)} \end{aligned}$$

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

- we can write the decision variable d_{i+1} for the next slope

$$d_{i+1} = 2 \Delta y x_{i+1} - 2 \Delta x y_{i+1} + c \quad \text{--- (4)}$$

$$\Rightarrow d_{i+1} - d_i = 2\Delta y(x_{i+1} - x_i) - 2\Delta x(y_{i+1} - y_i) \quad \text{--- (4)}$$

Since $x_{i+1} = x_i + 1$ we have:

$$d_{i+1} - d_i = 2\Delta y(x_{i+1} - x_i) - 2\Delta x(y_{i+1} - y_i)$$

$$d_{i+1} = d_i + 2\Delta y - 2\Delta x(y_{i+1} - y_i) \quad \text{--- (6)}$$

Special cases

If chosen pixel is the top pixel T (i.e. $d_i \geq 0$) $\Rightarrow y_{i+1} = y_i + 1$

$$d_{i+1} = d_i + 2\Delta y - 2\Delta x$$

If chosen pixel is the bottom pixel S (i.e. $d_i < 0$) $\Rightarrow y_{i+1} = y_i$

$$d_{i+1} = d_i + 2\Delta y$$

Finally we calculate d_1

$$d_1 = \Delta x [2m(x_1 + 1) + 2b - 2y_1 - 1]$$

$$d_1 = \Delta x [2(mx_1 + b - y_1) + 2m - 1]$$

Since $mx_1 + b - y_1 = 0$ & $m = \frac{\Delta y}{\Delta x}$, we have

$$d_1 = 2\Delta y - \Delta x$$

D algorithm

Given $P_1'(x_1', y_1')$ to $P_2'(x_2', y_2')$ such that $x_1' < x_2'$ & $0 < mx_1$

Step 1 :- put $x = x_1'$ & $y = y_1'$

$$\Delta x = x_2' - x_1', \quad \Delta y = y_2' - y_1'$$

$$dT = 2(\Delta y - \Delta x)$$

$$ds = 2\Delta y$$

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

Step 2 :- Set $\boxed{\text{pixel plot}(x, y)}$

Step 3

Step 3 :- Repeat steps while $x < x_2'$

i) $x = x + 1$

ii) If $d < 0$

then set $d = d + ds$

else

(i) $y = y + 1$

(ii) $d = d + dT$

3) plot (x, y)

Step 4 : END

Advantages :-

- faster as compared to DDA
- simple as it involves only integer arithmetic
- avoid generation of duplicates
- can be implemented using H/W
 \because it does not use multiplication & division

Books Referred :-

Computer Graphics by
Donald Hearn & Baker

CIRCLE DRAWING ALGOS :-

- A circle is a symmetrical figure.
- Any circle - Generating algorithm can take advantage of the circle's symmetry to plot 8 pts. for each value that the algorithm calculates.
- 8-way symmetry is used by reflecting each calculated pt. around each 45° axis.

- **e.g**

$$P_1 = (x, y)$$

$$P_2 = (y, x)$$

$$P_3 = (-y, x)$$

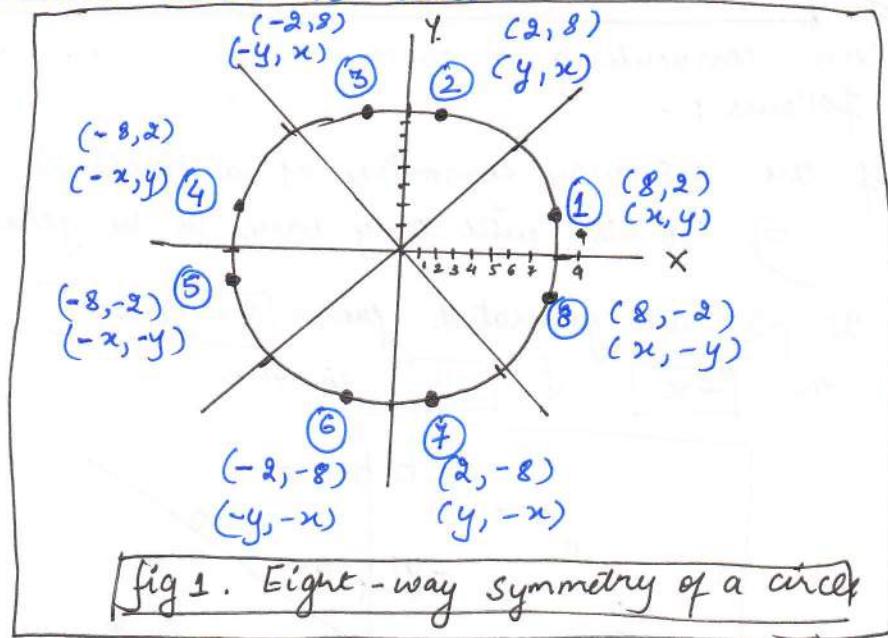
$$P_4 = (-x, y)$$

$$P_5 = (-x, -y)$$

$$P_6 = (y, -x)$$

$$P_7 = (x, -y)$$

$$P_8 = (x, y)$$



Defining a Circle

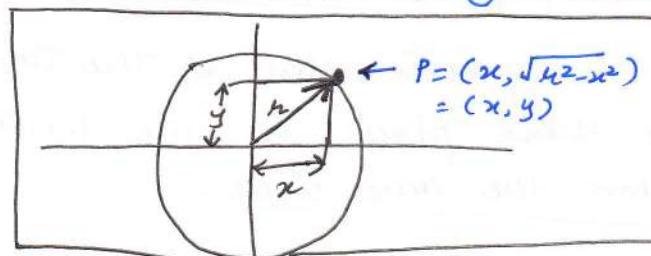
:- There are 2 standard methods of mathematically defining a circle centered at the origin.

(1) It defines a circle with the second-order polynomial

Eq^n.

$$y^2 = x^2 - r^2$$

where
 x = x coordinate
 y = y coordinate
 r = radius of the circle



Disadvantage → It is very inefficient method :: for each pt., both x & y must be squared and subtracted from each other, then the sq. root of the result must be found.

fig 2. Circle defined with a second-degree polynomial Eq^n.

(2) 2nd method of defining a circle makes use of trigonometric funcs

$$x = r \cos \theta$$

$$y = r \sin \theta$$

where
 θ = current angle
 r = radius of the circle

x = x coordinate
 y = y coordinate

Disadvantage :-

computation of the values of $\sin\theta$ & $\cos\theta$ is even more time-consuming than the calculations required by the first method.

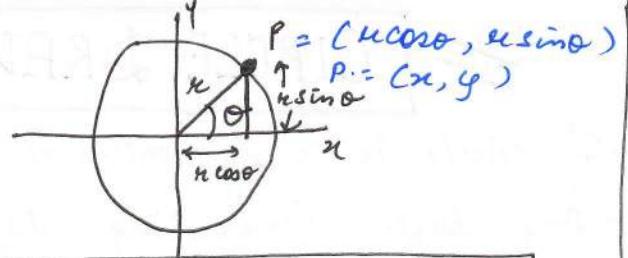


fig 3. Circle defined with Trigonometric func.

(A) BRESENHAM'S CIRCLE ALGORITHM :-

Scan converting a circle using Bresenham's algorithm works as follows :-

- If the 8-way symmetry of a circle is used to generate a circle
⇒ points will only have to be generated through a 45° angle
- If pts. are generated from $[90^\circ \text{ to } 45^\circ]$, moves will made only in the $[+x]$ & $[-y]$ direction.

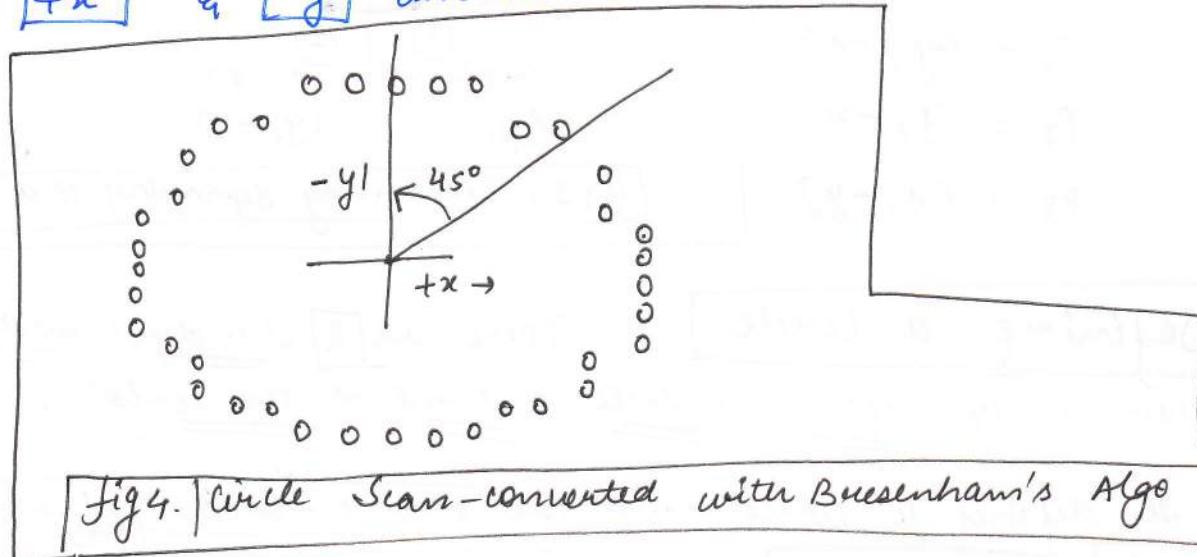


Fig 4. Circle Scan-converted with Bresenham's Algo

Best approximation of the true circle → will be described by those pixels in the raster that fall the least distance from the true circle.

Each new point closest to the true circle can be found by taking either of two actions.

- move in the $[x]$ direction one unit
- or move in the $[-y]$ direction one unit.

Assume (x_i, y_i) are the coordinates of the last scan-converted pixel upon entering step i (2)

As the coordinates of T are $(x_i + 1, y_i)$
& those of S are $(x_i + 1, y_i - 1)$

the following distances can be calculated as:-

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

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

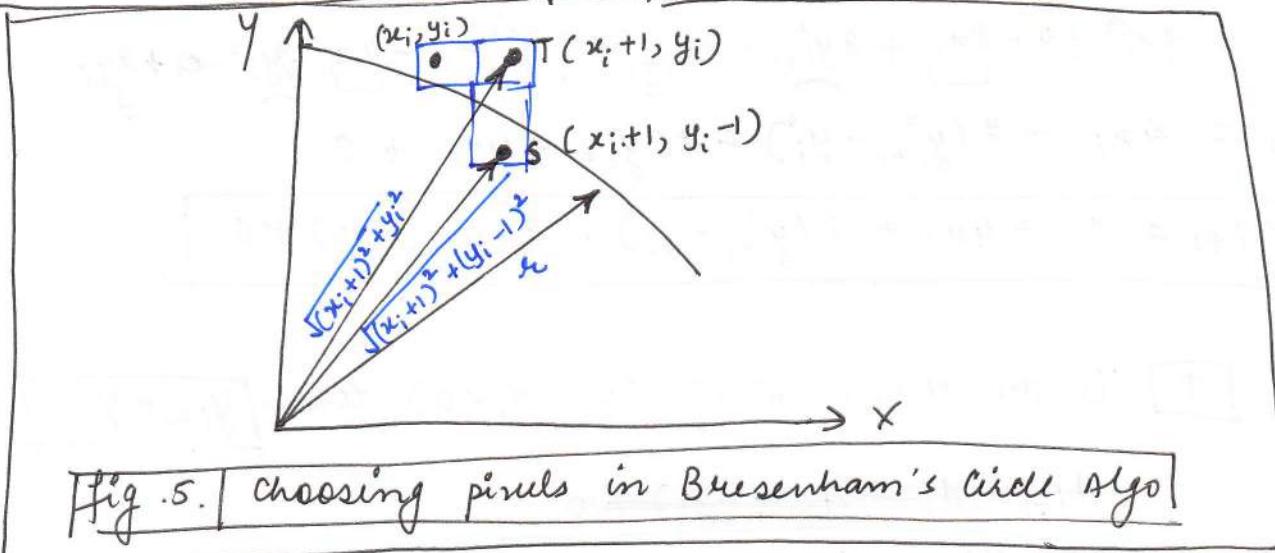


Fig. 5. choosing pixels in Bresenham's Circle Algo

The fun D → provides a relative measurement of the distance from the center of a pixel to the true circle.

Since $D(T)$ will always be +ve (T is outside the true circle)
 $D(S)$ will always be -ve (S is inside " ")

a decision variable $|d_i|$ may be defined as follows:-

$$\begin{aligned} d_i &= D(T) + D(S) \\ \therefore d_i &= 2(x_i + 1)^2 + y_i^2 - 2(y_i - 1)^2 - 2\alpha^2. \end{aligned} \quad \text{--- (1)}$$

$D(T) \rightarrow$ dist. from origin to pixel T
 $D(S) \rightarrow$ dist. from origin to pixel S

* when $|d_i| < 0$

$\Rightarrow |D(T)| < |D(S)|$ & pixel T is chosen.

* when $|d_i| \geq 0$

$\Rightarrow |D(T)| \geq |D(S)|$ & pixel S is chosen.

→ we can also write the decision variable $|d_{i+1}|$ for the next step:-

$$d_{i+1} = 2(x_{i+1} + 1)^2 + y_{i+1}^2 - (y_{i+1} - 1)^2 - 2\alpha^2$$

Hence

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

Since $x_{i+1} = x_i + 1$ we have

$$\begin{aligned} d_{i+1} - d_i &= 2(x_i + 2)^2 + y_{i+1}^2 + y_{i+1}^2 + 1 - 2y_{i+1} - 2(x_i + 1)^2 - y_i^2 - (y_i - 1)^2 \\ &= 2(x_i^2 + 4 + 4x_i) + 2y_{i+1}^2 - 2y_{i+1} - 2(x_i^2 + 1 + 2x_i) - y_i^2 - y_i^2 + 1 + 2y_i \\ &= \cancel{2x_i^2} + 8 + \cancel{8x_i} + \cancel{2y_{i+1}^2} - \cancel{2y_{i+1}} - \cancel{2x_i^2} - 2 - \cancel{4x_i} - \cancel{2y_i^2} - \cancel{1} + \cancel{2y_i} \end{aligned}$$

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

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

* If T is the chosen pixel (i.e. $d_i < 0$) then $y_{i+1} = y_i$ &

$$d_{i+1} = d_i + 4(x_i - y_i) + 10$$

$$d_{i+1} = d_i + 4x_i + 6$$

* If S is the chosen pixel (i.e. $d_i \geq 0$) then $y_{i+1} = y_i - 1$

$$d_{i+1} = d_i + 4(x_i - y_i) + 10$$

$$y_{i+1} = y_i - 1$$

Hence we have :-

$$d_{i+1} = \begin{cases} d_i + 4x_i + 6 & \text{if } d_i < 0 \\ d_i + 4(x_i - y_i) + 10 & \text{if } d_i \geq 0 \end{cases}$$

Algo :-

Step 1 :- put $x = D$; $y = R$ in eq ① we get $d = 3 - dx$

Step 2 :- Repeat steps while $x \leq y$

(a) Set pixel (x, y)

(b) If $(d < 0)$ then Set $d = d + 4x + 6$
else

(i) Set $d = d + 4(x - y) + 10$

(ii) $y = y - 1$ (end if)

Step 3 :- (c) $x = x + 1$ end loop
END.

(B) MIDPOINT CIRCLE ALGORITHM :-

- It is based on the following function for testing the spatial relationship between an arbitrary point (x, y) & a circle of radius r centered at the origin :-

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

-①

- Now consider the coordinates of the pt. halfway b/w pixel T & pixel S $\left(x_{i+1}, y_{i+1} - \frac{1}{2} \right), \left(x_{i+1}, y_{i+1} + \frac{1}{2} \right)$
This is called the mid pt. & we use it to define a decision parameter :-

$$P_i = f\left(x_{i+1}, y_{i+1} - \frac{1}{2}\right) = (x_{i+1})^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - r^2 \quad \text{(from eq)}$$

- * if P_i is -ve, \Rightarrow mid pt is inside the circle & we choose pixel T
- * if P_i is +ve (or equal to 0) \Rightarrow mid pt is outside the circle (or on the circle) & we choose pixel S

- Why, the decision parameter for the next step is :-

$$P_{i+1} = (x_{i+1} + 1)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - r^2 \quad -③$$

- Since $x_{i+1} = x_i + 1$, we have

$$\begin{aligned} P_{i+1} - P_i &= [(x_i + 1) + 1]^2 - (x_i + 1)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - \left(y_i - \frac{1}{2}\right)^2 \\ &= x_i^2 + 4x_i + 4 - x_i^2 - 2x_i + y_{i+1}^2 + \frac{1}{4} - y_{i+1} - y_i^2 + \frac{1}{4} - y_i \\ &= 2(x_i + 1) + 1 + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i) \end{aligned}$$

$$P_{i+1} = P_i + 2(x_i + 1) + 1 + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i) \quad -④$$

* If pixel T is chosen $\Rightarrow P_i < 0$

we have

$$Y_{i+1} = y_i$$

* If pixel S is chosen $\Rightarrow P_i \geq 0$

we have

$$Y_{i+1} = y_i - 1$$

Thus.

Thus

$$P_{i+1} = \begin{cases} P_i + 2(x_i + 1) + 1 & \text{if } P_i < 0 \\ P_i + 2(x_i + 1) + 1 - 2(y_i - 1) & \text{if } P_i \geq 0 \end{cases}$$

- (5)

We can continue to simplify this in terms of (x_i, y_i) & get

$$P_{i+1} = \begin{cases} P_i + 2x_i + 3 & \text{if } P_i < 0 \\ P_i + 2(x_i - y_i) + 5 & \text{if } P_i \geq 0 \end{cases}$$

- (6)

Algorithm :-

Step 1 :- put $x=0$, $y=r$ in eq (2)
we have $P = 1-r$

Step 2 :- Repeat steps while $x \leq y$

1) Plot (x, y)

2) If $P < 0$

then set $P = P + 2x + 3$

else

(i) $P = P + 2(x-y) + 5$

(ii) $y = y - 1$ (end if)

3) $x = x + 1$ (end loop)

Books Referenced
Computer Graphics

→ by Schaum's series

Step 3 :- END.