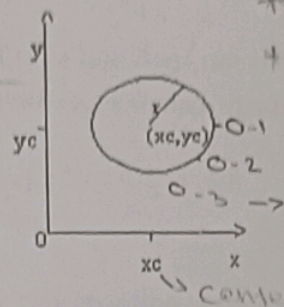
UNIT-II

CIRCLE GENERATION ALGORITHM

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

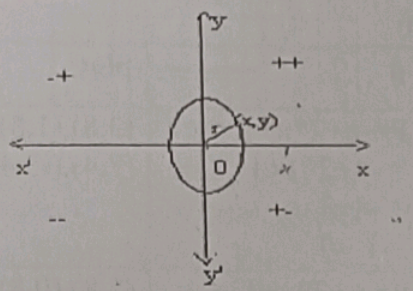


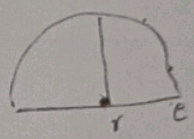
Equation: $(x - x_c)^2 + (y-y_c)^2 = r^2$. If the center is at the origin (0,0), then the equation is $x^2 + y^2 = r^2$

Another way to calculate the points of a circle is by means of parametric polar form that is $x = x_c + r\cos\theta$ $y = y_c + r\sin\theta$

When a display is generated with these equation using a fixed angular step size, a circle is plotted with equally spaced points along the **circumference.**

SYMMENTRY OF A CIRCLE





MID POINT CIRCLE DRAWING ALGORITHM

```
Procedure Circle (xc,yc,r: int)
Var   x,y,p: int;
      Procedure Plot;
      begin
            setpixel (xc+x,yc+y,1); setpixel (xc-x, yc+y,1);
            setpixel (xc+x, yc-y,1); setpixel (xc-x, yc-y,1);
            setpixel (xc+y, yc+x,1);setpixel (xc-y, yc+x,1);
            setpixel (xc+y, yc-x,1); setpixel (xc-y, yc -x ,1);
       end;
Begin
      x=0,y=r;
```

```
Plot;
p=1-r;
while (x<y)
{
 If (p<0)
       x =x+1;
 else
  {
     x = x+1;   y = y-1;
  }
 if (p<0)
    p = p+2*x+1;
 else
    p = p+2(x-y)+1;  plot;
}
End.
```

Example: Draw a circle with radius r=5, centered at a period (2,3).

Given $x_c=2, y_c=3, r=5$;

x=0,y=5; Plot;

[(2+0,3+5); = (2,8) (2-0,3+5); = (2,8)
(2+0,3-5); =(2,-2) (2-0,3-5); =(2,-2)
(2+5,3+0); =(7,3) (2-5,3+0); =(-3,3)
(2+5,3-0); =(7,3) (2-5,3-0); =(-3,3)]

p=1-r; = 1=5; =-4 <0

Index	X	y	p	plot
	0+1	5	P= p+2*x+1 = -4+2*1+1 = -4+3 =-1 <0	(3.8),(1,8),(3,-2),(1.-2). (7,4),(-3,4).(7,-2),(-3.-2)
	1+1 = 2	5	P= p+2*x+1 =-1+2*2+1 =-1+4+1 =4 >0	(7,8),(0,8),(4,-2),(0.-2). (7,5),(-3,5),(7.-1),(-3.-1) ……..

CHARACTER GENERATION

The characters scan be displayed in variety of sizes and styles. The overall design for a set of characters [family] is called as typeface. Earlier it was called as fonts. The term fonts refer to a set of casing metal characters. Font in a particular size and format.

Typefaces are divided into two major groups.

1. serif
2. san serif

1) Serif

(Ex) T (times new roman)A serif type has small lines or accents at the end of main strokes.

2) San serif

(Ex) T (Arial) San serif type one does not have accents.

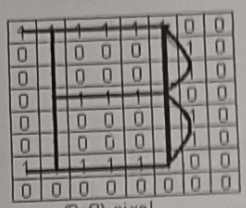
There are two different representation are used for storing the computer fonts.

(i) Bit map fonts
(ii) Outlines fonts.

Bit map fonts

A simple method for representing a character in a particular type face by means of a rectangle grid pattern.

(ex)

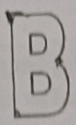


(8 x 8) pixel

Bit map fonts simple to define and display. That is character grid only needs to be mapped to a frame buffer. But, these types of fonts require more memory space.

ii)outline fonts(post script font)

In this scheme a character is designed using straight line and curve sections. Eg:



- Outline fonts required less storage. we can produce bold type, Italic or different sizes. By manipulating the curve definitions for the character outlines.

ATTRIBUTES OF OUTPUT PRIMITIVES

Any parameter that affects the way. A primitives is to be displayed is referred to as attribute parameters.

(ex) color, size ,etc…

To incorporate attribute options into a graphic package is to extend the parameter list associated with each primitive function to include the appropriate attributes.

LINE ATTRIBUTES

Basic attributes: type, width, and color
Other attributes: select a pen or brush

Line type

line type can be:

solid _____

dashed --------

dotted ……..

We modify the line algorithm to generate such line by setting the length and spacing of displayed solid sections along the line path. A dashed line can be displayed by generating a inter-dash spacing similar to solid lines. A dotted line can be displayed by generating very short dashes with the spacing equal to or greater than the dash size.

Function:

Begin

Setlinetype[i]

End

Input value

Where i is an integer value (1,2,3,4) to generate line type solid, dashed, dot tend or dash dot tend.

Line width (thickness)

The line width option is depend on the capability of the output device. A heavy line on the monitor could be displayed as adjusting parallel lines, while a pen plotter require pen changes.

Function:

Setlinewidth(w); where w is a positive number to indicate the relative width of a line.

1 for standard width
<1 for thin line
>1 for thick line
>2 for thick line

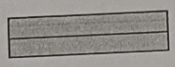
line darkness

For a raster display, a standard width line is generated by single pixels at each position. Other width lines are displayed as positive integers multiples of the standard line by plotting additional pixels along the adjacent parallel line path.

Line shape (CAPS)

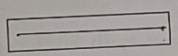
For heavy lines, we can adjust the shape of the line ends. To give them a better appearance by adding the line CAPS.

Butt cap



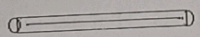
These types of line caps is obtained by adjusting the end position of the component parallel lines. So, that the thickness is displayed with square ends, that are perpendicular to the line path.

Round cap



Round cap is obtained by adding a filled semi circle at end of Butt cap.

projecting square cap

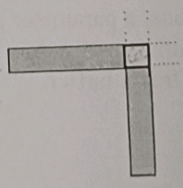


In this type, simply extend the line end & add Butt cap. That are positioned one half of the line width beyond the specified end points.

Thick lines join

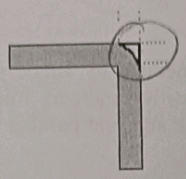
We can generate the thick poly lines that are smoothly joined at the cost additional processing at the segment end points.

Miter join



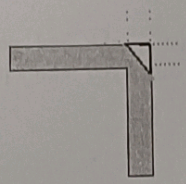
It is produced by extend in the outer boundaries of each of the line two lines until the meet

Round join



It is generated by capping the connection between the two segments with a circular boundary whose diameter is equal to the line width.

Bevel join



It is generated by displaying the line segment with Butt caps and filling the triangle gap where the segment meet .

Pen and brush options

With some packages, line can be displayed with pen and brush options. We have the options like shape, size and pattern. These shapes can be stored in a pixel mask that identifies the arrays of pixel positions. That are to be set along the line path.

Pen shape	line
Δ	ΔΔΔΔΔΔΔ
●	●●●●●●●
♣	♣♣♣♣♣♣♣
*	*******

Line color

When a system have a color options, a parameter giving the current color is included in the list of attributes. A number of colors choices depends on the number of bits available per pixel in the frame buffer.

Function

Setlinecolor(c)

(Ex) setlinecolor(3) - cyan
Setlinecolor(1) - blue
Setlinecolor(4) -red

CURVE ATTRIBUTES

The parameters are some as line segments. We can display the curves with varying colors, width, dot or dashed patterns and available pen and brush options.

Color and Gray scale levels

Various colors and intensity option can be made available to user depending on the system. Color options are numerically coded with values ranging from 0 to some positive number. For CRT monitor these color codes are converted into intensity level setting for the electron beams. For the color blotters. The codes to control the inkjet deposits or pen selections.

We can store the color codes directly in the frame buffer or we can store the color codes in a separate table and use pixel values as an index into this table. For the direct scheme, we have three bits of storage per pixel.

Code	R	G	B	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

In this case only eight colors can be displayed. By adding more bits per pixel to the frame buffer increases the number of colors choice. For six bits ie, two bits per each colors, we can get 64 colors.

For a full color RGB system with 1024*1024 resolution and 24 bits per pixel (R(8)+G(8)+B(8)) will require 3mp storage for the frame buffer.

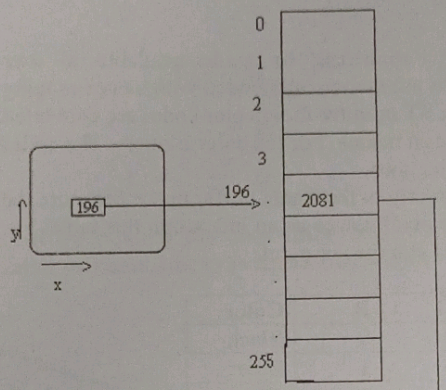
Color tables

Another scheme for storing color values is color table or color look up table or video look up table. Here the frame buffer values are now used as instances to the color table.

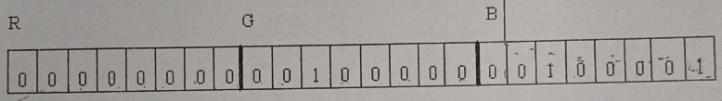
Each pixel can reference any one of the 256 enable positions and each entry in the table uses 24 bits for the RGB color. For ex

The color code 2081. A combination of green and blue color is displayed at (x, y). A system that have color table can have a choice of 17 millions colors.

Color lookup



5



Gray scale For monitor with no color options (ie, Black & White monitors). We can set the shades of Gray or Gray scale for the display. Numeric values from zero to one can be used to specify Gray scale levels. Which are then converted to binary codes for storage in the raster.

Buffer - temprou
Strage

Frame buffer	Intensity values	Color
0 (0,0)	0	Black
1 (0,1)	0.33	Dark green
2 (1,0)	0.67	Light green
3 (1,1)	1	white

AREA FILL ATTRIBUTES

For filling a defined region, we have a choice of solid color, pattern fill and choice for particular color and patterns. These fill options can be applied to polygon regions or to areas defined with curve boundaries.

↳ to defined with th
curve boundaries

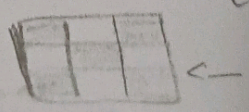
Fill styles

Area are displayed with three basic fill styles.

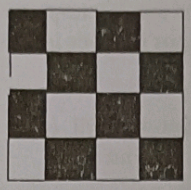
- Hollow with color border Δ

↳ to fill the so
with the boun
area.

↳ It have a sho
solid color, pa
fill to various
places.



- Filled with solid color ▲
- Filled with a pattern are designed

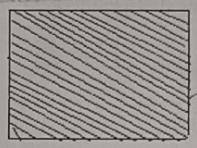


Function

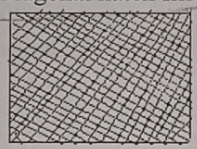
Setinteriorstyle[s;]

Where s can be any one of the above three styles.

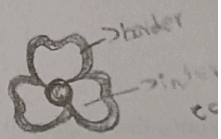
Another value for fill style is hatch. Which is used to filled on area with selected hatching patterns parallel or crossed line.



Diagonal hatch fill



Diagonal gross hatch fill

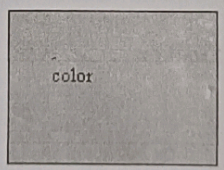


Follow area displayed using only the boundary outline with interior color. A solid fill displayed in a single up to and including the borders of the region.

Function

Setinteriercolor[c;]

As c is the designed color code.



Other fill options include specification for the edge type. Edge width and edge color of a region.

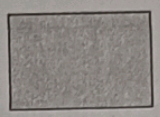
Pattern fill function

Setpatternstyle[s;]

Where s is the pattern index. Pattern fill can be implemented by modifying the scan line procedure so that a selected pattern is super imposed on the scan lines.

Soft fill

We can refined the filled areas. So, that the fill color is combined with the background color is referred to as soft fill or Dint Fill.



CHARACTER / FILL ATTRIBUTES

Character is controlled by the attributes such a font, size,color and orientation.

Font is a set of characters with a particular designed style.

i) arial
ii) Times new roman
iii) Dual rural
iv) Swiss
v) ……etc

There are many text options are available to the user that is we are having more number of font styles.

Fonts can also be displayed with underlining styles.

1. solid _____
2. dotted .....
3. double ═══

Fonts also displayed in following styles

i) **BOLD**
ii) *Italic*
iii) out line
iv) *shadow*

Function

Settextfont(t);

Color

To set a color to text we have the function

Settextcolor(c);

(Ex) color 1;
Color 2;
Color 3;

Color setting for displaying the text are stored in the system attribute list and used by the procedures that load character definitions into the frame buffer.

Text size

We can adjust the text size by scaling only character width. The character size is specified by points. That is one point equal to

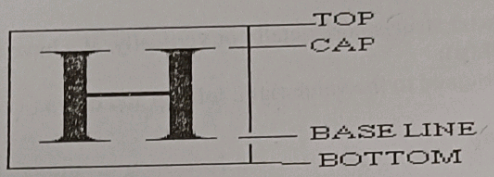
1point = 0.013 inch

ex)

8 point
12 point
16 point

Character height

The character height is defined by the distance between the base line and the cap line of the character.



Function

Setcharheight(h); Where h is a real number greater than zero

height1 height2

Character Width

Set the width of the text will have the function.

Setcharexpancian(w);

Where w is a positive real number that scales the body width of the characters.

Character spacing

Spacing between the characters are controlled by the function.

Setcharspace(s);

Where s is a real number negative values for 's' over lock the character bodies. Positive values inserts space to spread out the displayed characters. Amount of spacing to be applied is determined by multiplying the values(s) by the character height.

(ex)

Space [Normal]

S p a c e [positive]

Orientation

For orientation for a character string is set by the direction of the character up vector.

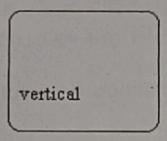
Function

Setcharpvector(v);

Where v is assigned to two values. That specifies the x and y is vector components.

(ex)

V = (1,1) The direction of the up vector is 45^0



Arrangement

Arrange the character strings horizontally or vertically. We have a function.

Settextpath(p);

Where p can be assigned to the value right, left, up and down.

(ex)

T **up**
S
E
B

T S E B B E S T

right B **left**
E
S
T **down**

Alignment

Alignment attribute specifies how text is to be positioned with respect to the staring co-ordinate.

Function

Setalign(h,v);

The horizontal alignment 'h' can be set to left, center, right.

Vertical alignment 'V' can be set to top, cap, half, base or bottom.

Marker

A marker symbol is a single character that can be displayed in different colors and different sizes.

Function

Setmarkertype(m);

Were m is an integer value.

ex)

M = 1 *
M = 2 #
M = 3 ♣
M = 4 ●
M = 5 ♥
M = 6 ♠
M = 7 Δ
M = 8 ◊
…

ANTI-ALIASING

ALIASING

Display primitives generated by raster algorithms have a jagged or stair step appearance. Because of the sampling problems. (That is, converting real values into integer pixel position.)

This distortion of information due to low frequency sampling (Under sampling is called **aliasing.**

The displayed can be improved by applying some of the anti-aliasing methods.

- ✓ Super sampling straight lines.
- ✓ Pixel weight mask.
- ✓ Area sampling.
- ✓ Filtering techniques.
- ✓ Pixel phasing.
- ✓ Compensation of line intensity differences is one way to increase the sampling rate is to display the object at higher resolution. But, that is not complete solution to aliasing problem.

Super sampling

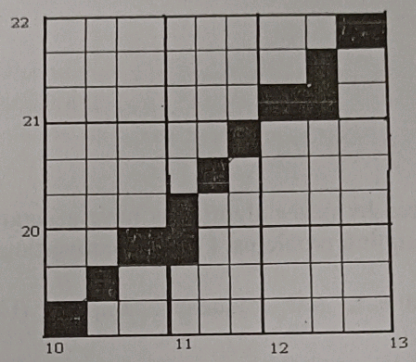
Another method to increase sampling rate by considering the screen as if it where converted with fine grid that is actually available. We can than use multiple sampole points across the fine grid to determine the intensity level for each pixel. This technique of sampling the result and through resolution as called as **super sampling or post filtering.**

Super sampling straight line segment

Super sampling line can be done in many ways. For the Gray Scale display of a line. We divide each pixel into a number of sub pixels and count the number of sub pixel that are along the line path. The intensity level for each pixel is than set to a value the proportional to sub pixel count.

Here each pixel area is divided into a equal sub pixel. The shaded region show that the sub pixels would be selected by a line algorithm. For this example

Three intensity levels are used. The pixel 10,20 is set to maximum intensity (level 3). Pixel at level 21 and 12. 22 are each set to next highest intensity level to land the pixels at 11, 20 and 12, 21 are set to lowest intensity above 0 (level 1). Thus the line intensity is spread out over a greater number of pixel and the stars effect is smoothed by display the line path.



In this example we considered pixels areas finite size. But be treated the mathematical entity with 0 width. If the take the finite width of the line into account. We can perform the sub sampling by getting each pixel intensity proportional to the number of sub pixels inside the polygon representing the line area.

1) 1am
2D Geometric transformation (translation, Rotation & scaling).

2) 5m
Reflection & polygon clipping.

UNIT III

2D-Geometric Transformations

Changes in place orientation, size and shape of an object are accomplished with geometric transformation that alters the coordinate descriptions of objects.

2D Basic Transformations

- Translation } move one place to another
- Rotation
- Scaling

Others

- Reflection
- Shear

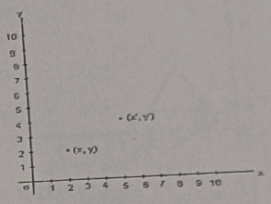
Translation

It is used to place the object from one location to another. If T_x, T_y are the given translation distances then the original position (x, y) moved to (x', y') is given by

$x' = x + T_x$; $y' = y + T_y$

Matrix form: $P' = P + T$

where $p = \begin{pmatrix} x \\ y \end{pmatrix}, P' = \begin{pmatrix} x' \\ y' \end{pmatrix}, T = \begin{pmatrix} T_x \\ T_y \end{pmatrix}$



The pair (T_x, T_y) is called Translation vector or shift vector.

Example: (Point Translation)

(x, y) = (2, 2)

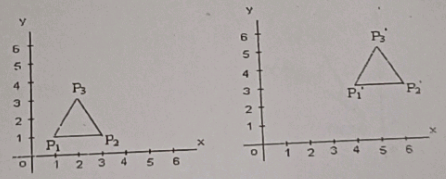
If T_x=3, T_y= 2 then

(x', y')= (x + T_x, y + T_y) = (5, 4)

Translation is a rigid body transformation that moves objects without deformation, (i.e., without changing the size)

A straight line segment is translated by applying the equations to each of the line endpoints and redrawing the line between the new endpoints. Polygons are translated by adding the transformation vector to the position of each vertex and regenerate the polygon again. Example: [Polygon Translation]

Before **After**



$P_1 = (1, 1)$
$P_2 = (3, 1)$
$P_3 = (2, 3)$

If $T_x = 3$, $T_y = 2$
Then

$P'_1 = (4, 3)$
$P'_2 = (6, 3)$
$P'_3 = (5, 5)$

P1 = (P1 + Tx) = (1,1) + 3 = (1,

Rotation

2D Rotation is used to repositioning the object along a circular path in the xy plane. To generate rotation we need rotation angle θ and the position called rotation point or pivot point. Positive values of θ define counter clockwise rotations. (Anti-clockwise). Negative values of θ rotate the object in clockwise direction (clockwise).

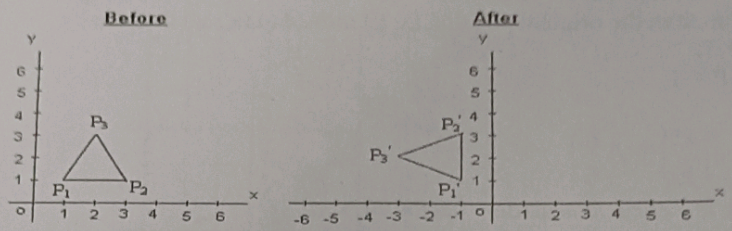
Transformation Equation: (about Origin) :

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

Matrix form: $P' = RP$

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

Example form: [Polygon Rotation]



$P_1 = (1, 1)$
$P_2 = (3, 1)$
$P_3 = (2, 3)$

If $\theta = 90°$ then

$x' = x\cos90° - y\sin90°$

$= x(0) - y(1)$

$x' = -y$

$y' = x\sin90° + y\cos90°$

$y' = x$

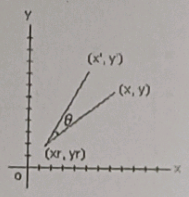
$P_1' = (-1, 1)$
$P_2' = (-1, 3)$
$P_3' = (-3, 2)$

Rotation of a point about any arbitrary position is given by

$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$

$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$

Example



A line segment is rotated by applying the above equations to the two endpoints and redraws the line between the new endpoints. Similarly for polygons are rotated by displaying each vertex through the specified rotation angle and redraw the polygon using the new vertices.

Scaling

It is used to alter the size of the objects. For the scaling factors S_x, S_y the transformation is given by

$x' = x.S_x$

$y' = y.S_y$

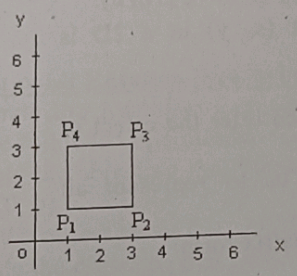
The matrix form is: $P' = S.P$

where $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, P = \begin{bmatrix} x \\ y \end{bmatrix}, S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$

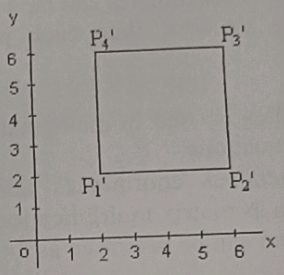
If $S_x = S_y$ then the scaling is called uniform scaling. If both S_x and $S_y = 1$ then the size of the object is unchanged. Unequal values of S_x and S_y result in a differential scaling. If the scaling factors value < 1, move objects closer to the co-ordinate origin and reduce the size of object. If values (S_x, S_y) are >1 move coordinate positions further from the origin and enlarge the size of the object.

Example: [Uniform scaling]

Before



After



$P_1 = (1, 1)\ P_2 = (3, 1)$

$P_3 = (3,3)\ P_4 = (1, 3)$

if $S_x = S_y = 2$

Then

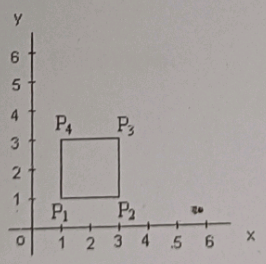
$P_1' = (2, 2)\ P_2' = (6, 2)$

$P_3' = (6, 6)\ P_4' = (2, 6)$

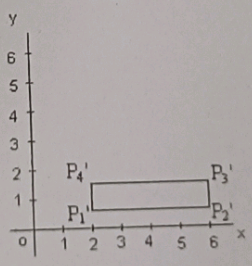
Example 2:

Differential scaling

Before



After



$P_1 = (1, 1)$ If $S_x = 2$, $S_y = 0.5$ $P_1' = (2, 0.5)$
$P_2 = (3, 1)$ Then $P_2' = (6, 0.5)$
$P_3 = (3, 3)$ $P_3' = (6, 1.5)$
$P_4 = (1, 3)$ $P_4' = (2, 1.5)$

Matrix representations and Homogeneous coordinates

Many graphics application involve sequence of transformations, For example in the case of animation, the object to be translated and rotated at each increment of the motion. So the matrix representations of translation, rotation and scaling can be reformulated so that such transformation sequences can be effectively processed.

All the 2D transformations are represented by 2x2 matrix form. Now we expand the 2x2 into 3x3 matrix form. This gives us to express all the equations as matrix multiplications.

To express any 2D transformation as a matrix multiplication, we represent each position (x, y) with the homogenous coordinate triple (x_h, y_h, h)

where $x = x_h / h \quad y = y_h / h$

Therefore the general homogeneous coordinates can be written as $(x.h, y.h, h)$.

A convenient choice is simply set h=1, then each point (x, y) in a 2D is represented as (x, y, 1).

The term Homogenous Coordinates is used in mathematics to refer the effect of this representation on Cartesian equations.

Expressing positions in homogeneous coordinates allow us to represent all geometric transformation equation as matrix multiplications.

For translation, the 2x2 matrix form can be rewritten as 3x3 matrix form.

$P' = T.P.$

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

T- Translation

For Rotation: $P' = R.P$

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

For Scaling: $P' = S.P$

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

Composite Transformations

We can setup a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of the individual transformations. Forming the products of transformation matrices is called as concatenation or composition of matrices.

Example: Two successive translations for a point can be calculated as follows:

$(x, y) \xrightarrow{T_1} (x', y') \xrightarrow{T_2} (x, y)$

i.e., $p'' = t_2.p'$

$$\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx_2 \\ 0 & 1 & ty_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & tx_1 \\ 0 & 1 & ty_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & tx_1 + tx_2 \\ 0 & 1 & ty_1 + ty_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Similarly two successive rotations R_1, R_2 applied to a point can be calculated from

$P'' = R_2.R_1.P$

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

For two successive scaling we have

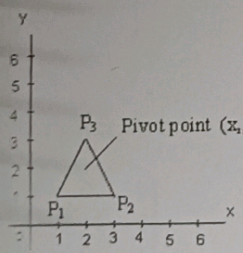
$P'' = S_2.S_1.P$

General pivot point Rotation (or) Arbitrary point Rotation

We can generate the rotation of an object about any selected point (x_r, y_r) by performing the following operations.

1. Translate the object so that Pivot point position is moved to the coordinate origin (0, 0).
2. Rotate the object about the origin.
3. Translate the object so that pivot point is returned to its Original position.

Example



$P_1 = (1, 1)$, $P_2 = (3, 1)$, $P_3 = (2, 3)$

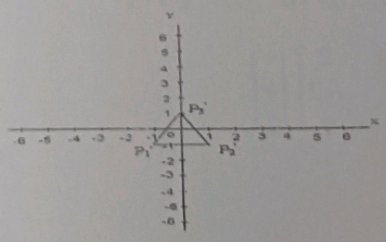
$(x_r, y_r) = (2, 2)$.

i) **Translate object to the origin.**

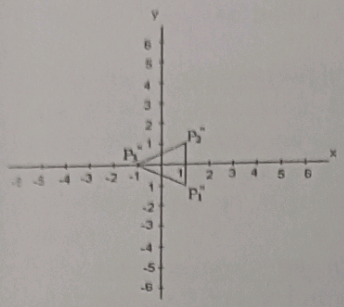
$P' = T_0.P$

where T_0=(-2,-2)

then $P_1' = (-1,-1)$, $P_2' = (1,-1)$, $P_3' = (0,1)$



ii) **Rotate the object (90°)**



$$P'' = R.P'$$
$$= \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ \\ \sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix}$$
$$= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -y' & x' \end{bmatrix}$$

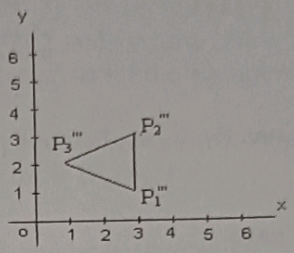
then $P_1'' = (-1,-1)$, $P_2'' = (1, 1)$, $P_3'' = (-1, 0)$

iii)Translate the origin to (x_r, y_r)

$P''' = T.P''$

T = (2, 2)

$P_1''' = (3, 1)$, $P_2''' = (3, 3)$ $P_3''' = (1, 2)$



The composite transformation matrix for this sequence is given by

P''' $= T.P''$

$= T.R.P''$

$= T.R.T_0.P.$

$$p''' = \begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -T_x \\ 0 & 1 & -T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

For the given figure:

$$= \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ & 0 \\ \sin 90^\circ & \cos 90^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

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

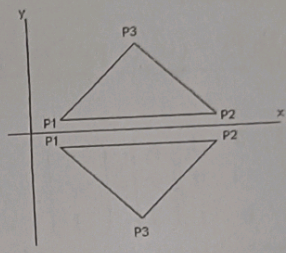
$$= \begin{bmatrix} 0 & -1 & 4 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$x''' = -y + 4$
$y''' = x$
$P_1 = (1, 1), P_2 = (3, 1), P_3 = (2, 3)$
$P_1''' = (3, 1)\ P_2''' = (3, 3)\ P_3 = (1, 2)$

Other Transformations

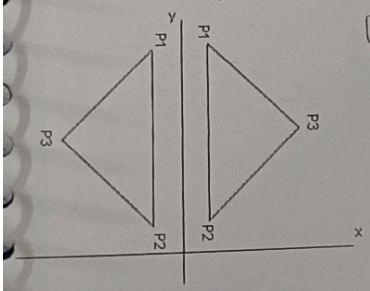
Reflection

It is a transformation that produces a mirror image of on object. A mirror image for a 2D is generated relative to an axis of reflection by rotating the object 180° about the reflection axis.



The reflection about the line y=0, x-axis is given by the following matrix.

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

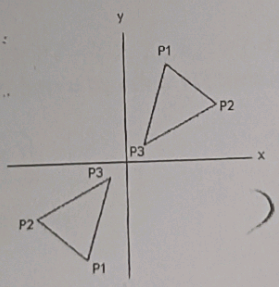


Reflection about the y-axis is given by the following matrix:

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

The reflection relative to the coordinate origin is given by

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



The matrix form

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

Example:
$P_1 = (2, 3), P_2 = (1, 1), P_3 = (3, 1)$

Then Ref_x is

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

i.e

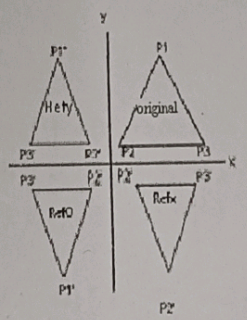
$$\boxed{\begin{aligned} x' &= x \\ y &= -y \end{aligned}}$$

$P_1' = (2, -3), P_2' = (1, -1), P_3' = (3, -1)$

Ref_y is =

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

i.e $\boxed{\begin{aligned} x' &= -x \\ y' &= y \end{aligned}}$ $P_1' = (-2, 3), P_2' = (-1, 1), P_3' = (-3, 1)$



Ref_0 is =

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

i.e

$$\boxed{\begin{aligned} x' &= -x \\ y' &= -y \end{aligned}}$$

$P_1' = (-2, -3), P_2' = (-1, -1), P_3' = (-3, -1)$

Shear

A transformation that distorts the shape of an 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 shear.

An x-direction shear relative to x-axis is given by

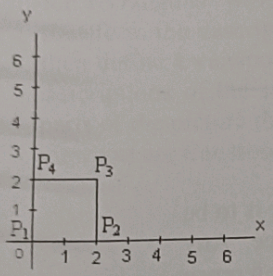
$$x' = x + s_h * y$$
$$y' = y$$

Any real number can be assigned to the shear parameter (S_h). Matrix form:

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

Example: x-direction shear.

Before



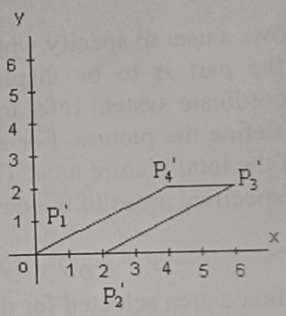
$P_1 = (0, 0), P_2 = (2, 0), P_3 = (2, 2), P_4 = (0, 2)$

If $S_h = 2$ then $x' = x + s_h * y = x + 2y$

$y' = y$

After:

$P_1' = (0, 0), P_2' = (2, 0), P_3' = (6, 2), P_4' = (4, 2)$



Negatives of S_h shift the coordinate positions to the left. For example:

If $S_h = -2$ then

$x' = x + s_h * y \quad = x - 2y$

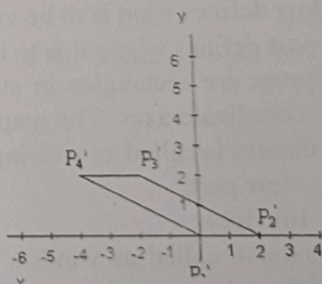
$y' = y$

$P_1' = (0, 0), P_2' = (2, 0), P_3' = (-2, 2), P_4' = (-4, 2)$

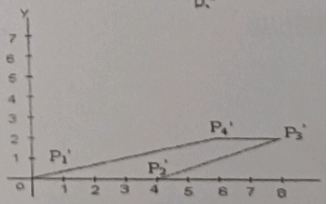
An x-direction shear relative to other reference lines is given by

$$x' = x + s_h * (y - y_r)$$
$$y' = y$$

where y_r is the reference point.



For example, $y_r = -1$ and $S_h = 2$ we have



$x' = x + 2(y - (-1)) = x + 2(y + 1)$
and $y' = y$

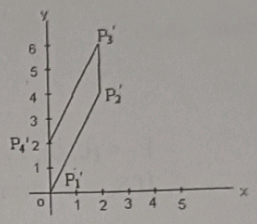
Therefore,
$P_1' = (0, 0)$, $P_2' = (4, 0)$, $P_3' = (8, 2)$, $P_4' = (6, 2)$

Similarly the y-direction shear relative to y-axis
$x' = x$
$y' = y + s_h . x$

Matrix form:

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

Example:
If $S_h = 2$
$x' = x$
$y' = y + 2x$
$\therefore P_1' = (0, 0)$, $P_2' = (2, 4)$, $P_3' = (2, 6)$, $P_4' = (0, 2)$



2D-Viewing

A graphics package allows a user to specify which part of a defined picture is to be displayed and where the part is to be displayed on the display device. Any convenient Cartesian coordinate system refer to as the world coordinate reference frame, can be used to define the picture. For a 2D picture, a view is selected by specifying a sub-area of the total picture area. The picture parts within the selected area then mapped on to specified areas of the device coordinates.

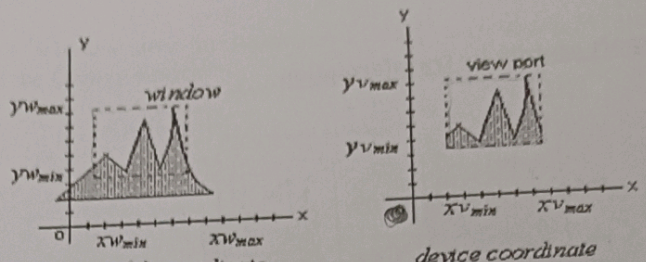
The viewing pipeline

- A world coordinate area selected for display is called a window.
- An area on display device to which a window is mapped is called a view port.
- The window defines what is to be viewed.
- The viewport defines where it is to be displayed.

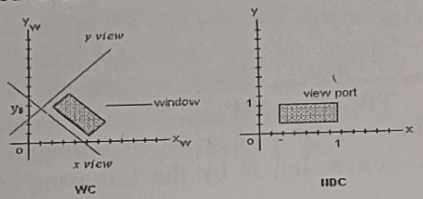
Windows and viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes. The mapping of a part of a world coordinate scene to device coordinates is called as viewing transformation.

(i.e.,) window to view port.
(world) (device)

In a 2D system this is also called as window to viewport transformations or the windowing transformation.

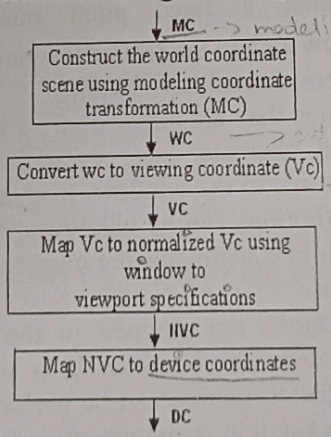


First we construct the scene in the world coordinates using the standard output primitives and attributes. Next, to obtain a particular orientation for the window we can setup a 2D viewing coordinate system in the world coordinate plane and define a window in the viewing coordinate system. The viewing coordinate reference frame is used to provide a method for setting up arbitrary orientations, for rectangular windows. once the viewing frame is established we can then transform the descriptions in world coordinate (w_c) to viewing coordinate (v_c).We then define a viewport in normalized coordinate (i.e., from 0 to 1) and map the v_c descriptions of the scene to normalized coordinate (N_c). At the final step all the parts of the picture that lie outside the viewport are clipped and the contents of the viewport are transformed to device coordinates (D_c).

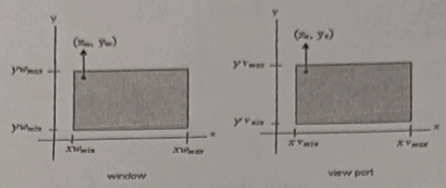


2D - viewing Transformation Pipeline

First construct the world scene using modeling coordinate transformation (MC)



Window to viewport coordinate transformation



A point at position (x_w, y_w) in the window is mapped into the position (x_v, y_v) in the associated viewport. To maintain the same relative placement we require that

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \quad and$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Solving these expressions we have

$xv = xv_{min} + (xw - xw_{min}).s_x$

$yv = yv_{min} + (yw - yw_{min}).s_y$

where the scaling factors

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} \quad and \quad s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

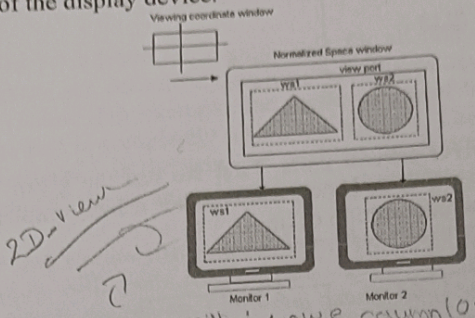
These set of equations can also be derived with a set of transformations that converts window to viewport. The conversion is by the following sequence of transformations

- Perform scaling transformation using a fixed point position of (xw_{min}, yw_{min}) that scales the window area to the size of the viewport.
- Translate the new window area to the position of the viewport.

Character strings can be handled in 2 ways when they are mapped into a viewport. The simplest mapping maintains a constant character size, even though the viewport area may be enlarged or reduced relative to the window. In systems that allows for change in character size, string definitions can be windowed the same as other primitives. For characters formed with line segments, the mapping can be carried out as a sequence of line transformations.

From normalized coordinates, object descriptions are mapped to the various display devices. Any number of output devices can be open in a particular application and another window to viewport transformation can be performed for each output device. This mapping is called workstation transformation. It is done

by selecting a window area in normalized space and viewport area in the coordinates of the display device.



CLIPPING

Any procedure that identifies those portions of a picture that are either inside or outside of a specified region is called as clipping algorithm or Clipping. The region against which an object is to be clipped is called a clip window.

Applications: Extracting the part of a defined scene for viewing, Identifying visible surfaces in 3D views, Anti aliasing lines or objects, Picture copying, moving, erasing or duplicating.

Depending on the applications the clip window can be general polygon or it can be curved boundaries. Clipping algorithms can be applied in world coordinates, so that only the contents of the window interior are mapped to device coordinates. In other way, the complete world coordinate picture can be mapped first to the device coordinates or normalized device coordinates, then clipped against the viewport boundaries.

Types: Point clipping, Line clipping, Area clipping [Polygon clipping], Curve clipping, Text clipping.

Point clipping

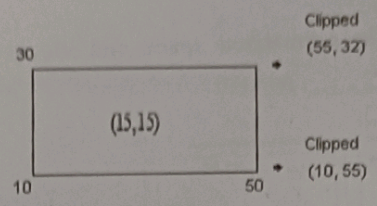
Assuming that the clip window is a rectangle, we save a point P =(x, y) for display if the following are satisfied.

i.e., $xw_{min} \leq x \leq xw_{max}$

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

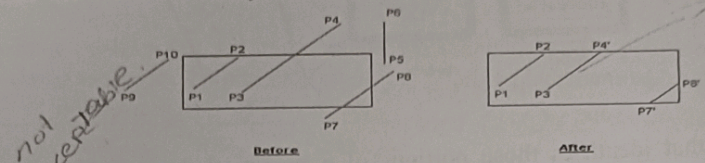
where $xw_{min}, xw_{max}, yw_{min}$ and yw_{max} are the edges of the clip window.

If any of the inequalities not satisfied the point is clipped. (i.e..) not saved for display.



Line Clipping

The line clipping procedure involves several parts. First we test the given line segment lies completely inside the window. If not, test whether it lies completely outside the window. Finally find the intersection of lines.

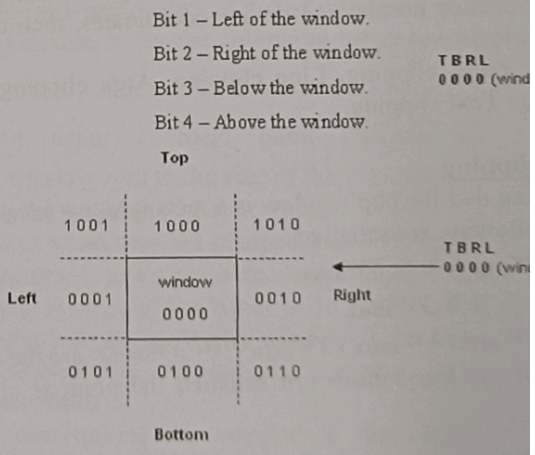


Here the line (P_1, P_2) is completely inside, therefore both the end points are saved for the (P_3, P_4), P_3 is inside the window and P_4 is outside the window. Therefore we clipped against the top window we get P_4'. Lines (P_5, P_6) and (P_9, P_{10}) are completely outside the window. Therefore the lines are discarded. Line (P_7, P_8) clipped against the bottom window and right side of the window, finally we get P_7' and P_8'.

Cohen-Sutherland Line Clipping Algorithm

Every line endpoint in a picture is assigned a four binary code called a Region code. It identifies the location of the point relative to the boundaries of the clipping rectangle. Each bit position in region code is used to indicate one of the four relative coordinate positions of the point with respect to clip window. The coding is as follows: (see above figure)

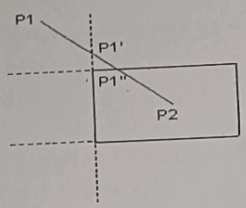
A value 1 in any bit position indicates that the point is in that relative position. Otherwise the bit is set to 0.



Bit values in the region code are determined by comparing the endpoint coordinates (x, y) to the clip boundaries. For examples, bit 1 is set to 1 if x is less than xw_{min} the other three bit values are determined by similar comparison.

Once we have established the region code for all line endpoints we can easily determine which are completely inside the window and which are completely outside the window. For example, the region codes of both endpoints of line are 0000 then the line is completely inside the window. Therefore accept these lines. Any lines that have 1 in the same bit position for each endpoints then the line is completely outside the window. Hence reject these lines.

The method that can be used to test lines for clipping is to perform the logical AND operation with both region code. If the result is not **0000** the line is completely outside the region. Lines that cannot be identified as completely inside or completely outside a clip window by these tests are checked for intersection with the window boundaries. We begin the clipping process for a line by comparing an outside endpoint to a clipping boundary to determine how much of the line can be discarded. Then the remaining part of the line is checked against the other boundaries, and continues until either the line is totally discarded or a section is found inside a window.



Consider the (P_1, P_2). The point P_1 is outside the window. When P_1 is clipped against the left window we get P_1'and then P_1' is clipped against top of the window we get P_1".For a line with endpoint coordinates (x_1, y_1) and (x_2, y_2), the y coordinate of intersection point with a vertical boundary is given by:

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

where x is set to either xw_{max} or xw_{min} and the slope $m = \frac{y_2 - y_1}{x_2 - x_1}$

Similarly intersection with a horizontal boundary the x-coordinate is given by

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

where y is set to yw_{max} or yw_{min}

```
Procedure Cohen(wmin, wmax: dcpt2; n: int, pts: wcpts);
type
boun = (left, right, bottom, top);code = array[boun]of Boolean;

Var i = int;
function encode (pt: wcpt2);
Var c: code;
        if pt.x < wmin.x then c[left]=true;
        else
                c[left]=false;
        if pt.x > wmax.x then c[right]=true;
        else
                c[right]= false;
        if pt.y < wmin.y then c[bottom] = true;
        else
                c[bottom]=false;
        if pt.y> wmax.y then
                c[top]=true;
        else
        c[top]= false;
}
function accept (c1, c2:code)
var k: boun;
{       accept=true ;
        for k=left to top
        {if c1[k] or c2[k] then accept = false;
        }
}
function reject(c1,c2:code)
var k: boun;
{       reject=false ;for k=left to top
        if (c1[k] and c2[k] ) then reject = true;

function ptinside(c: code)
        if (c[left] or c[right] or c[bottom] or c[top]) then
                ptinside = false;
        else  ptinside= true;

procedure swappts(var p1,p2: wcpt2)
var t: wcpt2;
  t=p1;p1=p2;p2=t; }
procedure swapcodes(c1,c2:code)
var t: code;
```

```
t=c1;c1=c2;c2=t; }
procedure Clipline( p1,p2: wcpt2)
var
c1, c2: code;done, draw: boolean; m:real;
        done =false; draw= false;
while not done
        c1=encode (p1); c2=encode (p2);
        if (accept (c1, c2)) then
        { done=true; draw = true;
        }
        else
        if (reject (c1, c2)) then done = true;
        else
        { if ptinside(c1) then
        {       swappts (p1, p2);
                swapcodes (c1, c2);
        }
        m = (p2.y - p1.y)/(p2.x - p1.x)
        if c1[left]  then
        {       p1.y = p1.y + (wmin.x - p1.x) * m;
                p1.x = w min.x;}
        else if c1[right] then
                {
                        p1.y = p1.y + (wmax.x - p1.x) * m;
                        p1.x = wmax.x;
                }
                else if c1[bottom] then
                {       p1.y = p1.y + (wmin.y - p1.y) / m;
                        p1.x = wmin.y;
                }
                else if c1[top] then
                {       p1.y = p1.y + (wmax.y - p1.y) / m;
                        p1.x = w max.y;
                }
                }  //else
                }//while not
if draw then  lineDDA (round(p1.x),round(p1.y),round(p2.x),round(p2.y));
begin         /*Cohen*/
      for i=1 to n-1
                {       clipline (pts[i],pts[i+1]);
                        clipline (pts[n], pts[1]);
                }
end.
```