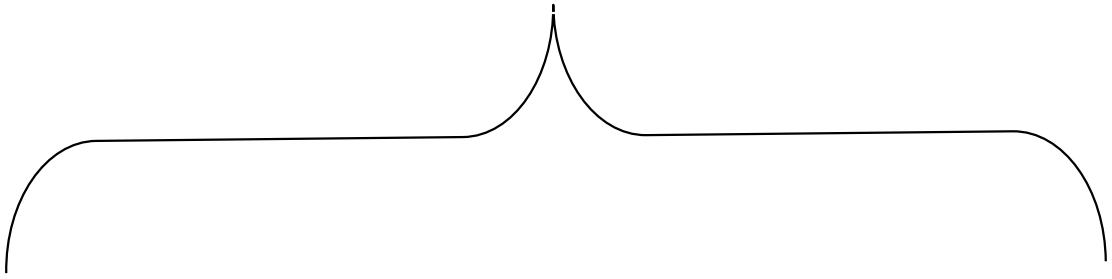
Introduction of Computer Graphics

Mode النمط



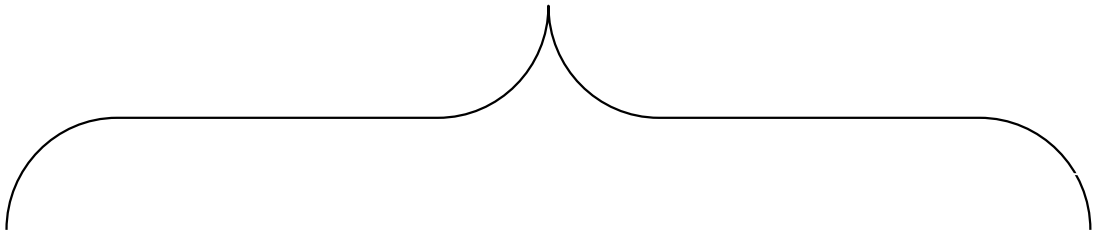
Text mode

نمط النصوص

Graphics mode

نمط الرسوم

Computer Graphics



Non-interactive

غير متفاعل

Interactive

متفاعل

- **في حالة النصوص : التعامل مع الحروف والارقام والرموز .**
- **في حالة الرسوم : التعامل مع النقاط (Pixel) وهي مختصر عناصر الصورة Picture element**
- **كل حرف يظهر على الشاشة في text mode يتمثل في الذاكرة بـ Byte 2 .**
- **حيث ان الـ Byte – 2 مقسمة الى**

البايت الثاني: صفة الحرف	البايت الاول : التمثيل الداخلي
A attribute	**للحرف ASCII**

توفر الشاشة نمطين :

1. النمط الاول (**text mode**) نمط النصوص :– يتم تمثيل الحروف في موقع ثابت في الذاكرة وكل حرف يظهر ضمن هذه النمط يتمثل في الذاكرة بـ **2 - Byte** . البايت الاول يمثل التمثيل الداخلي للحرف **ASCII** اما البايت الثاني فيمثل صفة الحرف **Attribute** .
2. النمط الثاني :– (**Graph mode**) نمط الرسوم :– يتعامل مع الـ (**Pixel**) وهي كلمة مختصرة الى عنصر الصورة **Picture element** وكل (**Pixel**) يتمثل داخل الذاكرة في موقع يختلف عن الـ **text mode** وهذا الموقع هو (**AooooH**) .
3. يحجز كل **Pixel** عدد من الـ **Bits** حسب عدد الوان الشاشة (نوع الشاشة) .

أ اذا كانت الشاشة غير ملونة يمكن تمثيل الـ **Pixel** بـ **bit-1** وتاخذ القيم { **1-on** ، **0-off** }

ب اذا كانت الشاشة تحتوي على **4** الوان يمكن تمثيل الـ **Pixel** بـ **bit-2** وتاخذ القيم { **00** ، **01** ، **10**، **11**}

ج اذا كانت الشاشة تحتوي على **8** الوان يمكن تمثيل الـ **Pixel** بـ **bit- 3** وتاخذ القيم { **000** ، **--** ، **111**}

يمكن توليد الالوان من ثلاثة الوان رئيسية هي [**1-Red 2-Green 3-Blue**] {**RGB**}

Computer graphics: the generation, representation, manipulation, processing, or evaluation of graphic images by a computer.

- Requires more work that using text mode.
- You must develop methods for drawing lines and characters.
- Single characters is made up of many pixel arranged in pattern that forms the character.
- You can light pixel any when your display.

Graphics types we will consider are:

1.1.1. Binary Graph

Binary graph are the simplest type of graphs and can take on two values, typically black and white, or ‘0’ and ‘1’. A binary graph is referred to as a 1 bit/pixel graph because it takes only 1 binary digit to represent each pixel.

These types of graphs are most frequently in computer vision application where the only information required for the task is general shapes, or outlines information.

For example, to position a robotics gripper to grasp an object or in optical character recognition (OCR). Binary graphs are often created from gray-scale graphs via a threshold value is, those values above it are turned white (‘1’), and those below it are turned black (‘0’).

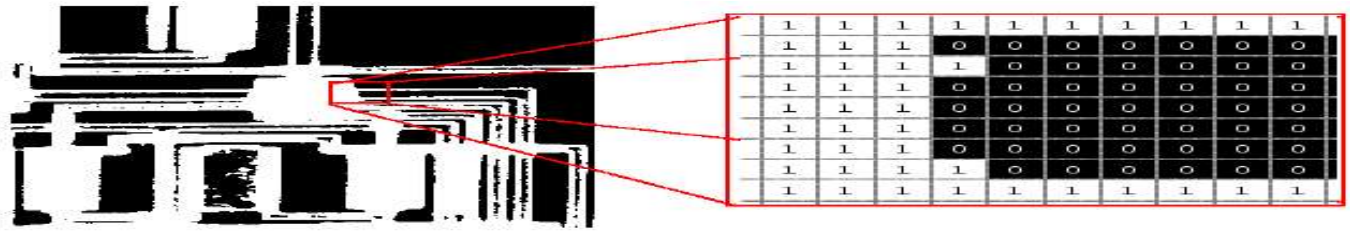


Figure (1.1): Binary Graphs.

1.2.2. Gray-scale graphs.

Gray-scale graph is a range of monochromatic shades from black to white. Therefore, a grayscale graph contains only shades of gray (brightness information only) and no color information.

Gray-scale graph table			
No. of pixe l	Binary value 0 = off 1 = on	Range Value	Gray color
1		0-1	Black/White
2	00/01/10/11	0-3	B/DarkB/WhiteB/ White
3	000/001/010/011/100/101/110/111	0-7	B/DarkB/---/ WhiteB/White
4	0000/0001------/1111	0-15	B/DarkB/---/ WhiteB/White
. .			
8	00000000/00000001--/11111111	0-255	B/DarkB/---/ WhiteB/White

The number of different brightness level available. (0) value refers to black color, (255) value refers to white color, and all intermediate values are different shades of gray varying from black to white. The typical graph contains 8 bit/ pixel (data, which allows us to have (0-255) different brightness (gray) levels. The 8 bit

representation is typically due to the fact that the byte, which corresponds to 8-bit of data, is the standard small unit in the world of digital computer.

Figure (1.2) : Gray Scale Graph

1.2.3. Color graph

Color graph can be modeled as three band monochrome graph data, where each band of the data corresponds to a different color. The actual information stored in the digital graph data is brightness information in each spectral band. When the graph is displayed, the corresponding brightness information is displayed on the screen by picture elements that emit light energy corresponding to that particular color. Typical color graphs are represented as red, green, and blue or RGB graphs. Using the 8-bit monochrome standard as a model, the corresponding color graph would have 24 bit/pixel – 8 bit for each color bands (red, green and blue). The following figure we see a representation of a typical RGB color graph.

IR(r,c) IG(r,c) IB(r,c)

The following figure illustrate that in addition to referring to arrow or column as a vector, we can refer to a single pixel red ,green, and blue values as a color pixel vector –(R,G,B).

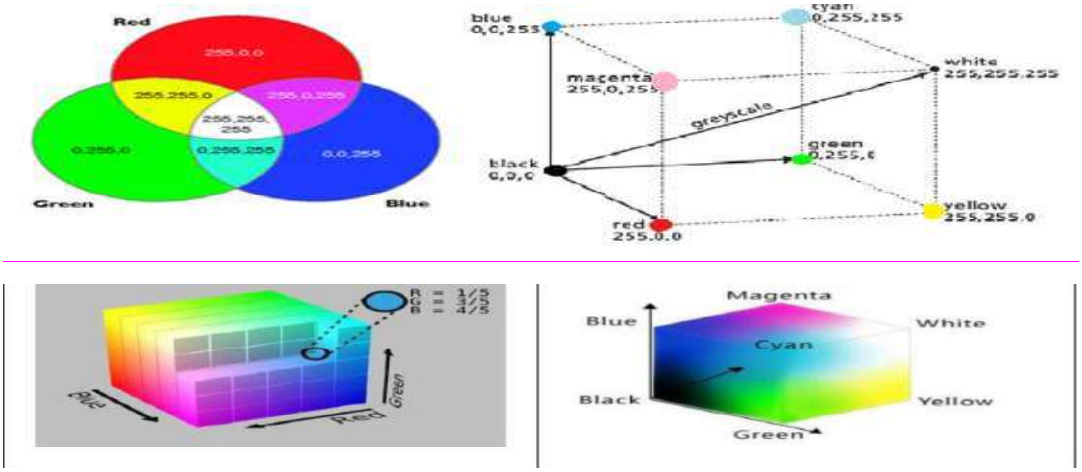


Figure (1.3): a color pixel vector consists of the red, green and blue pixel values (R, G, B) at one given row/column pixel coordinate (r,c).

the RGB color model used in color CRT monitor, in this model ,Red,Grren and Blue are added together to get the resultant color White.

Red	Green	Blue	Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
1	0	0	Red
0	1	1	Cyan
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

For many applications, RGB color information is transformed into mathematical space that decouples the brightness information from the color information.

The lightness is the brightness of the color, and the hue is what we normally think of as “color” and the hue (ex: green, blue, red, and orange). The saturation is a measure of how much white is in the color (ex: Pink is red with more white, so it is less saturated than a pure red). Most people relate to this method for describing color.

Example: “a deep, bright orange” would have a large intensity (“bright”), a hue of “orange”, and a high value of saturation (“deep”).we can picture this color in our minds, but if we defined this color in terms of its RGB components, R=245, G=110 and B=20, most people have no idea how this color appears. Modeling the color information creates a more people oriented way of describing the colors.

1.4.4 . Multispectral graphs

Multispectral graphs typically contain information outside the normal human perceptual range. This may include infrared (تحت الحمراء),ultraviolet (فوق البنفسجية), X-ray, acoustic or radar data. These are not graphs because the information represented is not directly visible by the human system. Source of these types of graph include satellite systems, underwater sonar systems and medical diagnostics imaging systems.

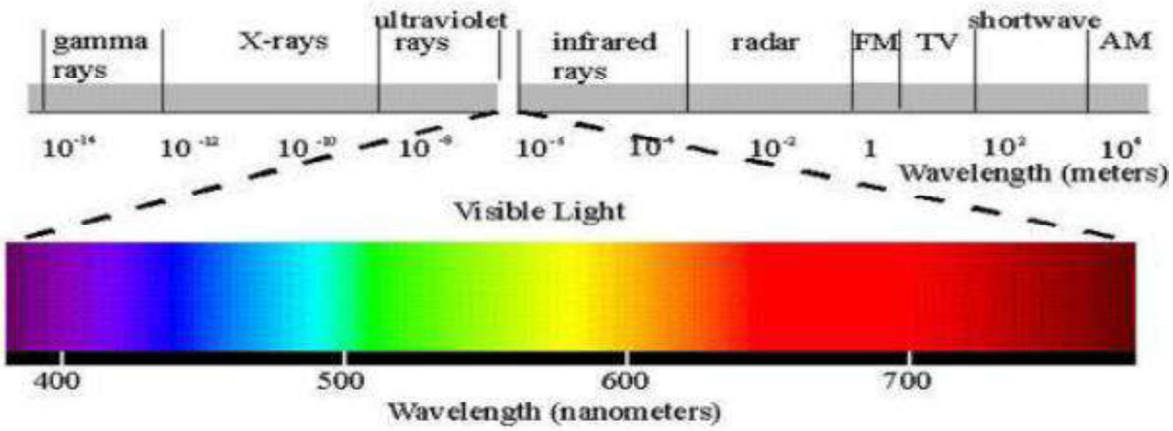
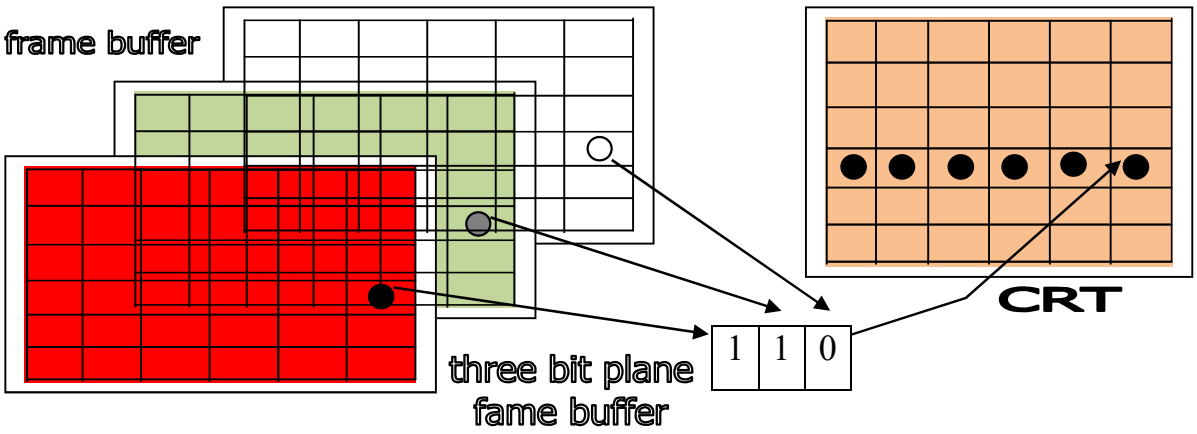


Figure (1.4) Electromagnetic spectrum.

1.5. B: Frame buffer

Each screen pixel corresponds to a particular entry in a two-dimension array residing in memory. This memory is called frame buffer or bit map. Frame buffer accessible to the central processing unit (CPU) of the main computer. This allowing repaid update of the stored image. The number of rows on the frame buffer array equal the number of raster lines on a display screen. The number of columns in this array equals the number of pixels on each raster line.

The term pixel is also used to describe the row and column location in the frame buffer array that corresponds to the screen location. A size 512*512 display screen required (262144) pixels memory location.

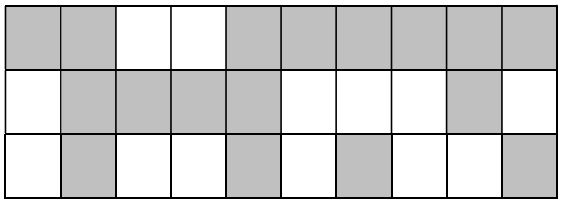


If we wish to display a pixel on the screen a specification values is placed into the corresponding memory location in the frame buffer array.

Each pixel in the frame buffer array is composed at several bits a single bit of place frame buffer to display a color or black and white quality image with shades of gray additional bit places are needed.

0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	1	1	1	0	1
1	0	1	1	0	1	0	1	1	0

➔



Frame Buffer **display screen**

Each screen location pixel and corresponding memory location in the frame buffer is accessed by (x, y). Integer coordinates pair. The (x) value refers to the column the (y) value refers to the row position.

1.3. Coordinates system

- Graphics screen consists of pixels ordered in horizontal and vertical lines.
- The sizes of axes are differing.
- CGA has low-resolution pixels that are large 320 horizontally and 200 vertically.
- VGA has high-resolution pixels so small 640 vertically and 480 horizontally. But SVGA resolution pixels are (1024 × 768).
- The smaller pixel the more pixels per average and the higher quality of the graphics display.

- Each adapter has one or more graphic images.
- You must know what graphics adopt is installed and which made is active.
- To light the pixel at the right corner of the screen you must know what the screen coordinate is.

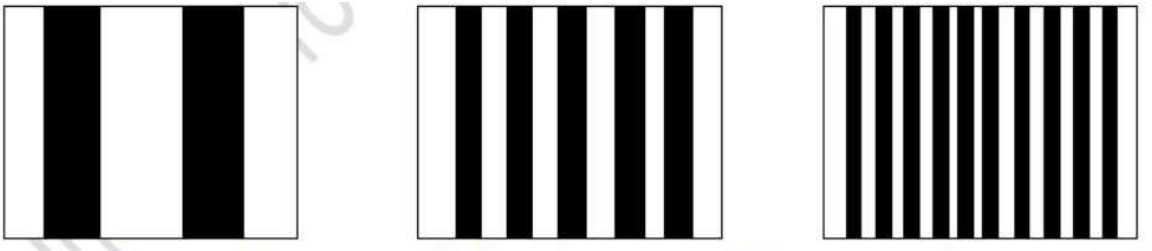
Note: **LCD**: Liquid Crystal Display, **LED**: Light-Emitting Diode, **HD**: HIGH Definition Pixels are the building blocks of every digital image. Clearly defined squares of light and color data are stacked up next to one another both horizontally and vertically

The resolution has to do with ability to separate two adjacent pixels as being separate, and then we can say that we can resolve the two. The concept of resolution is closely tied to the concepts of spatial frequency. There are two types of resolution:

1- Vertical resolution: the number of M rows in the image (image scan line).

2- Horizontal resolution: the number of N columns in the image.

Spatial frequency resolution: It is represent by the multiplication (M x N) and its closely tied to the concept of spatial frequency that refers to how rapidly the signal is changing in space, and the signal has two values for brightness 0 and maximum. If we use this signal for one line (row) of an image and then repeat the line down the entire image, we get an image of vertical stripes. If we increase this frequency the strips get closer and closer together, until they finally blend together as shown in figure below, note that the higher the resolution the more details (high frequency**).**



(a) Low frequency (f=2) (b) Median frequency (f=5) (c) High frequency (f=10)

.Figure (1.5) Resolution and Spatial frequency

In computers, resolution is the number of pixels (individual points of color) contained on a display monitor, expressed in terms of the number of pixels on the horizontal axis and the number on the vertical axis.

The sharpness of the image on a display depends on the resolution and the size of the monitor. The same pixel resolution will be sharper on a smaller monitor and gradually lose sharpness on larger monitors because the same numbers of pixels are being spread out over a larger number of inches. Display resolution is not measured in dots per inch as it usually is with printers (We measure resolution in pixels per inch or more commonly, Dots Per Inch (dpi)).

In computers, resolution is the number of pixels (individual points of color) contained on a display monitor, expressed in terms of the number of pixels on the horizontal axis and the number on the vertical axis .

The sharpness of the image on a display depends on the resolution and the size of the monitor .

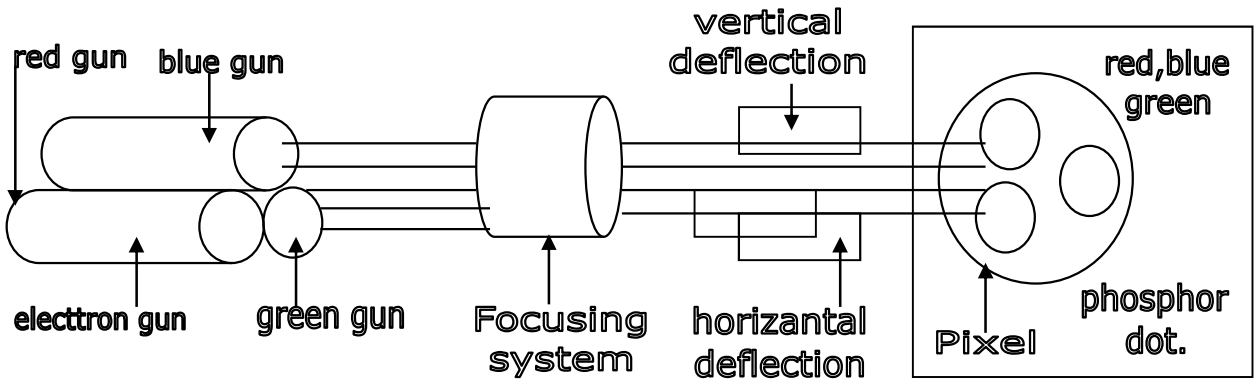
Display resolution is not measured in dots per inch as it usually is with printers (We measure resolution in pixels per inch or more commonly, dots per inch (**dpi)**) .

- A display with 240 pixel columns and 320 pixel rows would generally be said to have a resolution of 240*320 .

- Resolution can also be used to refer to the total number of pixels in a digital camera image. For example, a camera that can create images of 1600*1200 pixels will sometimes be referred to as a 2 megapixel resolution camera since 1600 * 1200 = 1,920,000 pixels, or roughly 2 million pixels. Where a megapixel (that is, a million pixels) is a unit of image sensing capacity in a digital camera. In general, the more megapixels in a camera, the better the resolution when printing an image in a given size.

1.4: Interactive computer graphics: observe has some control over the image, it involves communication between the computer and user, and the displayed picture is modified appropriately to signals, the computer receives from the input device e.g. (keyboard, mouse, joystick, etc.). It appears that picture is changing instantaneously in response to the observers' commands.

How the cathode ray tube (CRT) works:



The CRT consists of three electron guns: red, green, and blue. They emit a beam of negatively charged electrons towards a positively charged phosphor-coated screen. Along the way, the electron beam passes through the focusing system that concentrates the beam so that by the time the electrons reach the screen they have converged to a small dot.

The beam then passes through the deflection system (horizontal and vertical) which deflects the beam to strike any point on the screen. When this focused electron beams strikes the screen the phosphor emits a dot of visible light.

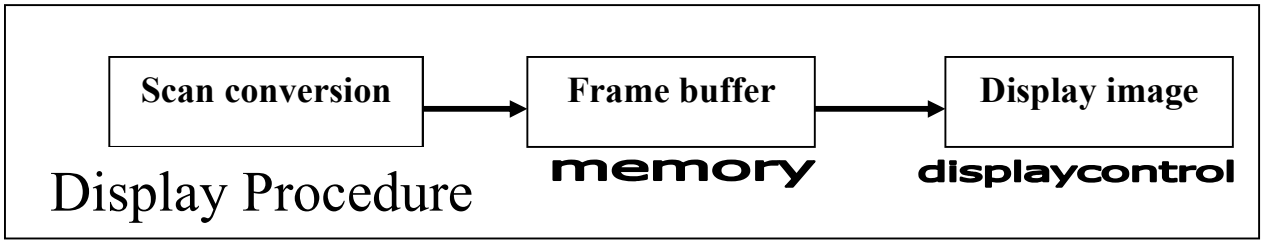
The video display is divided into very small dots called **"pixels"** (picture elements) each pixel is composed of a triangular pattern of red, green, and blue phosphor dot.

The CRT has three electron guns one for each of the three primary colors: red, green, and blue each electron gun hits its corresponding phosphor dot causing that particular color to appear on the screen the light on the display screen starts to fade as soon as the beam moves to another location.

So the beam must refresh the screen by illuminating pixels 30 to 60 times each second.

1.5: Raster – scan display

The video display in microcomputer is divided into very small rectangle or dots these dots are referred to as picture elements or pixels. We can consider the CRT screen to consist of grid of line made up of pixels the horizontal line made up of pixels the horizontal line are called raster-scan-line and video display is referred to as raster-scan-display.



Raster scan display

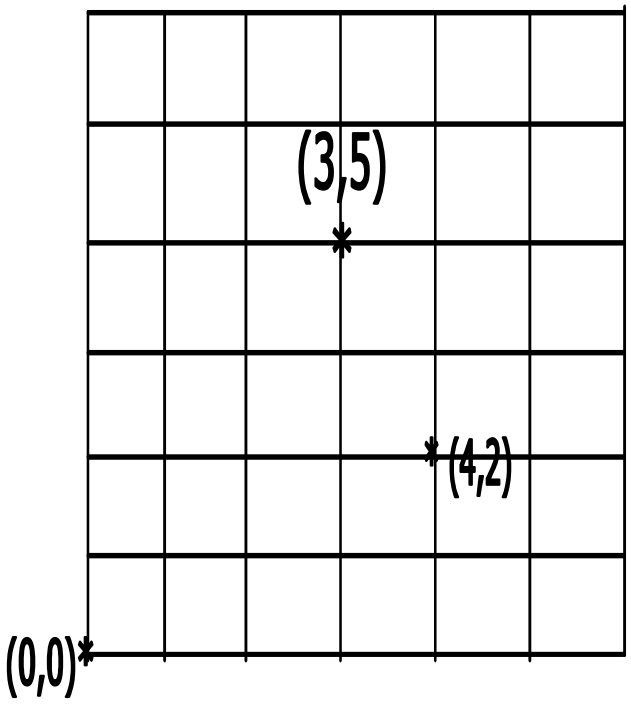
Raster scan display (n) = Scan Conversion (n) ➔Frame Buffer (n) ➔Display image (n)

1.5. A: Scan conversion

Image is usually defined in terms of equations for example x+ y=5 or graphic description such as "draw line from point to point" scan conversion is: the process of converting their abstract representation of an image into the appropriate pixel value in the frame buffer. An inexpensive microcomputer graphics system uses the CPU and library of software routines to perform scan conversion. "The process of representing continuous graphics objects as a collection of a discrete pixel is called scan conversion".

1.5. C: Plotting points

To draw a picture on a raster display, we must determine the corresponding points in the frame buffer that make up the picture. To perform this we must write a scan conversion point-plotting algorithm. Both frame buffer and display screen are given a two-dimensional coordinates system with the origin at the lower-left corner. Each pixel is described in no negative integer (x, y) coordinates pair the x value starts at the origin (0) and increases from left to right (there is no standard for the location of two origins).



1.6: Applications of computer graphics:

1. **CAD** (Computer Aided design): use of a computer to aid in the design of product.
2. **CAM** (Computer Aided Manufacturing): use of a computer to control the manufacturing of products.
3. **CAI** (Computer Aided Instructing): use of computer to display animated pictures to illustrate educational concept.
4. **CAE (Computer Aided Engineering)** use of a computer as engineer work .
5. **Simulation**: use of a computer to experience circumstance that otherwise would be too expensive or catastrophic to experience in reality E.g.: flight simulators, simulating unclear reactors.
6. **GUI** (Graphical User Interfaces): simplify the user of computer programs by giving them user friendly interfaces.
7. **Entertainment**: Computer games and movie making.
8. **Visualization:** The need for visualization today has increased dramatically and many advanced technologies will see the need for visualization. Data visualization helps us gain insights into the information to analyze and study the compatibility of the systems that we have around us.
9. **Image Processing:** Specific types of photos or photographs need to be edited to be used at various locations. One of the many techniques of computer graphics is to transform existing images into optimized ones to enhance their understanding.

Graphics Devices

1.Input Units

A) Key board . B) Mouse. C) Light pen .

D) Tach screen. E) etc…

2.Display Unit .

تقسم الاجهزة المظهرة للصورة حسب طبيعة الصورة الناتجة وهي :-

(a Monitor : - تكون الصورة فيه متغيرة (نقطة ، رمز ، خط)

(b Plotter:-تكون الصورة فيه ثابتة .

(c Printer: -تكون الصورة فيه على شكل حروف وارقام فقط .

(d صور تخطيط القلب تكون على شكل خطوط و نقاط فقط .

3. Frame buffer

. جزء من الذاكرة يخزن قيم النقاط التي تمثل الصورة .

FB :- هو جزء من الذاكرة الرئيسية ومصمم بتقنية **multiport memory** لان التعامل معها يتم عن طريق **CPU** الرئيسي و **DC** الخاص بالرسم.

Frame Buffer:- is the array which contains an internal representation of the image . it collects and stores pixel values for using by the device. It constructs (designs) as multiport memory technique.

FB ⟶ DC ⟶ Display Unit.

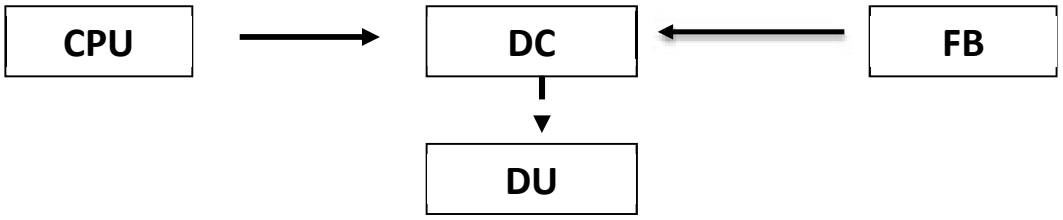
4.DC -: Display controller .

وهو CPU خاص بالتعامل مع FB لعرضها على الشاشة .

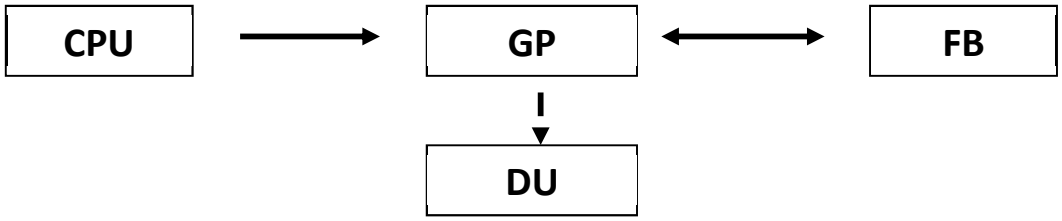
CPU هو الذي يقوم بتحديث ووضع المعلومات في FB ثم يقوم بقرائتها وعرضها على الشاشة .

أ – في الحاسبات القديمة مثل الوركاء وصخر تكون الـ CPU الرئيسية هو الذي يقوم بالرسم .

ب في الحاسبات الحديثة يكون DC مفصولاً عن الـ CPU .



ج – في السنوات التالية ظهرت الحاجة الى Graphic Processor وهو الذي يقوم بتغيير محتويات FB ويكون موقعه بين CPU ، FB .



ء – ثم ظهرت بعد ذلك Math Co Processor . (كارت الشاشة)

What are the methods of programming graphics? *اساليب البرمجة للرسوم :-*

1- استخدام لغة اعتيادية باضافة اوامر خاصة (Pascal ، ------)

2- استخدام لغة خاصة بالرسم LOGO ، HPGL.

3- التطبيقات المتخصصة بالرسم CAD .

4- انواع من البرمجيات تكون غير مختصة بالرسم ، ولكن نتائجها تكون على شكل رسوم مثل Lotus، Spss حيث تكون النتيجة رسم او جدول .

نختار التطبيق الافضل حسب :- (How we can choose the best application)

1. مرونة الاستخدام .

2. سهولة الاستخدام وسهولة التعلم .

3. تحقيق الهدف .

انواع اجهزة الرسم :-

1-نوع يعتمدالنقطة اساس الصورة .

2-نوع يعتمد الخط اساس الصورة .

3-نوع يعتمد الرمز اساس الصورة مثل IBM Golve Ball وقد اندثرت حالياً .

Object primitives (عناصر الصورة)

1- Point 2- Line 3- Arc (circle)

2- النقطة (**point or pixel**) ومميزاتها لها طول ، عرض ، لون ، وشدة اضاءة .

Pixel :- is the smallest addressable screen element ,
It is the smallest piece of the display screen which we can control.

2. Line: a line can be described as a single point that continues for a distance, or as the connection between two points (x1, y1) ,(x2, y2) .

The line is one of the main components of design including principles such as shape, color, texture, value, perspective, and form. Lines can appear in many different forms some examples may be straight, curved, continuous, dotted, thick, thin, real, and implied. A line can be used to create structure and tone in illustrations and other artworks.

To find distance of line calculate $dx=X_{end}-X_{start}$, $dy=Y_{end}-Y_{start}$ and slop $M=dy/dx$.

2.1 drawing lines

مواصفات الخط المستقيم :– اللون ، شدة الاضاءة ، نوعية الخط ، العرض .

نوعية الخط المستقيم :–

Requirements for an acceptable line drawing algorithm

1. The line should appear to be straight.(يجب ان يظهر الخط مستقيماً قدر الامكان)
2. The line should terminate accurately.(يجب ان يظهر وينتهي باحداثيات ثابتة)
3. The line should have constant density.(يجب ان تكون له كثافة ثابتة)
4. The line density should be independent of line length and angle.(يجب ان تكون الكثافة غير معتمدة على طول الخط او زاويته)
5. Line should be drawn rapidly.

2.1.1. Horizontal and vertical lines:

The simplest lines to drawn are horizontal and vertical lines, the screen coordinates of the point an **a horizontal line** *are obtained by keeping the value of y constant and repeatedly incrementing or decrementing the x value by one unit.* The following lines of code draw a horizontal line between the two points, (x1, y1), (x2, y2)

Input:(x1,y1),(x2,y2)

Output: Draw a **Horizontal line** Because (y2-y1)=0 ➔ { dy=0}

1: For x =x1 to x2 step sign(x2-x1)

2: Plot (x, y), color

3: Next x

Note: - sign (n) return either **1** {n>0} or **0** {n=0} or **-1** { n<0}

To draw a vertical line the x value is fixed and the y value varies.

Input:(x1,y1),(x2,y2)

Output: Draw a **vertical line** Because (x2-x1)=0 ➔ { dx=0}

1: For y =y1 to y2 step sign (y2-y1)

2: Plot (x, y), color

3: Next y

2.1.2. Diagonal lines:

To draw a **diagonal line with slop equal to 1** *we repeatedly ejective Change by same unit both the x and y values from the starting to the ending pixels.* **The slop** *is defined as the change in y value divided by change in x values*: ***m= Δy/ Δx = (y2-y1)/(x2-x1)***.

The following lines of code draw a diagonal line with slop equal to +1 or -1 only: ***x =x1; y =y1;While (x<=x2) { plot (x, y, color); x =x +sign(x2-x1); y=y +sign(y2-y1); }***

Input :(x1, y1), (x2, y2)

Output: Draw a slop line m= -1 or m=1

1: x =x1; y =y1

2: For x =x1 to x2 step sign(x2-x1) // x=x +sgn(x2-x1)

3: Plot (x, y), color

4: y=y+ sign(y2-y1) //sign(y2-y1)= either +1 or -1

5: next for // repeat in step 2 until x =x2

Note: ***the slop is zero*** *then* ***line is a horizontal dy(y2-y1)= (y1-y2)=0***

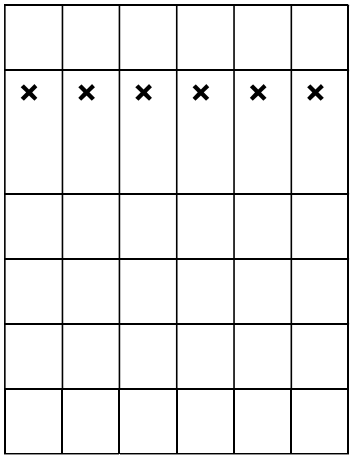
and if ***slop is undefined*** *then* ***line is a vertical dx(x2-x1)= (x1-x2)=0***

but if ***slop line is not equaled +1 or -1*** *then* found three methods to draw diagonal lines are:

A>Y=mX+b.

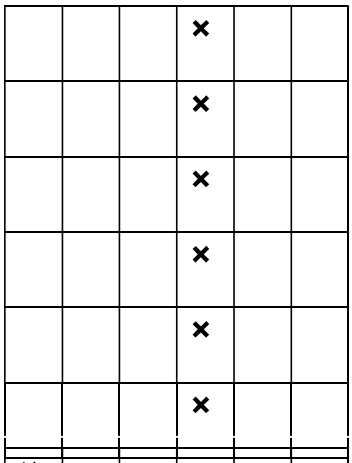
عندما نريد رسم خط افقي

لاتوجد مشكلة



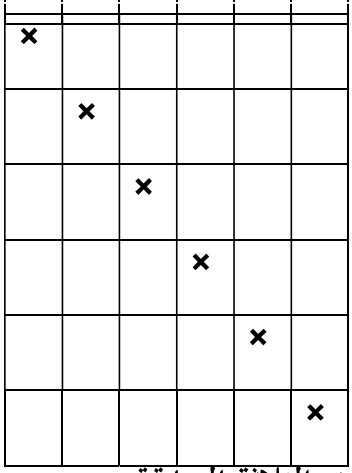
عندما نريد رسم خط عمودي

لاتوجد مشكلة



عندما نريد رسم خط قطري

لاتوجد مشكلة

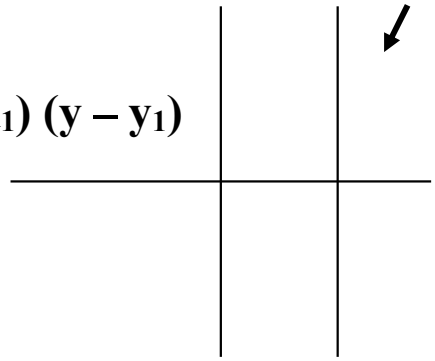


لكن المشكلة تظهر عندما نريد رسم خط بين نقطتين لاتنتميان للحالات الثلاثة السابقة .

Equation of line

$$\frac{y2-y1}{x2-x1} = \frac{y-y1}{x-x1} \quad \ldots\ldots (1)$$

$y_2 \neq y_1$



$$(y_2 - y_1)(X - X_1) = (X_2 - X_1)(y - y_1)$$

Vertical line

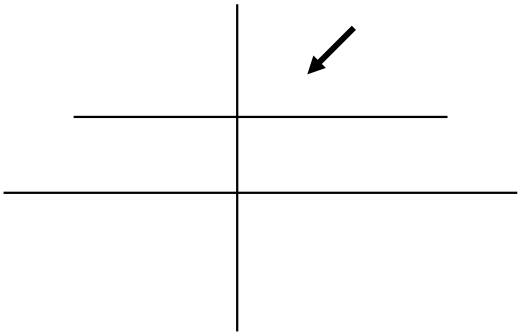
$y_2 \neq y_1$

$X = X_1$

$X_2 = X_1$

Horizontal Line

$X_1 \neq X_2$



$X_2 \neq X_1$

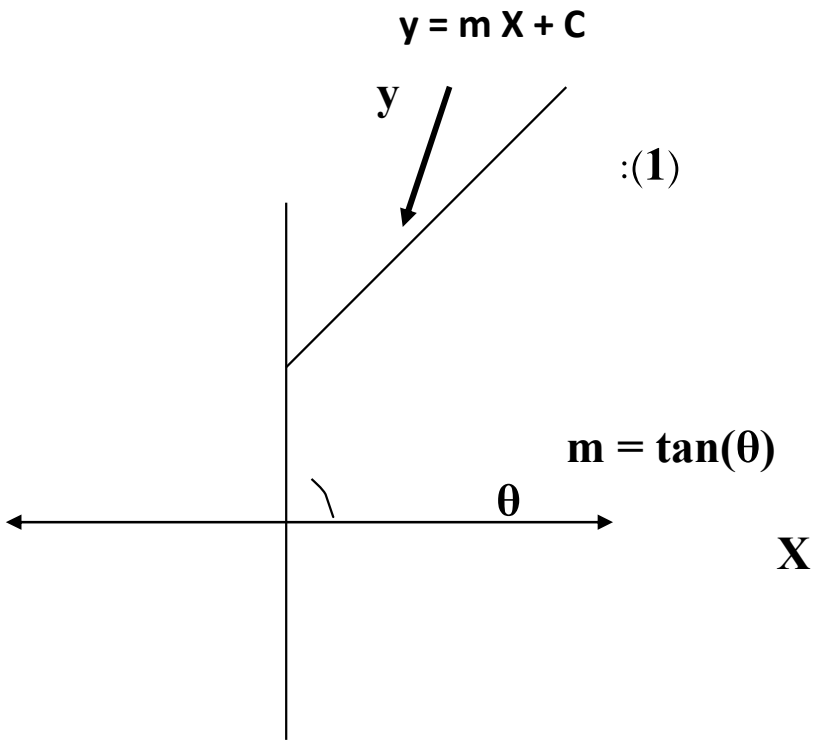
$y = y_1$

$y_2 = y_1$

$$m = \frac{y2-y1}{x2-x1}$$

بالتعويض في المعادلة رقم :(1)

$$y - y_1 = m\,(X - X_1)$$
$$y = m\,(X - X_1) + y_1$$
$$= m\,X + (y_1 - m\,X_1)$$
$$= m\,X + C$$



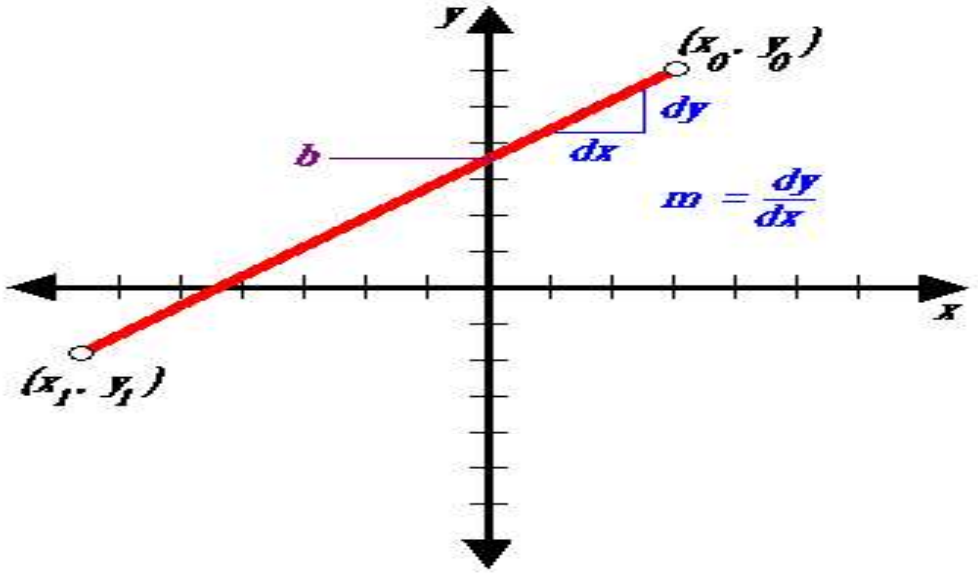
Using line equation 'Y=mX+b':

this method is advantaged the line equation to finds point where they belongs in line. The constant **b** is found by if it assigns line point in this equation then you find the value of **b**. For example if the endpoints of line is (x1, y1) (x2, y2) then the ***b = y1-mx1*** or ***b=y2-mx2***. The following algorithm is used this method below:

1: dx=x2-x1: dy=y2-y1
2:if (dx=0) that lead draw the *vertical line* & exit
3:if (dy=0) that lead draw the *horizontal line* & exit
4: m=dy/dx : b=y1-mx1 { OR b=y2-mx2}
5: for x=x1 to x2 step sgn(dx) {X+ =1 or X- =1}
5.1: y=m*x+b
5.2 plot point (x, y), color

6: next x {goto 5 where un-finish}

Simple Line



Based on *slope-intercept algorithm* from algebra*:*

y = mx + b

b=y-mx

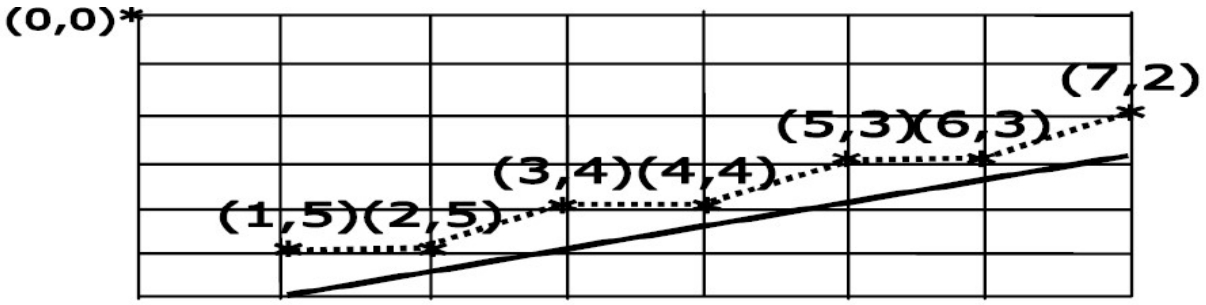
Simple approach:

increment x, solve for y

Floating point arithmetic required

Example: trace the line equation to draw the line that endpoints are (1, 5), (7, 2), and draw the line in screen coordinate.
dx=7-1=**6**; **dy**=2-5=**-3**; **m**= -3/6= -1/2 =**-0.5**; **b**=5- (-0.5)*1=**5.5**

X	Y	Point (x, y)	Plot in screen
1	1*-0.5+5.5	(1,5)	(1,5)
2	2*-0.5+5.5	(2,4.5)	(2,5)
3	3*-0.5+5.5	(3,4)	(3,4)
4	4*-0.5+5.5	(4,3.5)	(4,4)
5	5*-0.5+5.5	(5,3)	(5,3)
6	6*-0.5+5.5	(6,2.5)	(6,3)
7	7*-0.5+5.5	(7,2)	(7,2)



خوارزمية المسح التفاضليDigital

Digital Differential Analyzer

خوارزمية المسح التفاضلي

Digital DifferentialDigital Differential Analyzer

تعتبر خوارزمية المسح التفاضلي

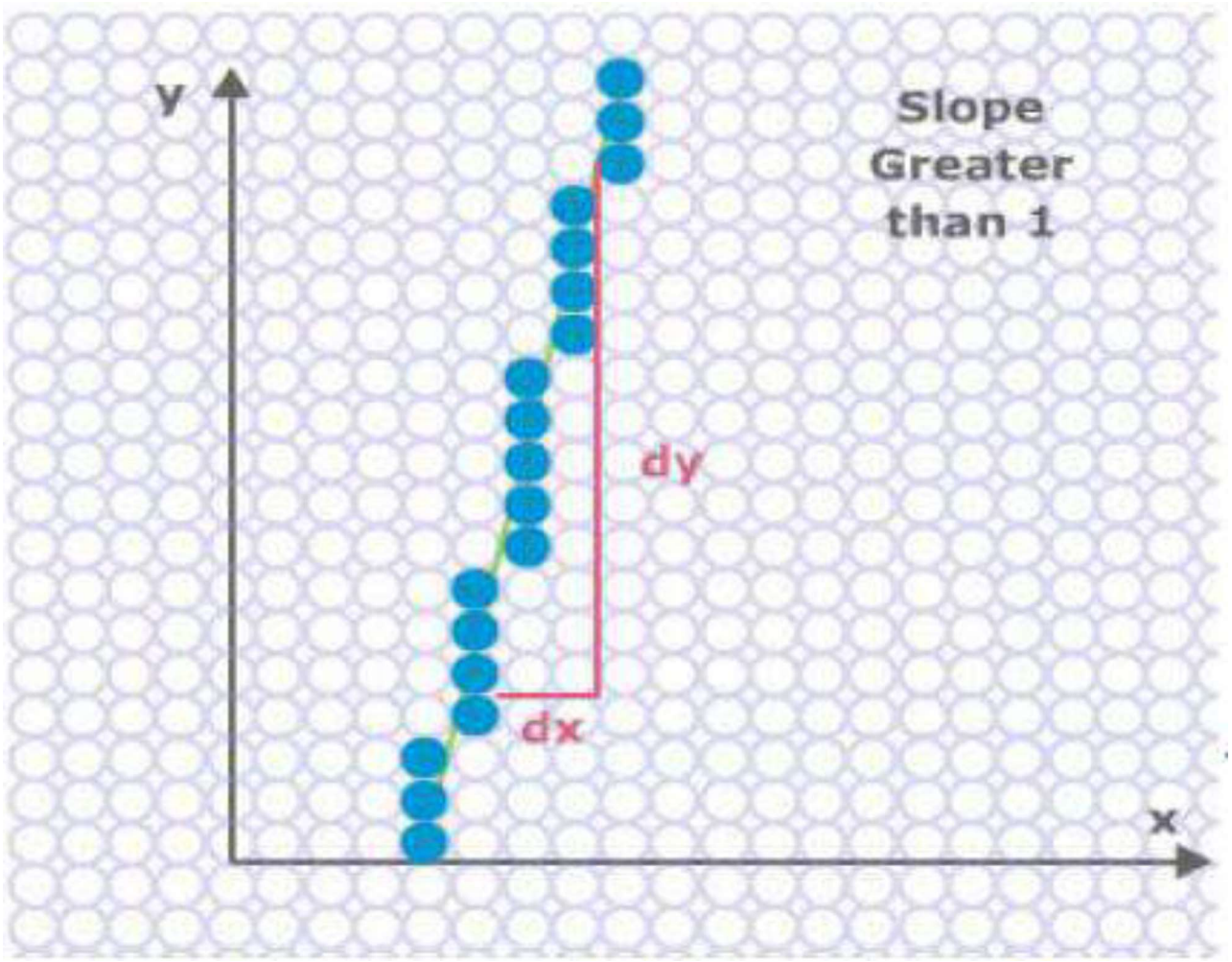
D igital Differential Analyzer

Digital Differential Analyzer والتي تعرف اختصارًا ب
DDA

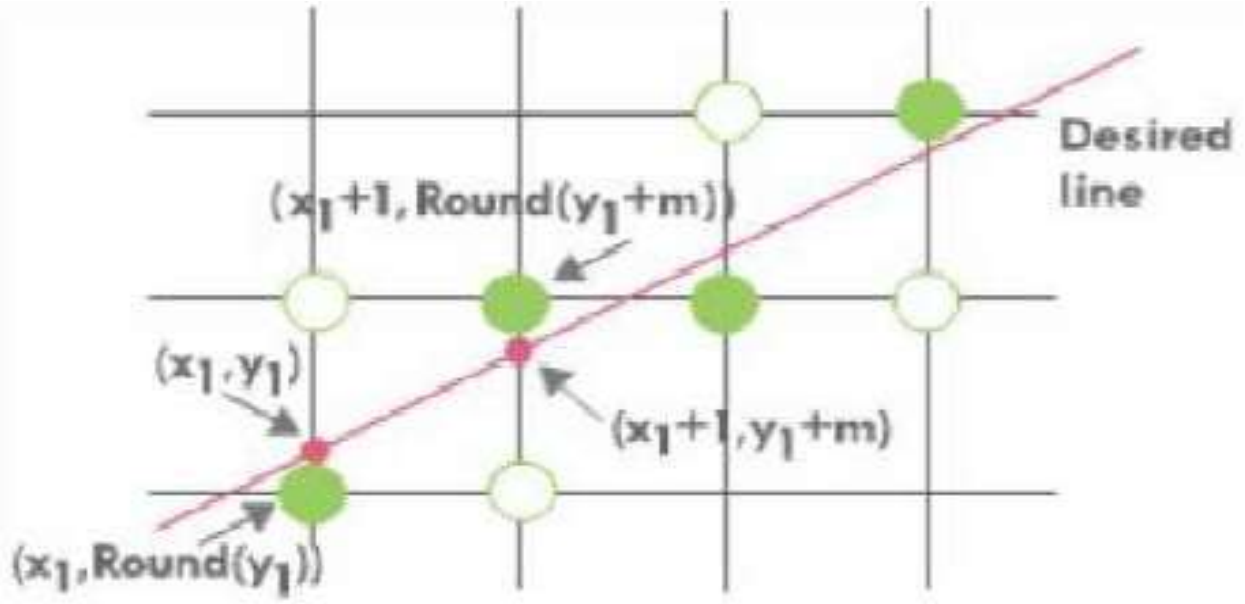
Digital Differential Analyzer
DDA

خوارزمية مسح متزايدة

incremental alg incremental algorithmm بمعنى
أنها تهدف إلى إلغاء عملية الضرب وتحويلها إلى عملية جمع.



DDA



تعتبر خوارزمية **DDA** أسرع من خوارزمية **slope-intercept** نتيجة إلغاء عملية الضرب. لكنها ما زالت تعاني من تزايد الأخطاء التراكمية نتيجة عمليات التدوير، أضف إلى أن العمليات الحساب تتم بالفاصلة العائمة.

Digital Differential Analyzer (DDA)

(Xb, Yb) (Xe , Ye)

Begin:-

If ABS (Xe –Xb) ≥ ABS (Ye – Yb)
Then Length = ABS (Xe – Xb) .
Else Length = ABS (Ye – Yb).
End if
Dx = (Xe – Xb) / length
Dy = (Ye – Yb) / length
X = Xb + 0.5 × sign (Dx)
Y = Yb + 0.5 × sign (Dy)
For I = 1 To Length
Plot (integer (X) , integer (y))
X = X + DX
Y = Y + DY
Next I
END.

$$\text{Sign (X)} \begin{cases} -1 & x < 0 \\ 0 & x = 0 \end{cases}$$

Ex: 1- Consider the line from (0,0) to (3,6) .

$X_B = 0$ $X_E = 3$
$Y_B = 0$ $Y_E = 6$

Length = 6
D x = 0.5
D y = 1
X = 0.5
Y =0.5

Ex: 1- Consider the line from (0,0) to (3,6) .

$X_B = 0$, $X_E = 3$

$Y_B = 0$, $Y_E = 6$

Length = 6 , D x = 0.5 D y = 1 X = 0.5 Y =0.5

I	X	Y	PLOT (X,Y)
1	0.5	0.5	(0,0)
2	1.0	1.5	(1,1)
3	1.5	2.5	(1,2)
4	2.0	3.5	(2,3)
5	2.5	4.5	(2,4)
6	3.0	5.5	(3,5)

Ex: 2- Consider the line from (0,0) to (10,-10) .

$X_B = 0$, $X_E = 10$

$Y_B = 0$, $Y_E = -10$

Length = 10 , D x = 1 , $D_Y = -1$ $X = 0.5$, $Y = -0.5$

I	X	Y	PLOT (X,Y)
1	0.5	- 0.5	(0,0)
2	1.0	- 1.5	(1,-1)
3	2.5	- 2.5	(2,-2)
…			
…..			
……			
10			(9,-9)

سلبيات خورازمية DDA

- تتعامل مع الاعداد الحقيقية والتعامل مع العدد الحقيقي فيه سلبيات من حيث الدقة والسرعة "

- التعامل مع العدد الحقيقي يأخذ ٢٠ مرة بقدر التعامل مع الصحيح بالنسبة الى وقت CPU

- الزيادة تكون مرة في X ومرة في Y وبعبارة اخرى هناك زيادة في المحور السيني والمحور الصادي

3.4 B the simple DDA

The simple *Digital Differential Analyzer* is a line drawing algorithm that generates line from their differential equations. It work on the principle of
simultaneously incrementing x and y by small steps proportional to the first derivatives of x and y. The slop a line between the two points (x1,y,) , (x2,y2) is given by m=(y2-y)/(x2-x1).

The algorithm starts with the initial values x=x1 & y=y1, the coordinates
are then incremented by Δy *and* Δx respectively to find the next points coordinates. The value of x and y are rounded to integers and a pixel is set at that point. This step is repeated until the second end point (x2, y2) is reached.

algorithm

1: dx=x2-x1:dy=y2-y1

2: if (abs(dx)>abs(dy)) then length=abs(dx)
else length=abs(dy)

3: xinc=dx/length: yinc=dy/length

4: x=x1: y=y1

5: for i=0 to length
5.1: plot pixel(x, y)
5.2: x=x+xinc : y=y+yinc
6:next i

Ex/: **trace the simple DDA algorithm to draw a line between the two points (23,33), (29,40).**

Sol/: dx=29-23=6
dy=40-33=7
Length=7
Xinc=6/7 =0.857
Yinc=7/7=1.

X	Y	Plot(X)	Plot(y)
23	33	23	33
23.857	34	24	34
24.714	35	25	35
25.571	36	26	36
26.429	37	26	37
27.286	38	27	38
28.143	39	28	39

Example: Consider the line from (0, 0) to (6, 6), use DDA to rasterize the line. Let us perform some initial calculations

Initially X1 = 0 , Y1 = 0 , X2 = 6 , Y2 = 6

So our first step would be to evaluate the length
Length = abs(x2 – x1) = 6 – 0 = 6
Length = abs(y2 – y1) = 6 – 0 = 6
So length = 6 (in any X or Y direction)
Had (x2 – x1) been larger then X direction would have been chosen otherwise Y direction.

So our next step would be to evaluate Δx and Δy as in our algorithm

$\Delta x = x_2 - x_1 / \text{length} = 6 - 0 / 6 = 1$

$\Delta y = y_2 - y_1 / \text{length} = 6 - 0 / 6 = 1$

$x = x_1 + 0.5 * \text{sign}(\Delta x)$

$= 0 + 0.5 * \text{sign}(1)$ /* **since sign(1) returns 1*/**

$x = 0.5$

Similarly, $y = 0.5$

Evaluating through main loop gives us

i	*plot*	*x*	*y*
1		0.5	0.5
2	(0, 0)	1.5	1.5
3	(1, 1)	2.5	2.5
4	(2, 2)	3.5	3.5
5	(3, 3)	4.5	4.5
6	(4, 4)	5.5	5.5
7	(5, 5)	6.5	6.5
8	(6, 6)	7.5	7.5

So we can see from the above table how DDA method evaluates and plot new points (0.5 is taken as 0 in above table because we are considering floor value).

Ex/ **find the point which are plotted for the line with the tow end point?**

(x1,y1)=(2,1)
(x2,y2)=(7,3)

Dx=x2-x1=7-2=5
Dy=y2-y1=3-1=2

M=dy/dx=2/5=0.4
dx>dy
xi=5/5=1
yi=2/5=0.4

i	x	y	Plot(x,y)
	2	1	2,1
	3	1.4	3,1
	4	1.8	4,2
	5	2.2	5,2
	6	2.6	6,3
	7	3	7,3

H.W1/ trace the simple DDA algorithm to draw a line between the two point (29, 40), (23, 33).

H.W2/ trace the simple DDA algorithm to draw a line between the two point (0, 0), (6, 6).

H.W3/ trace the simple DDA algorithm to draw a line between the two point (3, 8), (10, 6).

Bresenham's line drawing algorithm

- This algorithm is designed so that each iteration changes one of the
- coordinate values by ±1. The other coordinate may or may not change
- depending on the value of an error term maintained by the algorithm. This
- error term record the distance measure perpendicular to the axis of
- greatest movement, between the exact path of the line and the actual dots
- generated.

Bresenham's line drawing algorithm

- ***If the x-axis is the axis of greatest movement, then at each iteration the***
- ***x coordinates of the line is incremented, and the slop of the line by***
- ***dy/dx is added to the error term. Whether to increment the y coordinate***
- ***of the current point. A positive e value indicates that the exact path of***
- ***the line lies above the current point, therefore they coordinate is***
- ***incremented, and 1 is decremented from e.***

Bresenham's line drawing algorithm

- ***If e is negative the y coordinate value is left unchanged.***
- **And verse vise if y-axis is greatest movement**
- ***Impartment note:-If X-axis is greatest movement the initial e =dy/dx***
- **But *If Y-axis is greatest movement the initial e =dx/dy***

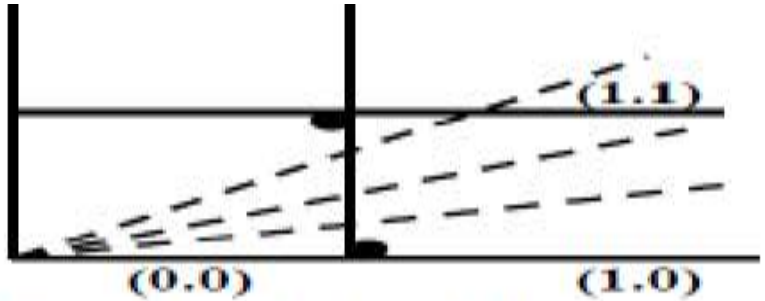
this algorithm is X-axis greatest movement and slop is positive

- ***1:dx=x2-x1: dy=y2-y1: e=(dy/dx)-0.5: x=x1: y=y1;***
- ***2:For i=0 to |dx|***
- ***2.1: Plot pixel (x, y,)***
- ***2.2: If (e>0) then if (y1>y2) y=y-1***
- ***Else y=y+1***
- ***e=e-1***
- ***2.3: if(x1>x2) then x=x-1***
- ***else x=x+1***
- ***2.4: e=e+ (dy/dx)***
- ***3:next i***

3.Bresenham's algorithm

- Developed by Bresenham J.E.
- Designed so that each iteration changes one of the coordinates values by ± 1.
- The other coordinate may or may not change depending on the value of an error term maintained by the algorithm.
- The error term records the distance measured perpendicular to the axis of greatest movement between the exact path of the line and the actual dots generated.

Bresenhem's algorithm



$X_E >= X_B$

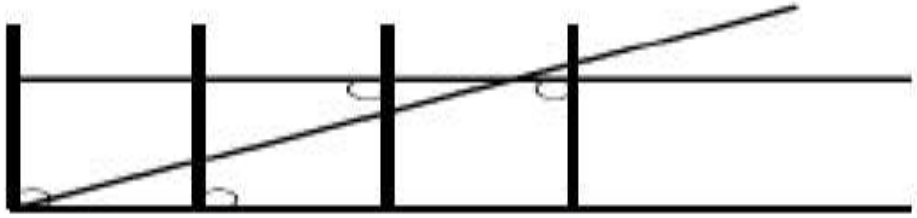
$y_E >= y_B$

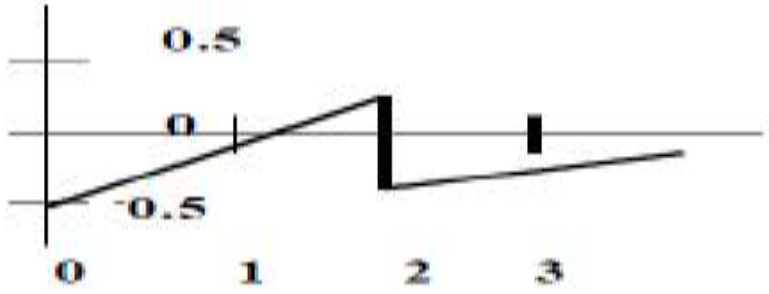
$0 \leq (\Delta y / \Delta X) < \frac{1}{2} \leq$ plot (1.0)

$\frac{1}{2} \leq (\Delta y / \Delta X) \leq 1$ plot (1.1)

initialize error to -0.5

$e = e + (\Delta y / \Delta X)$





this algorithm is X-axis greatest movement and slop is positive

- **Ex :3- Consider line from**
- **(0,0) to (8,2) .**
- **x1 = 0, y1= 0 ,x2 = 8, y2= 2 .**
- **Dx = 8 Dy = 2**
- **m=DY/DX=2/8 =0.25**
- **e=m-0.5**
- **e = 0.25 – 0.5 = -0.25**

i	x	y	e	Plot(x,y)
1	0	0	-0.25 +m	0,0
2	1	0	0 -1+m	1,0
3	2	1	-0.75 +m	2,1
4	3	1	-0.5 +m	3,1
5	4	1	-0.25 +m	4,1
6	5	1	0 -1+m	5,1
7	6	2	-0.75 +m	6,2
8	7	2	-0.5 +m	7,2

this algorithm is X-axis greatest movement and slop is positive

- **EX/Consider line from (1,2) to (7,4) .**

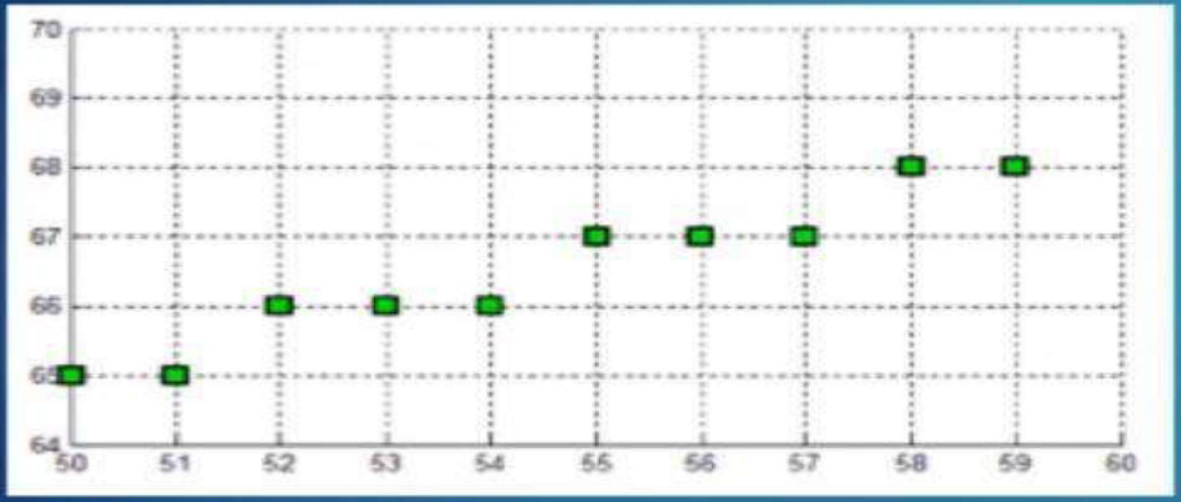
- **x1 = 1, y1= 2 ,x2 = 7, y2= 4 .**
- **Dx = 6, Dy = 2**
- **m=DY/DX=2/6 =0.333**
- **e=m-0.5**
- **e = 0.333 – 0.5 = -0.166**

i	x	y	e	Plot(x,y)
1	1	2	-0.166	1,2
2	2	2	0.167	2,2
3	3	3	-0.5	3,3
4	4	3	-0.167	4,3
5	5	3	0.166	5,3
6	6	4	-0.501	6,4

Ex. / trace the line where the end points (50, 65), (59, 68) by Bresenham's algorith

Sol. / dx= 59-50= 9 ; dy= 68-65= 3 ; m= dy/dx =3/9 = 0.333

e=0.333 – 0.5 = -0.167



i	*X*	*Y*	*e*	
0	*50*	*65*	*-0.167*	*+m*
1	*51*	*65*	*0.166*	*-1+m*
2	*52*	*66*	*-0.501*	*+m*
3	*53*	*66*	*-0.168*	*+m*
4	*54*	*66*	*0.165*	*-1+m*
5	*55*	*67*	*-0.502*	*+m*
6	*56*	*67*	*-0.169*	*+m*
7	*57*	*67*	*0.164*	*-1+m*
8	*58*	*68*	*-0.503*	*+m*
9	*59*	*68*	*-0.17*	

Note: this algorithm uses in **slop of line positive** but in **negative slop** replaces *e>0 to* ***e<0***, and *e=e-1 to* ***e=e+1*** and coordinate y=y+1 to y=y-1 Or x=x+1 to x=x-1 to obtain algorithms:

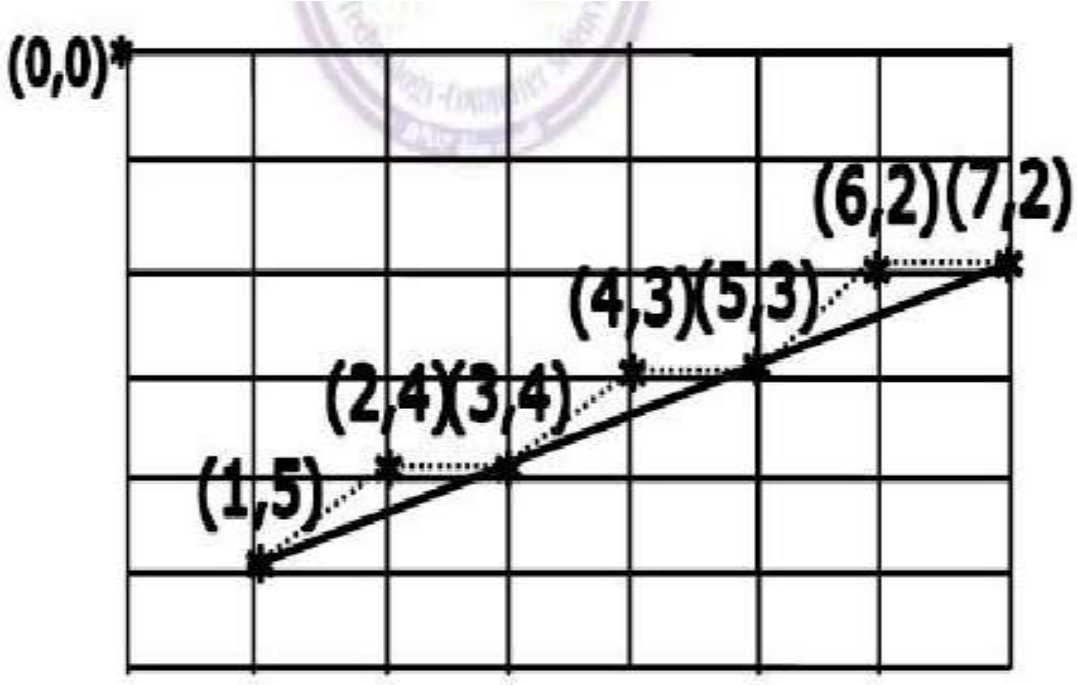
- ***1:dx=x2-x1: dy=y2-y1: e=(dy/dx)+0.5: x=x1: y=y1;***
- ***2:For i=0 to |dx|***
- ***2.1: Plot pixel (x, y,)***
- ***2.2: If (e<0) then if (y1>y2) y=y-1***
- ***Else y=y+1***
- ***e=e+1***
- ***2.3: if(x1>x2) then x=x-1***
- ***else x=x+1***
- ***2.4: e=e+ (dy/dx)***
- ***3:next i***

slop of line in negative slop

- Ex. / trace the Bresenhams algorithm to draw a line between the two points
- (1, 5), (7, 2).
- Sol. / dx= 7-1= 6;
- dy= 2-5= -3
- m=dy/dx=-3/6= -0.5
- *{Negative slop} uses algorithm*
- *modify of Bresenhams*
- e=-0.5 + 0.5 = 0
- **Note:- try e=-0.5**

slop of line in negative slop

I	X	Y	E	PLOT (X,Y)
1	1	5	0	1,5
2	2	5	-0.5	2,5
3	3	4	0	3,4
4	4	4	-0.5	4,4
5	5	3	0	5,3
6	6	3	-0.5	6,3
7	7	2	0	7,2



Bresenham's line drawing algorithm

- This algorithm is designed so that each iteration changes one of the
- coordinate values by ±1. The other coordinate may or may not change
- depending on the value of an error term maintained by the algorithm. This
- error term record the distance measure perpendicular to the axis of
- greatest movement, between the exact path of the line and the actual dots
- generated.

Bresenham's line drawing algorithm

- ***If the x-axis is the axis of greatest movement, then at each iteration the***
- ***x coordinates of the line is incremented, and the slop of the line by***
- ***dy/dx is added to the error term. Whether to increment the y coordinate***
- ***of the current point. A positive e value indicates that the exact path of***
- ***the line lies above the current point, therefore they coordinate is***
- ***incremented, and 1 is decremented from e.***

Bresenham's line drawing algorithm

- ***If e is negative the y coordinate value is left unchanged.***
- **And verse vise if y-axis is greatest movement**
- ***Impartment note:-If X-axis is greatest movement the initial e =dy/dx***
- **But *If Y-axis is greatest movement the initial e =dx/dy***

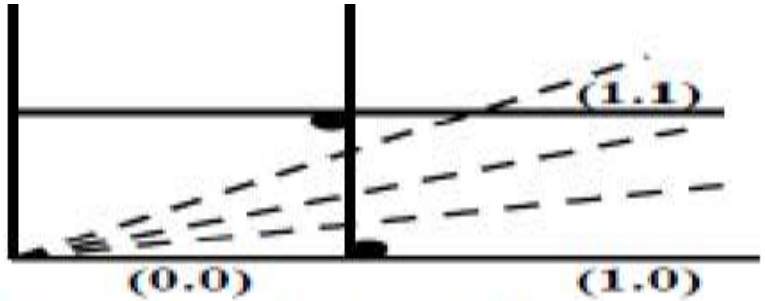
this algorithm is X-axis greatest movement and slop is positive

- ***1:dx=x2-x1: dy=y2-y1: e=(dy/dx)-0.5: x=x1: y=y1;***
- ***2:For i=0 to |dx|***
- ***2.1: Plot pixel (x, y,)***
- ***2.2: If (e>0) then if (y1>y2) y=y-1***
- ***Else y=y+1***
- ***e=e-1***
- ***2.3: if(x1>x2) then x=x-1***
- ***else x=x+1***
- ***2.4: e=e+ (dy/dx)***
- ***3:next i***

3.Bresenham's algorithm

- Developed by Bresenham J.E.
- Designed so that each iteration changes one of the coordinates values by ± 1.
- The other coordinate may or may not change depending on the value of an error term maintained by the algorithm.
- The error term records the distance measured perpendicular to the axis of greatest movement between the exact path of the line and the actual dots generated.

Bresenhem's algorithm



$X_E >= X_B$

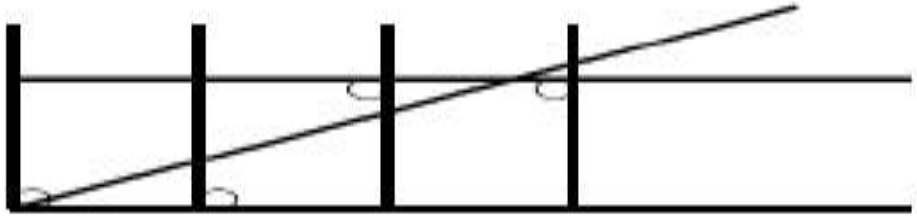
$y_E >= y_B$

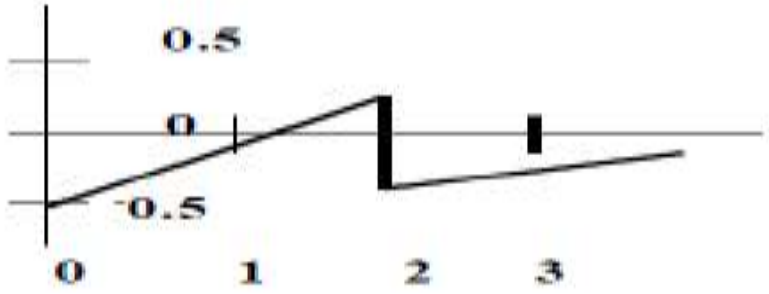
$0 \leq (\Delta y / \Delta X) < \frac{1}{2} \leq$ plot (1.0)

$\frac{1}{2} \leq (\Delta y / \Delta X) \leq 1$ plot (1.1)

initialize error to -0.5

$e = e + (\Delta y / \Delta X)$





this algorithm is X-axis greatest movement and slop is positive

- **Ex :3- Consider line from**
- **(0,0) to (8,2) .**
- **x1 = 0, y1= 0 ,x2 = 8, y2= 2 .**
- **Dx = 8 Dy = 2**
- **m=DY/DX=2/8 =0.25**
- **e=m-0.5**
- **e = 0.25 – 0.5 = -0.25**

i	x	y	e	Plot(x,y)
1	0	0	-0.25 +m	0,0
2	1	0	0 -1+m	1,0
3	2	1	-0.75 +m	2,1
4	3	1	-0.5 +m	3,1
5	4	1	-0.25 +m	4,1
6	5	1	0 -1+m	5,1
7	6	2	-0.75 +m	6,2
8	7	2	-0.5 +m	7,2

this algorithm is X-axis greatest movement and slop is positive

- **EX/Consider line from (1,2) to (7,4) .**

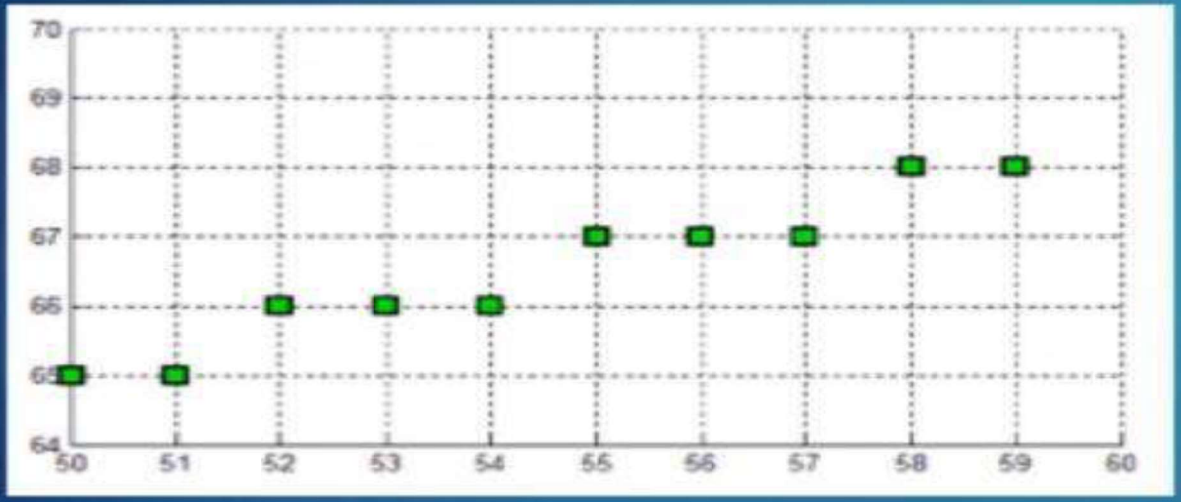
- **x1 = 1, y1= 2 ,x2 = 7, y2= 4 .**
- **Dx = 6, Dy = 2**
- **m=DY/DX=2/6 =0.333**
- **e=m-0.5**
- **e = 0.333 – 0.5 = -0.166**

i	x	y	e	Plot(x,y)
1	1	2	-0.166	1,2
2	2	2	0.167	2,2
3	3	3	-0.5	3,3
4	4	3	-0.167	4,3
5	5	3	0.166	5,3
6	6	4	-0.501	6,4

Ex. / trace the line where the end points (50, 65), (59, 68) by Bresenham's algorith

Sol. / dx= 59-50= 9 ; dy= 68-65= 3 ; m= dy/dx =3/9 = 0.333

e=0.333 – 0.5 = -0.167



i	*X*	*Y*	*e*	
0	*50*	*65*	*-0.167*	*+m*
1	*51*	*65*	*0.166*	*-1+m*
2	*52*	*66*	*-0.501*	*+m*
3	*53*	*66*	*-0.168*	*+m*
4	*54*	*66*	*0.165*	*-1+m*
5	*55*	*67*	*-0.502*	*+m*
6	*56*	*67*	*-0.169*	*+m*
7	*57*	*67*	*0.164*	*-1+m*
8	*58*	*68*	*-0.503*	*+m*
9	*59*	*68*	*-0.17*	

Note: this algorithm uses in **slop of line positive** but in **negative slop** replaces *e>0 to* ***e<0***, and *e=e-1 to* ***e=e+1*** and coordinate y=y+1 to y=y-1 Or x=x+1 to x=x-1 to obtain algorithms:

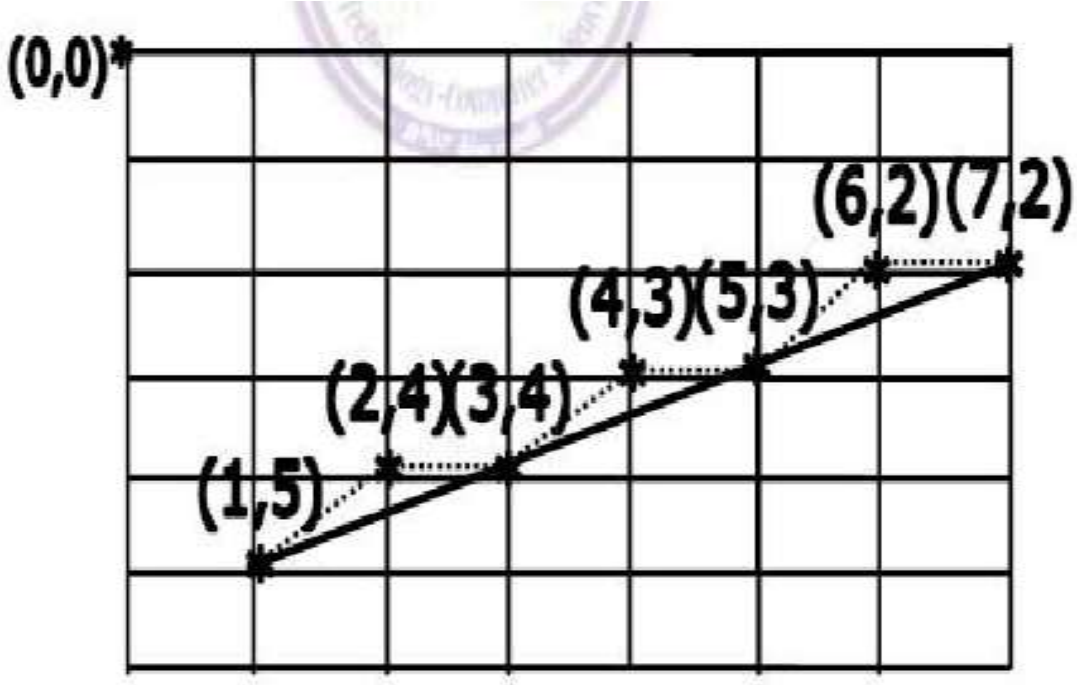
- ***1:dx=x2-x1: dy=y2-y1: e=(dy/dx)+0.5: x=x1: y=y1;***
- ***2:For i=0 to |dx|***
- ***2.1: Plot pixel (x, y,)***
- ***2.2: If (e<0) then if (y1>y2) y=y-1***
- ***Else y=y+1***
- ***e=e+1***
- ***2.3: if(x1>x2) then x=x-1***
- ***else x=x+1***
- ***2.4: e=e+ (dy/dx)***
- ***3:next i***

slop of line in negative slop

- Ex. / trace the Bresenhams algorithm to draw a line between the two points
- (1, 5), (7, 2).
- Sol. / dx= 7-1= 6;
- dy= 2-5= -3
- m=dy/dx=-3/6= -0.5
- *{Negative slop} uses algorithm*
- *modify of Bresenhams*
- e=-0.5 + 0.5 = 0
- **Note:- try e=-0.5**

slop of line in negative slop

I	X	Y	E	PLOT (X,Y)
1	1	5	0	1,5
2	2	5	-0.5	2,5
3	3	4	0	3,4
4	4	4	-0.5	4,4
5	5	3	0	5,3
6	6	3	-0.5	6,3
7	7	2	0	7,2



Integer Bresenham algorithm

Bresenham's algorithm as presented above requires the use of :

(1) Floating point arithmetic. (2) Division.

Bresenham algorithm requires the use of floating point arithmetic and division to calculate the slop of the line and to evaluate the error term. The speed of the algorithm can be increased by using integer arithmetic and eliminating the division.

Since only the sign of the error term is important, the simple Transformation

تتطلب خوارزمية بريسنهام استخدام حساب النقطة العائمة والقسمة لحساب ميل الخط وتقييم الخطأ بشرط. يمكن زيادة سرعة الخوارزمية باستخدام عدد صحيحا لحساب والقضاء على القسمة.

$$E = E * 2 * \Delta X$$

of the error term in the previous algorithm yields an integer algorithm. This allows the algorithm to be efficiently implemented in hardware..

من مصطلح الخطأ في الخوارزمية السابقة ينتج عددا صحيحا للخوارزمية. وهذا يسمح بتنفيذ الخوارزمية بكفاءة

Integer Bresenhem's Algorithm (for first Octant) .

So : $E=\{ (DY / DX) - 0.5 \} * 2 \Delta X \Rightarrow E= 2 \Delta Y - \Delta X$

$E= \{ E-1 \} * 2 \Delta X \Rightarrow E= E - 2 \Delta X$

$E= \{ E + (DY / DX) \} * 2 \Delta X \Rightarrow E = E+ 2 \Delta Y$

Integer Bresenhem's Algorithm

هي تطوير الخوارزمية الاولى بالغاء عملية القسمة بايجاد

E وذلك بضرب قيمة E * 2DX

Bresenham's integer algorithm for the first octant { $0 \leq \Delta Y \leq \Delta X$ }
i.e. slop between zero and one

```
X = XB
Y = YB
Dx = XE –XB
Dy = YE – YB
     E = 2*Dy – Dx
          For i = 1 to Dx
          Plot (x, y )
           While (E ≥ 0 )
           y = y +1
           E = E-2 *Dx
            end while
                        X = x +1
                        E = E+ 2 *Dy
                       Next i .
                       End .
```

Ex : line from (0,0) to (8,2) ?

x1 = 0, y1= 0 , x2 = 8, y2= 2;

Dx= X2-X1= 8-0 = 8;

Dy = Y2-Y1=2-0 =2 ;

M=dy/dx=2/8=0.25

E = 2*Dy – Dx;

E = 4 – 8 = -4;

```
For i = 1 to Dx
        Plot (x, y )
         While (E ≥ 0 )
         y = y +1
         E = E-2 *Dx
          end while
                    X = x +1
                    E = E+ 2 *Dy
                   Next i .
```

I	E	X	Y	PLOT	Y	E	X	E
1	-4	0	0	0,0			1	0
2	0	1	0	1,0	1	-16	2	-12
3	-12	2	1	2,1			3	-8
4	-8	3	1	3,1			4	-4
5	-4	4	1	4,1			5	0
6	0	5	1	5,1	2	-16	6	-12
7	-12	6	2	6,2			7	-8
8	-8	7	2	7,2			8	-4

Ex : line from (1,1) to (8,5) ?

Sol/

x1 = 1, y1= 1 , x2 = 8, y2= 5;

Dx= X2-X1= 8-1= 7;

Dy = Y2-Y1=5-1=4 ;

M=dy/dx =4/7= 0.571

E = 2*Dy – Dx; ايجاد القيمة الابتدائية للقرار

E = 2* 4-7=1;

I	X	Y	E	PLOT(X,Y)
1	1	1	1	1,1
2	2	2	-5	2,2
3	3	2	3	3,2
4	4	3	-3	4,3
5	5	3	5	5,3
6	6	4	-1	6,4
7	7	4	7	7,4
8	**8**	**5**	**1**	**8,5**

for i=1to dx

e=e-2*dx if e>=0

e=e+2*dy

General Bresenham's algorithm

- A full implementation of Bresenham algorithm requires
- modification for lying in the other octanats. These can easily be
- developed by considering quadrant in which the line lies and its
- slope. When the absolute magnitude of the slop of the line is > 1 ,
- Y is incremented by one end Bresenham error is used to determine
- when to increment X.
- Whether X or Y incremented by +(-) 1 depends on the quadrant.

- تعديل الاستلقاء في الأوكتانات الأخرى. هذه يمكن أن Bresenham يتطلب التنفيذ الكامل لخوارزمية تكون بسهولة تم تطويره من خلال النظر في الربع الذي يقع فيه الخط وميل. عندما يكون الحجم المطلق في نهاية واحدة متى تزيد Bresenhamعن طريق استخدام خطأ Y لميل الخط أكبر من ١يتم زيادة بمقدار +(-) ١ على الربع. Y أو X يعتمد زيادة X

General Bresenham's algorithm

- X = XB
- Y = YB
- Dx =abs (XE – XB)
- Dy = abs (YE – YB)
- S1 = Sign (XE – XB)
- S2 = Sign (YE – YB)
- if (Dy > Dx)
- Then Dy ⟷ Dx (Interchange)

توضيح (Temp=dx ,dx=dy,dy=temp)

- Interchange = 1
- Else
- Interchange = 0
- end it

```
E= 2 *Dy – Dx
For i = 1 to Dx
Plot ( x, y )
   While (E >= 0 )
         if   Interchange = 1
then x = x + S1
Else y = y + S2
         end if
E = E - 2 *Dx
  end while
          if Interchange = 1
Then y = y + S2
Else x = x + S1
             end if
E = E + 2 *Dy
next i
End .
```

Ex : Consider the line from (3,6) to (2,2) using General Bresenham algorithm.

- **x = 3 ,y= 6 ,Dx = 1 Dy = 4** **, Dx = 4 ; Dy = 1** **E = 2*Dy – Dx;**
- **S1 = -1 ; S 2 = -1**
- **Interchange = 1**
- **E = -2**
- **for I = 1 to 4**
- **plot(x,y)**
- **while (E>= 0)**
- **x = x –1**
- **E = E – 8**
- **End while**
- **y = y - 1**
- **E = E + 2**
- **next I**
- **end.**

I	E	X	Y	PLOT(X,Y)	X	E	Y	E
1	-2	3	6	3,6			5	0
2	0	3	5	3,5	2	-8	4	-6
3	-6	2	4	2,4			3	-4
4	-4	2	3	2,3			2	-2

H.W

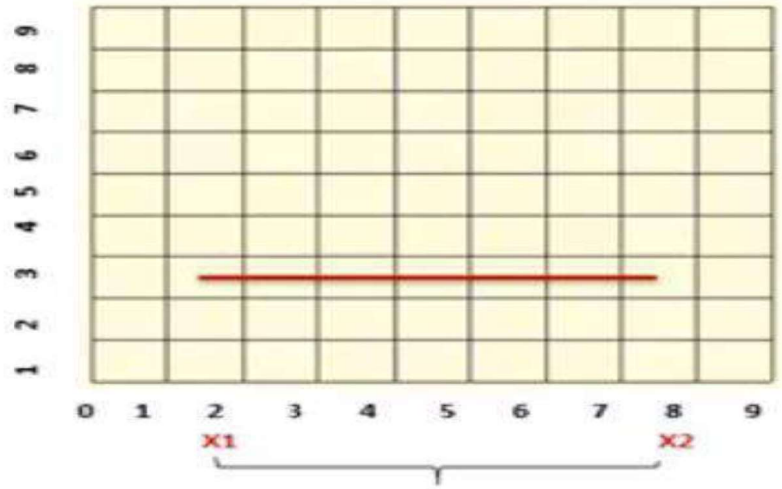
- Draw the line from (0,0) to (-8,-4) using General Bresenham algorithm
- Draw the line from (,) to (,) using General Bresenham algorithm

slope

Slope

Slope measures the steepness of a line. $m = \frac{\Delta y}{\Delta x}$. There are four types of slope (zero, undefined, positive (m<1, m=1, m>1), negative).

1- Slope of Zero



$$m = \frac{\Delta y}{\Delta x} = \frac{0}{6} = 0$$

slope

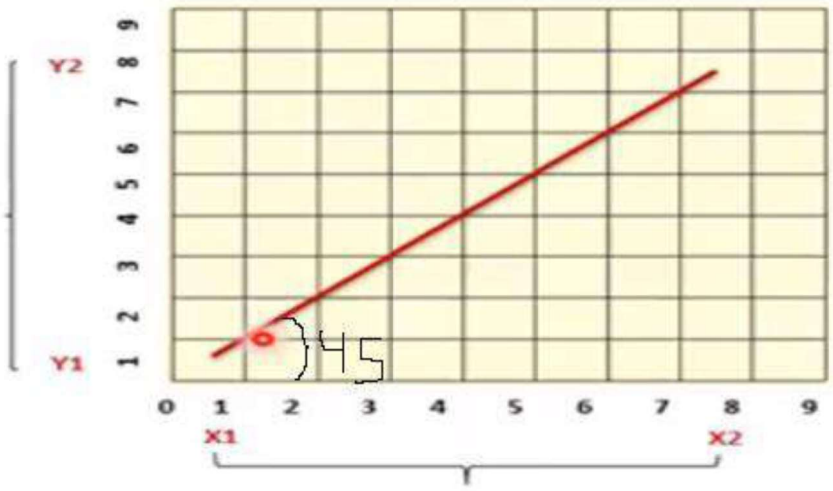
Slope

2- Undefined Slope

$$m = \frac{\Delta y}{\Delta x} = \frac{6}{0} = \text{undefined}$$

Slope

3- Positive slope (m<1, m=1, m>1)



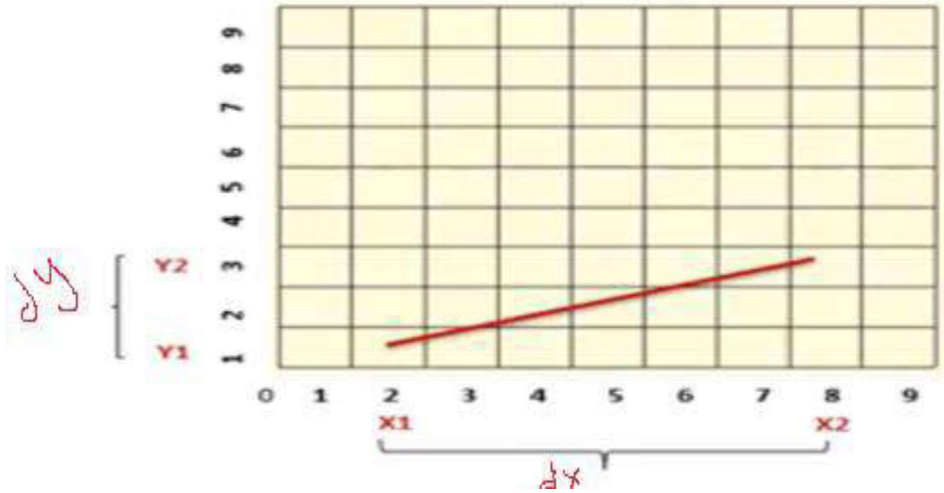
$$\Delta X = \Delta Y, \quad m = 1$$

$$m = \frac{\Delta y}{\Delta x} = \frac{7}{7} = 1$$

Slope

3- Positive slope (m<1, m=1, m>1)

A positive slope means that two variables are positively related that is, when x increases, so does y, and when x decreases, y decreases also.



$\Delta X > \Delta Y, \; m < 1$

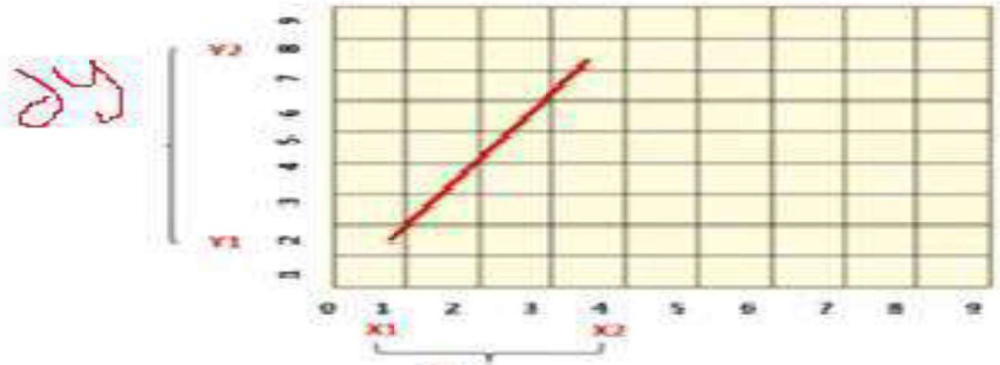
$m = \frac{\Delta y}{\Delta x} = \frac{2}{6} = 0.3$

$\because dx > dy \qquad \therefore x = x+1$

slope

Slope

3- Positive slope (m<1, m=1, m>1)

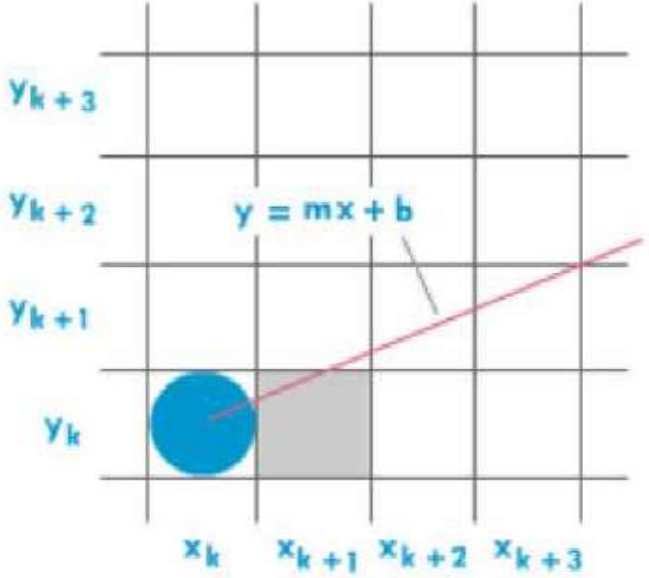


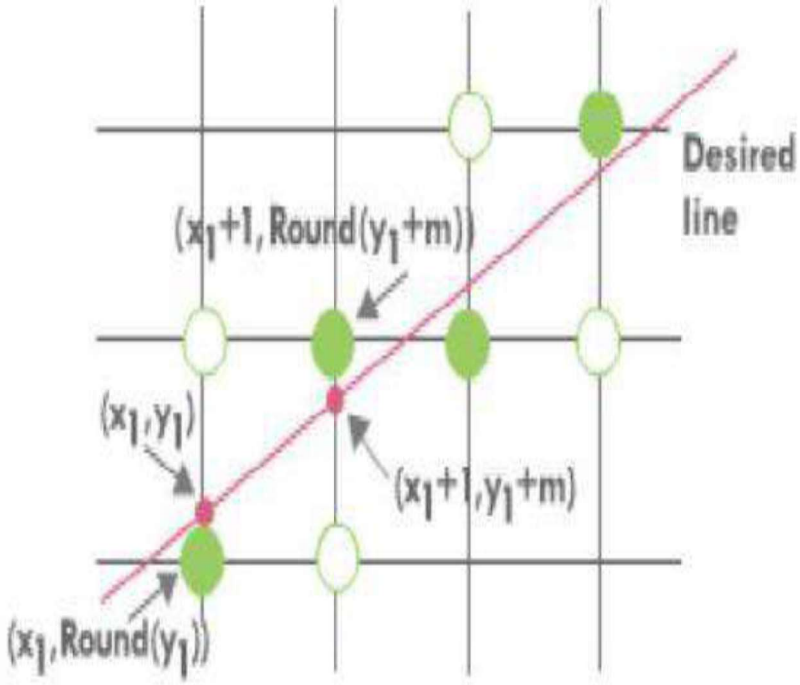
$\Delta X < \Delta Y, \quad m > 1$

$m = \frac{\Delta y}{\Delta x} = \frac{6}{3} = 2$

line

فمثلا عند إظهار النقطة (x_k , y_k) ، نريد تحديد البكسل الجديد المراد إظهاره من إجل x_{k+1} حيث $x_{k+1} = x_k + 1$ من الواضح إننا أمام خيارين وحيدين يتوجب إختيار إحداهما فقط وهما البكسل (x_{k+1} , y_k) أو البكسل (x_{k+1} , y_{k+1})





Circle Drawing Algorithms

Circle Drawing Algorithms

Introduction

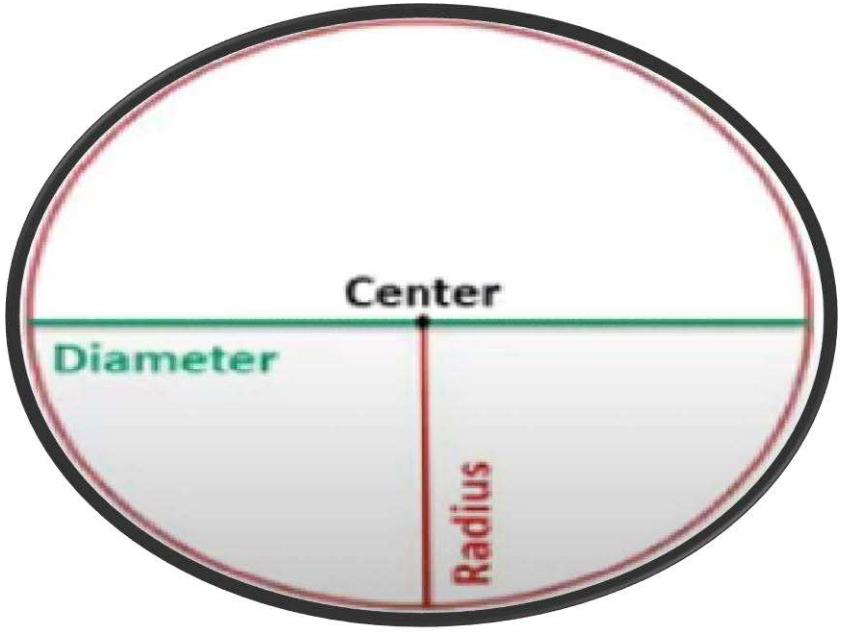
Circle : the set of all points on plane that are fixed distance from a center.

Each circle has :

Center.

Diameter.

Radius.



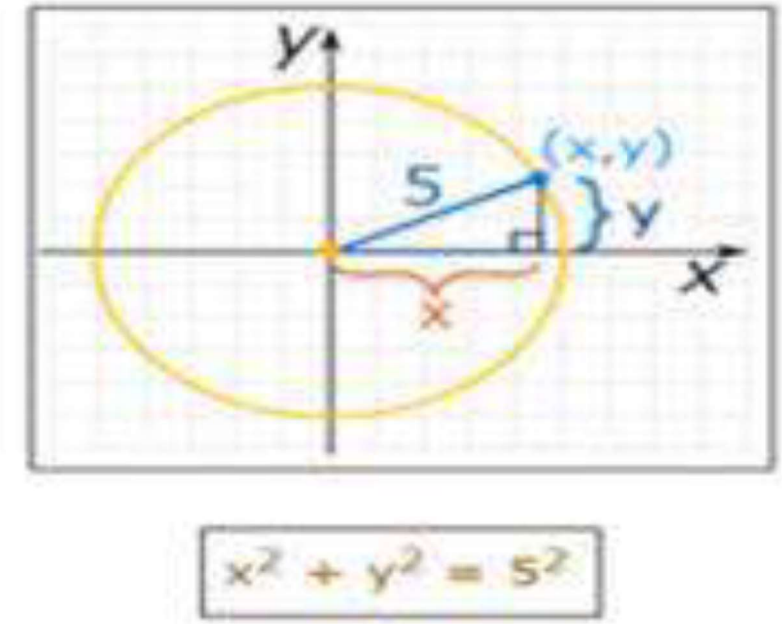
Introduction

Equation of the circle.

For any point on the circle (X,Y) and the center at the origin (0,0),

The equation of the circle is:

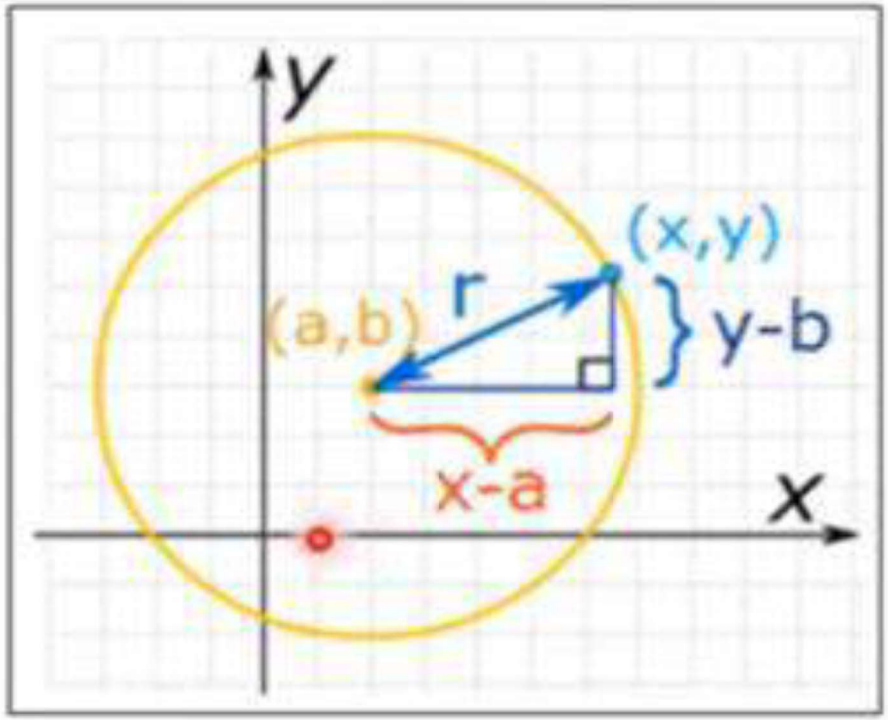
$X^2 + Y^2 = R^2$**,Where R is the radius of the circle.**



Introduction

If the center of the circle at point (a,b) then the equation will be:

$$(X-a)^2+(Y-b)^2 = R^2$$



Introduction

Also ,for any point on a circle (X,Y)and the center at origin (0,0),the equation of the circle is:

$X^2 + Y^2 - R^2 = 0$

Above equation also called circle function f(X,Y) ,Where:

F(x,y)=$X^2 + Y^2 - R^2$

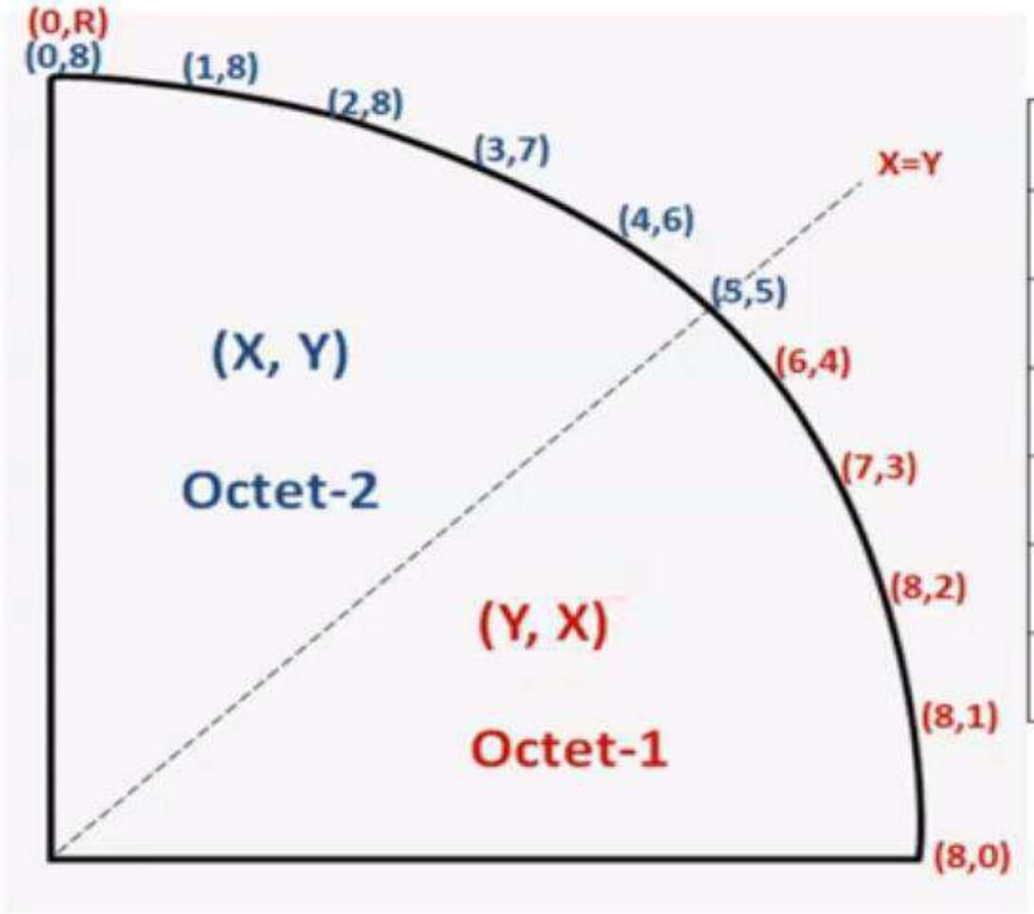
iF the center at point (a,b) ,the equation (circle function)is

F(x,y)=$(X-a)^2+(Y-b)^2 - R^2$

Note: To find the position of a pixel in relation to the circle, we must use circle function as following:

$$F(x,y)=\begin{cases} \text{Pixel is outside circle} & f(x,y) > 0 \\ \text{Pixel is on a circle} & f(x,y) = 0 \\ \text{Pixel is inside circle} & f(x,y) < 0 \end{cases}$$

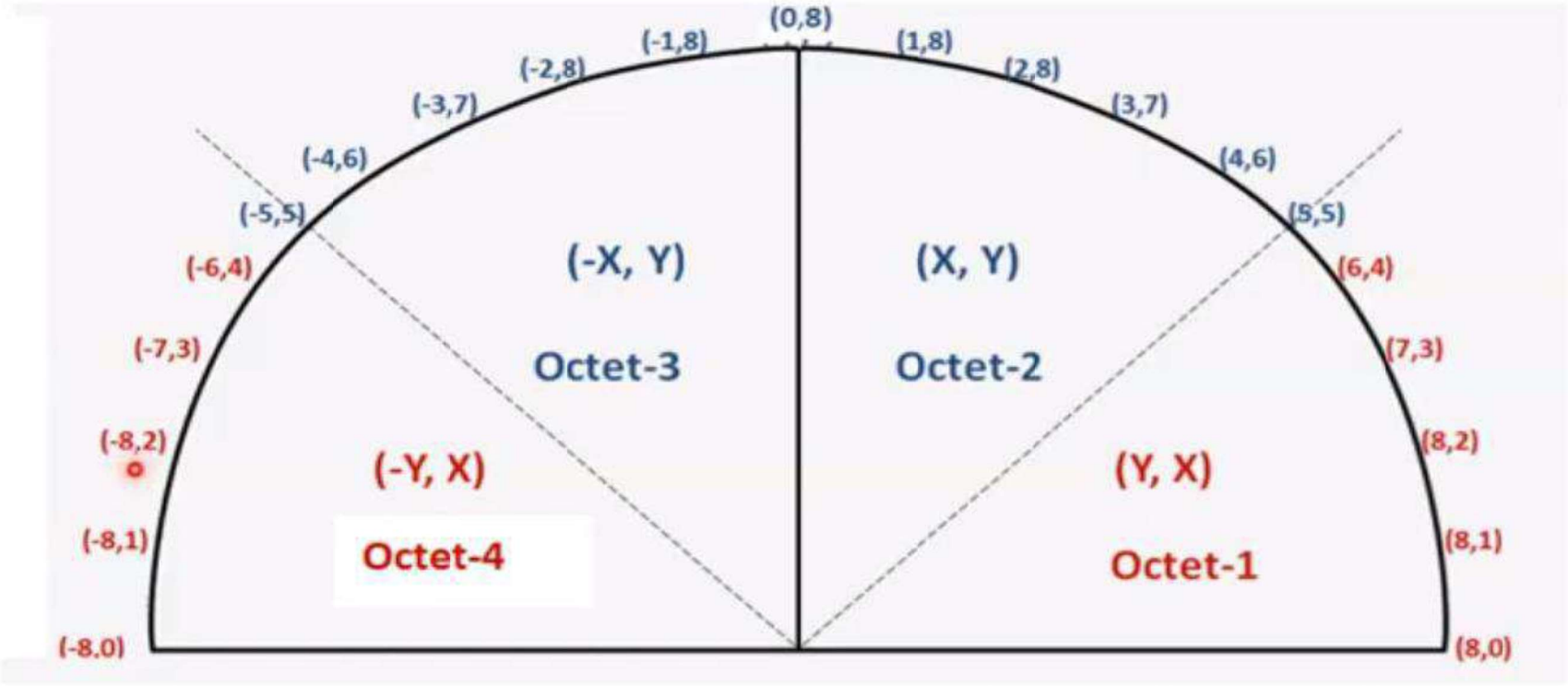
8-Way Symmetry



Octet-2 Points	Octet-1 Points
(0, 8)	(5, 5)
(1, 8)	(6, 4)
(2, 8)	(7, 3)
(3, 7)	(8, 2)
(4, 6)	(8, 1)
(5, 5)	(8, 0)

Introduction

8-Way Symmetry



8-Way Symmetry

- 8-Way symmetry is based on (Mirror reflection). If we see Right hand, in the mirror we will Left hand, similarly if we see pixel (x, y), in the mirror we will see (y, x). So, point (x,y) in octect-2 will become point (y, x) in octect-1 after reflection. Point (-x, y) in octet-3 will become point (-y, x) in octet-4 and so on.

(x, y)

(-x, y)

(-x,-y)

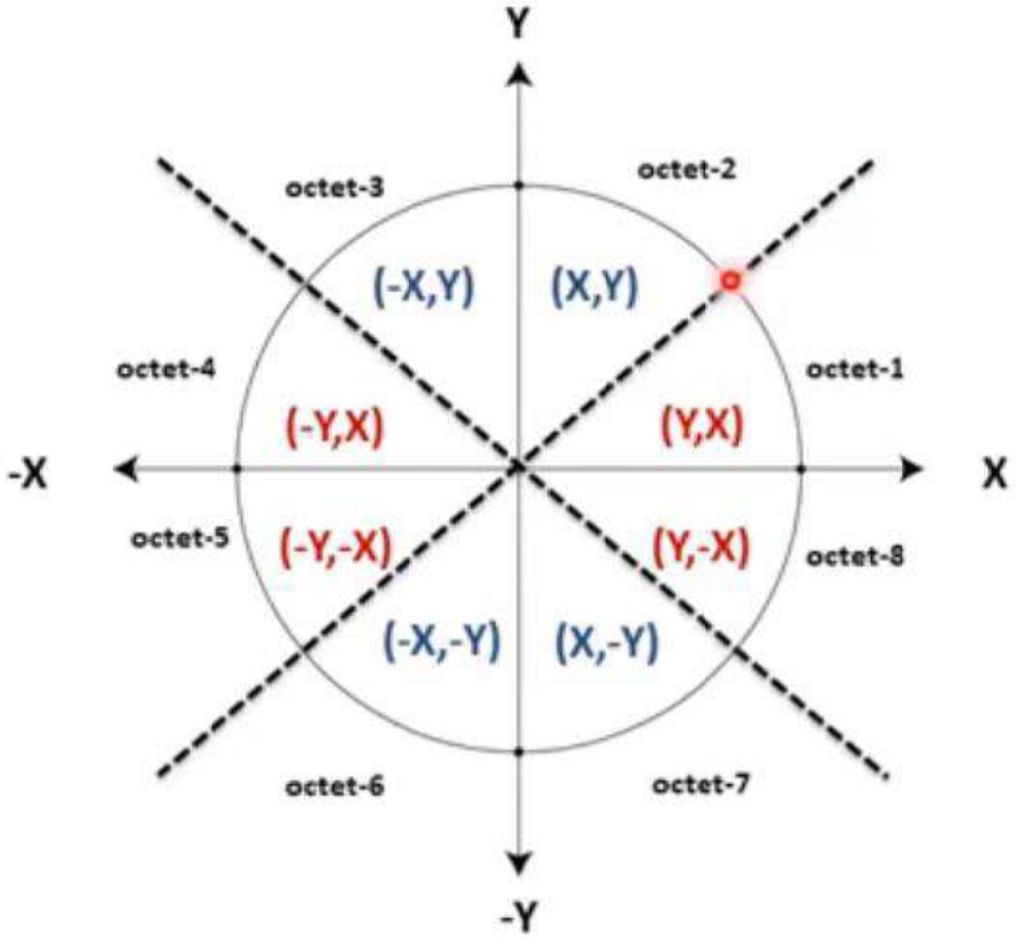
(x,-y)

(y, x)

(-y, x)

(-y,-x)

(y,-x)



Circle Drawing Algorithms

Circles are probably the most used curves to elementary graphics. They often serve as building blocks to generate artistic images. This sheet describes circle drawing algorithm.

Type circle drawing

1-circle generation algorithm.

2-circle generation bresenham's algorithm.

3-circle drawing by using circle equation.

Circle generation Algorithm

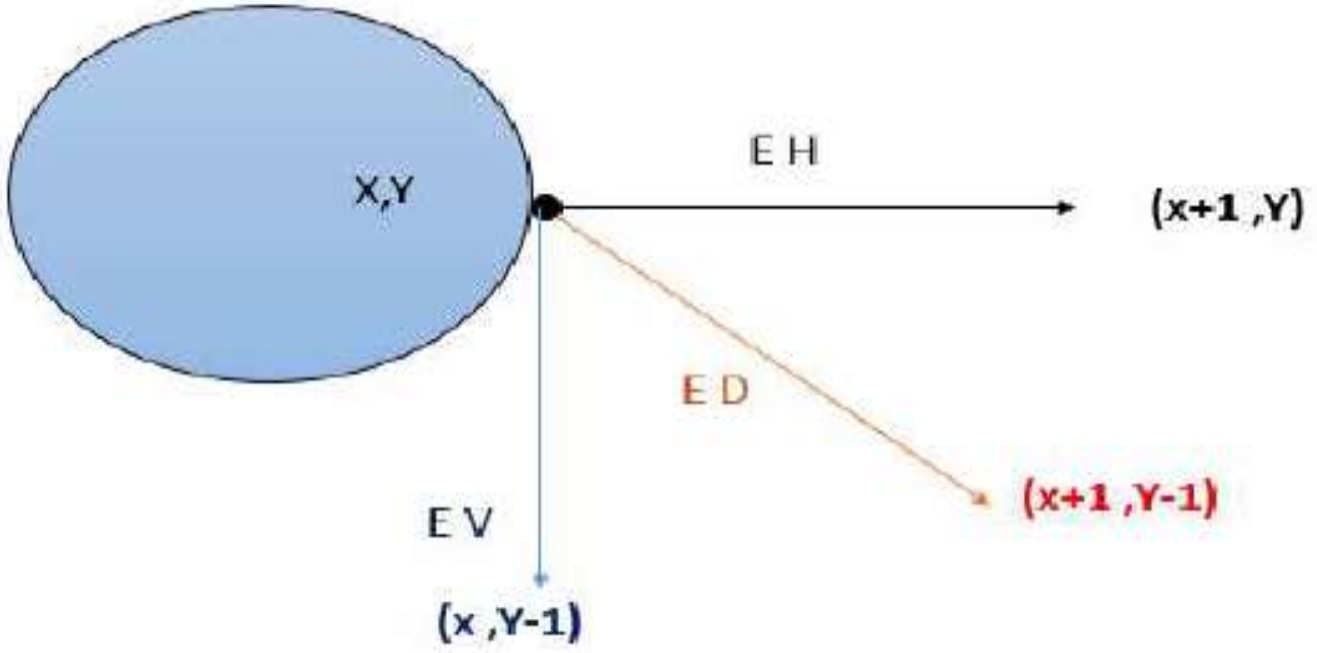
For any given point on the circle in clockwise to generation of it there are only three possible selections for the next pixel which best represents the circle horizontally to the right, diagonally downward to the right and vertically downward. These are labeled: $H\,m$, $D\,m$, $V\,m$,

The algorithm chooses the pixel

which minimizes the square of the distance between

one of these pixels and the true circle, i.e. the minimum of:

Circle Drawing Algorithms



minimum of:

$E_H = |(x+1)^2 + y^2 - R^2|$

$E_D = |(x+1)^2 + (y-1)^2 - R^2|$

$E_V = |x^2 + (y-1)^2 - R^2|$

1. Circle generation Algorithm

X=XC Y=YC + R

While (y >= 0)

$E_H = |(x+1)^2 + y^2 - R^2|$

$E_D = |(x+1)^2 + (y-1)^2 - R^2|$

$E_V = |x^2 + (y-1)^2 - R^2|$

Min = minimum (Eh, Ed, E v)

If min = Eh then X=X+1: go to 1

If min = Ed then X=X+1:Y=Y-1: go to 1

If min = Ev then y = y - 1

1 End while

Ex /Find the coordinates of the circle when (Xc,Yc)=(0,0) and Radios = 8 using circle generation algorithm?

X	Y	Plot	E H	E D	E V	X	Y
0 1 2 3	8 8 8 7 . . 0	0,8 1,8 2,8 3,8	1 4 9 1	14 11 6	15 14 11	1 2 3	8 8 7

$E_H = |(x+1)^2 + y^2 - R^2|$ **$|(0+1)^2 + 8^2 - 8^2|$** **=1**

$E_D = |(x+1)^2 + (y-1)^2 - R^2|$ **$|(0+1)^2 + (8-1)^2 - 8^2| = |(1)+49-64| = 14$**

$E_V = |x^2 + (y-1)^2 - R^2|$ **$|(0)^2 + (8-1)^2 - 8^2| = |0+49-64| = 15$**

Min = minimum =1 EH → (X+1,Y)

Circle Generation – Bresenham`s Algorithm

One of the most efficient and easiest to drive of the circle algorithms is due to Bresenham. To begin, note that only one octant of the circle need be generated. The other parts can be obtained by successive reflections. This is illustrated in Fig. 5. If the first octant (0 to 45 ccw) is generated, the second octant can be obtained by reflection through the line y=x to yield the first quadrant. The results in the first quadrant are reflected through the line x=0 to obtain those in the second quadrant.

Circle Generation – Bresenham`s Algorithm

The combined result in the upper semicircle are reflected through the line y=0 to complete the circle. Bresenham`s Algorithm is consider the first quadrant of an origin- centered circle. If the algorithm begins at x=0 , y=R, then for clockwise generation of the circle y is a monotonically decreasing function of x in the first quadrant. Here the clockwise generation starting at x=0, y=R is chosen.

The center of the circle is (0,0). See Figure (6).

Circle Generation – Bresenham`s Algorithm

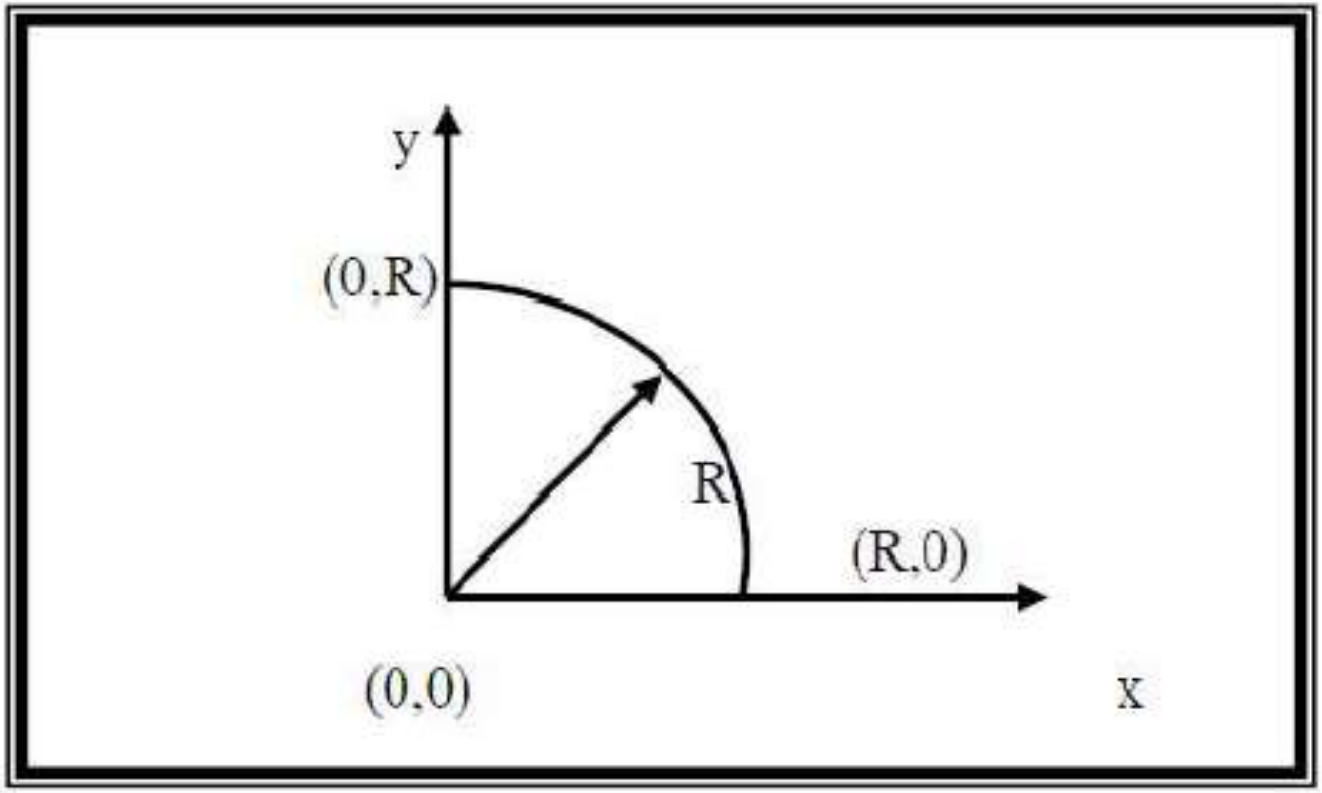


Figure 6: First quadrant of a circle.

Circle Generation – Bresenham`s Algorithm

The circle equation, when the center (a,b), and the radius R, is:

$$(x-a)^2+(y-b)^2=R^2 \qquad \cdots\cdots\cdots(1)$$

And when the center of this circle is the origin (0, 0), then the equation:

$$x^2+y^2=R^2 \quad \text{Because of a, b} = 0.$$

The different between the square of the distance from the center of the circle to the diagonal pixel at (+1 , −1) *i i x y* and the distance to a point on the circle R^2 is:

The different between the square of the distance from the center of the circle to the diagonal pixel at (x_i+1, y_i-1) and the distance to a point on the circle R^2 is:

$$\Delta_i=(x_i+1)^2+(y_i-1)^2-R^2 \qquad \cdots\cdots\cdots\cdots(2)$$

If $\Delta_i<0$ then find $\delta=m_H-m_D$ $\cdots\cdots\cdots\cdots(3)$

If $\Delta_i=0$ then find $\delta=m_D$ $\cdots\cdots\cdots\cdots(4)$

If $\Delta_i>0$ then find $\delta=m_D-m_V$ $\cdots\cdots\cdots\cdots(5)$

Circle Generation – Bresenham`s Algorithm

Bresenham`s suggestions that we can find Δ_n depending on Δ_0:

1- Horizontal Case:

$$\begin{aligned}\Delta_n &= (x_n+1)^2 + (y_n-1)^2 - R^2 \\ &= (x_n)^2 + (y_n-1)^2 - R^2 + 2x_n + 1 \\ &= (x_0+1)^2 + (y_0-1)^2 - R^2 + 2x_n + 1 \\ \Delta_n &= \Delta_0 + 2x_n + 1\end{aligned}$$

$$x_n = x_0 + 1, \quad y_n = y_0$$

2- Diagonal Case:

$$\begin{aligned}\Delta_n &= (x_n+1)^2 + (y_n-1)^2 - R^2 \\ &= (x_n)^2 + (y_n)^2 - R^2 + 2x_n - 2y_n + 2 \\ &= (x_0+1)^2 + (y_0-1)^2 - R^2 + 2x_n - 2y_n + 2 \\ \Delta_n &= \Delta_0 + 2x_n - 2y_n + 2\end{aligned}$$

$$x_n = x_0 + 1, \quad y_n = y_0 - 1$$

Circle Generation – Bresenham`s Algorithm

3- Vertical Case:

$$\Delta_n = (x_n + 1)^2 + (y_n - 1)^2 - R^2$$
$$= (x_n + 1)^2 + (y_n)^2 - R^2 - 2y_n + 1$$
$$= (x_\circ + 1)^2 + (y_\circ - 1)^2 - R^2 - 2y_n + 1$$
$$\Delta_n = \Delta_\circ - 2y_n + 1$$

$$x_n = x_\circ \, , y_n = y_\circ - 1$$

By simplified the equation (3), (5) we get:

$$\delta = 2\Delta + 2y - 1$$
$$\overline{\delta} = 2\Delta - 2x - 1$$

When x=0, y=R, and use these values in equation (2):

$$\Delta = 1 + (R-1)^2 - R^2 \quad \Rightarrow \quad \Delta = 2(1 - R)$$

2. *Bresenham's algorithm*

x= xc
y = R+yc
Limit = yc
1 Plot (x, y)
Δ= $(x+1)^2+(y-1)^2-R^2$
If y ≤ limit then 4

If Δ < 0 then 2
If Δ = 0 then 20
if Δ > 0 then 3
2δ = 2 Δ + 2y –1
if δ ≤ 0 then 10
if δ > 0 then 20
3 δ = 2 Δ – 2x-1
If δ ≤ 0 then 20
if δ > 0 then 30
10 x = x + 1 : go to 1
20 x = x + 1 : y = y - 1: go to 1
30 y = y - 1 : go to 1
4 END

Ex/Find the coordinates of the circle when (Xc,Yc)=(0,0) and Radios = 8 using Bresenham's circle algorithm(I)?

x	Y	Plot	Δ	δ	δ'	x	y
0	8	(0,8)	-14	-13	-	1	8
1	8	(1,8)	-11	-7	-	2	8
2	8	(2,8)	-6	3	-	3	7
3	7	(3,7)	-12	-11	-	4	7
4	7	(4,7)	-3	7	-	5	6
5	6	(5,6)	-3	5	-	6	5
6	5	(6,5)	1	-	-11	7	4
7	4	(7,4)	9	-	3	7	3
7	3	(7,3)	4	-	-7	8	2
8	2	(8,2)	18	-	19	8	1
8	1	(8,1)	17	-	17	8	0
8	0	(8,0)	18	-			

3.Bresenham's Circle Algorithm (II)

x = Xc
y = R+yc
limit = yc
Δ = 2 (1 - R)
1 Plot (x,y)
if y ≤ limit then 4

if Δ < 0 then 2
if Δ = 0 then 20
if Δ > 0 then 3
2 δ=2Δ+2y-1
if δ ≤ 0 then 10
if δ > 0 then 20
3 δ' = 2Δ-2x-1
if δ' ≤ 0 then 20
if δ' > 0 then 30
10 x = x + 1 : Δ = Δ + 2x + 1: go to 1
20 x = x + 1 : y = y - 1: Δ = Δ + 2x - 2y + 2: go to 1
30 y = y – 1 : Δ = Δ - 2y + 1: go to 1

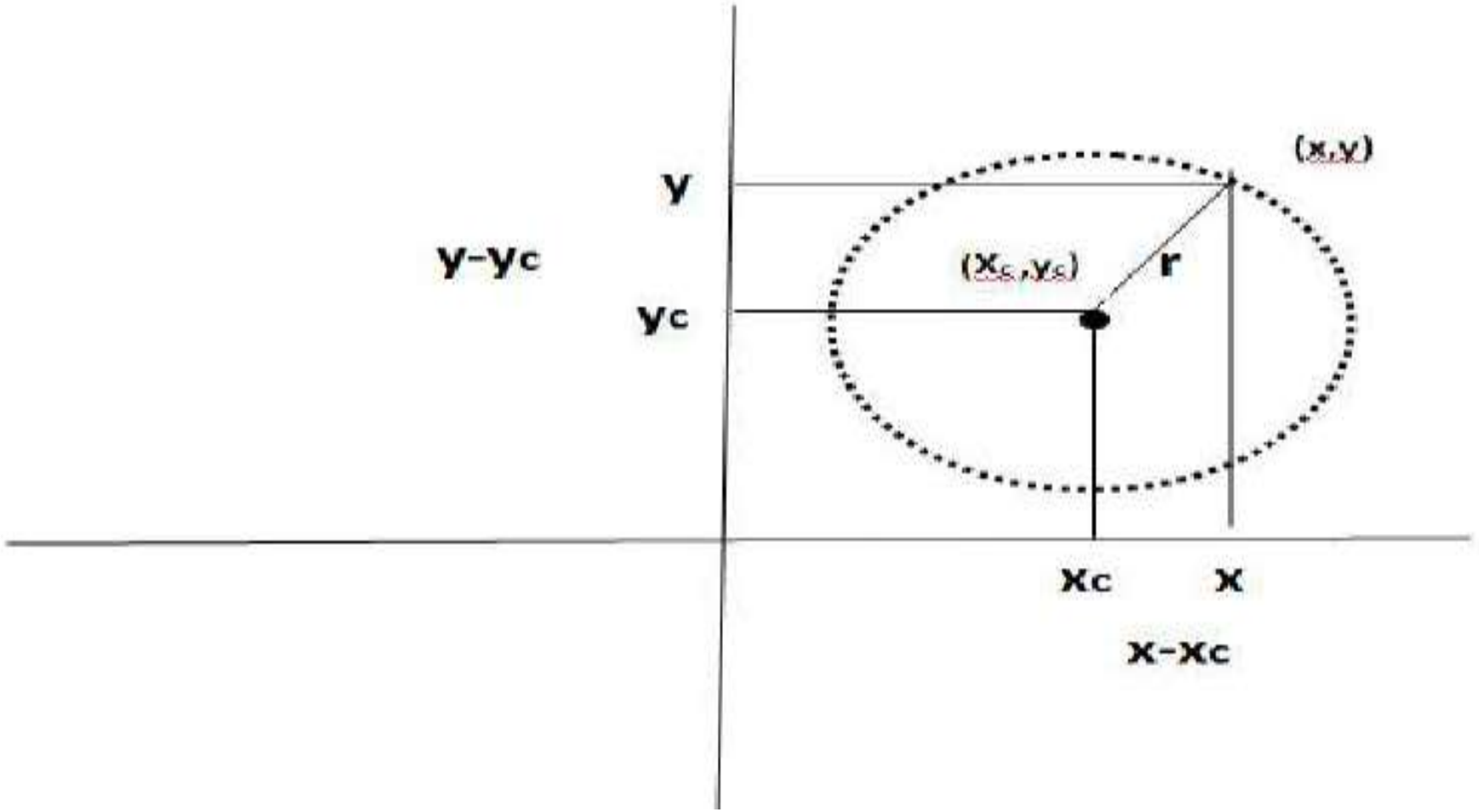
4 END

ex /Find the coordinates of the circle when (Xc,Yc)=(0,0) and Radios = 8 using Bresenham's circle algorithm(II)?

χ	y	Plot	Δ	δ	$'\delta$	x	y	Δ_N
0	8	0,8	-14	-13	-	1	8	-11
1	8	1,8	-11	-7	-	2	8	-6
2	8	2,8	-6	3	-	3	7	-12
3	7	3,7	-12	-11	-	4	7	-3
4	7	4,7	-3	7	-	5	6	-3
5	6	5,6	-3	5	-	6	5	1
6	5	6,5	1	-	-11	7	4	9
7	4	7,4	9	-	3	7	3	4
7	3	7,3	4	-	-7	8	2	18
8	2	8,2	18	-	19	8	1	17
8	1	8,1	17	-	17	8	0	18
8	0	8,0	18	-	-			

Circle Drawing By Using Circle Equation

A circle is specified by the coordinates of its center (X c, Y c) and its radius (r) .



Circle Drawing Algorithm (By Using Circle Equation)

1- x= -r

2- while(x=r)

3- y1= +SQRT (r*r – SQR (x-))

4. 2= -SQRT (r*r – SQR (x-))

5. Plotpixel (x,y1) plotpixel (x,y2)

6. X=x+1

7. Repet step2 until x= +r

4. Circle Drawing By Using Circle Equation

$(\chi-\chi c)^2 + (y-yc)^2 = R^2$ ------(1)

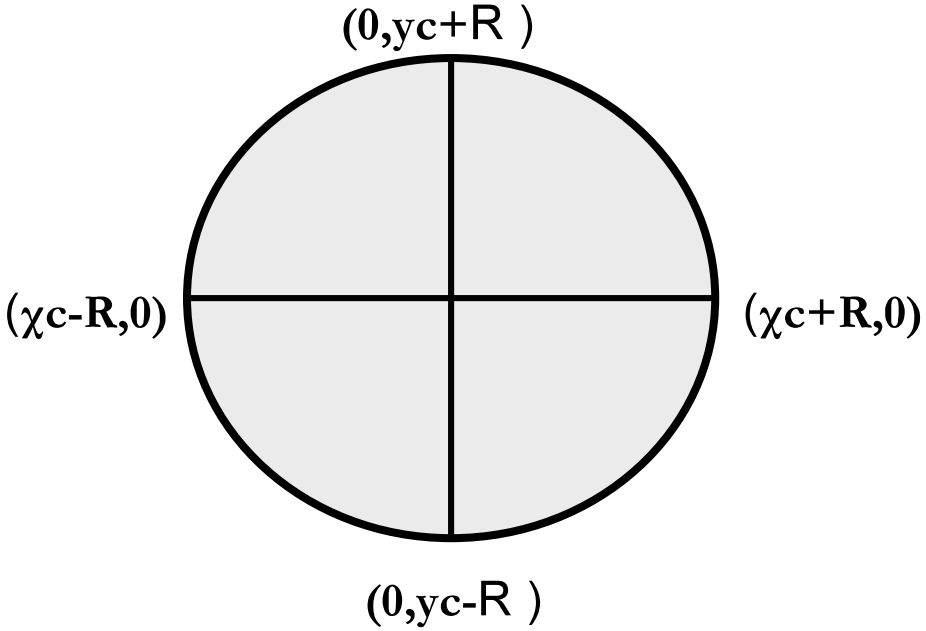
X , Y:- point on the boundary

Xc , Yc:- the center of the circle

R:- ½ Radious

From equation (1) we can find that

$$y = Yc \pm \sqrt{R^2 - (X-Xc)^2}$$



For X = (Xc-R) to (Xc + R)

$$y = yc \pm \sqrt{R^2 - (\chi-\chi c)^2}$$

This method of drawing a circle is inefficient it because :-

•We are not taking advantage of the symmetry of a circle .

•The amount of processing time required to perform the squaring and square root operations repeatedly .

•X value are equally spacd(the differ by one unit) the Y value are not .

The circle is denot and flat near the Y-axis and has large gaps and the steep near the X-axis.

ex : R=5 Xc=0 Yc=0

For X: = -R To + R

$$y = \pm\sqrt{R^2 - X^2}$$

X	Y	plot
-5	0	(-5,0)
-4	-3,+3	(-4,-3),(-4,3)
-3	-4,+4	(-3,-4),(-3,4)
-2		
-1		
0	-5,+5	(0,-5),(0,5)
1		
2		
3	-4,+4	(3,-4),(3,4)
4	-3,+3	(4,-3),(4,3)
5	0	(5,0)

Aspect Ratio

Aspect Ratio: is the ratio of the horizontal width to the vertical height.

- The horizontal and vertical plots of an equal number of pixels have different lengths.
- The ratio is a consequence of non – square pixels and rectangular display screen.

Aspect Ratio

Ex/

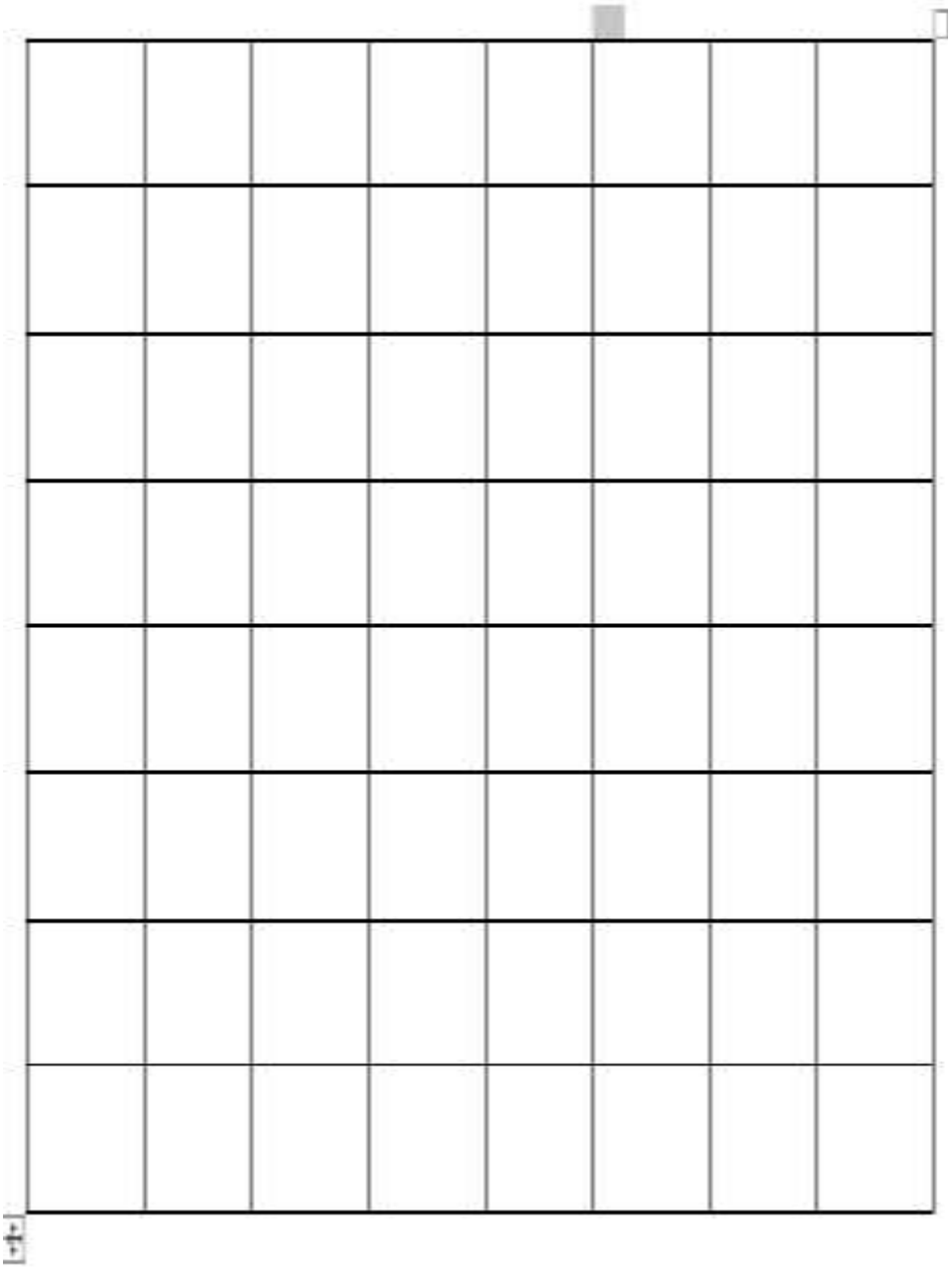
If we plot eight pixels horizontally on the display screen and then we measure the line, we find that it is 0 .3 cm wide. If we plot eight pixels vertically on the display screen and then we measure the line, we find that its 0.4 cm height.

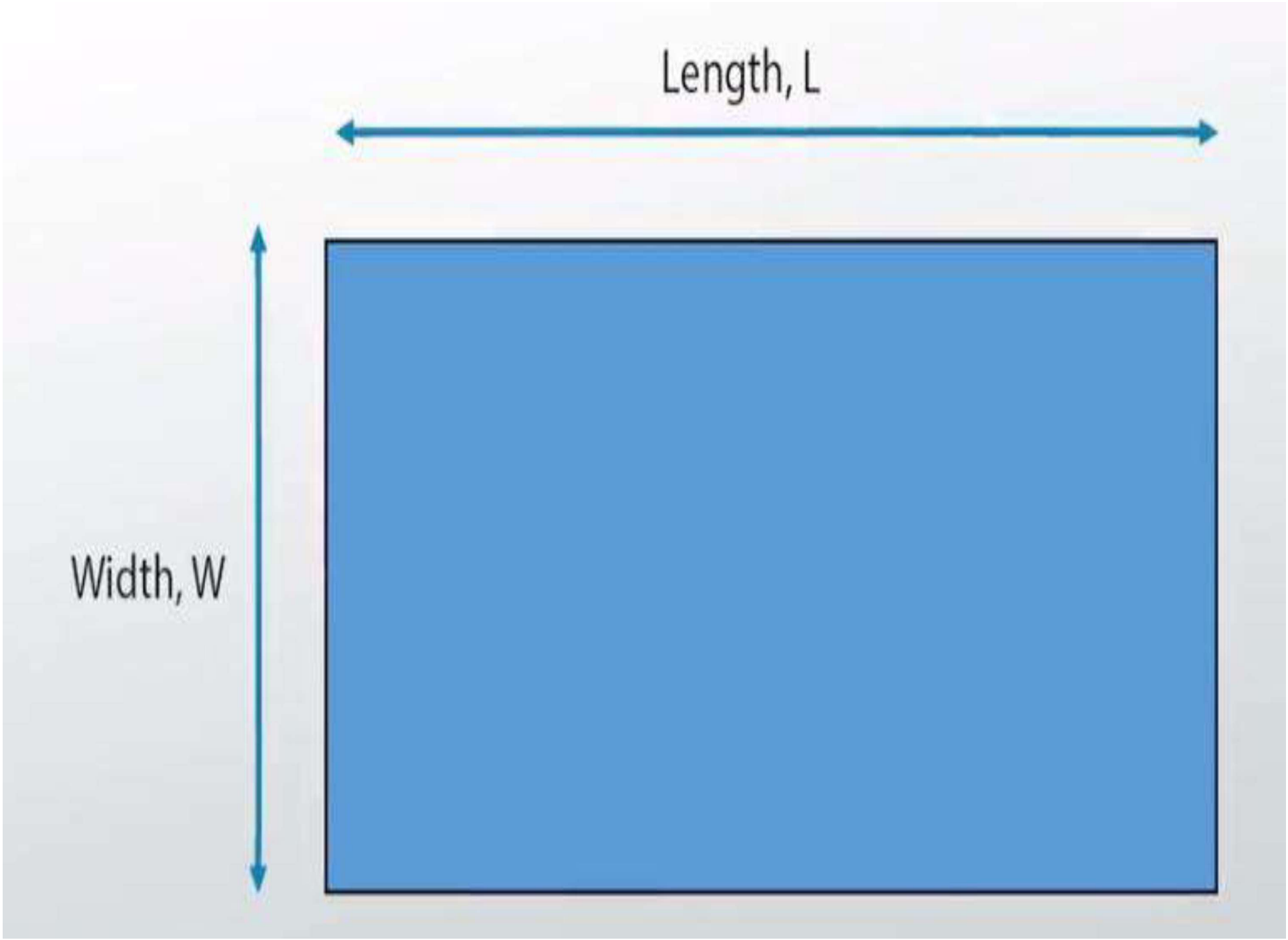
The **A.R = 0.4 / 0.3 = 1.33 .**

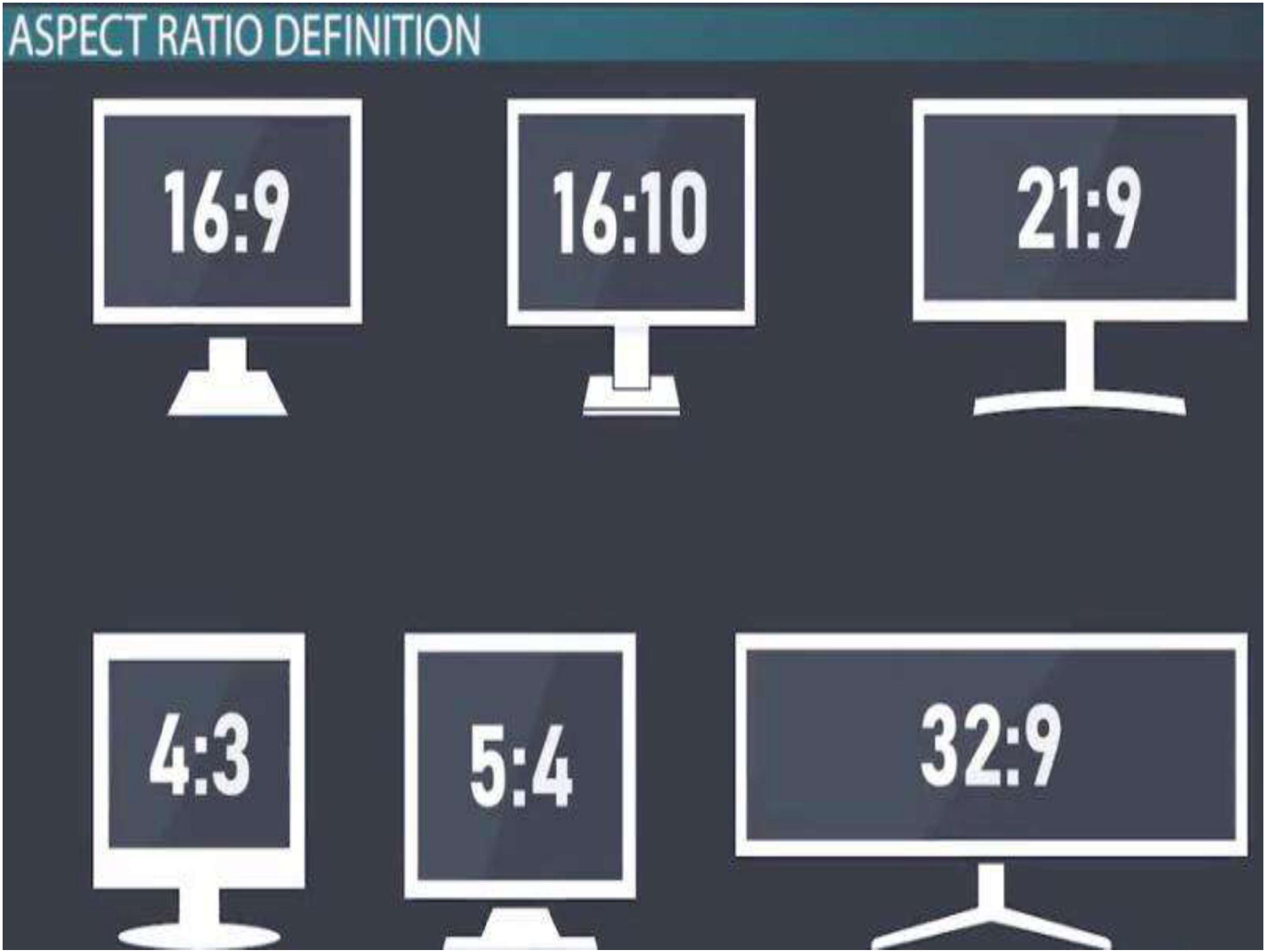
New number of pixels = Old number of pixels * AR .

New number of pixels = 8 * 1.33 = 10.64 ≈ 11 pixels.

11 – 8 = 3 pixels will be added to the horizontal line.




Length, L
Width, W


ASPECT RATIO DEFINITION
16:9
16:10
21:9
4:3
5:4
32:9

The normalized device coordinate

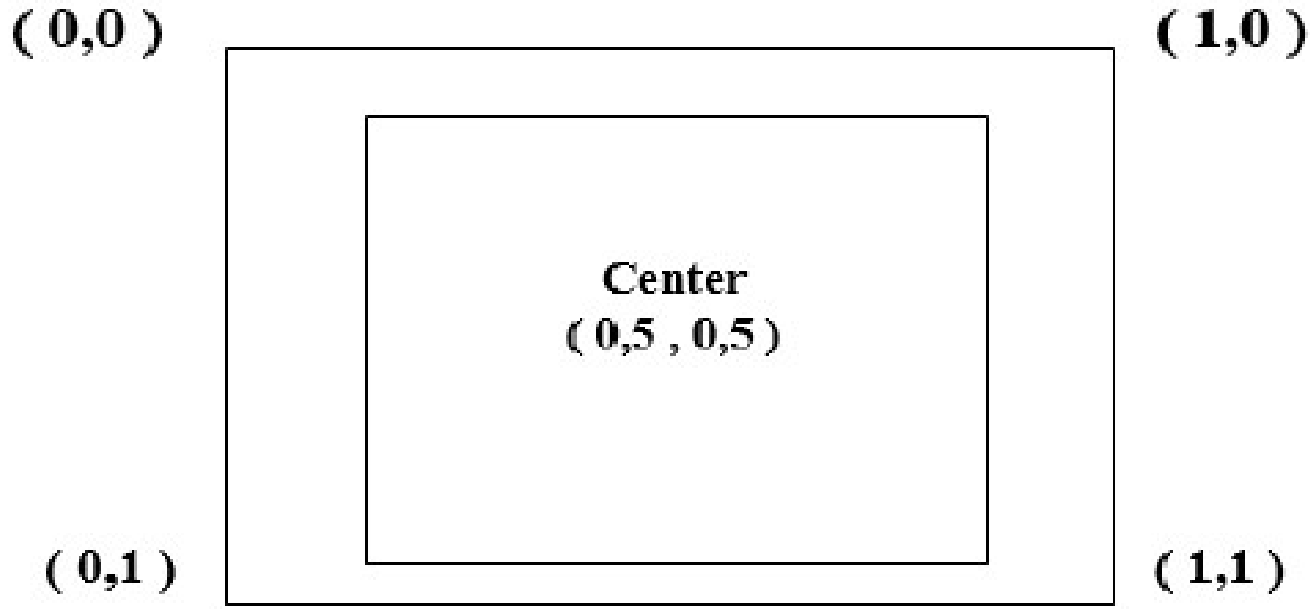
Different display devices may have different screen sizes as measured in pixels.

If we wish our program to be devices independent we should specify the coordinates in some units other than pixels and then use the interpreter to convert these coordinates to the appropriates values for particular display , we are using device independent units are called the normalized device coordinates.

In theses units, the screen measures (1) unit wide and (1) unit high, the lower left corner of the screen is the origin , and the right upper corner is the point (1 , 1) , the point(0.5 , 0.5) is the center of the screen no matter what the physical or resolution of the actual display device may be .

Appropriate pixel values for the particular display we are using the device Independent units are called the normalized devices coordinates.

قيم البكسل المناسبة للشاشة المحددة التي نستخدمها في الجهاز تسمى الوحدات المستقلة بإحداثيات الأجهزة الطبيعية.



Normalized device coordinates

The interpreter uses a simple linear formula to convert from normalized device coordinates to the actual device coordinates suppose that for actual display the index of the left most pixel is the (**width – start**) and that there are (**width**) pixels in the horizontal direction .

يستخدم المترجم صيغة خطية بسيطة للتحويل من إحداثيات الجهاز الطبيعية إلى إحداثيات الجهاز الفعلية، افترض أنه بالنسبة للعرض الفعلي فإن مؤشر البكسل الأيسر هو (العرض - البداية) وأن هناك (عرض) بكسل في الاتجاه الأفقي

Suppose also that the bottom most pixel is (**high – start**) and the number of Pixels in the vertical direction is (**height**). In the normalized coordinates the screen is one unit wide, so the normalized coordinates it is (**width**) units wide, so the normalized **X** position should by multiplied by (**width / 1**) to convert it actual screen units . At position **Xn = 0** in normalized coordinates we so the should

get Xs = **width – start** in actual screen coordinate,

conversion formula should be:-

Xs = width * Xn + width – start.

Similarly for the vertical direction

Ys =height * Yn + height – start

Xs / Xn =width / 1

Xs = width * Xn + width – start

Screen

normalized

Normalization

هي عملية تحويل إحداثيات الشاشة إلى إحداثيات جديدة وتفيد في عملية تنفيذ البرامج على جميع أنواع الشاشات ومهما كان حجم الشاشة .

وتجري عملية التحويل كما يلـــي : -

- أخذ جميع إحداثيات الصورة على max (x-1) وmax (y-1) وتكون محصورة ما بين (0 , 1) سواء على المحور X أو المحور Y .

Ex //

If we have points P1(0,0) , p2(200,200) p3(1023,1023) , convert these points from screen 1024 * 1024 to appropriate coordinates in screen 800*600 ?

العملية الأولى:- تقسيم إحداثيات الشاشة المصدر ناقص واحد:

MaxX = 1023(1024 = 0..1023).
MaxY = 1023(1024 = 0..1023).

We use the equation
Xr = X / maxX ; Yr = Y / maxY

For all points so we get:-
p1 * (0 /1023 , 0 /1023) = ≈(0,0)
P2 * (200 /1023 , 200 /1023) = ≈(0.195,0.195)
P3 * (1023 /1023 , 1023 /1023) = ≈ (1,1)

العملية الثانية :- نضرب النسبة في الشاشة الجديدة ناقص واحد :

MaxX = 799(800 = 0..799).

MaxY = 599(600 = 0..599).

By using the equation

Xn = Xr * maxX ; Yn = Yr * maxY

Xn = Xr * maxX ; Yn = Yr * maxY

For all points so we get:-

p1 * = (0*799, 0*599) ≈ (0,0)

P2 * = (0.195*799 , 0.195*599)≈(156 , 116)

P3 * = (1*799, 1*599) ≈ (799,599)

So that the points will be:-

1024 * 1024	800 * 600
(0,0)	(0,0)
(200,200)	(156,116)
(1023,1023)	(799,599)

Advantages of Aspect ratio

In the existence of a picture, it retains equilibrium (ratio between vertical and horizontal pixel).
Pictures do not bend as the aspect ratio is raised.

في وجود الصورة، فإنها تحتفظ بالتوازن (النسبة بين البكسل الرأسي والأفقي).لا تنحني الصور عند زيادة نسبة العرض إلى الارتفاع.

The Most Common Aspect Ratios

Today, there are a few aspect ratios that are most commonly used. You will encounter these over and over, no matter what platform you work in. It's helpful to be familiar with these ratios, although there are many other aspect ratios used besides these.

- 3:2
- 4:3
- 1:1
- 16:9
- 9:16


4:3
3:2
9:16
16:9
1:1
4:3
3:2
16:9
1:1
9:16

TWO DIMENSIONAL GEOMETRIC TRANSFORMATIONS

TWO DIMENSIONAL TRANSFORMATION

1- Translation
2-Scaling
3-Rotation
4-Shearing
5-Reflection

Two Dimensional Transformation

- **Introduction:**
- Geometric transformations provide a mean by which an image can be constructed or modified. The transformations we examine in this sheet are translation, scaling, rotations, reflection and sharing.

The advantage of used the translation: -

- Details appear more clearly.
- Reduces a picture more of if is visible.
- Change the scale of a symbol.
- Rotate it through some angle.

Translation

1 :- **Translation** .

A point (x,y) is translated to a new position (x,y) by moving it H units in the horizontal direction and V units in the vertical direction .

$$X' = X + H$$
$$Y' = Y + V$$

IF

H > 0 the point moves to the right

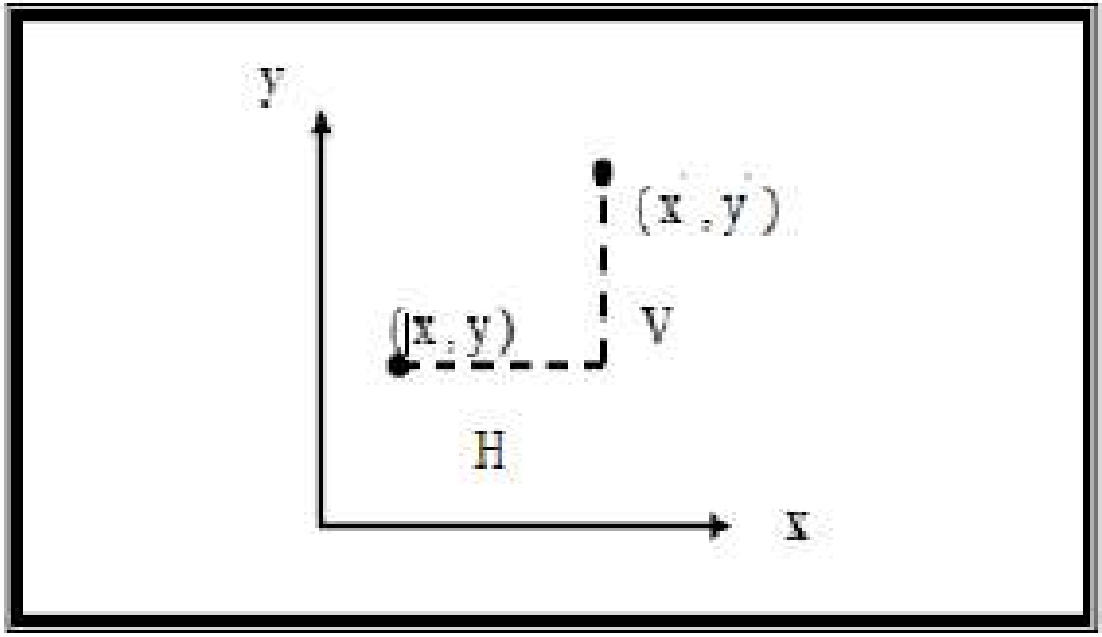
H < 0 the point moves to the left.

V > 0 the point moves to the up.

V < 0 the point moves to the down.

Translation

To translate an object in an image we must translate every point defining this object. All points are displaced the same distance and the object is drawn using these transformed points.



Translation

EX :- consider a triangle defined by its three vertices (20,0) , (60,0) , (40,100) being translated 100 units to the right and 10 units up?

Solution: -

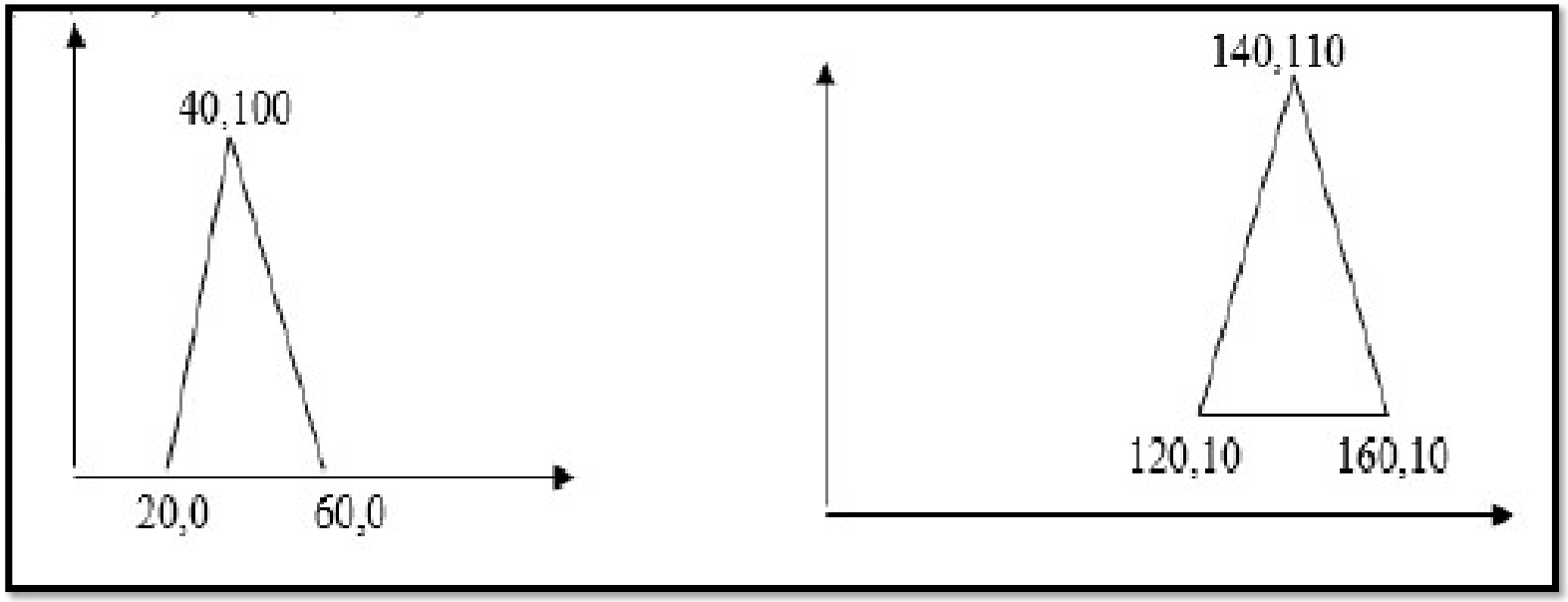
H = 100; V=10 ;By using the equations :-

We get :-

(20,0) → (120,10)

(60,0) → (160,10)

(40,100) → (140,110)



Translation

H . W

1- Consider a triangle defined by it three vertices (40 , 0) , (100 , 0) ,
(60 , 100) be translated 40 units to the left and 40 units down.

2-Consider a triangle defined by it three vertices (40 , 0), (80 , 0), (60 , 100) be translated 120 units to the right and 20 units up.

Scaling

2 :- scaling

We can change the size of an object or the entire image by multiplying the distances between points by an **enlargement or reduction factor**. This factor is called scaling factor and the operation that change the size is called **scaling.**

- ***if***
- 0< Sx <1 the object is reduction in X-direction.
- 1< Sx the object is enlarged in X-direction.
- 0< Sy<1 the object is reduction in Y-direction.
- 1< Sy the object is enlarged in Y-direction.

Scaling

Whenever a scaling is performed there is one point that remains at the same location. This is called the fixed point of the scaling transformation.

If the fixed point is at the origin (0,0) then the point (x,y) can be scaled by a factor Sx in the X-direction and factor Sy in the Y-direction to the new point (x,y) where :-

$x` = x * S_x$

$y` = y * Sy$

- If the Sx # Sy the resulting object is a distortion of the original.
- It is possible to choose any point (Xp,Yp) as the fixed point of scaling . Three steps we need: -

Scaling

1 :-Translate

Translate the point (Xp,Yp) to the origin . Every points (x,y) is moved to a new point (x,y).

$$X' = X - Xp$$
$$Y' = Y - Yp$$

2 :- Scaling

Scale these translated points with the origin as the fixed point

$$X'' = X' * Sx$$
$$Y'' = Y' * Sy$$

3 :-Translate

Translate the origin back to the fixed point (Xp,Yp)

$$X''' = X'' + Xp$$
$$Y''' = Y'' + Yp$$

By substitution we get X = (X - Xp) * Sx + Xp

Y = (Y – Yp) * Sy + Yp

Scaling

Example : Scale the triangle **(4 , 4) , (7 , 8) , (10 , 5)** by **Sx = 2** and **Sy = 2**, about the origin point.

Solution:

The new points are:**(8 , 8) , (14 , 16) , (20 , 10)**

To scale an object about a pivot point (xp , yp), we perform the following three steps:

Step 1: Translate

x` = x - xp

y` = y - yp

Step 2: Scaling

x` = x * Sx

y` = y * Sy

Step 3:

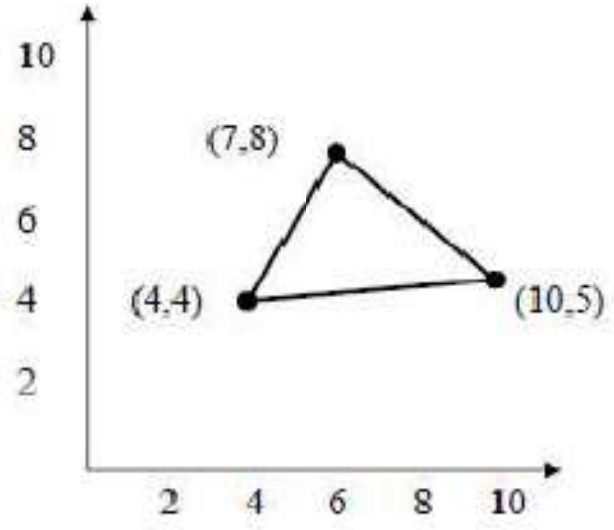
x``` = x`` + xp

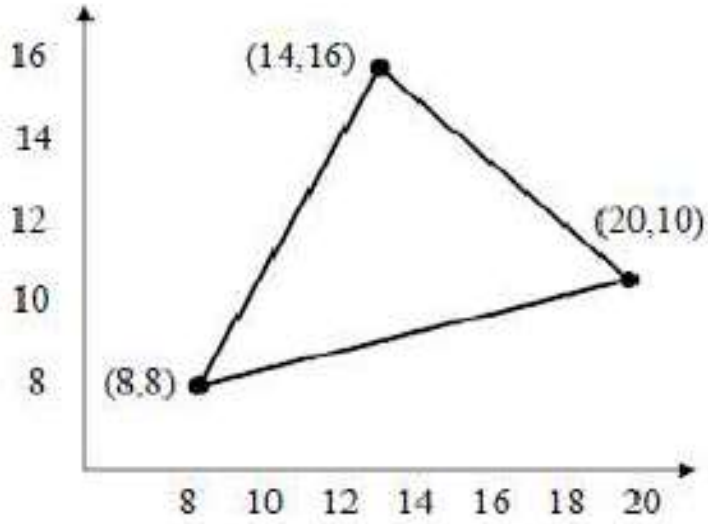
y``` = y`` + yp

By substituting: بالتبديل

X`= (x - xp)*Sx+ xp

Y`= (y- yp)*Sy+yp





ملاحظات مهمة

" scaling factor" اذا كان معامل التحجيم :

- *اكبر من واحد يعني تكبير الصورة او الرسم
- *اذا كان معامل التحجيم اصغر من واحد يعني تصغير الصورة او الرسم
- *اذا كان يساوي واحد يعني لا يوجد تغيير في ابعاد الصورة او الرسم
- *sx=sy اذا كان معامل التحجيم متساويان للمحورين فيكون افضل حالة.

" fixed point" عندما يتم تنفيذ عملية التحجيم فهنالك نقطة واحدة تبقى في نفس مكانها وتدعى بالنقطة الثابتة .

sx=sy اذا كان معامل التحجيم متساويان فيكون افضل حالة.

ممكن حل السؤال حسب المتطلبات اما باستخدام القانون او باستخدام المصفوفات من نقطة الاصل.

في حالة يكون عملية التحجيم من غير نقطة الاصل نستخدم القواعد الثلاثة التي تم شرحها سابقا .

SX or SY اذا لم يعطي احد المعاملات في السؤال نفرض قيمته واحد.

Scaling

H . w :

1 . Magnify the triangle (0 , 0) , (8 , 10) , (12 , 4), 4 times its size, about the origin point.

2 . Magnify the above triangle 1 / 2 its size.

3 . Magnify the triangle (0 , - 3) , (- 6 , - 7) , (6 , - 7), 3 times its size, about the point (0,-3).

Rotation

- Another useful transformation is the rotation of an object about a specified pivot point. After the object has been rotated, it is still the same distance away from the pivot point, however its orientation has been changed. It is possible to rotate one or clockwise (negative angle) or counterclockwise (positive angle) direction.
- Any point (x , y) can be represented by its radial distance, **r**, from the origin and its angle, , off the x – axis as show in figure (8).

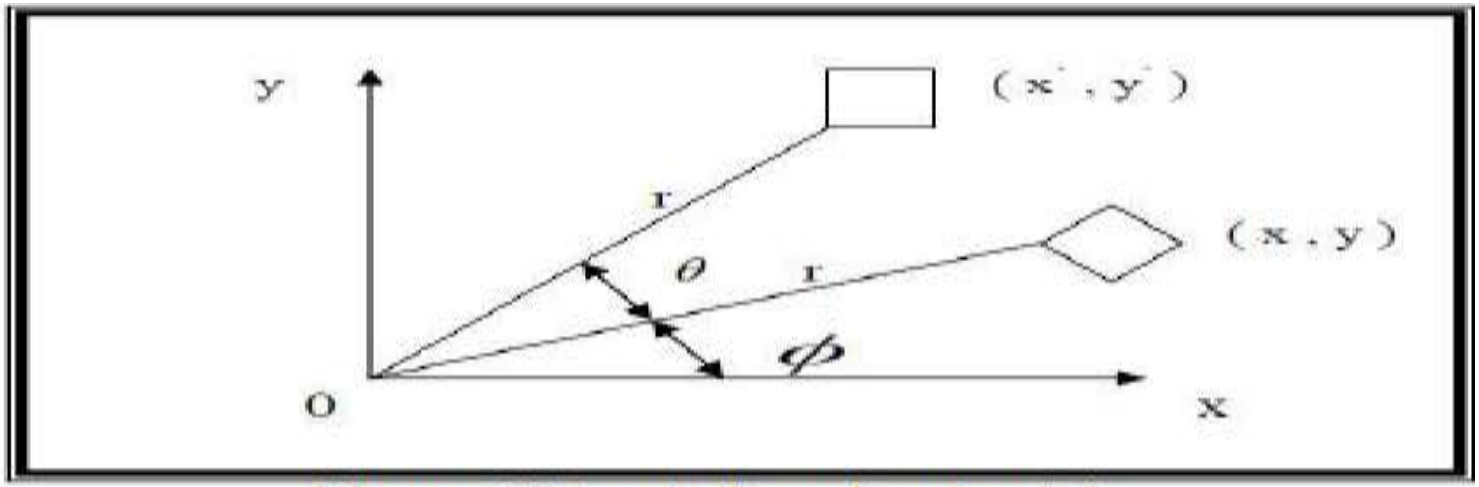


Figure (8) rotation about origin

Rotation

- You can rotate an object about a specified pivot point (Xp,Yp).
- After the object has been rotated it is still the same distance a way from the pivot point but its orientation has been changed.
- It's possible to rotate one or more objects or the entire picture about any point in the world space in either a clockwise or counter clockwise direction.
- Rotation about the origin
- Using the lows of sine and cosines we get: -

- Equation to rotate a point by an angle (θ) about the origin in the counter clockwise direction.

Rotation

- To rotate an object an angle (θ) about a pivot point use the following

three steps :-

1 :-Translate

$$X' = X - Xp$$
$$Y' = Y - Yp$$

Translate the point (Xp,Yp) to the origin . Every points (x,y) is moved to a new point (x,y).

2: - Rotation

Rotate these translated points with the origin as the fixed point.

$$X'' = X' * COS(\theta) - Y' * SIN(\theta)$$
$$Y'' = Y' * COS(\theta) + X' * SIN(\theta)$$

3: -Translate

Translate the origin back to the fixed point (Xp,Yp)

$$X''' = X'' + Xp$$
$$Y''' = Y''' + Yp$$

Rotation

- By substitution we get $X = (X - Xp) * COS(\theta)-(Y-Yp)*SIN(\theta)$

  $Y = (Y – Yp) * COS(\theta)+(X-Xp)*SIN(\theta)$
- To rotate in clockwise direction change the angle (θ) to (-θ) where: -
- $COS(-\theta) = COS(\theta)$
- $SIN(-\theta) = - SIN(\theta)$

- So to rotate a point (X,Y) through a clockwise angle (-θ) about the origin of the coordinate system we get :-

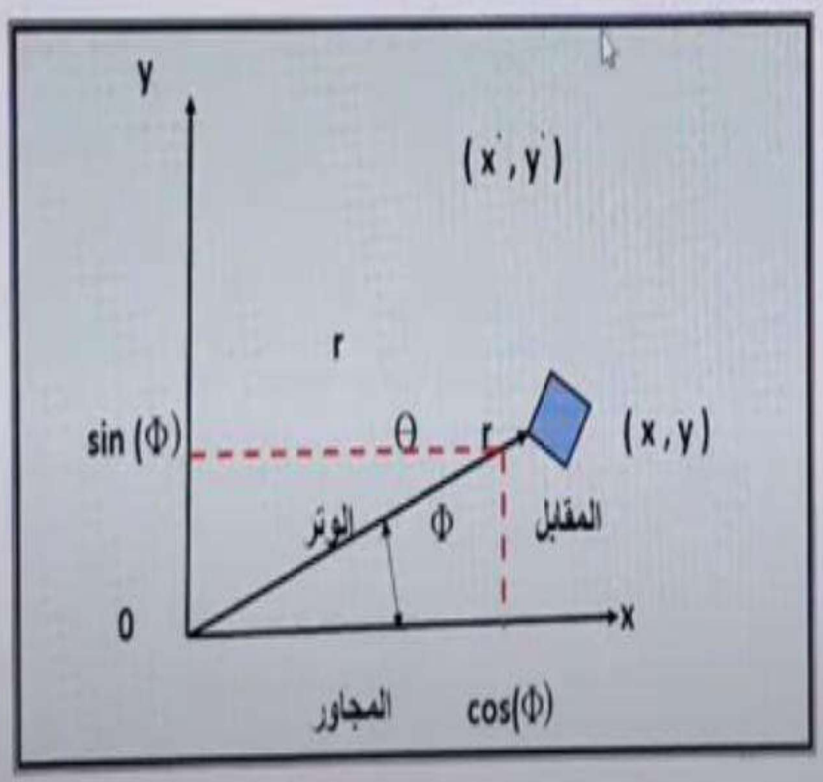
$$X' = X * COS(\theta) + Y * SIN(\theta)$$

$$Y' = Y * COS(\theta) - X * SIN(\theta)$$

Rotation

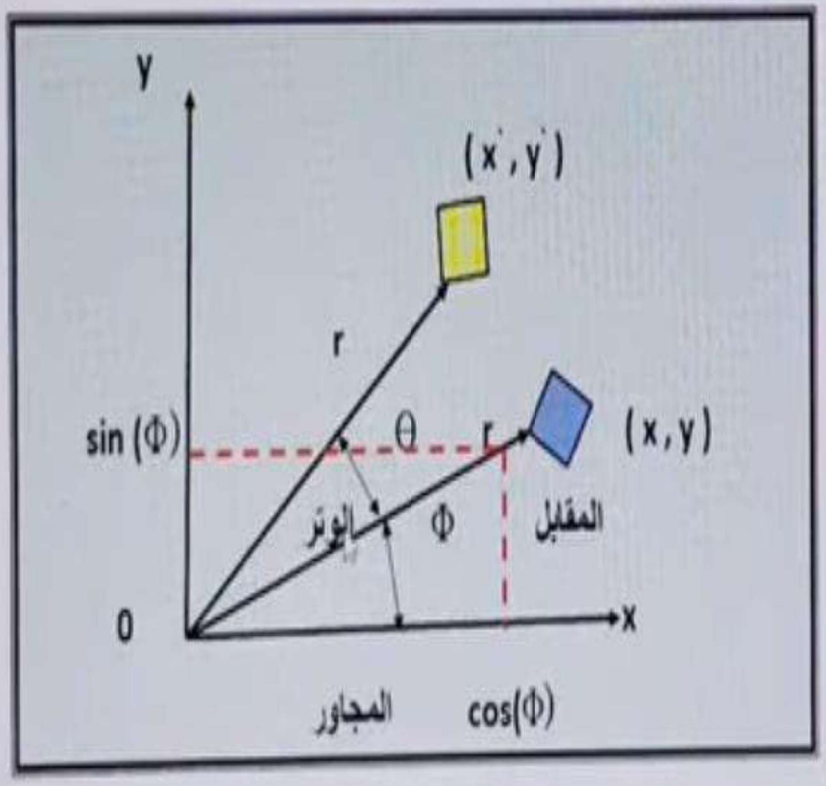
$$\cos(\Phi) = \frac{\text{المجاور}}{\text{الوتر}} = \frac{x}{r} \longrightarrow x = r\cos(\Phi)$$

$$\sin(\Phi) = \frac{\text{المقابل}}{\text{الوتر}} = \frac{y}{r} \longrightarrow y = r\sin(\Phi)$$



$$\cos(\Phi) = \frac{\text{المجاور}}{\text{الوتر}} = \frac{x}{r} \longrightarrow x = r\cos(\Phi)$$

$$\sin(\Phi) = \frac{\text{المقابل}}{\text{الوتر}} = \frac{y}{r} \longrightarrow y = r\sin(\Phi)$$



Rotation

Rotated an angle θ in the counterclockwise direction from the origin point :

$$x` = x * \cos(\theta) - y * \sin(\theta)$$
$$y` = y * \cos(\theta) + x * \sin(\theta)$$

Rotated an angle θ in the clockwise direction from the origin point :

$$x` = x \cos(\theta) + y \sin(\theta)$$
$$y` = -x \sin(\theta) + y \cos(\theta)$$

Rotation

EX :-

The triangle (20,0), (60,0), (40,100) rotate 45 clockwise direction about the origin.

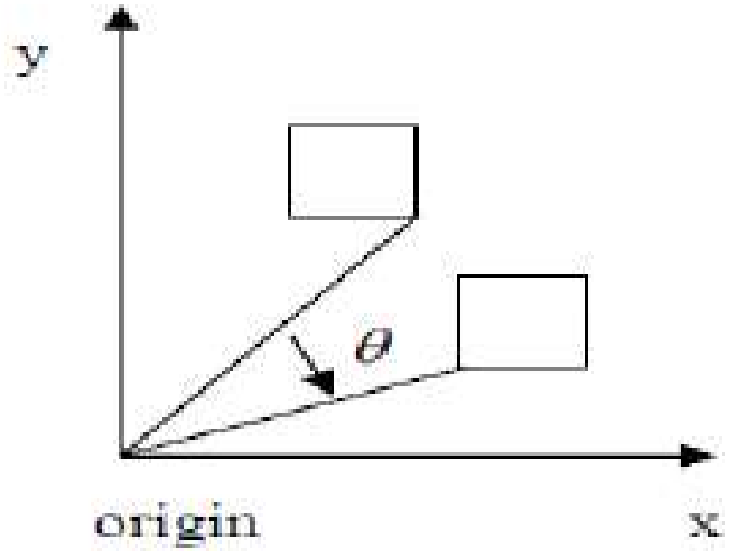
Solution

By using the equations:

$$X' = X * COS(\theta) + Y * SIN(\theta)$$
$$Y' = Y * COS(\theta) - X * SIN(\theta)$$

We get: -

(20,0) → (14,-14)
(60,0) → (42,-42)
(40,100) → (99,42)



Rotation

EX:- Rotate The triangle (7,8), (4,4), (10,5) 90 counter clockwise about the point (7,8) , step by step? **NOT: Sin (90)=1 , Cos(90)=0;**

- **Step 1: Translate:**

- x` = x – xp
- y` = y – yp

Xp=7, Yp= 8 ;

x`1=7-7=0	y`1=8-8=0	(0 ,0)
x`1=4-7=-3	y`1=4-8=-4	(-3 , -4)
x`1=10-7=3	y`1=5-8=-3	(3 , -3)

Rotation

Step 2: Rotate :

$$X'' = X' * COS(\theta) - Y' * SIN(\theta)$$
$$Y'' = Y' * COS(\theta) + X' * SIN(\theta)$$

X``1=0*COS(90)-0*SIN(90)=0	Y``1=0*COS(90)-0*SIN(90)=0	(0 , 0)
X`` 2=-3*(0)-(-4)*(1)=4	Y``2=4*(0)+(-3)*(1)=-3	(4 , -3)
X`` 3=3*(0)+3*(1)=3	Y`` =-3*(0)+3*(1)=3	(3 , 3)

Rotation

- **Step 3: *Translate* :**

$$X''' = X'' + Xp$$
$$Y''' = Y'' + Yp$$

X```1=0+7=7	Y```1=0+8=8	(7 , 8)
X```2=4+7=11	Y```2=-3+8=5	(11 , 5)
X```3=3+7=10	Y```3=3+8=11	(10 , 11)

Rotation

- **H . w :**

1 . Rotate the triangle (10 , 0) , (30 , 0) , (50 , 80) 45o counterclockwise about the origin.

2 . Rotate the triangle (7 , 8) , (4 , 4) , (10 , 5) 90o counter clockwise about the point (7 , 8).

Shearing

- ***4:- Shearing***
- A **shearing** transformation produces a distortion of an object or the entire image. There are two types of shearing (Y-shear & X-shear).
- A Y-shear transforms the point (x,y) to the point (x,y) , where :-

$$X' = X$$
$$Y' = Shy * X + Y$$

Shy # 0

- A Y-shear moves a vertical line up or down, depending on the sign of the shear factor Shy. A horizontal line is distorted into a slanted line with slop Shy.

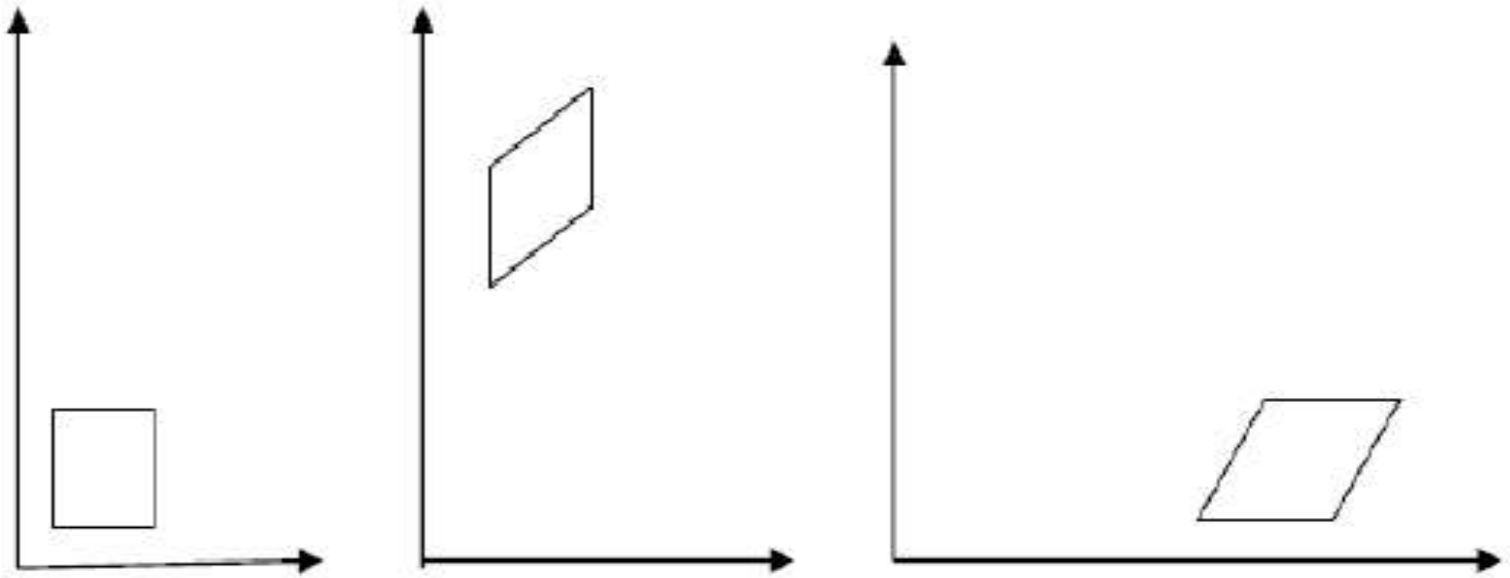
- A X-shear has the opposite effect. That is, the point (x,y) is transformed to the point (x,y) , where :-

$$X' = X + Shx * Y$$
$$Y' = Y$$

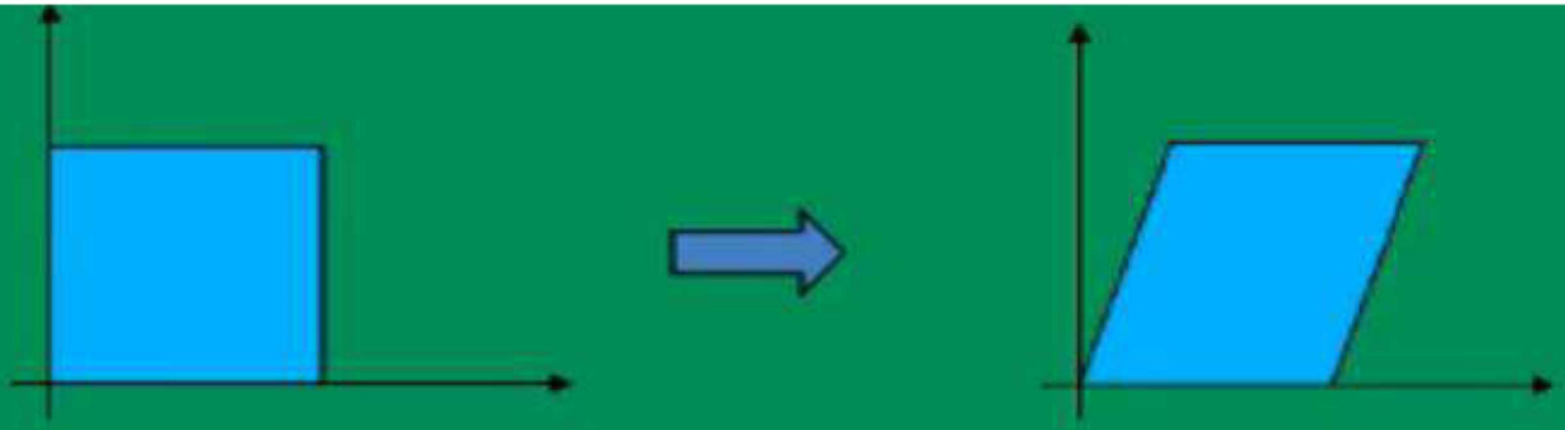
Shx # 0

A vertical line becomes slanted line with slop Shx and the horizontal lines are shifted to the right or left, depending on the sign of Shx.

Shearing



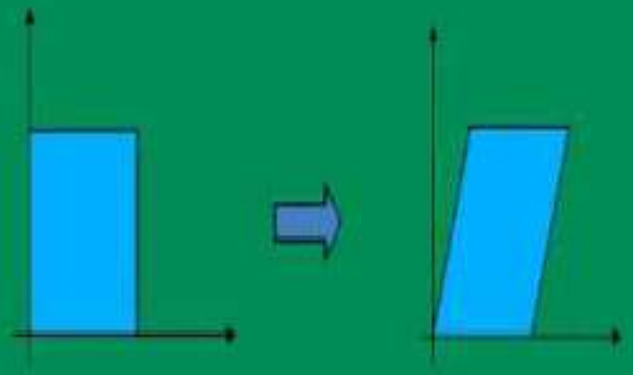
Shearing



- Y coordinates are unaffected, but x coordinates are translated linearly with y
- That is:
  - $y' = y$
  - $x' = x + y * h$

$$[x',y',1]=[x,y,1]*\begin{vmatrix}1 & Sh_y & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{vmatrix}$$

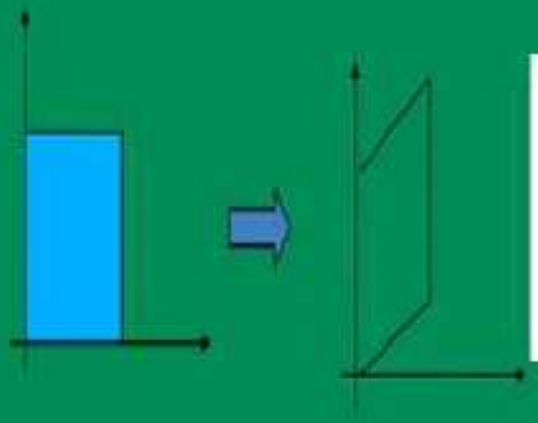
Shearing



- Y coordinates are unaffected, but x coordinates are translated linearly with y
- That is:
  - $y' = y$
  - $x' = x + y * h$

$$[x', y', 1] = [x, y, 1] * \begin{vmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Shearing in Y



$$[x', y', 1] = [x, y, 1] * \begin{vmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Shearing

Matrix Representation of Shearing

1:- Y-shear

$$[\,x'\ y'\,] = [\,x \quad y\,]\begin{bmatrix} 1 & shy \\ 0 & 1 \end{bmatrix}$$

2:- X-shear

$$[\,x'\ y'\,] = [\,x \quad y\,]\begin{bmatrix} 1 & 0 \\ shx & 1 \end{bmatrix}$$

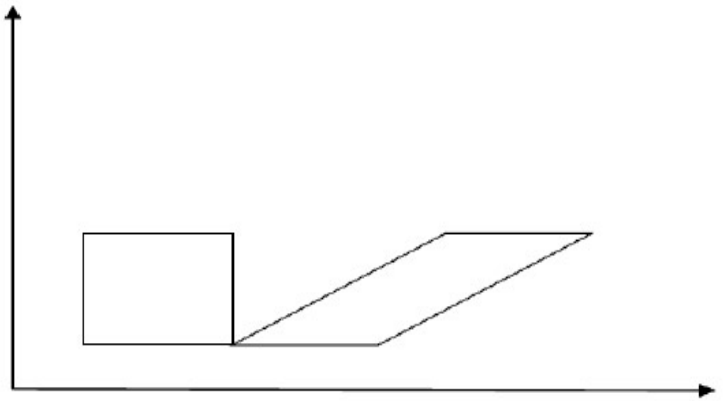
Shearing

- **Example :**Shear the object (1,1), (3,1),(1,3), (3,3) with
- a: shx = 2
- b: shy =2

solution:

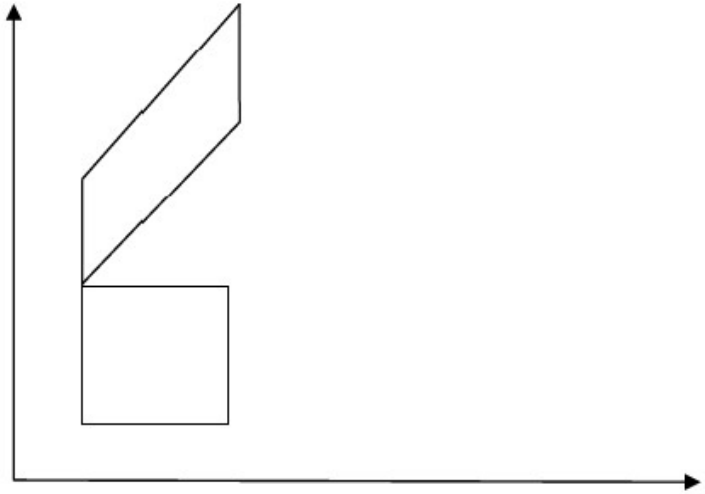
a: shx

$$\begin{bmatrix}1&1&1\\3&1&1\\1&3&1\\3&3&1\end{bmatrix} \times \begin{bmatrix}1&0&0\\2&1&0\\0&0&1\end{bmatrix} = \begin{bmatrix}3&1&1\\5&1&1\\7&3&1\\9&3&1\end{bmatrix}$$

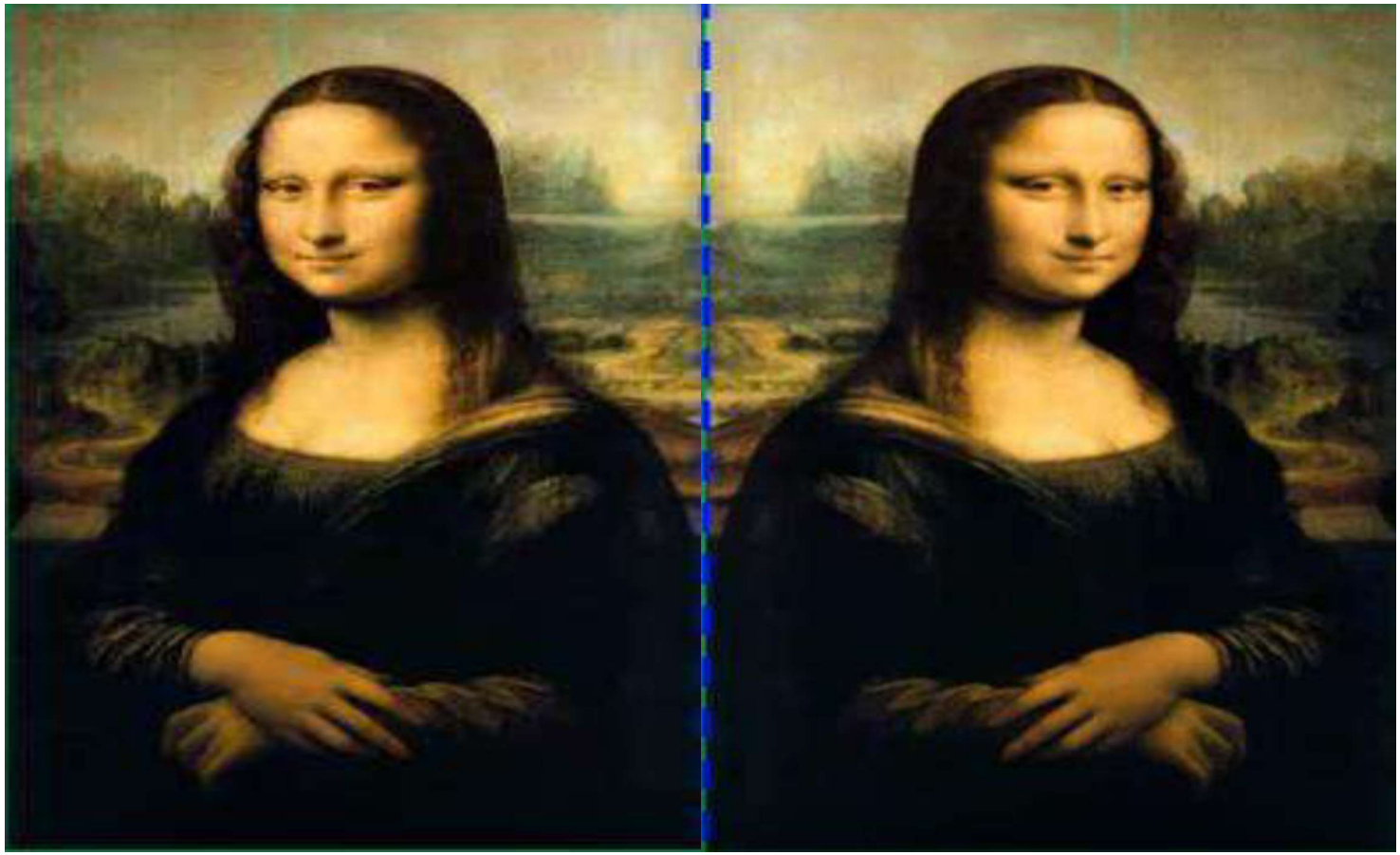


b: shy

$$\begin{bmatrix}1&1&1\\3&1&1\\1&3&1\\3&3&1\end{bmatrix} \times \begin{bmatrix}1&2&0\\0&1&0\\0&0&1\end{bmatrix} = \begin{bmatrix}1&3&1\\3&7&1\\1&5&1\\3&9&1\end{bmatrix}$$

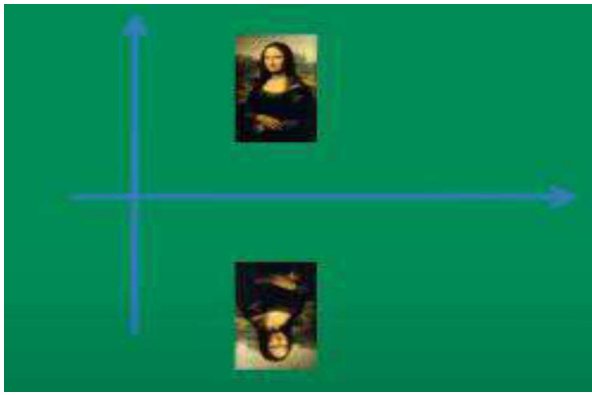


Reflection



Reflection

- It is a transformation that produced a mirror image of an object; the mirror image
- is generated relative to an axis of reflection.
- There are different types of reflection:



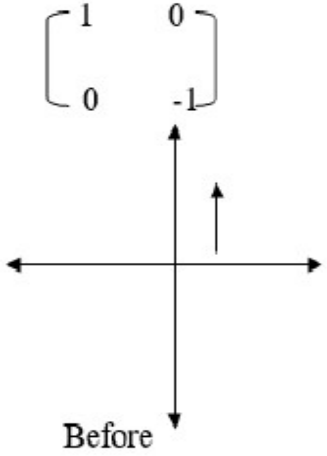
The following are just scales with negative scale factor: -

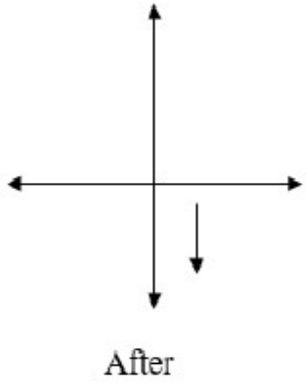
1 - Reflection about X – axis:

x`=x

y`=-y

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$





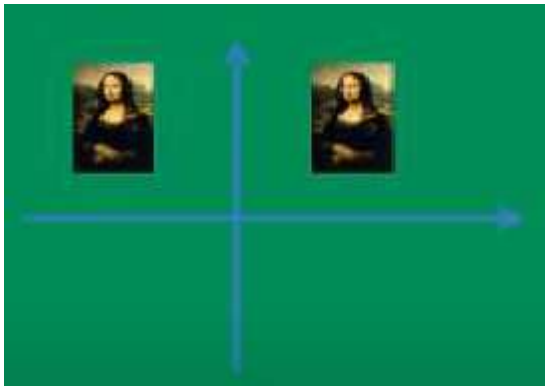
Reflection

2 - Reflection about Y – axis:

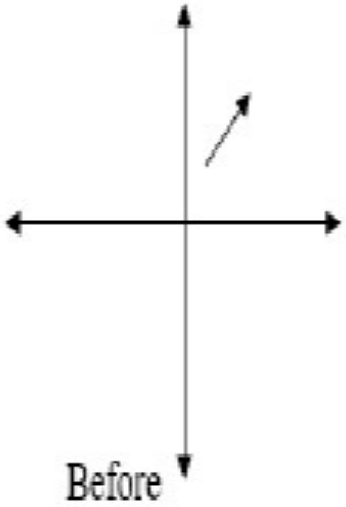
x`=-x

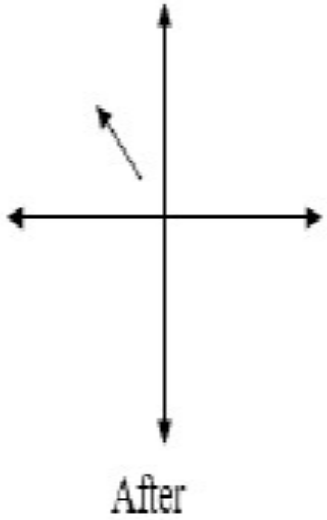
y`=y

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



$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$





Reflection

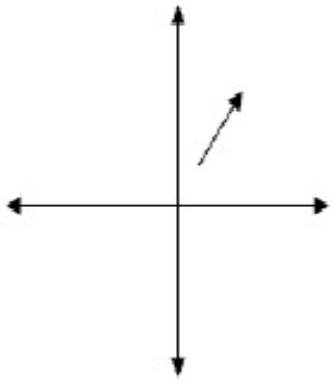
3 - Reflection about the origin (0, 0):

x=-x

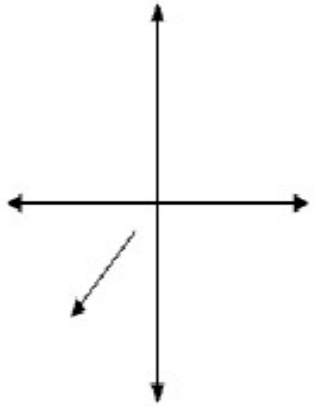
y=-y

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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



Before



After

Reflection

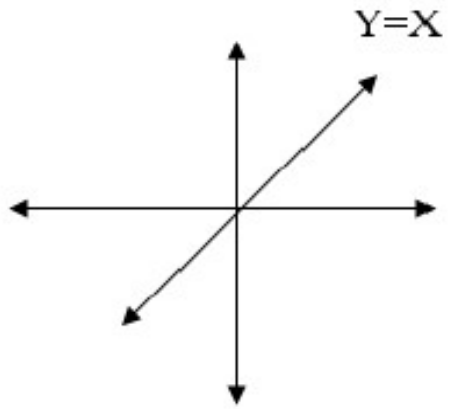
4 - Reflection about the line y = x :

x=y

y=x

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

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Reflection

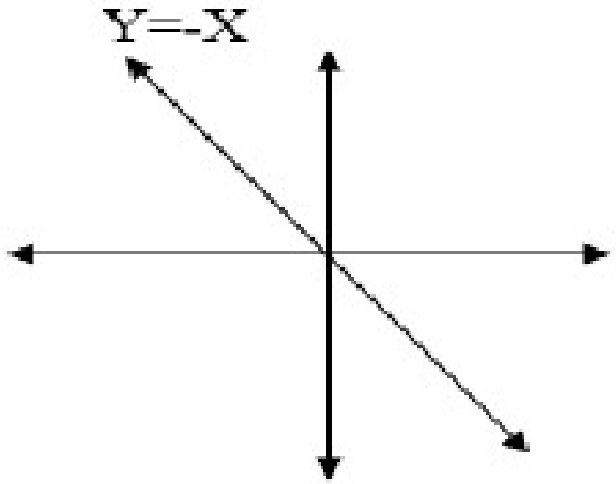
5 - Reflection about the line y = - x :

x=-y

y=-x

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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



Reflection

- **Example :**

Reflect the shape (20, 70), (40, 50), (60, 70), (40, 90), about:

1- X – axis , 2- Y- axis

3- origin (0,0) , 4- y = x , 5- y = -x , by used matrix representation, and draw the result.

1- X – axis:

x`=x

y`=-y

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 20 & -70 & 1 \\ 40 & -50 & 1 \\ 60 & -70 & 1 \\ 40 & -90 & 1 \end{bmatrix}$$

Reflection

2- Y- axis:

x`=-x

y`= y

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -20 & 70 & 1 \\ -40 & 50 & 1 \\ -60 & 70 & 1 \\ -40 & 90 & 1 \end{bmatrix}$$

3- origin (0,0):

x`=-x

y`=-y

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -20 & -70 & 1 \\ -40 & -50 & 1 \\ -60 & -70 & 1 \\ -40 & -90 & 1 \end{bmatrix}$$

Reflection

4- y = x:

x`=y

y`=x

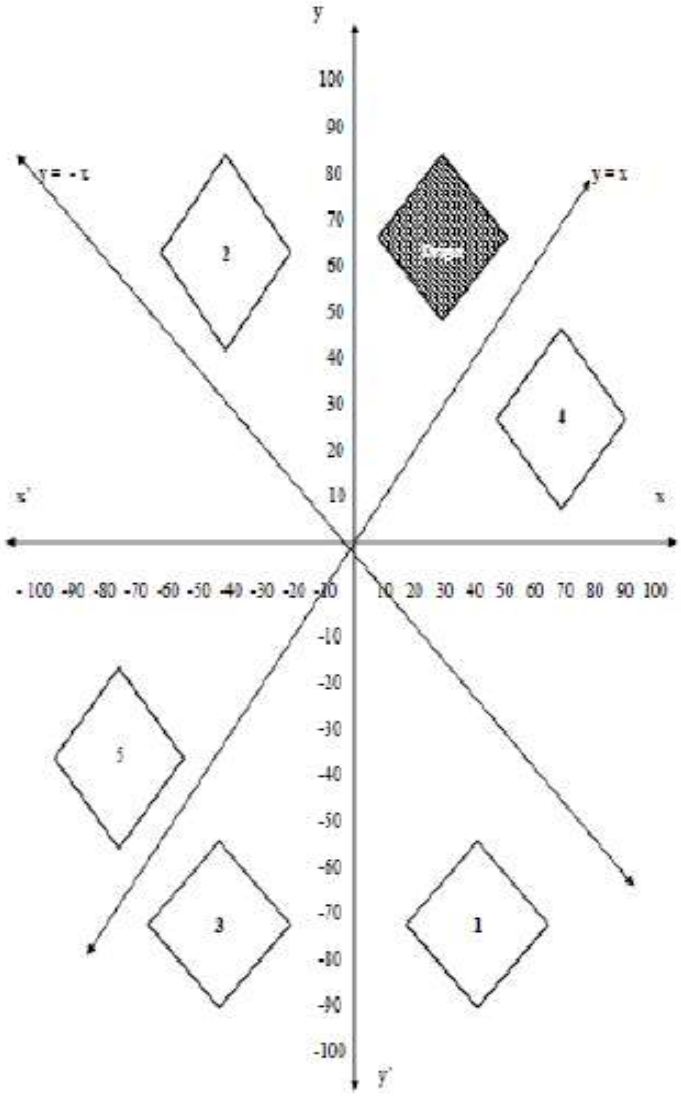
$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 70 & 20 & 1 \\ 50 & 40 & 1 \\ 70 & 60 & 1 \\ 90 & 40 & 1 \end{bmatrix}$$

5- y = -x:

x=-y

y=-x

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -70 & -20 & 1 \\ -50 & -40 & 1 \\ -70 & -60 & 1 \\ -90 & -40 & 1 \end{bmatrix}$$



2 * 2 Matrix representation of transformation

Transformation can be represented as a product of the row vector [x y] and a 2*2 matrix accept for the translation we can represent transformation as a product of 1*2 row vector and an appropriate 2*2 matrix .

Translation

If a point P1=(X1,Y1) is being a 1*2 vector . if we added it to some 1*2 vector T=[H V] , we will obtain another 1*2 matrix which we can interpret as another point :-

[X2 Y2] = P2 = P1 + T

Then X2 = X1 + H & Y2 = Y1 + V

Scaling

general 2*2 matrix

$$S = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix}$$

If T is the identity matrix $T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ NO Change

Ex :-

Stretch the image to three times width and two times height?

P2 = P1 T

$$T = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

Rotation

Rotation transformation matrix

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

EX:- rotate the point (2,3) counter clockwise by an angle (π/6) radios.

Solu:-

The rotation matrix is

$$R=\begin{bmatrix}\cos(\pi/6) & \sin(\pi/6)\\ -\sin(\pi/6) & \cos(\pi/6)\end{bmatrix}=\begin{bmatrix}0.866 & 0.5\\ -0.5 & 0.866\end{bmatrix}$$

Then the rotated point would be:-

$$[2\ \ 3]\begin{bmatrix}0.866 & 0.5\\ -0.5 & 0.866\end{bmatrix}=[0.232\quad 3.398\ ]$$

For the rotation matrix for an angle (θ) clockwise would be

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

Using matrices allow us to combine translation with other transformation by simple matrix multiplication, for example, rotating about a point other than the origin can be done by translation, rotation, translation. To do this we will use homogeneous coordinate, we use 3*3 matrixes instead of 2*2 introducing an additional dummy coordinate.

Matrix Representation of Shearing

1:- Y-shear

$$[\,x'\ y'\,]=[\,x\quad y\,]\begin{bmatrix}1 & shy\\ 0 & 1\end{bmatrix}$$

2:- X-shear

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

$[x'\ y'\ ] = [\ x \quad y\ ] \quad \begin{matrix} shx & 1 \end{matrix}$

*In general 3 * 3 Matrix representation of transformation*

1 :- Translation

$$[x'\ y'\ 1] = [\ x \quad y \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{bmatrix}$$

2 :- Scaling

$$[x'\ y'\ 1] = [\ x \quad y \quad 1] \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3 :- Rotation

$$[x'\ y'\ 1] = [\ x \quad y \quad 1] \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4 :- Shearing

a:- Y-shear

$$[x'\ y'\ 1] = [\ x \quad y \quad 1\ ] \begin{bmatrix} 1 & shy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

b:- X-shear

$$[x'\ y'\ 1] = [\ x \quad y \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ shx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5 :- Reflection

a:- Y-axis

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

b:- X-axis

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

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

c:- Origin

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

Rotation about an arbitrary point

The transformation matrix for counter clockwise rotation about Point (Xc,Yc)

The three transformation steps are:-

1:- Translation to the origin

$$T1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{bmatrix}$$

2:-Rotation

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

3:- Translation back to (Xp,Yp)

$$T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xc & Yc & 1 \end{bmatrix}$$

To transform a point

$[x'\ y'\ 1] = ((([x\ \ y\ \ \ 1]\ T1\ )\ R\ )\ T2)$

We must form an overall transformation matrix

$[x'\ y'\ 1] = [x\ \ y\ \ 1](T1\ \ (R\ \ \ T2))$

$$T1\ \ R\ \ T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xc & Yc & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ Xc & Yc & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ -Xc*\cos(\theta)+Yc*\sin(\theta)+Xc & -Xc*\sin(\theta)-Yc*\cos(\theta)+Yc & 1 \end{bmatrix}$$

Ex:- find the 3 * 3 matrix that will rotate an object by 90 degree about the point (15,15) ?

$$T1\ \ R\ \ T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xc & Yc & 1 \end{bmatrix}$$

$$T1\ \ R\ \ T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -15 & -15 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 15 & 15 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -15 & -15 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 15 & 15 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 30 & 0 & 1 \end{bmatrix}$$

Scaling about an arbitrary point

The transformation matrix for counter clockwise rotation about Point (Xc,Yc)

The three transformation steps are:-

1:- Translation to the origin

$$T1= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{bmatrix}$$

2:-Scaling

$$R= \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3:- Translation back to (Xp,Yp)

$$T2= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xc & Yc & 1 \end{bmatrix}$$

To transform a point

$[x'\ y'\ 1] = ((([x\ \ y\ \ \ 1]\ T1\ )\ S\ )\ T2)$

We must form an overall transformation matrix

$[x'\ y'\ 1] = [x\ \ y\ \ 1](T1\ \ (S\ \ \ T2))$

$$T1\ \ R\ \ T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xc & Yc & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ Xc & Yc & 1 \end{bmatrix}$$

$$= \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ -Xc*Sx+Yc & -Yc*Sy+Yc & 1 \end{bmatrix}$$

Ex:- find the 3 * 3 matrix that will scaled an object twice in X&Y directions about the point (15,15) ?

$$T1\ R\ T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xc & -Yc & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xc & Yc & 1 \end{bmatrix}$$

$$T1\ R\ T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -15 & -15 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 15 & 15 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ -30 & -30 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 15 & 15 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ -15 & -15 & 1 \end{bmatrix}$$

Transformation procedures

```
Type
  Matrix = array[1..3,1..2] of real;
```

Create the identity matrix as transformation matrix.

```
Procedure   Identity (Var H : Matrix);
Var
  I,J : Integer;
Begin
   For I := 1 to 3 Do
        For J := 1 to 2 Do
             IF (I=J) THEN H[I,J] := 1
                      ELSE H[I,J] := 0;
End;
```

Scaling

$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \Leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

```
Procedure   Scale (Sx,Sy : Real ;   Var H : Matrix);
Var
  I : Integer;
Begin
   For I := 1 to 3 Do
        Begin
            H[I,1] := H[I,1] * Sx;
            H[I,2] := H[I,2] * Sy;
        End;
End;
```

Translation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{bmatrix} \Leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

```
Procedure   Translate (Tx,Ty : Integer ;   Var H : Matrix);
 Begin
     H[3,1] := H[3,1] + Tx;
     H[3,2] := H[3,2] + Ty;
 End;
```

Rotation

$$\begin{bmatrix} Cos(A) & Sin(A) & 0 \\ -Sin(A) & Cos(A) & 0 \\ 0 & 0 & 1 \end{bmatrix} \Leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

```
Procedure   Rotate (A : Real ;   Var H : Matrix);
Var
  Temp , S ,C : Real ;
  I  : Integer;
Begin
   C := COS (A) ; S := SIN (A) ;
   For I := 1 to 3 Do
        Begin
            Temp := H[I,1] * C – H[I,2] * S;
            H[I,2] := H[I,1] * S + H[I,2] * C;
            H[I,1] := Temp;
        End;
End;
```

The following procedure transform a single point, the point coordinates are passed to the procedure as argument and the transformed point is returned in the same variable.

```
Procedure   Do_Trans ( H : Matrix ; Var X,Y : Real);
Var
  Temp: Real;
Begin
   Temp := X * H[1,1] + Y * H[2,1] + H[3,1];
   Y      := X * H[1,2] + Y * H[2,2] + H[3,2];
   X      := Temp ;
End;
```

Two – Dimensional Viewing Transaction and Clipping Windows and View Ports

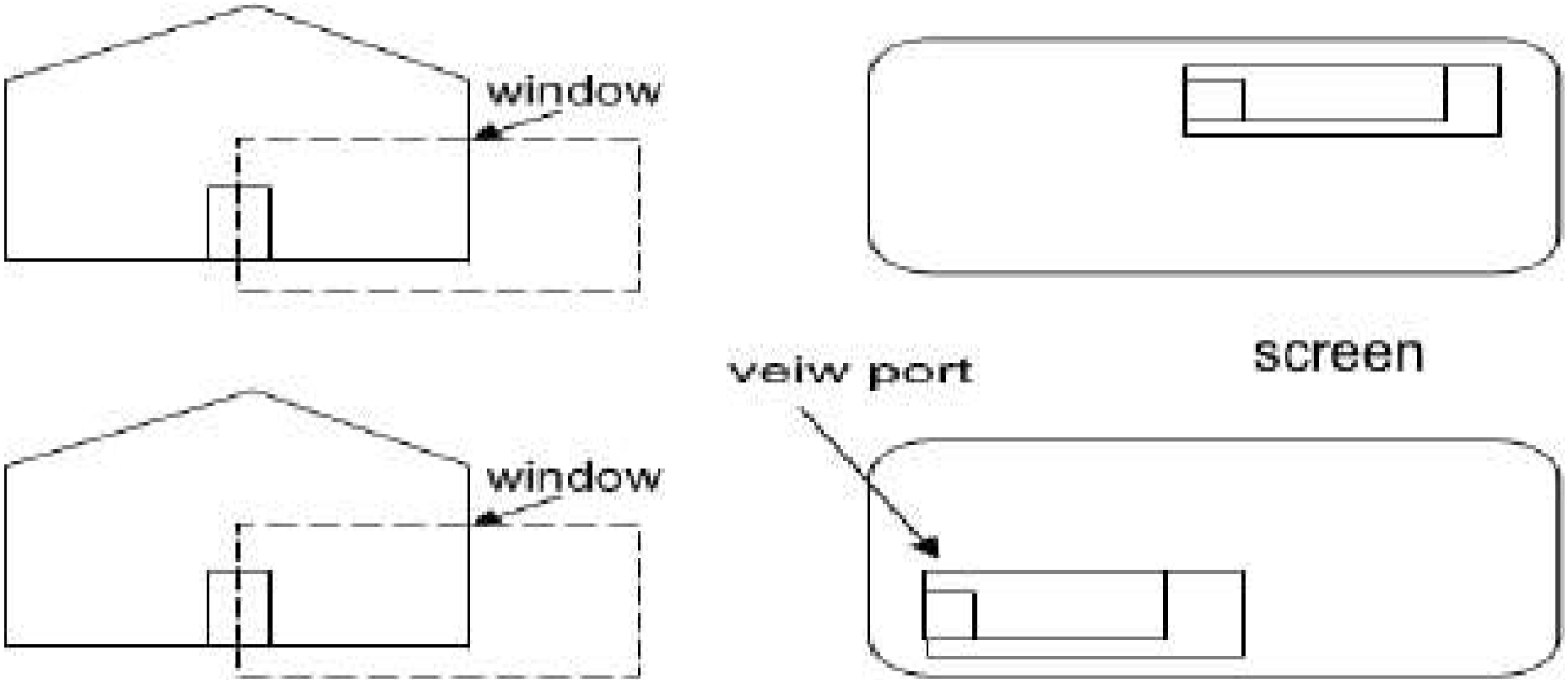
viewing Transaction and clipping
Windows and view ports

- The method for selecting and enlarging portions of a drawing enclosed in a rectangular region is called ***Widowing*****.** The rectangular region is called a ***window*****.**
- The technique for not showing that part of the drawing which one is not interested in is called ***clipping***.
- **Clipping** is a process which divided each element of the picture into its visible and invisible portions allowing the invisible portion to be discarded.
- If we imagine a box about a portion of the object so we could display what is enclosed in the box such a box called a ***Window***.

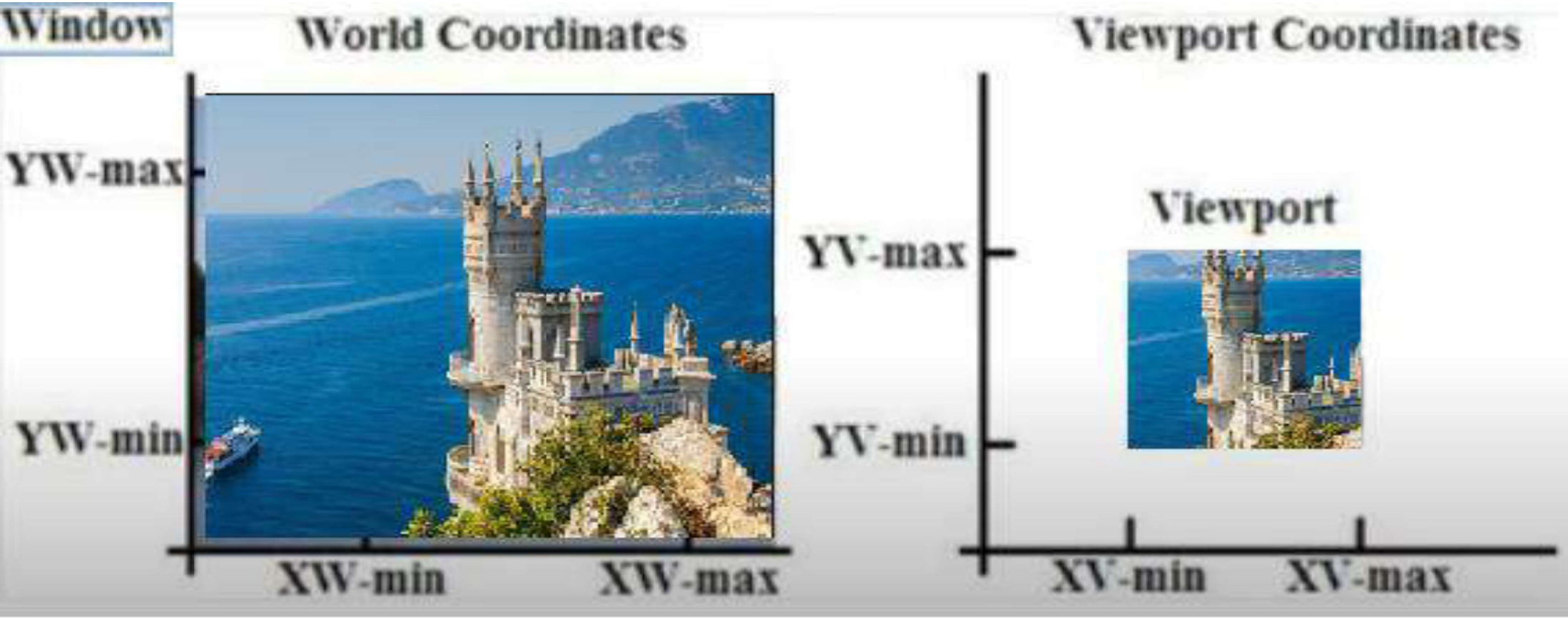
viewing Transaction and clipping
Windows and view ports

- If we do not wish to use the entire screen for display, we can imagine a box on the screen and have the image confined to that box such a box in the screen space is called a ***view port***.
- When the window is changed , a different part of the object at the same position on the display.
- If we change the view port, we see the same part of the object drawn at a different place on the display.

viewing Transaction and clipping
Windows and view ports



viewing Transaction and clipping
Windows and view ports



Viewing Transformation

★ Mapping from object space to image space.

1. - Change the window size to become the size of the view port (Scaling).
2. - Position the window at the desired location on the screen (Translate) by moving the Lower-left corner of the window to the view port Lower-left corner location.

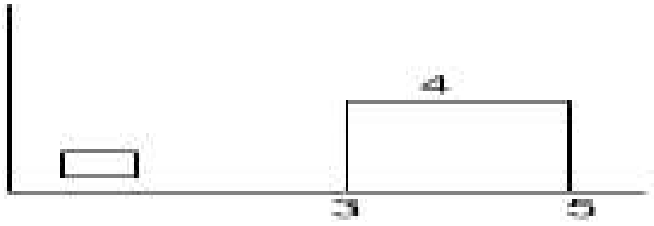
To do this we need 3 steps:

First :- Translate the lower-left corner of the window at the origin .

Second :- The object and the window are scaled until the window has the dimensions of the view port.

Third :- Move the view port to its correct position on the screen .

Ex/ A window has left and right boundaries of 3 of 5 and lower and upper boundaries of 0 and 4. The view port is the upper-right quadrant of the screen with boundaries at 0.5 and 1.0 for both X and Y direction.



The first translation matrix would be

The length of the window is

X1=3;X2=5

Y1=0;Y2=4

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{bmatrix} \quad \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{vmatrix}$$

To origon

5 – 3 = 2 in the X direction

4 – 0 = 4 in the Y direction

The length of the view port is

1.0 – 0.5 = 0.5 in the X & Y direction

The X scale factor is 0.5/2 = 0.25

The Y scale factor is 0.5/4 = 0.125

Ex/

The scaling transformation matrix is

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{vmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Finally to position the view port requires a translation of:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{bmatrix} \quad \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{vmatrix}$$

The viewing transformation is then:

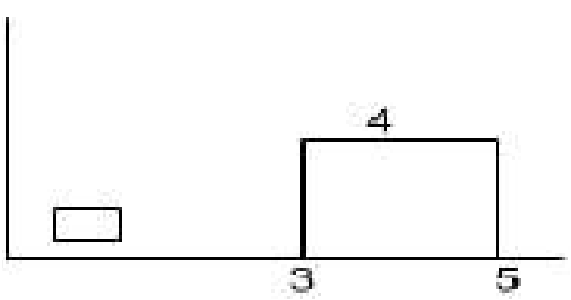
$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{vmatrix} \begin{vmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{vmatrix} = \begin{vmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ -0.25 & 0.5 & 1 \end{vmatrix}$$

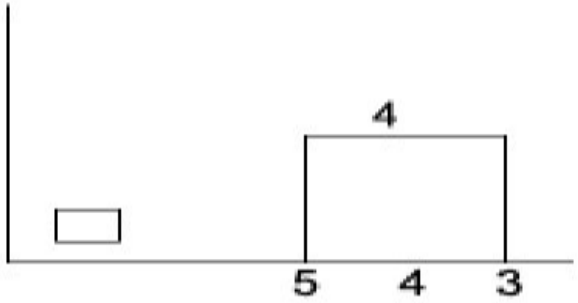
In general the viewing transformation is

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -WXL & -WYL & 1 \end{vmatrix} \begin{vmatrix} \frac{(VXH - VXL)}{(WXH - WXL)} & 0 & 0 \\ 0 & \frac{(VYH - VYL)}{(WYH - WYL)} & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ VXL & VYL & 1 \end{vmatrix}$$

$$\begin{vmatrix} \frac{(VXH - VXL)}{(WXH - WXL)} & 0 & 0 \\ 0 & \frac{(VYH - VYL)}{(WYH - WYL)} & 0 \\ VXL - WXL\frac{(VXH - VXL)}{(WXH - WXL)} & VYL - WYL\frac{(VYH - VYL)}{(WYH - WYL)} & 1 \end{vmatrix}$$

EX/





- V: View port
- W: Window
- X: Position of a vertical boundary
- Y: Horizontal boundary
- H: High boundary
- L: Low boundary

Introduce for Window and Viewport

A windows is specified by four world coordinate: **WXmin, WXmax, WYmin, and WYmax**. Similarly, a viewport is described by four normalized device coordinates: ***VXmin, VXmax, VYmin, and VYmax***. The objective of window-to-veiwport mapping is to convert the world coordinates **(WX, WY)** of an arbitrary point to its corresponding normalized device coordinates **(VX, VY)**. In order to maintain the same relative placement of the point in the viewport as in the window, we require:

Introduce for Window and Viewport

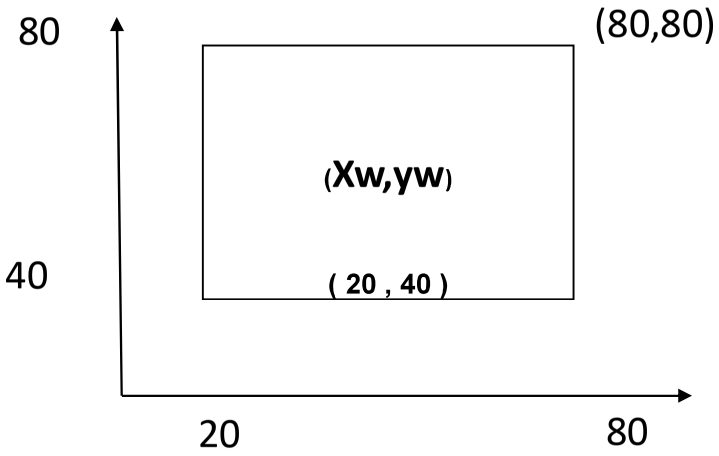
$$\frac{WX - WXmin}{WXmax - WXmin} = \frac{VX - VXmin}{VXmax - VXmin} \quad and$$

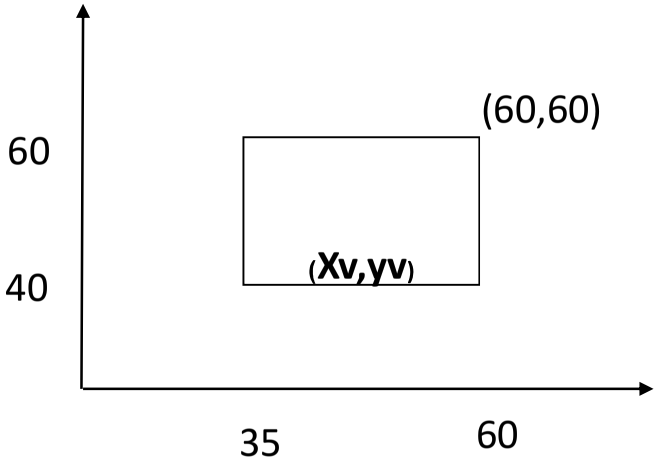
$$\frac{Wy - Wymin}{Wymax - Wymin} = \frac{Vy - Vymin}{Vymax - Vymin} \quad \text{thus}$$

- $VX = VXmin + (WX - WXmin) * Sx$
- $VY = VYmin + (WY - WYmin) * Sy$ where

$$Sx = \frac{VXmax - VXmin}{WXmax - WXmin} \quad and \quad sy = \frac{Vymax - Vymin}{Wymax - Wymin}$$

Example:-Given a window with lower left corner of(20,40)and upper right corner of (80,80)and a view port with lower left corner of(30,40)and upper right corner of (60,60)and (Xw,Yw)=(30,60) Find (Xv,Yv)?





XV=30+(30-20)*$\frac{60-30}{80-20} = 35$

YV=40+(60−40)$*\frac{60-40}{80-40} = 50$

XV,YV=(35,50)

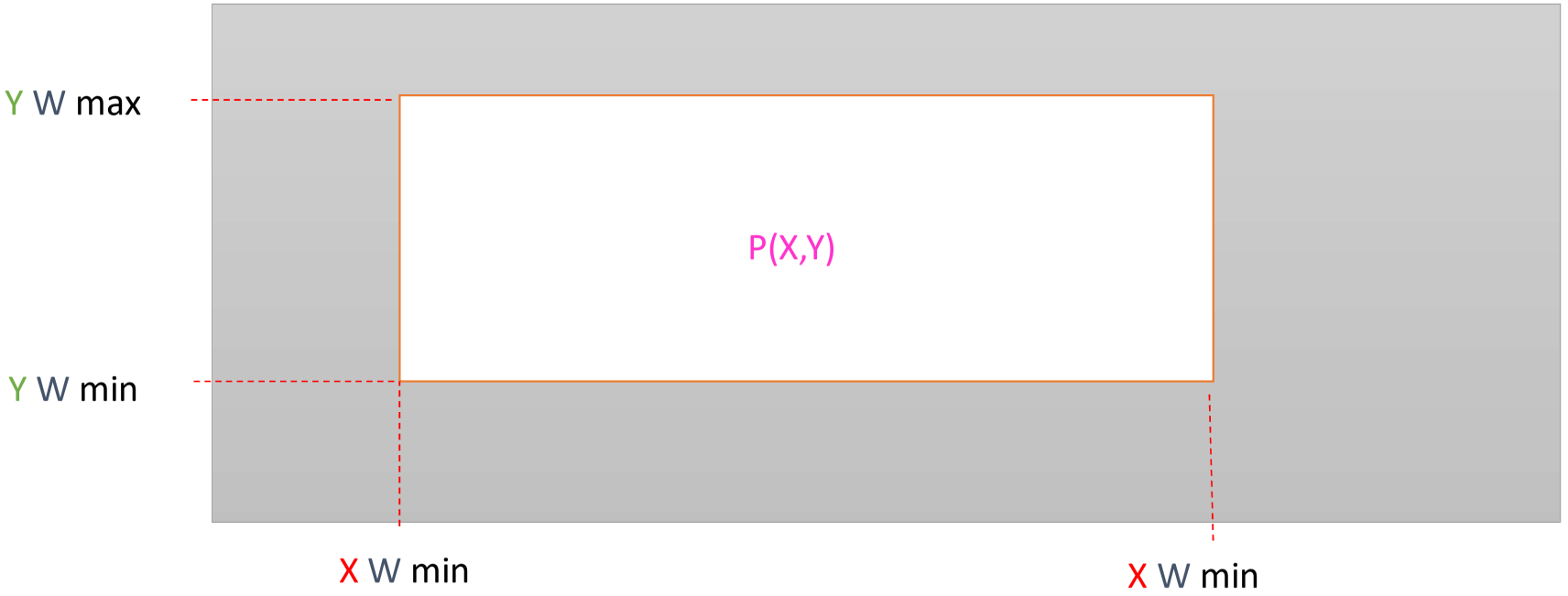
H.W:-

find the normalization transformation that maps a window whose lower left corner is at (1,3) and upper right corner is at (3,5) onto a viewport that has lower left corner at (0.2,0.5) and upper right corner (0.8,0.9) . what position of point p in window is (2.5,3.5) onto viewport coordinate?

Clipping Windows

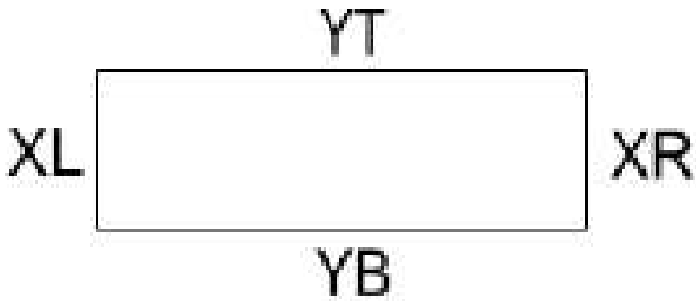
CLIPPING

- If we wish to display only a portion of the total picture, we use a window to select that portion of the picture which is to be viewed (like clipping or cutting out a picture from a magazine). This is known as clipping.
- The process of clipping determines which elements of the picture lie inside the window and so are visible.

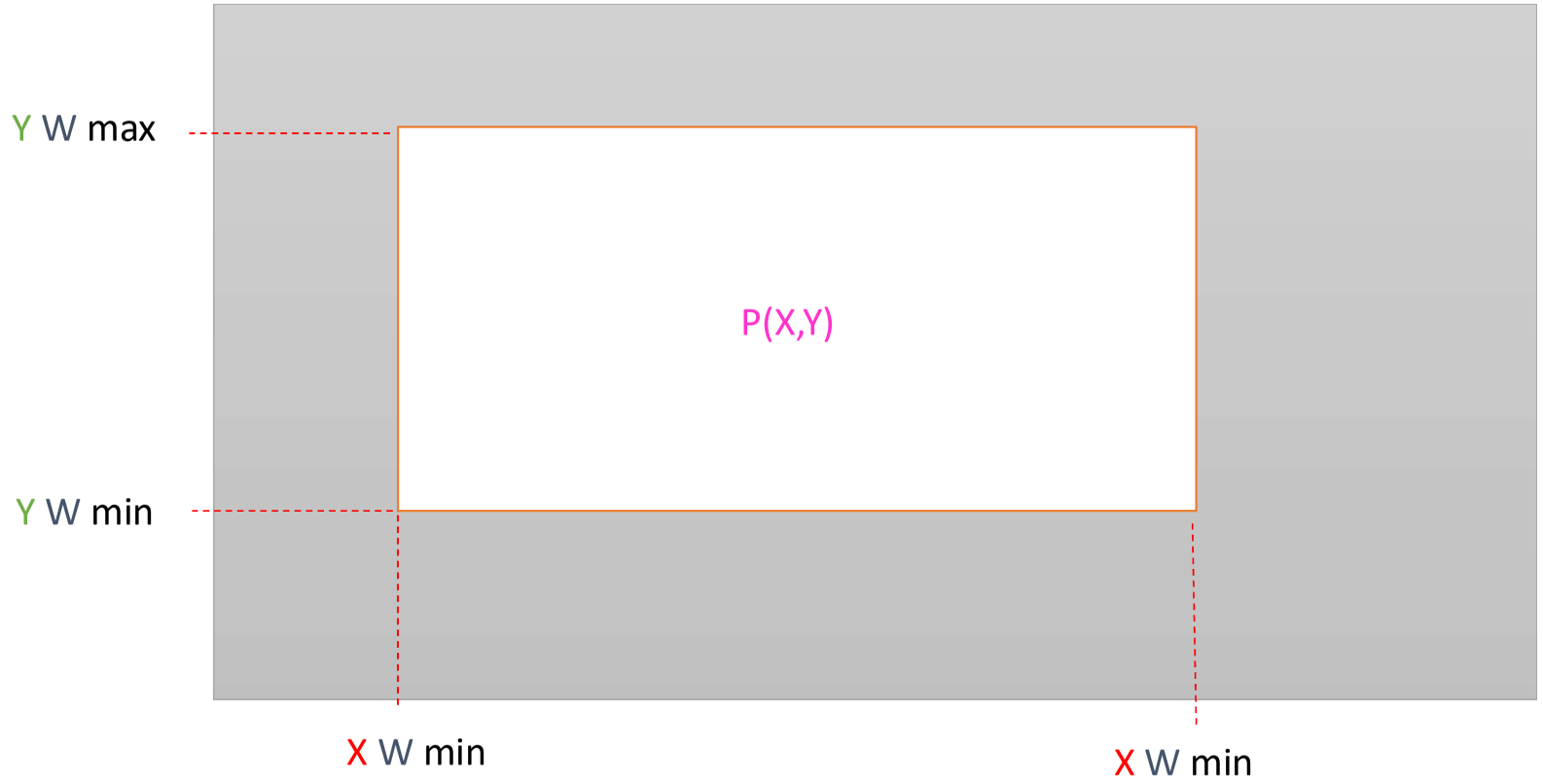


RECTANGULAR CLIPPING WINDOWS

The clipping window assumes to be rectangles whose sides are aligned with the coordinate area. The X extent is measured from X min to X max and the Y extent is measured from Y min to Y max.



Inside (x>=xw min)&&(x<=xw max)&&(y>=yw min)&&(y,=yw max)



There are three types of clipping

1 – Point 2 – Line 3 – Polygon

TBRL

Top

Bottom

Right

Left

Top

TBRL 1001	TBRL 1000	TBRL 1010
TBRL 0001	TBRL 0000	TBRL 0010
TBRL 0101	TBRL 0100	TBRL 0110

Left

Right

Bottom

1:- POINT CLIPPING

A point p(x,y) is inside the rectangular window (visible) if all the following inequalities are true.

$$X_{min} \leq X \leq X_{max} \qquad Y_{min} \leq Y \leq Y_{max}$$

- If any of these inequalities is false point P is outside the window and is not displayed (invisible).

Point Clipping algorithm

P(x , y) , نقطة إحداثياتهاXl , Xr , Yt , Yb معلومة

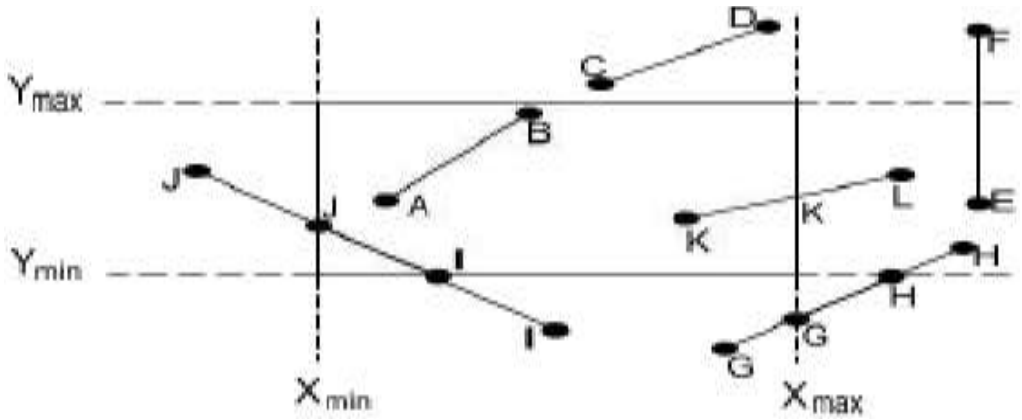
```
Procedure      out code ( p ; VAR  OP: integer ) ;
Begin
```

- Op := 0 ;
- If P.x < XL then op : = op OR 1 ;
- If P.x > XR then op : = op OR 2 ;
- If P.y < YB then op : = op OR 4 ;
- If P.y > YT then op : = op OR 8 ;

```
End;
```

2:-Line Clipping

First	bit	on	if	$x < XL$	to	the	left	of window.
Second	bit	on	if	$x > XR$	to	the	right	of window.
Third	bit	on	if	$y < YB$	to	the	below	of window.
Fourth	bit	on	if	$y > YT$	to	the	above	of window.



LINE SEGMENT CLIPPING:

1 – Visible:

Both endpoints of the line segment lie within the window (Line AB).

2 – Not visible:

The line segment definitely lies outside the window (Line CD and EF).

This will occur if the line segment from (X_1, Y_1) to (X_2, Y_2) satisfies any one of the following four in qualities.

$X_1, X_2 > X_{max}$ $Y_1, Y_2 > Y_{max}$ $X_1, X_2 < X_{min}$ $Y_1, Y_2 < Y_{min}$

3 – Clipping candidate:

The line is in neither category 1 nor 2 (Line GH, IJ, and KL)

LINE INTERSECTIONS AND CLIPPING:

We determine the intersection points of the lines in category (3) with the boundaries of the window. The intersection points subdivided the line segment into several smaller line segments which can belong only to category 1 (visible) or category 2 (not visible). The segment in category 1 will be the clipped line segment.

INTERSECTION POINT:

- If M is the slope of the line segment between points (X_1, Y_1) and (X_2, Y_2) then if $X_1 \neq X_2$

M = (Y2 – Y1) / (X2 – X1) then any point on the line is ***M = (Y – Y1) / (X – X1)***

★ If the line segment crosses a left or right window edge then $X_1 \neq X_2$ and M has non denominator.

★ If the line crosses a top or bottom widow edge then $Y_1 \neq Y_2$ and the reciprocal of the slop 1/m has a nonzero denominator.

The slop M is obtained from the two given end points.

The slop M is obtained from the two given endpoints

★ If we are testing against a left or right direction the X value is known (the left or right edge value).

Then the X value is substituted into the equation for Y.

$$Y = M * (X - X_1) + Y_1$$

★ If we are testing against a Top or Bottom direction the Y value is known (the Top or Bottom edge value).

Then the Y value is substituted into the equation for X.

$$X = 1/M * (Y - Y_1) + X_1$$

Simple visibility algorithm.

Check for totally visible lines.

- If ((xb < XL) OR (xb > XR)) then 1 .
- If ((xe < XL) OR (xe > XR)) then 1 .
- If ((yb < YB) OR (yb > YT)) then 1 .
- If ((ye < YB) OR (ye > YT)) then 1 .

Draw line

Go to 3

1 - Check for totally invisible lines

- if ((xb < XL) AND (xe < XL)) then 2
- if ((xb > XR) AND (xe > XR)) then 2
- if ((yb < YB) AND (ye < YB)) then 2
- if ((yb > YT) AND (ye > YT)) then 2

The line is partially visibly or diagonally crosses the corner; determine the intersections go to 3

2 lines is invisible

3 next lines

Cohen clipping Algorithm

{w (4) = (xl , xr ,yb , yt) } , {p(1)=p(x), p(2) = p(y)} ,{pc=0000}

Subroutine end point (p , w , pc , s)

If p (1) < w (1) then pc (4) = 1 else pc (4) = 0

If p (1) > w (2) then pc (3) = 1 else pc (3) = 0

If p (2) < w (3) then pc (2) = 1 else pc (2) = 0

If p (2) > w (4) then pc (1) = 1 else pc (1) = 0

S = 0

For I = 1 to 4 s = s + pc (i) Next i

Return

Subroutine logical (pc1 ,pc2 , inter)

Inter = 0

For I = 1 to 4 Inter = inter = pc1 (i) * pc2(i) Next I

Return

Subroutine Cohen (p1 , p2 , w , visible)

Call end point (p1 , w . pc . s1)

Call end point (p2 , w .pc2 , s2)

If S1 = 0 & S2 = 0 then visible = yes

Call logical (pc1 , pc2 , inter)

If inter < > 0 then visible = NO

else visible = partial

If S1 = 0 then 1

else p1 p2 end if

1 Return

THE ALGORITHM

- FLAG = 0
- IF P2 (1) – P1 (1) = 0
- THEN FLAG = - 1
- ELSE slop = (p2 (2) – p1 (2)) /(p2 (1) – (p1(1))
- End if
- For I = 1 to 4
- Call cohen (p1 , p2 . w1 visible)
- If visible = yes then 2
- If visible = No then 3
- If flag = -1 and i ≤ 2 then 1
- If i ≤ 2 then
- Inter = slop * (w (i) – p1 (1)) $_{+}$ p1 (2)
- P2 (1) = w (I)
- P2 (2) = inter
- else if flag = -1 then
- p2 (2) = w (i)

```
 else inter = (1 /slop ) * ( w ( i ) – p1 ( 2) ) + p1 ( 1 )
   p2 ( 1 ) = inter
        p2 ( 2 ) = w ( i )
                end  if
         end if
1 next  i
2 draw  p1 p2
3 end
```

MIDPOINT SUBDIVISION

- The line segment is divided at its midpoint into two smaller line segments. The clipping categories of the two new line segments are then determined. Each segment in category 3 is divided again into smaller segment and categorized. The bisection and categorization process continues until all segments are in category 1 (visible) or category 2 (invisible).
- The midpoint coordination (X_m , Y_m) of a line segment joining

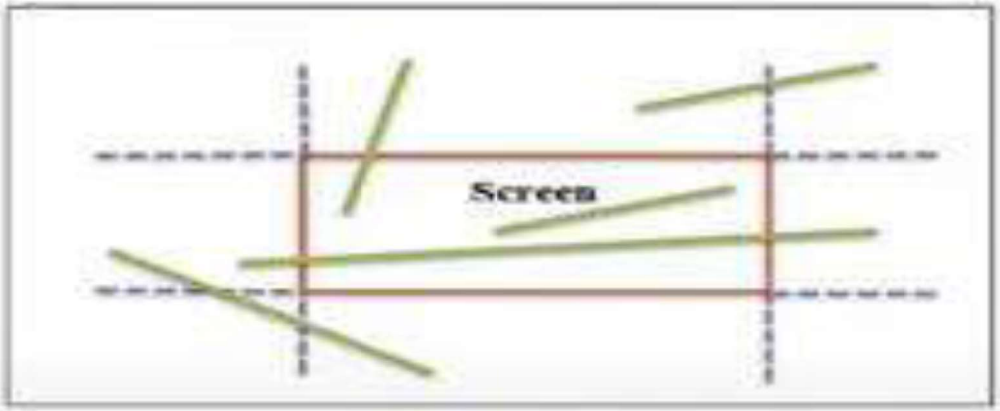
  P_1 (X_1 , Y_1) to P_2 (X_2 , Y_2) are given by

  Xm = (X1 + X2) / 2 , Ym = (Y1 + Y2) / 2

clipping

Line Clipping

It would be quite inappropriate to clip pictures by converting all picture elements into points and using point clipping, the clipping process would take far too long. We must instead attempt to clip large elements of the picture. Figure below shows a number of different lines with respect to the screen :



- **Notice that those lines which are partly invisible are divided by the screen boundary into one or more invisible portions but only one visible segment.**
- **This means that the visible segment of a straight line can be determined simply by computing its two endpoints.**

clipping

We divide the line clipping process into two phases:

1- Identify those lines which intersect the window and so need to be clipped

2- Perform the clipping.

All line segments fall into one of the following clipping categories:

1- Visible: both endpoints of the line segment lie within the window.

line (A----B)

2- Not visible: the line segment definitely lies outside the window.

This will occur if the line segment from (X1, Y1) to (X2, Y2) satisfies any one of the following four inequalities:

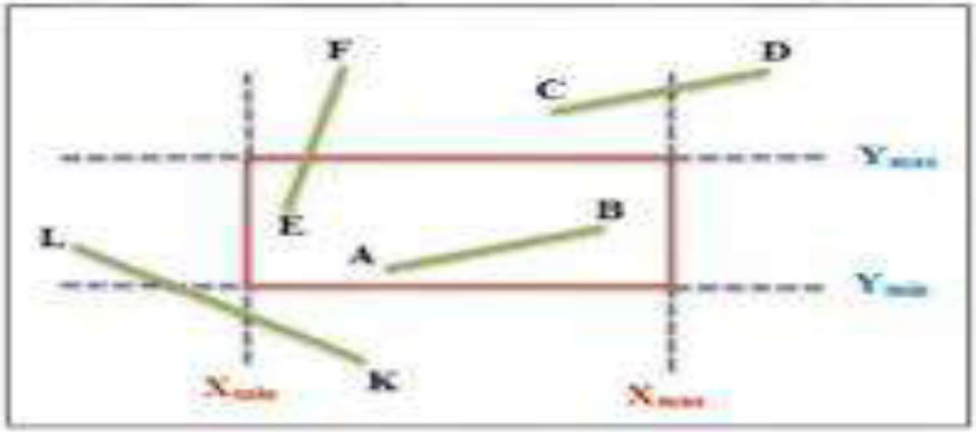
X1 , X2 > X max Y1 , Y2 > Y max

X1 , X2 < X min Y1 , Y2 < Y min

line (C----D) and (L----K)

3- Clipping candidate: the line is in neither category 1 nor category 2.

line (E----F)



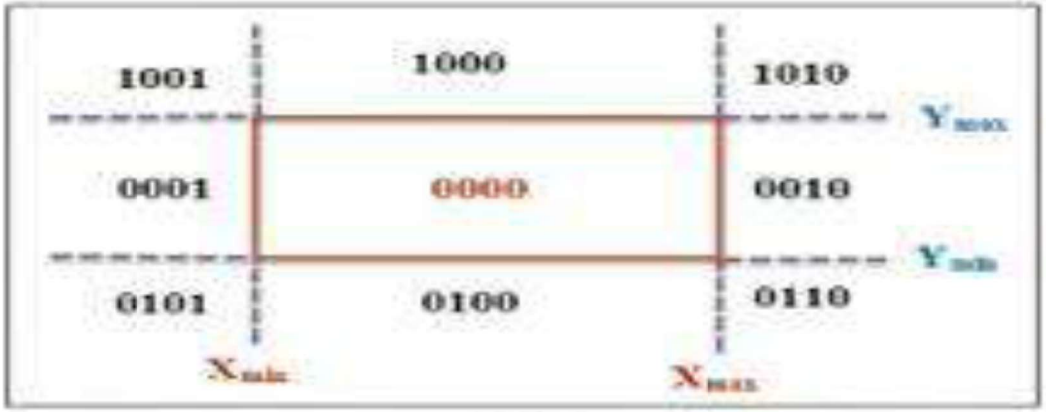
Cohen-Sutherland algorithm

Cohen – Sutherland algorithm

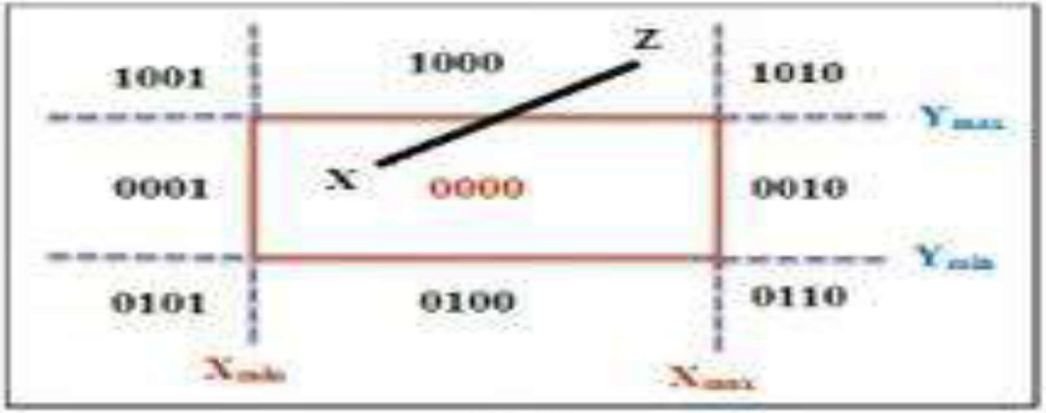
This algorithm is to find the category of the line segment. The algorithm proceeds in two steps:

1- Assign a 4-bit code to each endpoint of the line segments. The code is determined according to which of the following nine regions the endpoints lie in :

X= 0000 Z=1000



(A)



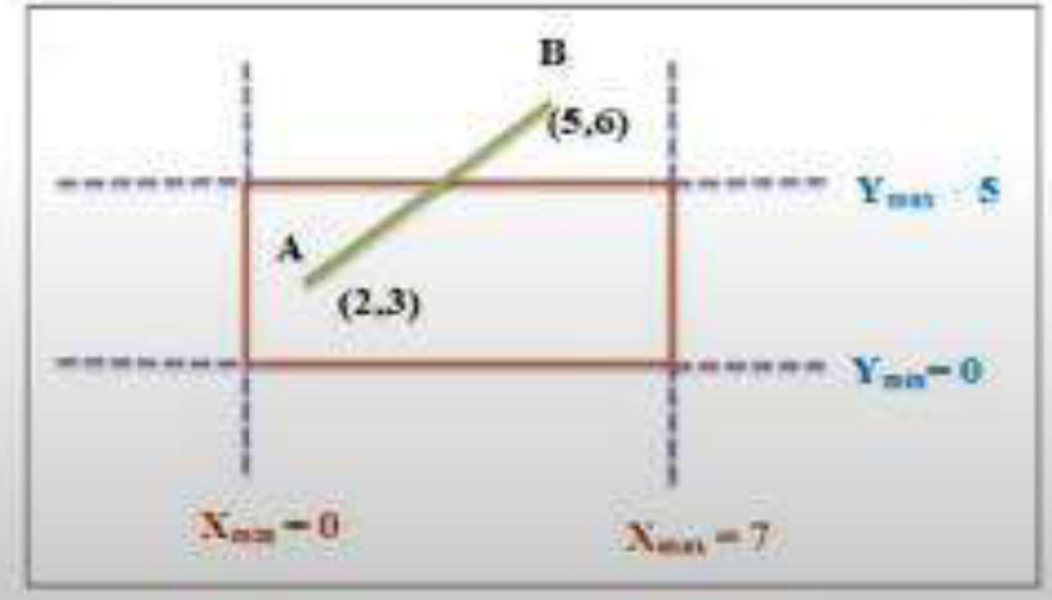
(B)

Starting from the leftmost bit, each bit of the code is set to true(1) or false(0) according to the schema:

Bit 1 ■ endpoint is above the window = sign (Y – Ymax)
Bit 2 ≡ endpoint is below the window = sign (Ymin -Y)
Bit 3 ≡ endpoint is to the right of the window = sign (X - Xmax)
Bit 4 ■ endpoint is to the left of the window = sign (Xmin - X)

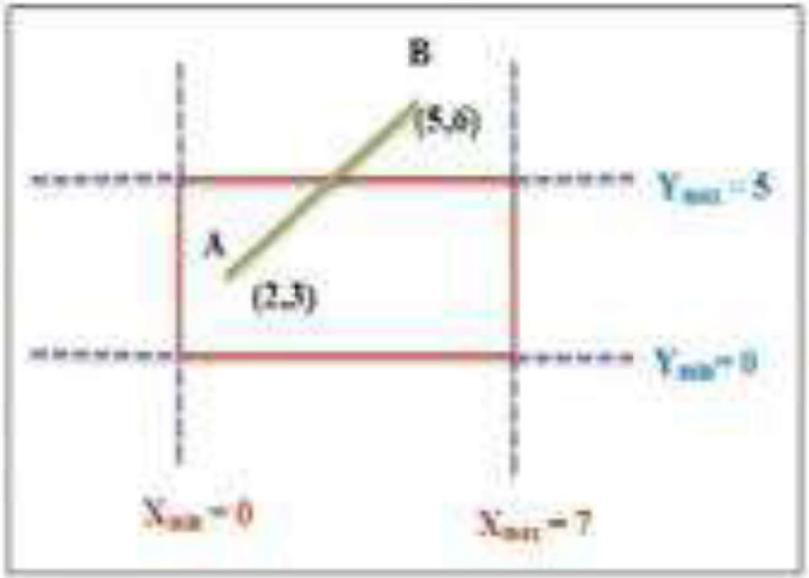
Note :

sign (a) = 1 if a is positive
sign (a) = 0 otherwise.



Example 1:

To fined the code for any end point

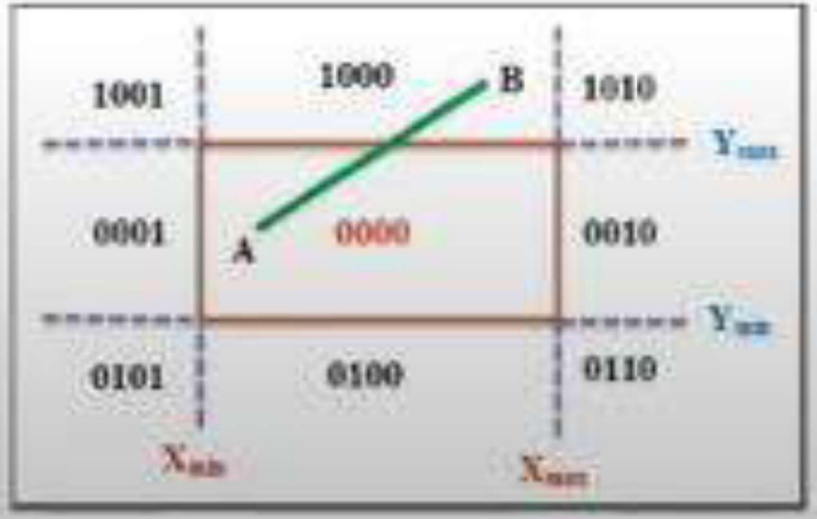


1- End point A (0000)

Bit 1 ≡ = sign (Y – Ymax) = sign (3 – 5) = -2 = 0
Bit 2 ≡ = sign (Ymin - Y) = sign (0 – 3) = -3 = 0
Bit 3 ≡ = sign (X - Xmax) = sign (2 – 7) = -2 = 0
Bit 4 ≡ = sign (Xmin - X) = sign (0 – 2) = -2 = 0

2- End point B (1000)

Bit 1 ≡ = sign (6 – 5) = 1 = 1
Bit 2 ≡ = sign (0 - 6) = -6 = 0
Bit 3 ≡ = sign (5 – 7) = -2 = 0
Bit 4 ≡ = sign (0 – 5)= -5 = 0



2- Determine The line segment is a candidate for clipping

1- Visible: The line segment is visible if both endpoint codes are 0000

A = 0000

B = 0000

2- Not visible: The line segment is not visible if the logical AND of the codes is not 0000

C = 1000

D = 1010 AND

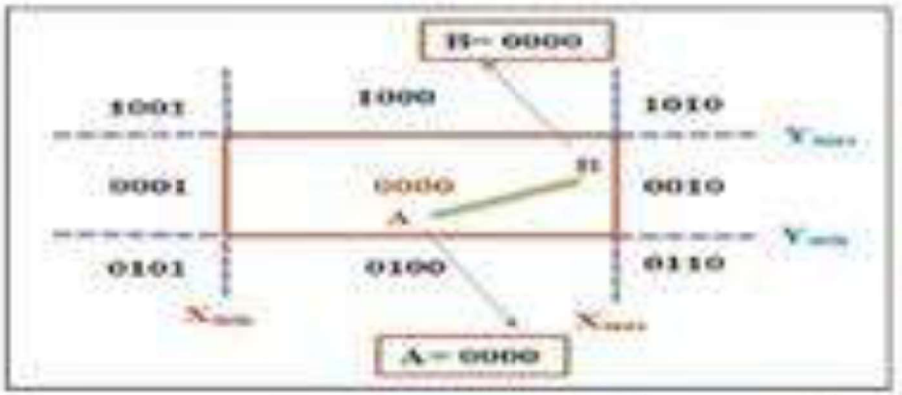
1000

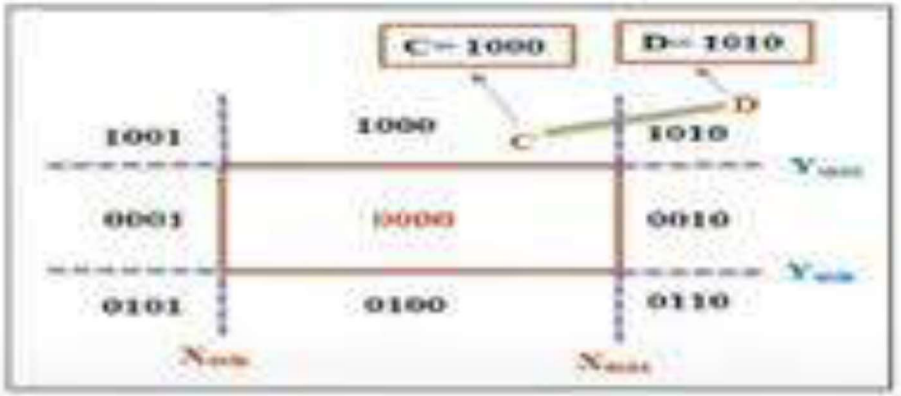
3- Clipping candidate: The line segment is a candidate for clipping if the logical AND of the endpoint codes is 0000.

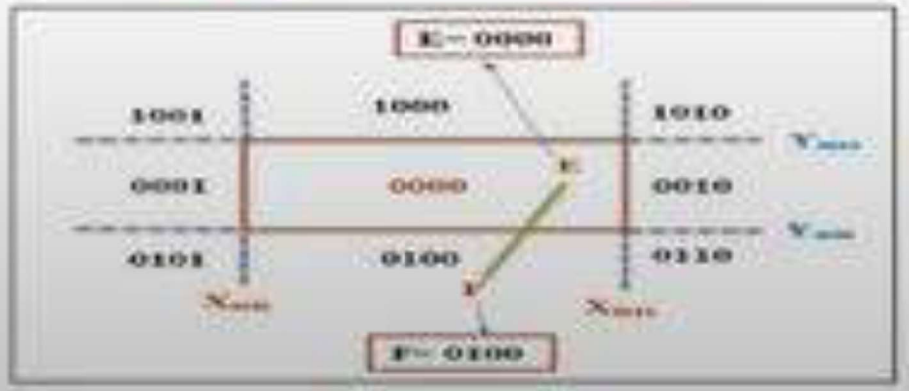
E = 0000

F = 0100 AND

0000







- The point of intersection of the line segment with an extended window edge, each of the four directions is tested in the order: left, right, top, bottom. The part of the line that is clearly outside is discarded.

- To calculate the point of intersection of the line segment with the window border, the point-slope formula is used. If M is the slope of a line segment between (X1, Y1) and (X2, Y2), then if $X1 \neq X2$:-

$$M = (Y2 - Y1) / (X2 - X1)$$

for any other point (X,Y) on the line :-

$$M = (Y - Y1) / (X - X1)$$

If we are testing against a left or right direction the X value is known (left or right) edge value. This X value is substituted into the equation:

$$Y = M * (X - X1) + Y1$$

If we are testing against a top or bottom, the Y value is known and substituted into :

$$X = (1/M) * (Y - Y1) + X1$$

تحديد نقطة التقاطع الخط مع حافة النافذة

- The point of intersection of the line segment with an extended window edge, each of the four directions is tested in the order: left, right, top, bottom. The part of the line that is clearly outside is discarded.

- To calculate the point of intersection of the line segment with the window border, the point-slope formula is used. If M is the slope of a line segment between (X1, Y1) and (X2, Y2), then if $X1 \neq X2$:-

$$M = (Y2-Y1) / (X2-X1)$$

for any other point (X,Y) on the line :-

$$M=(Y-Y1) / (X-X1)$$

If we are testing against a left or right direction the X value is known (left or right) edge value. This X value is substituted into the equation:

$$Y=M * (X-X1) + Y1$$

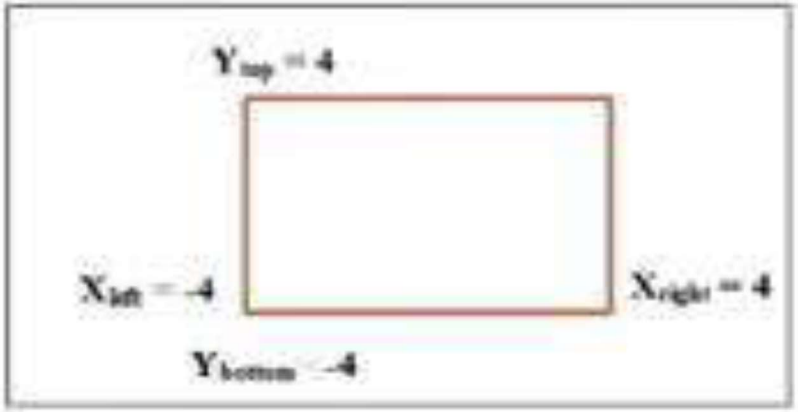
If we are testing against a top or bottom, the Y value is known and substituted into :

$$X= (1/M) * (Y-Y1) + X1$$

Example 2:-

If the clipping window is XL=-4; XR=4; YT=4; YB=-4 Clip the lines:

KL=(4,5)-(6,5) and AB=(1,1)-(1,3)



(A)

Solution :

For the line KL: code1 of (4,5) is 1000

code2 of (6,5) is 1010

code1 And code2 : 1000 And 1010 = 1000 (not zero)

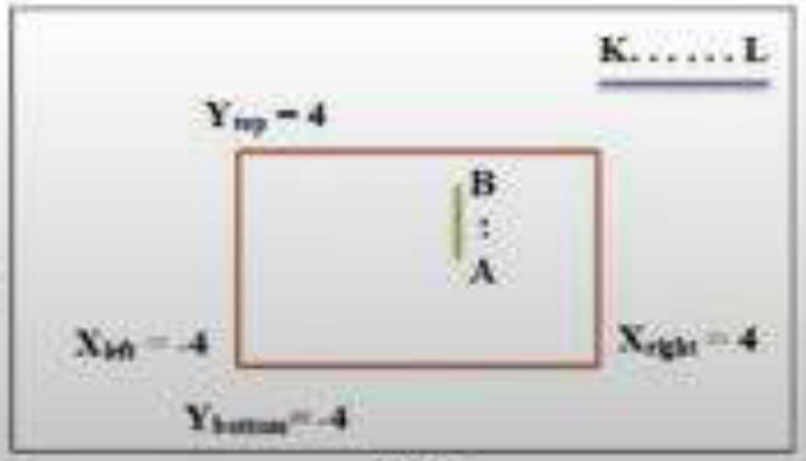
then the line KL is not visible

For the line AB: code1 of (1,1) is 0000

code2 of (1,3) is 0000

code1 And code2 : 0000 And 0000 = 0000 (zero)

then the line AB is visible.

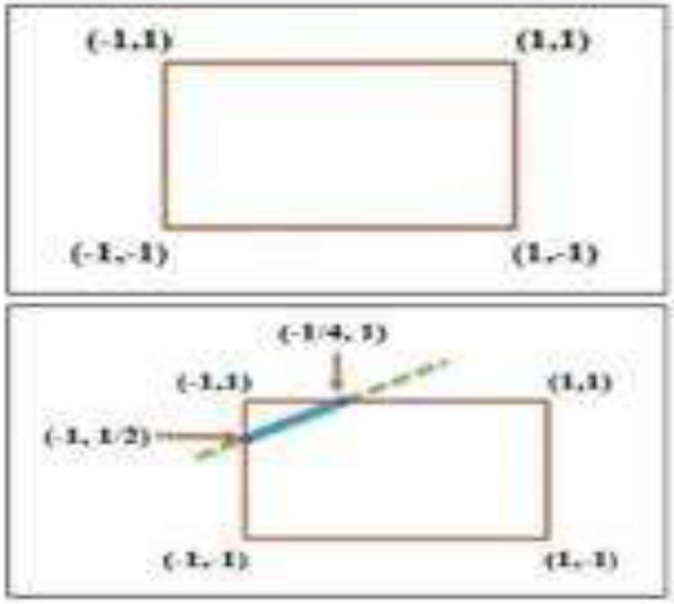


(B)

Clipping

Example 3:

Consider the clipping window
Clip the line (- 3/2 , 1/6) – (1/2 , 3/2)



Solution :

Code1 of (- 3/2 , 1/6) is 0001
Code2 of (1/2 , 3/2) is 1000
Code1 And Code2 : 0001 And 1000=0000
Then the line is a candidate for clipping

We must calculate the intersection points: **M= 1.34 / 2 = 0.6**

Left intersection : XL= -1
YL=0.6 (-1-(-3/2))+1/6=1/2
Then (-1, 0.5) is the first point of intersection

Right intersection : XR=1
YR=0.6 (1-(-3/2))+1/6=1.8
Then (1, 1.8) is rejected

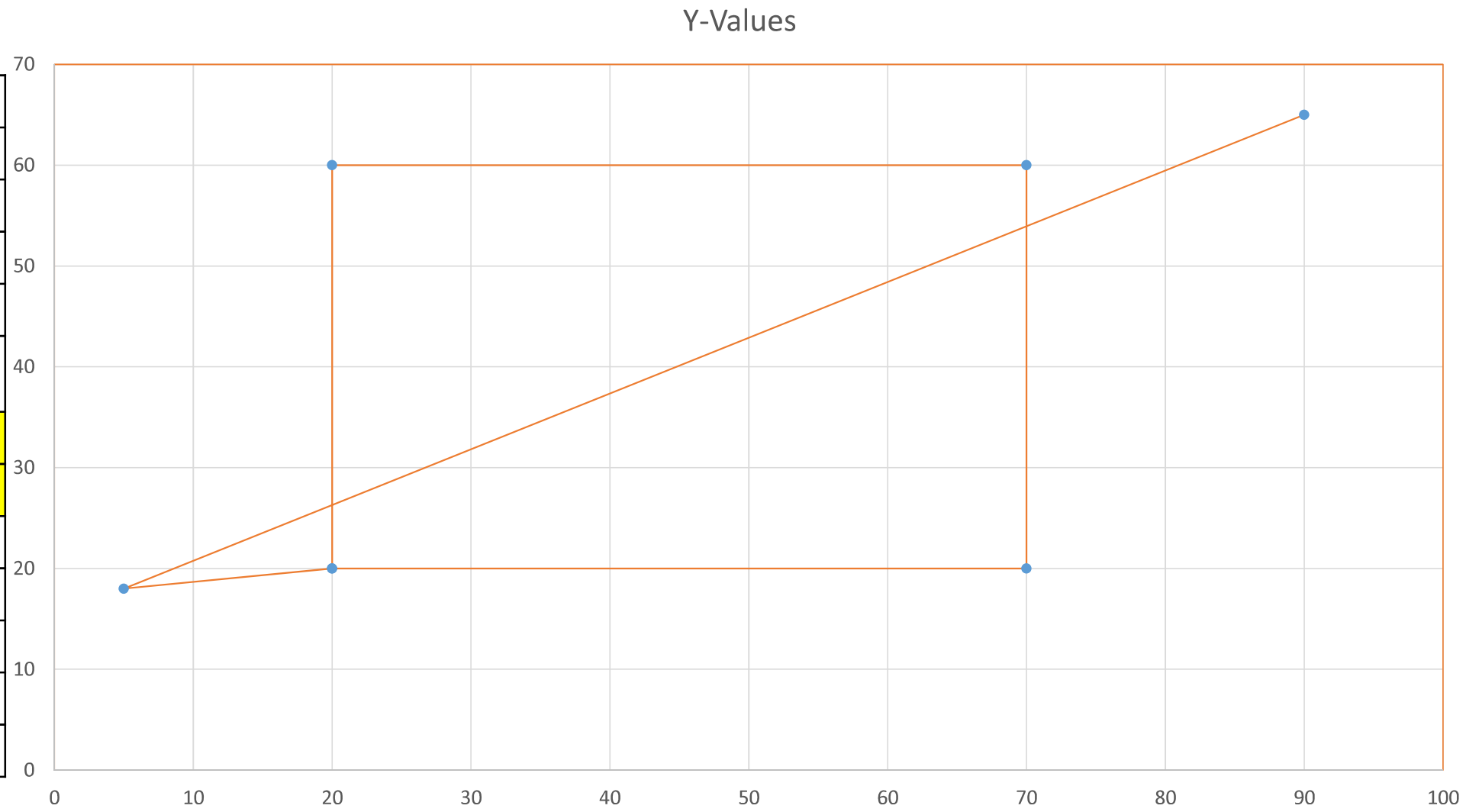
Top intersection : YT=1
XT =1.6 (1- 1/6)+(- 3/2) = - 1/4
Then (-1/4 ,1) is the second point of intersection

Bottom intersection : YB= -1
XB =1.6 (-1 - 1/6)+(- 3/2) = - 3.25
Then (-3.25 ,-1) is rejected

EX/ CONSIDER THE CLIPPING WINDOW CLIP THE LINE?

X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65

X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65



$$M = \frac{Y2-Y1}{X2-X1}$$

$$X = \frac{1}{M}\ (Y-Y1)+X1$$

$$Y = Y1 + M\ (X - X1)$$

خطوات الحل كالأتي :

1-Assign code of each regions as TBRL

2-before OR between the two end points of the line if OR=0000 trivially accept

3-perfrom and between the two end points of line if AND ≠ 0000 reject

4- Find intersection with required edge

5- Repeat write new points

1- ***Assign code of each regions as <u>TBRL</u>***

0101 ⟶ 0101

2- ***<u>OR</u> between the two end points of the line if OR=0000***

3-***<u>AND</u> between the two end points of line if AND ≠ 0000 reject***

4- ***Find intersection with required edge***

$$\begin{array}{r} 0101 \\ 0101 \\ \hline 1111 \end{array} \quad \text{OR}$$

≠ 0000 **cant accept**

=0000 cant reject

$$\begin{array}{r} 0101 \\ 0101 \\ \hline 0000 \end{array} \quad \text{AND}$$

X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65

$$M = \frac{Y2-Y1}{X2-X1} = \frac{65-18}{90-5} = 0.55294 \ ,$$

$$Y = Y1 + M(X - X1)$$

$$X = \frac{1}{M} * (Y\text{-}Y1) + X1 = \frac{1}{0.55} * (Y\text{-}18) + 5 =$$

$$Y = 18 + 0.553 * (X\text{-}5)$$

(5,18) ⟹ *0101* ***<u>TBRL</u>***

(5,18) → 0101 ***TBRL***

Y=ymin=20

X=1/5(20-18)+5 =8.617

(8.617,20) → 0001

(90,65) → 1010

OR

0001
1010
1011

OR 1011 ≠ 0000

AND

0001
1010
0000

AND 0000 = 0000

Find

x=xmin=20

Y=18+0.5*(20-5)=26.2941

X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65

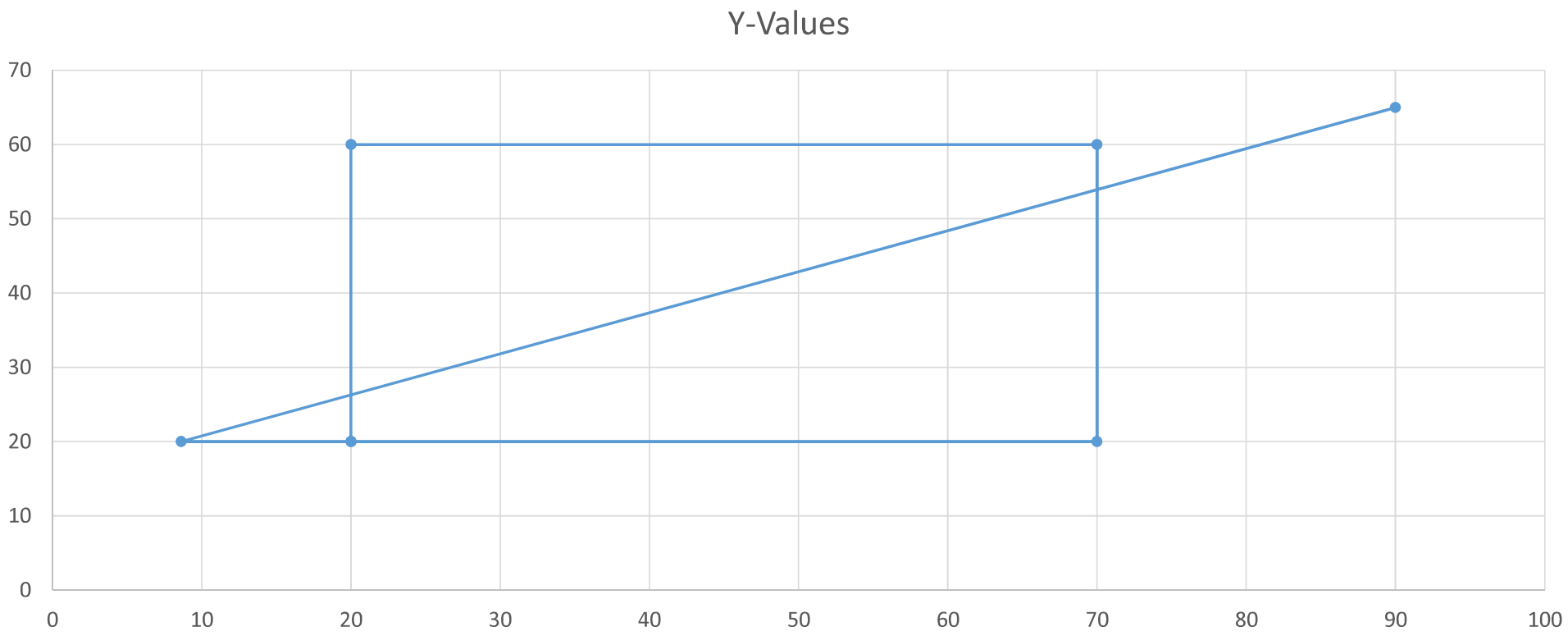
→

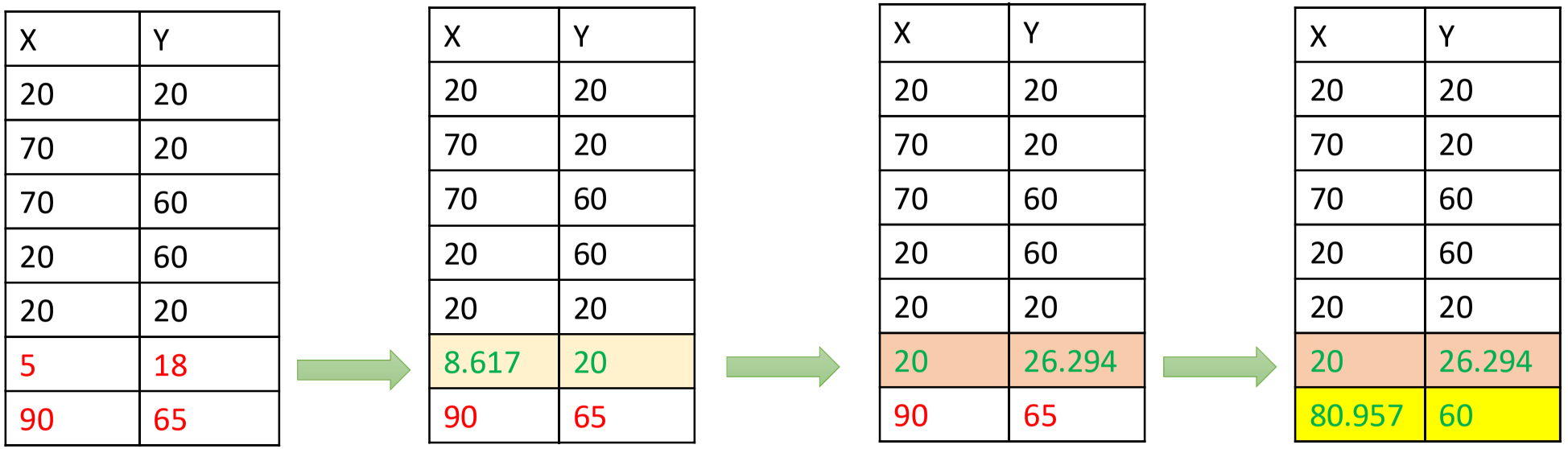
X	Y
20	20
70	20
70	60
20	60
20	20
8.617	20
90	65

→

X	Y
20	20
70	20
70	60
20	60
20	20
20	26.294
90	65

(x,y)=(8.617,20)




X Y
20 20
70 20
70 60
20 60
20 20
5 18
90 65
X Y
20 20
70 20
70 60
20 60
20 20
8.617 20
90 65
X Y
20 20
70 20
70 60
20 60
20 20
20 26.294
90 65
X Y
20 20
70 20
70 60
20 60
20 20
20 26.294
80.957 60

(20,26.294)

(80.957,60)

0000
0010
0100

OR 0010 ≠ 0000

0000
0010
0000

AND =0000

X	Y
20	20
70	20
70	60
20	60
20	20
20	26.294
80.957	60

X	Y
20	20
70	20
70	60
20	60
20	20
20	26.294
70	53.941

TBRL

Find ins right edge

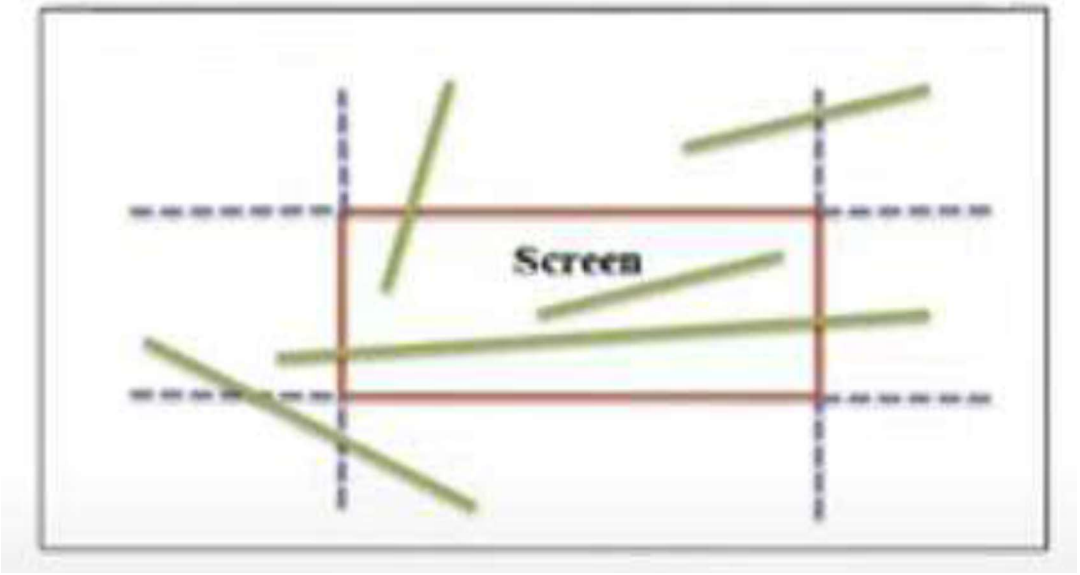
X=xmax=70

Y=18+0.55(70-5)=53.941

(70,5394) في الوسط =0000 or 0000/0000=0000 accept

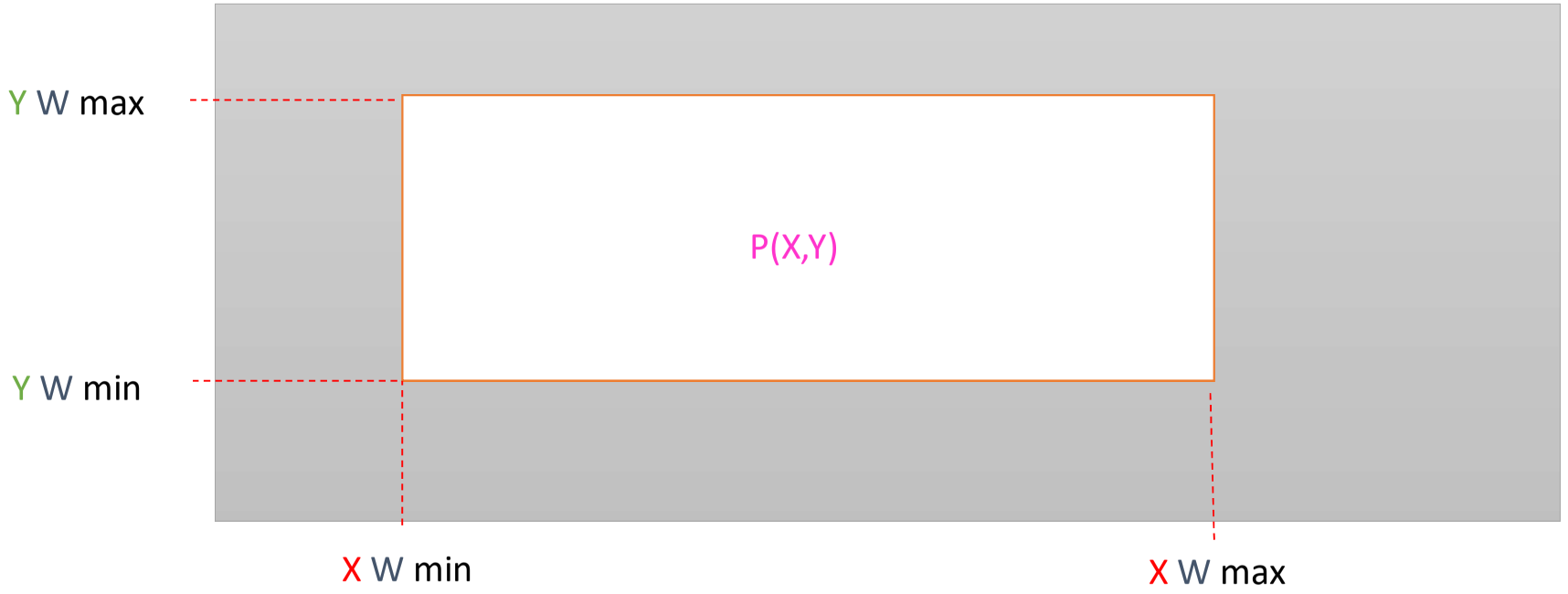
Clipping Windows

CLIPPING



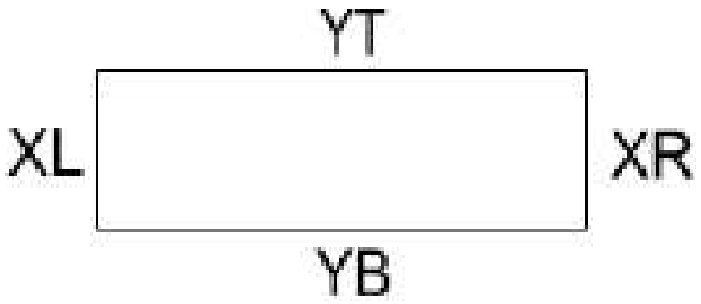
CLIPPING

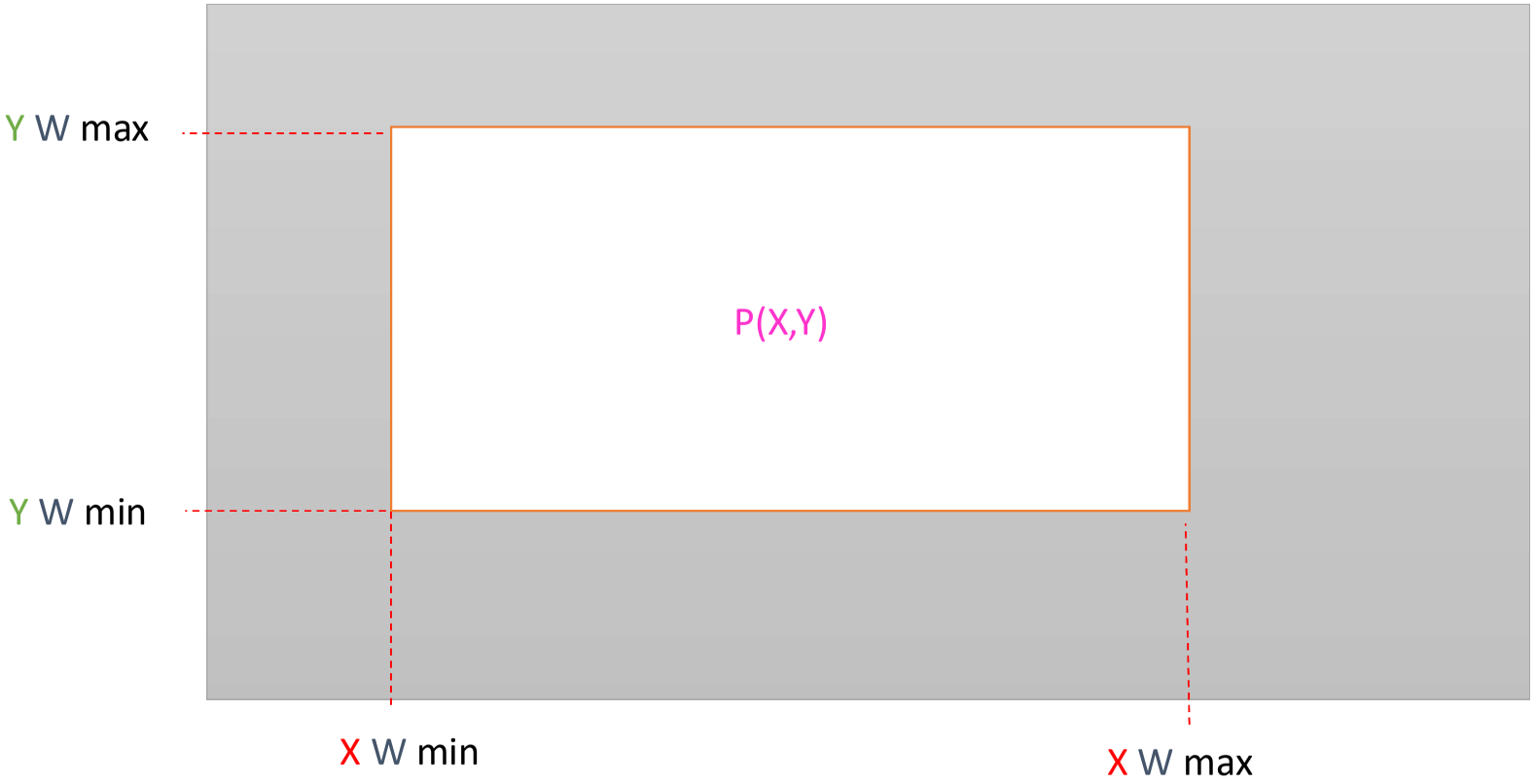
- If we wish to display only a portion of the total picture, we use a window to select that portion of the picture which is to be viewed (like clipping or cutting out a picture from a magazine). This is known as clipping.
- The process of clipping determines which elements of the picture lie inside the window and so are visible.



RECTANGULAR CLIPPING WINDOWS

The clipping window assumes to be rectangles whose sides are aligned with the coordinate area. The X extent is measured from X min to X max and the Y extent is measured from Y min to Y max.




Y W max
P(X,Y)
Y W min
X W min
X W max

There are three types of clipping

1 – Point 2 – Line 3 – Polygon

TBRL

Top

Bottom

Right

Left

Top

TBRL 1001	TBRL 1000	TBRL 1010
TBRL 0001	TBRL 0000	TBRL 0010
TBRL 0101	TBRL 0100	TBRL 0110

Left

Right

Bottom

1:- POINT CLIPPING

A point p(x,y) is inside the rectangular window (visible) if all the following inequalities are true.

$$X_{min} \leq X \leq X_{max} \qquad Y_{min} \leq Y \leq Y_{max}$$

- If any of these inequalities is false point P is outside the window and is not displayed (invisible).

Point Clipping algorithm

P(x , y) , نقطة إحداثياتهاXl , Xr , Yt , Yb معلومة

Procedure out code (p ; VAR OP: integer) ;

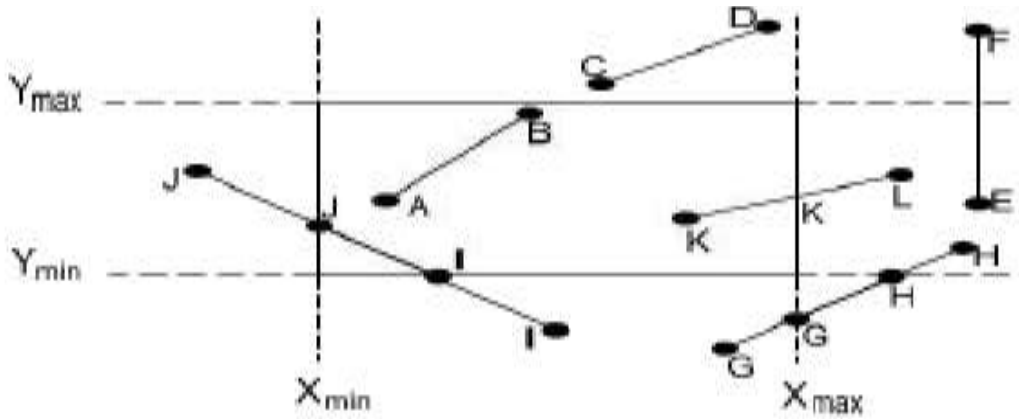
Begin

- Op := 0 ;
- If P.x < XL then op : = op OR 1 ;
- If P.x > XR then op : = op OR 2 ;
- If P.y < YB then op : = op OR 4 ;
- If P.y > YT then op : = op OR 8 ;

End;

2:-Line Clipping

First	bit	on	if	x < XL	to	the	left	of window.
Second	bit	on	if	x > XR	to	the	right	of window.
Third	bit	on	if	y < YB	to	the	below	of window.
Fourth	bit	on	if	y > YT	to	the	above	of window.



LINE SEGMENT CLIPPING:

1 – Visible:

Both endpoints of the line segment lie within the window (Line AB).

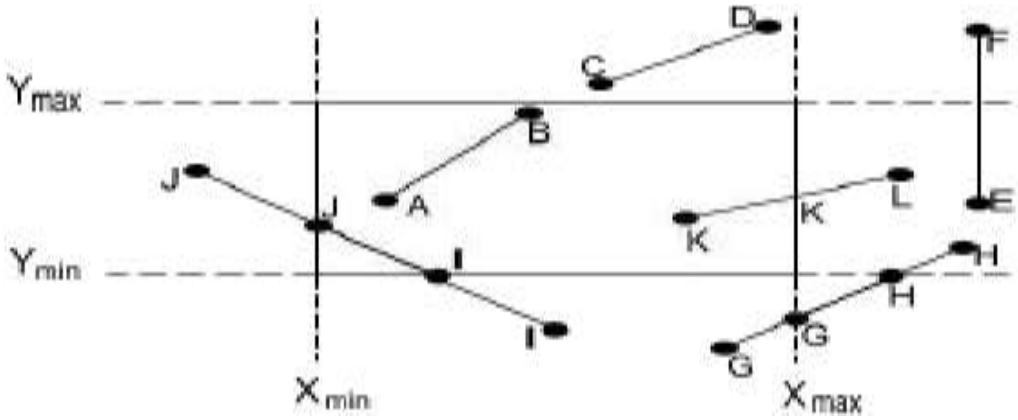
2 – Not visible:

The line segment definitely lies outside the window (Line CD and EF).

This will occur if the line segment from (X_1, Y_1) to (X_2, Y_2) satisfies any one of the following four in qualities.

$X_1, X_2 > X_{max}$ $Y_1, Y_2 > Y_{max}$ $X_1, X_2 < X_{min}$ $Y_1, Y_2 < Y_{min}$

3 – Clipping candidate:



الخط ليس في أي فئة

The line is in neither category 1 nor 2 (Line GH, IJ, and KL)

LINE INTERSECTIONS AND CLIPPING:

We determine the intersection points of the lines in category (3) with the boundaries of the window. The intersection points subdivided the line segment into several smaller line segments which can belong only to category 1 (visible) or category 2 (not visible). The segment in category 1 will be the clipped line segment.

We divide the line clipping process into two phases:

1- Identify those lines which intersect the window and so need to be clipped

2- Perform the clipping.

All line segments fall into one of the following clipping categories:

1- Visible: both endpoints of the line segment lie within the window.

line (A----B)

2- Not visible: the line segment definitely lies outside the window.

This will occur if the line segment from (X1, Y1) to (X2, Y2) satisfies any one of the following four inequalities:

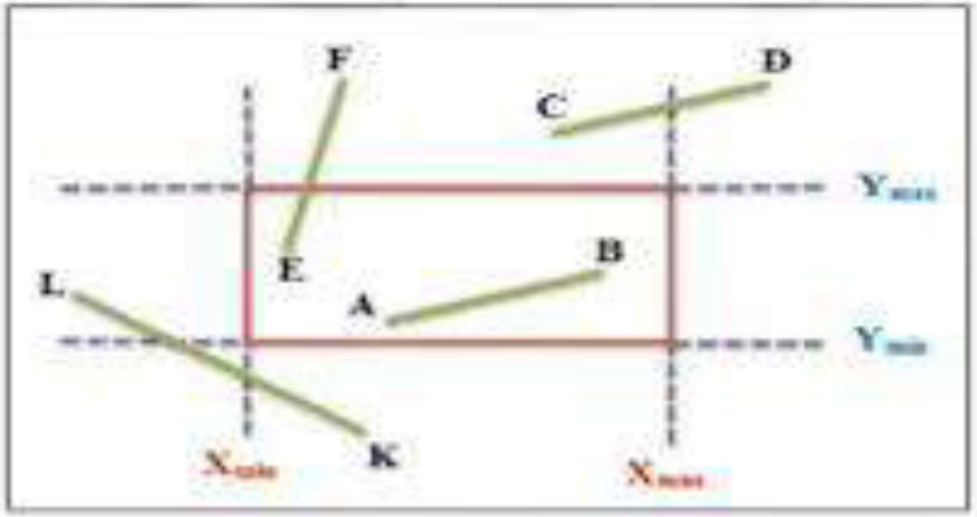
X1 , X2 > X max Y1 , Y2 > Y max

X1 , X2 < X min Y1 , Y2 < Y min

line (C----D) and (L----K)

3- Clipping candidate: the line is in neither category 1 nor category 2.

line (E----F)



INTERSECTION POINT:

- If M is the slope of the line segment between points (X_1, Y_1) and (X_2, Y_2) then if $X_1 \neq X_2$

M = (Y2 – Y1) / (X2 – X1) then any point on the line is ***M = (Y – Y1) / (X – X1)***

★ If the line segment crosses a left or right window edge then $X_1 \neq X_2$ and M has non denominator.

★ If the line crosses a top or bottom widow edge then $Y_1 \neq Y_2$ and the reciprocal of the slop 1/m has a nonzero denominator.

The slop M is obtained from the two given end points.

The slop M is obtained from the two given endpoints

★ If we are testing against a left or right direction the X value is known (the left or right edge value).

Then the X value is substituted into the equation for Y.

$$Y = M * (X - X_1) + Y_1$$

★ If we are testing against a Top or Bottom direction the Y value is known (the Top or Bottom edge value).

Then the Y value is substituted into the equation for X.

$$X = 1/M * (Y - Y_1) + X_1$$

Simple visibility algorithm.

Check for totally visible lines.

- If ((xb < XL) OR (xb > XR)) then 1 .
- If ((xe < XL) OR (xe > XR)) then 1 .
- If ((yb < YB) OR (yb > YT)) then 1 .
- If ((ye < YB) OR (ye > YT)) then 1 .

Draw line

Go to 3

1 - Check for totally invisible lines

- if ((xb < XL) AND (xe < XL)) then 2
- if ((xb > XR) AND (xe > XR)) then 2
- if ((yb < YB) AND (ye < YB)) then 2
- if ((yb > YT) AND (ye > YT)) then 2

The line is partially visibly or diagonally crosses the corner; determine the intersections go to 3

2 lines is invisible

3 next lines

Cohen clipping Algorithm

{w (4) = (xl , xr ,yb , yt) } , {p(1)=p(x), p(2) = p(y)} ,{pc=0000}

Subroutine end point (p , w , pc , s)

If p (1) < w (1) then pc (4) = 1 else pc (4) = 0

If p (1) > w (2) then pc (3) = 1 else pc (3) = 0

If p (2) < w (3) then pc (2) = 1 else pc (2) = 0

If p (2) > w (4) then pc (1) = 1 else pc (1) = 0

S = 0

For I = 1 to 4 s = s + pc (i) Next i

Return

Subroutine logical (pc1 ,pc2 , inter)

Inter = 0

For I = 1 to 4 Inter = inter = pc1 (i) * pc2(i) Next I

Return

Subroutine Cohen (p1 , p2 , w , visible)

Call end point (p1 , w . pc . s1)

Call end point (p2 , w .pc2 , s2)

If S1 = 0 & S2 = 0 then visible = yes

Call logical (pc1 , pc2 , inter)

If inter < > 0 then visible = NO

else visible = partial

If S1 = 0 then 1

else p1 p2 end if

1 Return

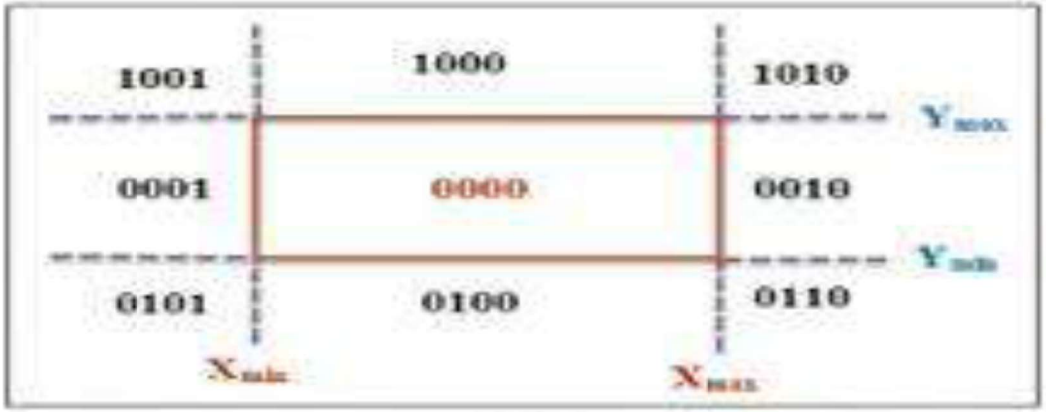
Cohen-Sutherland algorithm

Cohen – Sutherland algorithm

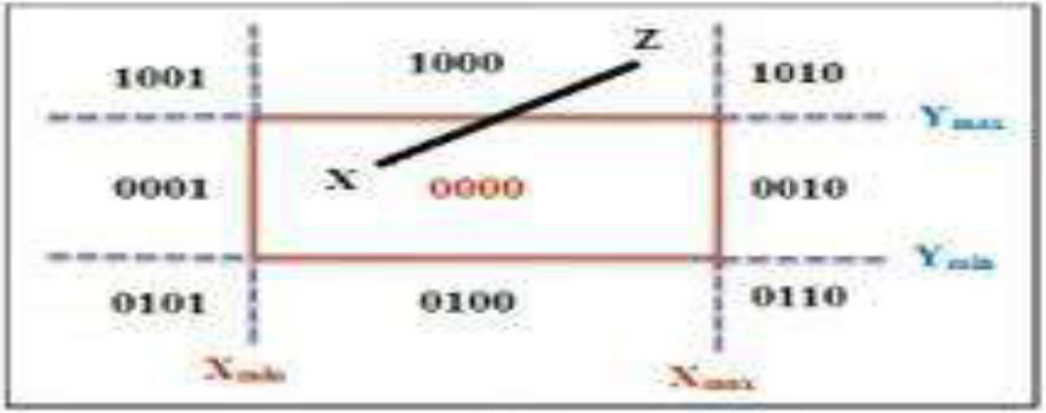
This algorithm is to find the category of the line segment. The algorithm proceeds in two steps:

1- Assign a 4-bit code to each endpoint of the line segments. The code is determined according to which of the following nine regions the endpoints lie in :

X= 0000 Z=1000



(A)



(B)

THE ALGORITHM

- FLAG = 0
- IF P2 (1) – P1 (1) = 0
- THEN FLAG = - 1
- ELSE slop = (p2 (2) – p1 (2)) /(p2 (1) – (p1(1))
- End if
- For I = 1 to 4
- Call cohen (p1 , p2 . w1 visible)
- If visible = yes then 2
- If visible = No then 3
- If flag = -1 and i ≤ 2 then 1
- If i ≤ 2 then
- Inter = slop * (w (i) – p1 (1)) $_{+}$ p1 (2)
- P2 (1) = w (I)
- P2 (2) = inter
- else if flag = -1 then
- p2 (2) = w (i)

```
else inter = (1 /slop ) * ( w ( i ) – p1 ( 2) ) + p1 ( 1 )
  p2 ( 1 ) = inter
      p2 ( 2 ) = w ( i )
              end  if
       end if
1 next  i
2 draw  p1 p2
3 end
```

MIDPOINT SUBDIVISION

- The line segment is divided at its midpoint into two smaller line segments. The clipping categories of the two new line segments are then determined. Each segment in category 3 is divided again into smaller segment and categorized. The bisection and categorization process continues until all segments are in category 1 (visible) or category 2 (invisible).
- The midpoint coordination (X_m , Y_m) of a line segment joining

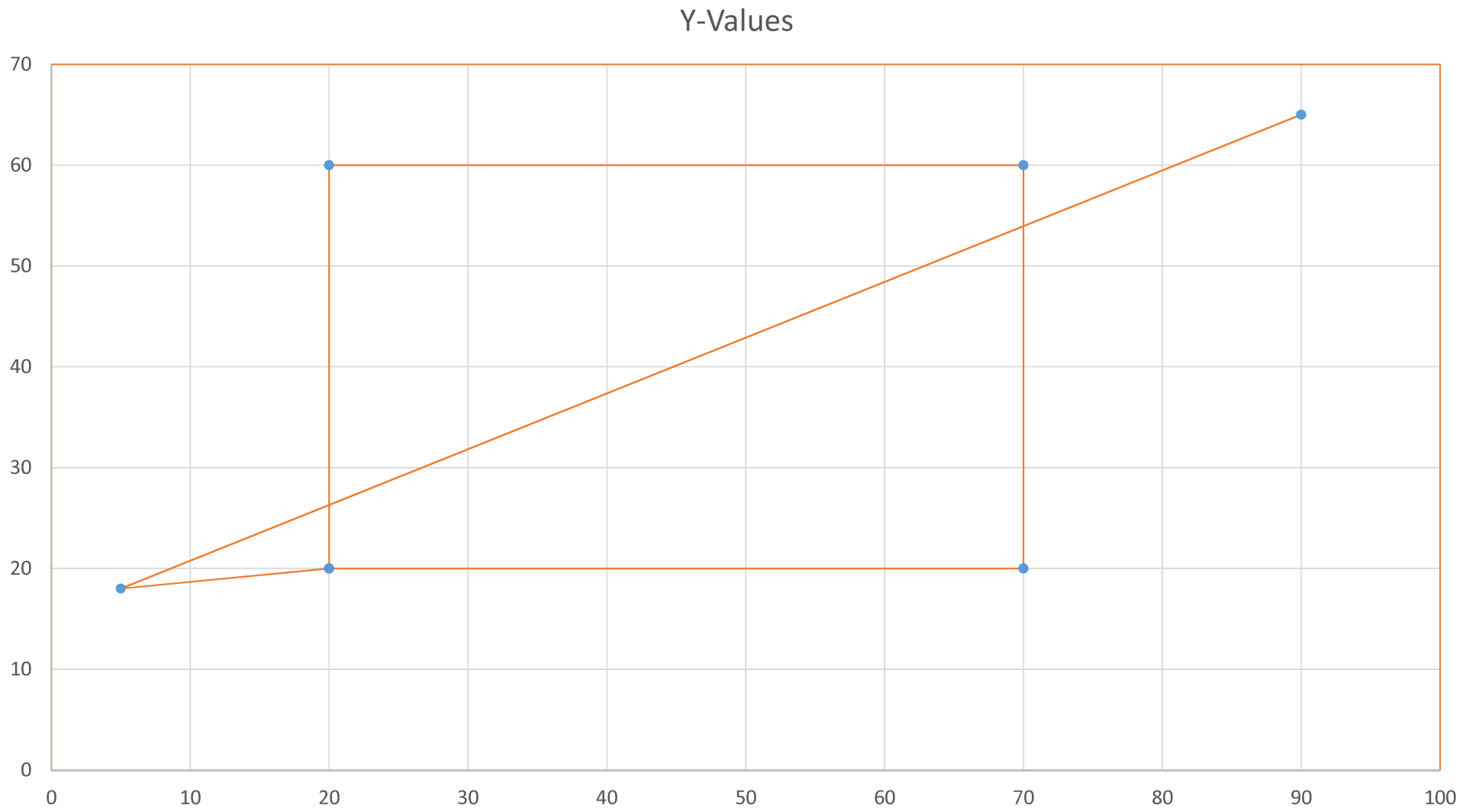
  P_1 (X_1 , Y_1) to P_2 (X_2 , Y_2) are given by

  $Xm = (X1 + X2) / 2$, $Ym = (Y1 + Y2) / 2$

EX/ CONSIDER THE CLIPPING WINDOW CLIP THE LINE?

X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65

X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65



$$M = \frac{Y2-Y1}{X2-X1}$$

$$X = \frac{1}{M}\ (Y-Y1)+X1$$

$$Y = Y1 + M\ (X - X1)$$

خطوات الحل كالأتي :

1-Assign code of each regions as TBRL

2-before OR between the two end points of the line if OR=0000 trivially accept

3-perfrom and between the two end points of line if AND ≠ 0000 reject

4- Find intersection with required edge

5- Repeat write new points

1- ***Assign code of each regions as <u>TBRL</u>***

0101 → 1010

2- ***<u>OR</u> between the two end points of the line if OR=0000***

≠ 0000 **cant accept**

3-***<u>AND</u> between the two end points of line if AND ≠ 0000 reject***

$$\begin{array}{r} 0101 \\ 1010 \\ \hline 1111 \end{array} \quad \text{OR}$$

=0000 cant reject

4- ***Find intersection with required edge***

$$\begin{array}{r} 0101 \\ 1010 \\ \hline 0000 \end{array} \quad \text{AND}$$

X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65

$$M = \frac{Y2-Y1}{X2-X1} = \frac{65-18}{90-5} = 0.55294 \ , \qquad Y = Y1 + M\,(X - X1)$$

$$X = \frac{1}{M} * (Y-Y1)+X1 = \frac{1}{0.55} * (Y-18)+5 = \qquad Y=18+0.553*(X-5)$$

(5,18) → *0101* ***<u>TBRL</u>***

(5,18) → 0101 ***TBRL***

Y=ymin=20

X=1/0.55*(20-18)+5 =8.617

(8.617,20) → 0001

(90,65) → 1010

OR

0001
1010
1011

OR 1011 ≠ 0000

AND

0001
1010
0000

AND 0000 = 0000

X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65

→

X	Y
20	20
70	20
70	60
20	60
20	20
8.617	20
90	65

→

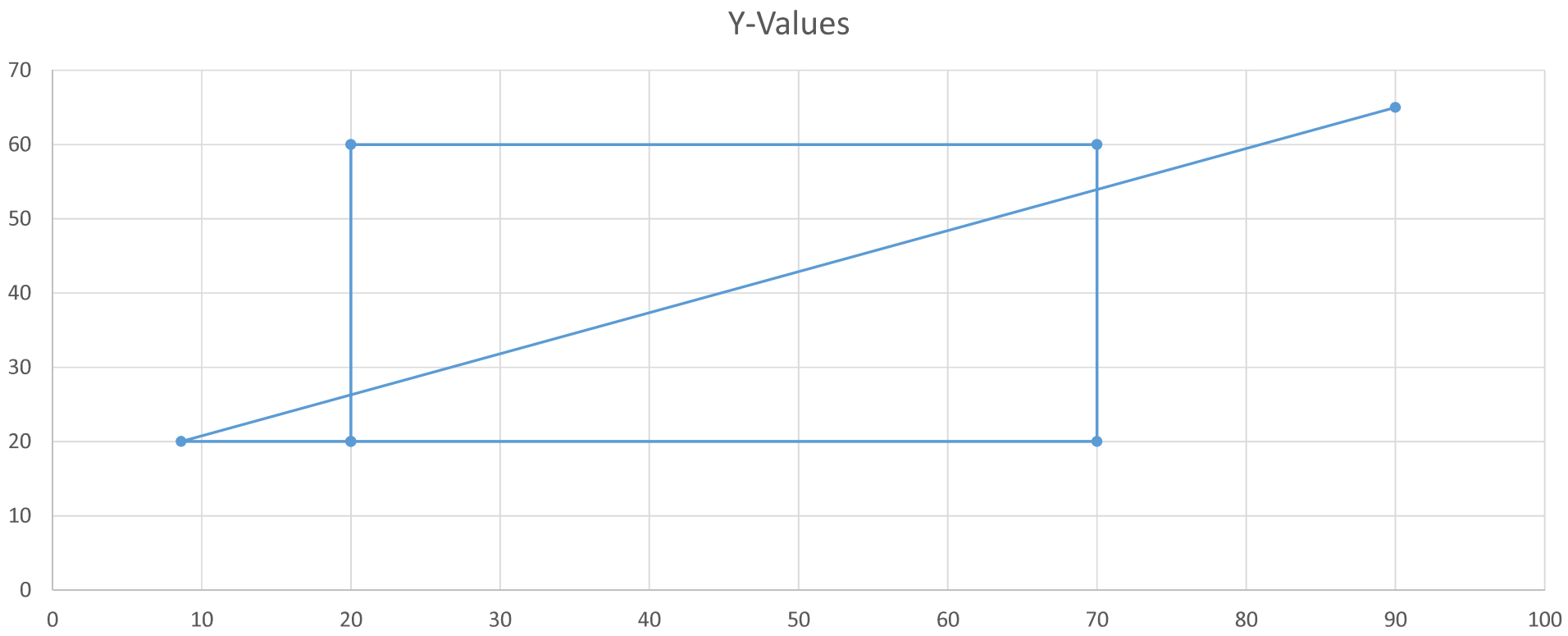
X	Y
20	20
70	20
70	60
20	60
20	20
20	26.294
90	65

Find

x=xmin=20

Y=18+0.5*(20-5)=26.2941

(x,y)=(8.617,20)



X	Y
20	20
70	20
70	60
20	60
20	20
5	18
90	65

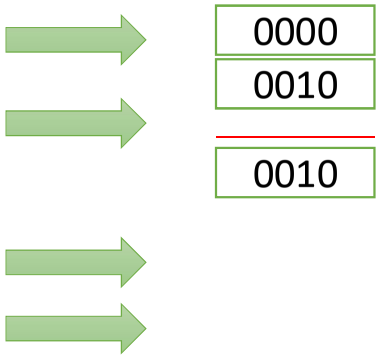
X	Y
20	20
70	20
70	60
20	60
20	20
8.617	20
90	65

X	Y
20	20
70	20
70	60
20	60
20	20
20	26.294
90	65

X	Y
20	20
70	20
70	60
20	60
20	20
20	26.294
80.957	60

(20,26.294)

(80.957,60)



OR 0010 ≠0000

AND =0000

X	Y
20	20
70	20
70	60
20	60
20	20
20	26.294
80.957	60

X	Y
20	20
70	20
70	60
20	60
20	20
20	26.294
70	53.941

TBRL

Find ins right edge

X=xmax=70

Y=18+0.55(70-5)=53.941

(70,5394) في الوسط =0000 or 0000/0000=0000 accept

Ex/ given a window with lower left corner of (1.1) and upper right corner of(5,5) that the x value of the intersection point of this window write the line of end points (2,3),(7,4)is:-

$M = \frac{Y2-Y1}{X2-X1}$ $X = \frac{1}{M} * (Y-Y1)+X1$ $Y = Y1 + M(X - X1)$

M=4-3/7-2=0.2

Y=33+0.6=3.6

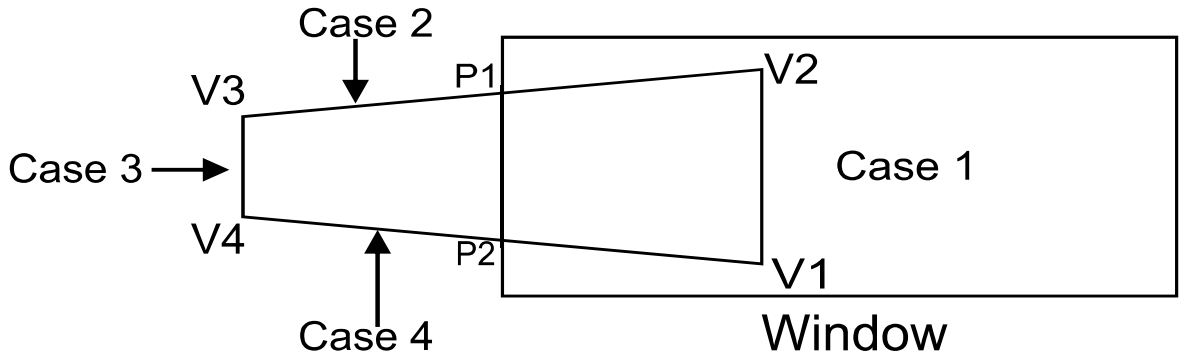
3– polygon clipping Algorithm

There are four possible cases

- **Case – 1** :- If the first point and the second point inside the window **then** store second point
- **Case - 2** :- If the first point inside and the second point outside the window **then** store the intersection.
- **Case – 3** :- If the first point and the second point outside the window **then** nothing.
- **Case – 4** :- If the first point outside and the second point inside the window **then** store the intersection

  and the second point .

EX/



The input polygon is V1 . V2 . V3 . V4

The result of this left clipping is the transformation of input list {V1, V2, V3, V4} to the output list {V2, P1, P2, V4}

- Clips a polygon against each edge of the window.

- عندما نعمل Clipping بالنسبة الى left تكون النتيجة هي V1 P2 P1 V2.
- عندما نعمل Clipping بالنسبة الى RIGHT تكون النتيجة هيP1 V2 V1 P2
- عندما نعمل Clipping بالنسبة الى TOP تكون النتيجة هي V2 V1 P2 P1
- عندما نعمل Clipping بالنسبة الى BOTTON تكون النتيجة هي P2 P1 V2 V1

- نطبق العملية ٤ مرات (Bottom , right , top , left)

- Left : - V2 P1 P2 V1
  Bottom : - P1 P2 V1 V2
- Right : - P2 V1 V2 P1
- Top : - V1 V2 P1 P2
- - For each edge it inputs a list of vertices and outputs a new list of vertices.
- - The input list is a sequence of consecutive vertices of the polygon obtained from the previous edge clipping.

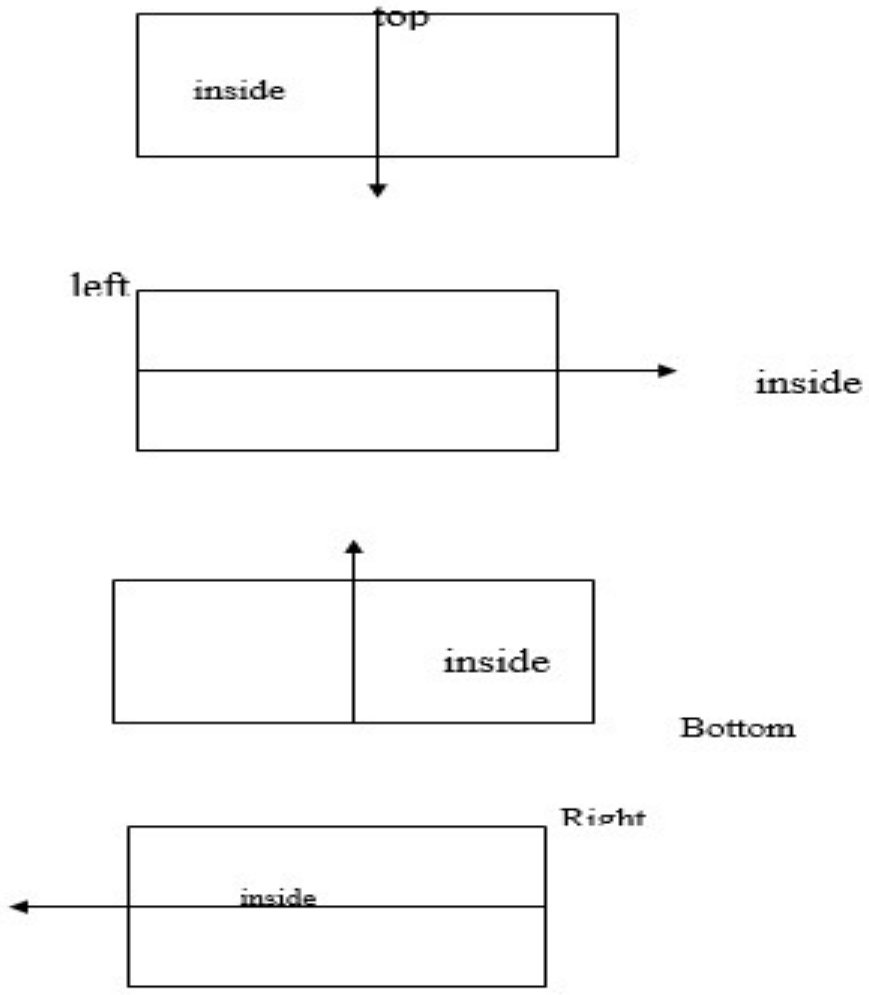
In general cases

P.y < = top inside
p.y >= top outside

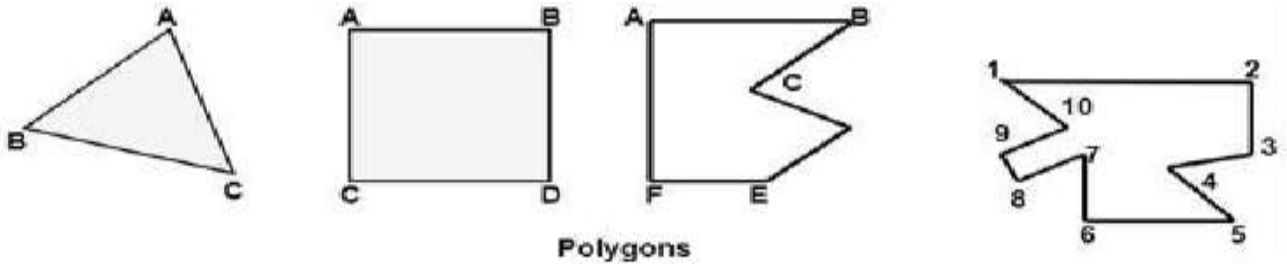
p.x > = left inside
p.x <= left outside

p.y > = Bottom inside
p.y < = Bottom
outside

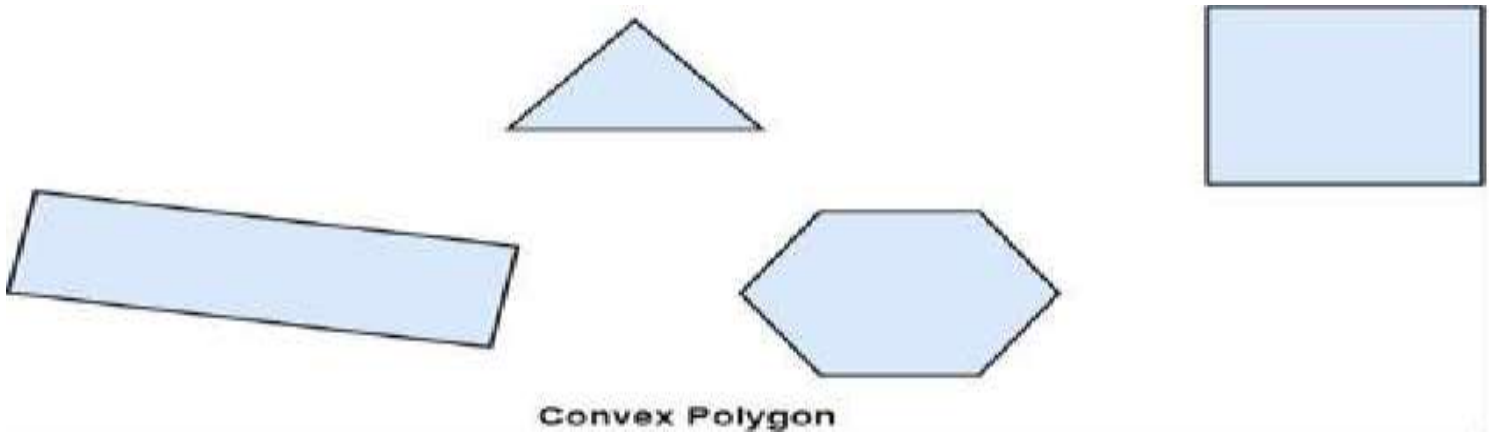
p.x < = right inside
p.x >= right outside



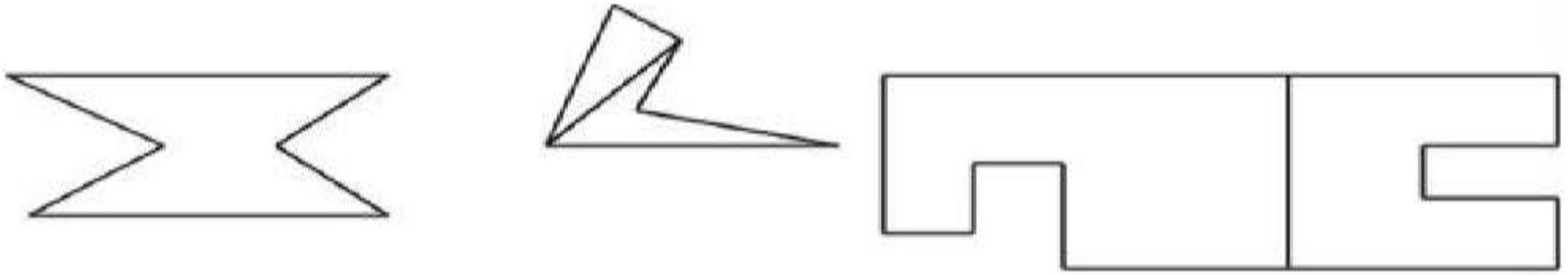
- **Polygon** is a representation of the surface. It is primitive which is closed in nature. It is formed using a collection of lines. It is also called as many-sided figure. The lines combined to form polygon are called sides or edges. The lines are obtained by combining two vertices.
- **Example of Polygon: {Triangle, Rectangle, Hexagon, Pentagon …etcs}**
- Following figures shows some polygons.



Polygons



Convex Polygon



Concave Polygon

Types of Polygons : 1-Convex 2- Concave

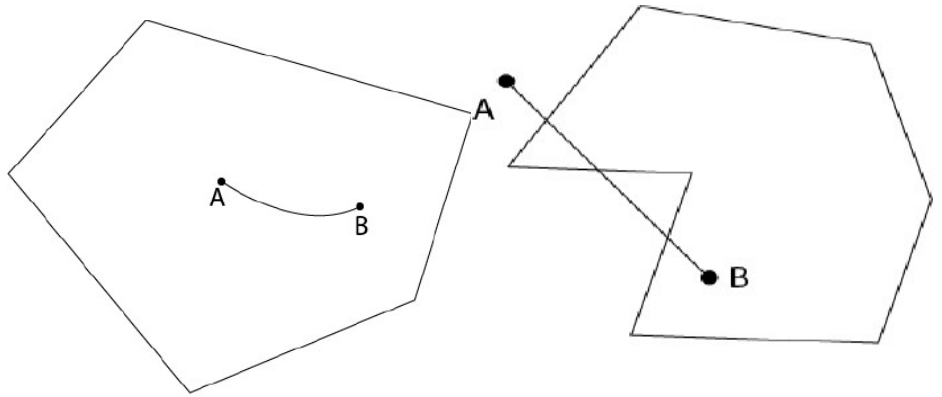
A polygon is called convex of line joining any two interior points of the polygon lies inside the polygon. A non-convex polygon is said to be concave. A concave polygon has one interior angle greater than 180°. So that it can be clipped into similar polygons.

- **1. Convex** if we take any two points inside of polygon and connect among by line and this line is fully inside the polygon.
- **2. Concave** if we take any two points inside of polygon and connect among by line and this line is not fully inside the polygon.

Note: *the head of polygon is called by* ***Vertices.***

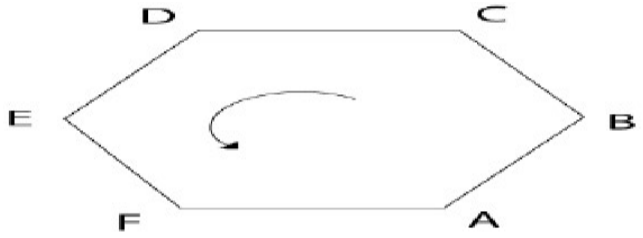
A polygon can be positive or negative oriented.

If we visit vertices and vertices visit produces counterclockwise circuit, then orientation is said to be positive

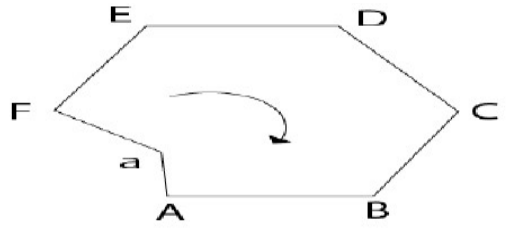


Convex polygon

Concave polygon



Polygon with positive orientation



Polygon with negative orientation

- If the sequences of capes of the polygon are anticlockwise called **positive orientation** otherwise called **Negative orientation** to know the point is inside the polygon we need:
- 1- Know the polygon positive orientation or Negative orientation.
- 2- Determine the point is inside of polygon by the equation:
- $C=(x2 - x1) (y - y1) - (y2 - y1) (x - x1)$
- If value of **C** is **positive** then point in left side otherwise is right side, if polygon is positive orientation then the point in left side is inside in polygon but if point in right side that is outside polygon and similar inverse of polygon is negative orientation.

EX/

Convex & Concave Windows

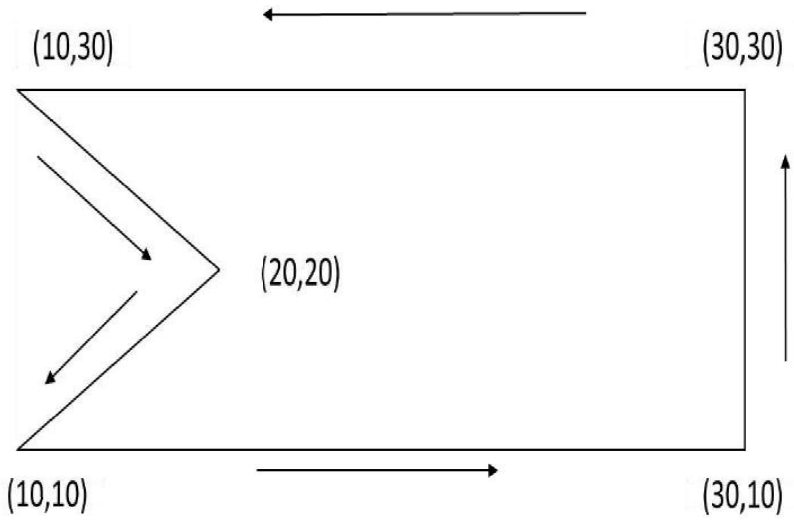
V1 = (x2 – x1 , y2 – y1) = (30 – 10 , 10 – 10) = (20,0)

V2 = (x2 – x1 , y2 – y1) = (30 – 30 , 30 – 10) = (0,20)

V3 = (x2 – x1 , y2 – y1) = (10 – 30 , 30 – 30) = (-20,0)

V4 = (x2 – x1 , y2 – y1) = (20 – 10 , 20 – 30) = (10,-10)

V5= (x2 – x1 , y2 – y1) = (10 – 20 , 10 – 20) = (-10,-10)



V1 * V2 = (20 * 20) – (10 * 0) = +400

V2 * V3 = (0 * 0) – (20 * -20) = +400

V3 * V4= (-20 * -10) – (0 * 10) = +200

V3 * V4= (-20 * -10) – (0 * 10) = +200

V4 * V5 = (10 * -10) – (-10 * -10) = -200

V5* V1 = (-10 * 10) – (-10 * 20) = +100

$$V1 \cdot V2 = \begin{vmatrix} 20 & 0 \\ 0 & 20 \end{vmatrix} = 400$$

$$V2 \cdot V3 = \begin{vmatrix} 0 & 20 \\ -20 & 0 \end{vmatrix} = 400$$

$$V3 \cdot V4 = \begin{vmatrix} -20 & 0 \\ 10 & -10 \end{vmatrix} = 200$$

$$V4 \cdot V5 = \begin{vmatrix} 10 & -10 \\ -10 & -10 \end{vmatrix} = -200$$

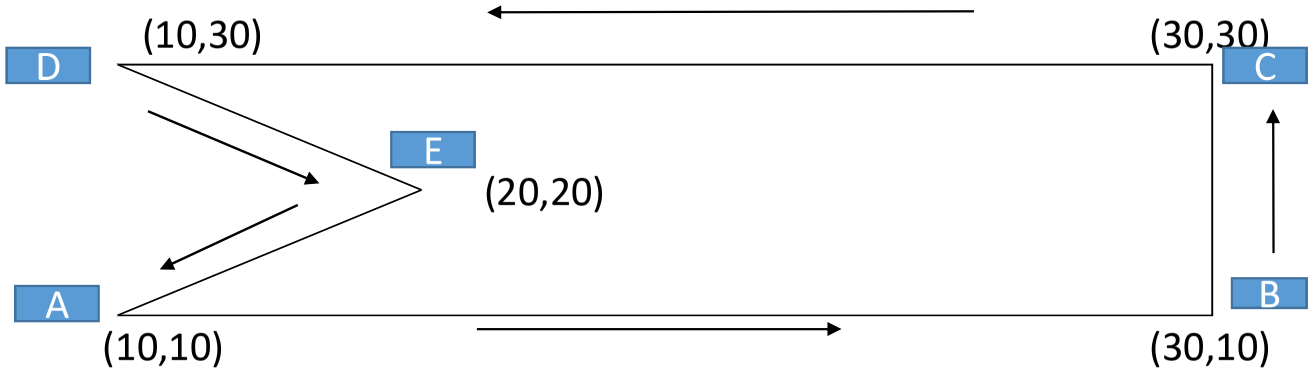
$$V5 \cdot V1 = \begin{vmatrix} -10 & -10 \\ 20 & 0 \end{vmatrix} = 200$$

طريقة الحل :-

١- اعط لكل رأس حرف

٢- اوصل كل زوج من الرؤوس

٣-اجراء تطبيق المعادلة

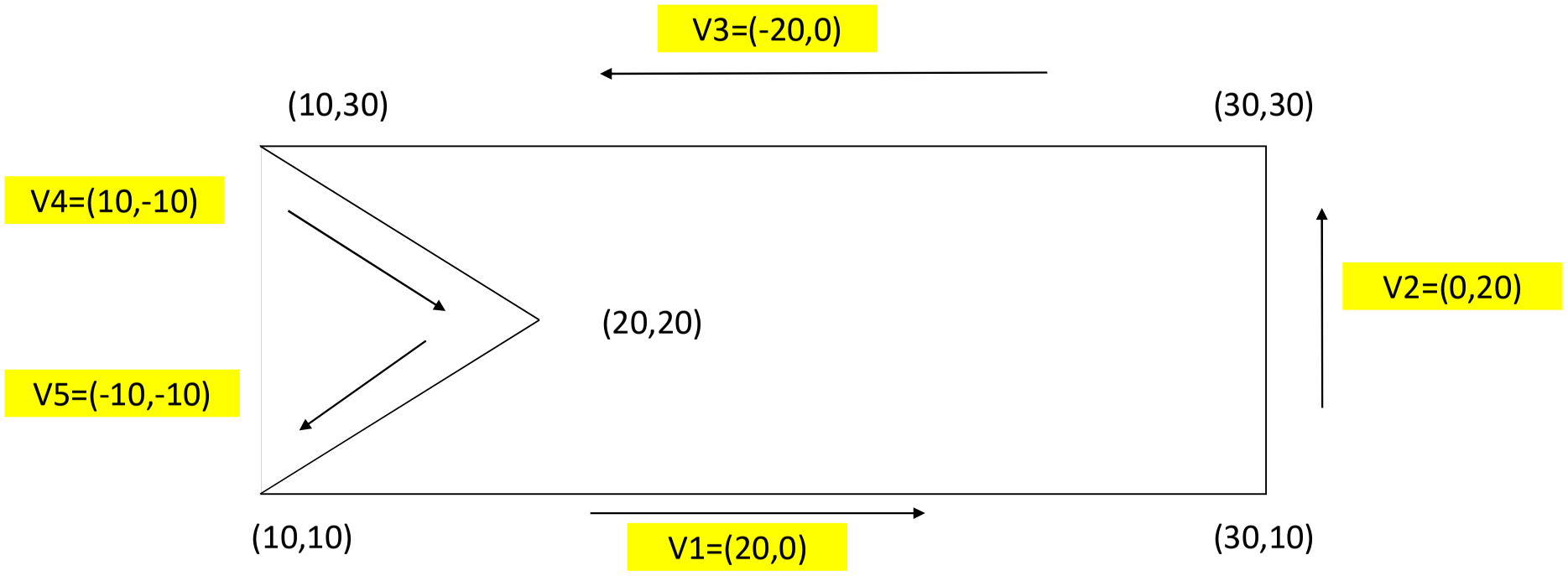


$$C=(x2-x1)(y-y1)-(y2-y1)(x-x1)$$

	X1	Y1	X2	Y2	X2-x1	Y2-y1	V	
AB	10	10	30	10	20	0	V1*V2	400
BC	30	10	30	30	0	20	V2*V3	400
CD	30	30	10	30	-20	0	V3*V4	200
DE	10	30	20	20	10	-10	V4*V5	-200
EA	20	20	10	10	-10	-10	V5*V1	100

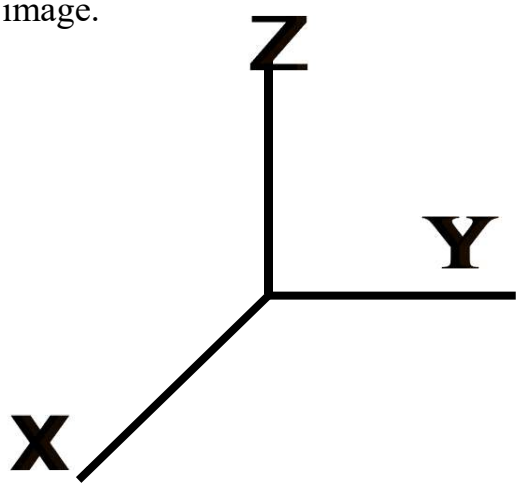
$$V1*V2=\begin{matrix}20 & 0\\ 0 & 20\end{matrix} \quad =(20*20)-(0*0)=400$$

الحل النهائي



Three-dimensional Transformation

- The world composed of three-dimensional images.
- Objects have height, width, and depth.
- The computer uses a mathematical model to create the image.



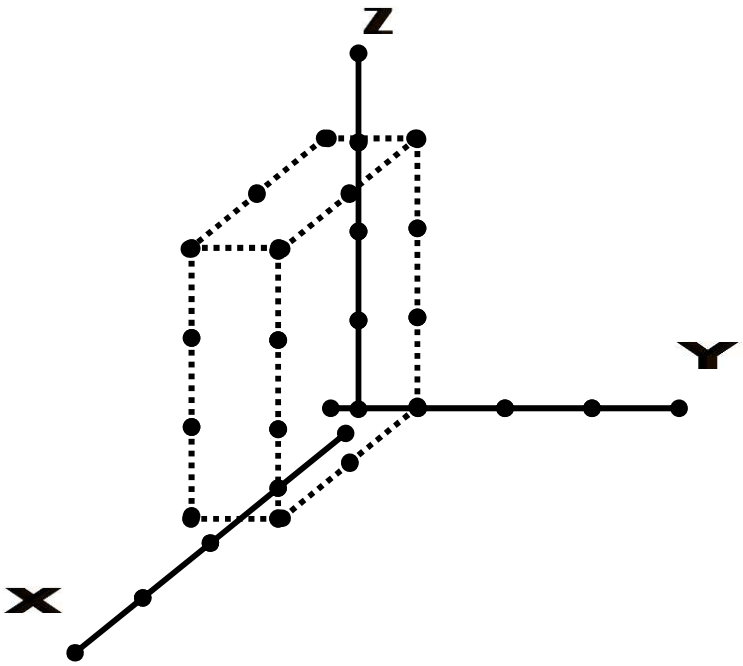
1-:Coordinate System:

A three dimensional coordinate system can be view as an extension of the two dimensional coordinate system.

The third-dimension depth is represented by the Z-axis which is at right angle to the x, y coordinate plane.

A point can be described by triple (x, y, z) of coordinate values

Ex./ Draw the figure: (0,0,3), (0,1,3), (2,0,3), (2,1,3), (0,1,0) (2,0,0), (2,1,0)



2: Transformation:

Transformations of 3 dimensions are simply extension of two-dimension transformation.

A three-dimensional point (x, y, z) will be associated with homogeneous row vector [x, y, z, 1].

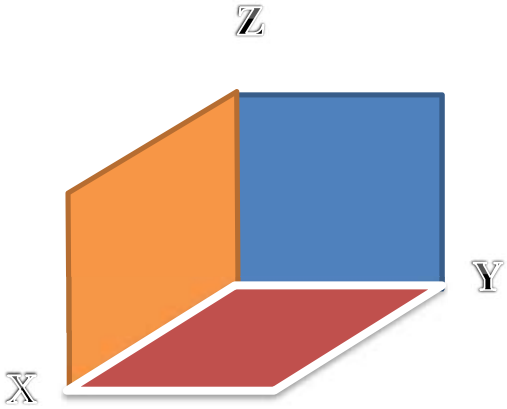
We can represent all three-dimensional linear transformation by multiplication of 4*4 matrixes.

2.1 Translate (shift, Move)

The new coordinate of a translate point can be calculate by using transformation.

$$T: \begin{cases} \underline{X} = X + a \\ \underline{Y} = Y + b \\ \underline{Z} = Z + c \end{cases}$$

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}$$



2.2:Scaling:

- Allows for a contraction or stretching in any of the x, y, or z direction. To scale an object:
    1. Translate the fixed point to the origin.
    2. Scale the object.
    3. Perform the inverse of the original translation.
- The scaling matrix with scale factors Sx, Sy, Sz in x, y, z direction is given by the matrix

And see that matrices are as follows. The window shift is given by

$$S: \begin{cases} \underline{X} = Sx * X \\ \underline{Y} = Sy * Y \\ \underline{Z} = Sz * Z \end{cases}$$

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3 Reflection

About origin: (X, Y, Z)➔ (-X, -Y, -Z)

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Mirror about Main Axes**
  - **X-axis: (X, Y, Z)➔ (X, -Y, -Z)**

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - **Y-axis: (X, Y, Z)➔ (-X, Y, -Z)**

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - **Z-axis: (X, Y, Z)➔ (-X, -Y, Z)**

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Mirror about Main Plane**
  - **Plane XY: (X, Y, Z)➔ (X, Y, -Z)**

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - **Plane YZ: (X, Y, Z)➔ (-X, Y, Z)**

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - **Plane XZ: (X, Y, Z)➔ (X, -Y, Z)**

$$[\underline{X}, \underline{Y}, \underline{Z}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.4: Shearing about main plane therefore shear 3D are:-

- **Shear XY ➔**

$x^{sh} = x + Shx*z$

$y^{sh} = y + Shy*z$ ➔ $[X^{sh}, Y^{sh}, Z^{sh}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ shx & shy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$z^{sh} = z$

- **Shear XZ ➔**

$x^{sh} = x + Shx*y$

$y^{sh} = y$ ➔ $[X^{sh}, Y^{sh}, Z^{sh}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ shx & 1 & shz & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$z^{sh} = z+ Shz*y$

- **Shear YZ ➔**

$x^{sh} = x$

$y^{sh} = y + Shy*x$ ➔ $[X^{sh}, Y^{sh}, Z^{sh}] = [X\ Y\ Z\ 1] \times \begin{bmatrix} 1 & shy & shz & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$z^{sh} = z+ Shz*x$

Note:

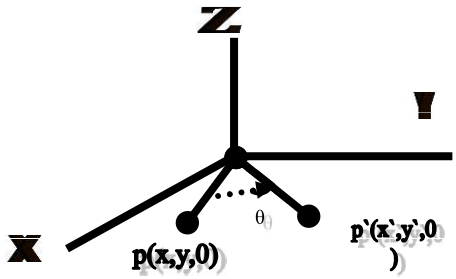
- if shear for example on plane XY is -3**, therefore shx= -3, shy= -3**
- **if** shear on z by -2 and shear on y by 5, therefore this shear at plane YZ and **shy= 5, shz= -2**
- **if it apply shear directly then center of shearing (0,0,0), but if center shearing not (0,0,0) need**
  a) **Shift center (Xc, Yc, Zc) into (0, 0, 0) by shifting transform.**
  b) **Apply shearing transform (or Scaling transform)**
  c) **Inverse step a (return center in the location (Xc, Yc, Zc))**
  d) **These step (a, b, c) apply in scaling transform.**

2.5 Rotation:

Rotation in three dimensions is considerably more complex than rotation in two dimensions. In two dimensions, a rotation is prescribed by an angle of rotation θ and center of rotation p.

Three dimensional rotations require the prescription of an angle of rotation and an axis of rotation.

The canonical rotations are defined when one of the positive x, y, or z coordinate axes is chosen as the axis of rotation. Then the construction of the rotation transformation proceeds just kike that of a rotation in two dimensions about the origin see figure above.



Rotation about the X-Axis **R(X, θ)**

$$X^r = X$$
$$Y^r = Y\cos(\theta) - Z\sin(\theta)$$
$$Z^r = Z\cos(\theta) + Y\sin(\theta)$$

1	0	0	0
0	Cos(θ)	Sin(θ)	0
0	-Sin(θ)	Cos(θ)	0
0	0	0	1

Rotation about the Y-Axis **R(Y, θ)**

$$X^r = X\cos(\theta) - Z\sin(\theta)$$
$$Y^r = Y$$
$$Z^r = Z\cos(\theta) + X\sin(\theta)$$

Cos(θ)	0	Sin(θ)	0
0	1	0	0
-Sin(θ)	0	Cos(θ)	0
0	0	0	1

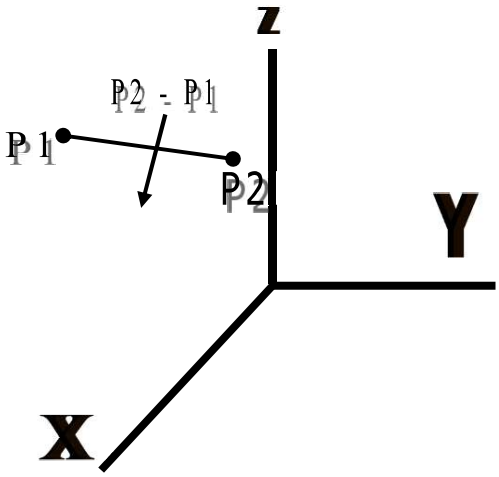
Rotation about the Z-Axis **R(Z, θ)**

$$X^r = X\cos(\theta) - Y\sin(\theta)$$
$$Y^r = Y\cos(\theta) + X\sin(\theta)$$
$$Z^r = Z$$

Cos(θ)	Sin(θ)	0	0
-Sin(θ)	Cos(θ)	0	0
0	0	1	0
0	0	0	1

note that the direction of positive angle of rotation is chosen in accordance to the right-hand rule with respect to the axis of rotation.

The general use of rotation about an axis ***L*** can be built up from these canonical rotations using matrix multiplication in next section.



2.6: <u>Rotation about an arbitrary Axis</u>

- It is like a rotation in the two-dimension about an arbitrary point but it is more complicated.
- Two points P1(x1, y1, z1) and P2(x2, y2, z2) Define a line.

The equation for the line passing through these Point are :

x= (x2 - x1) t + x1

y= (y2 - y1) t + y1 t: real value [0 to 1]

z= (z2 – z1) t + z1

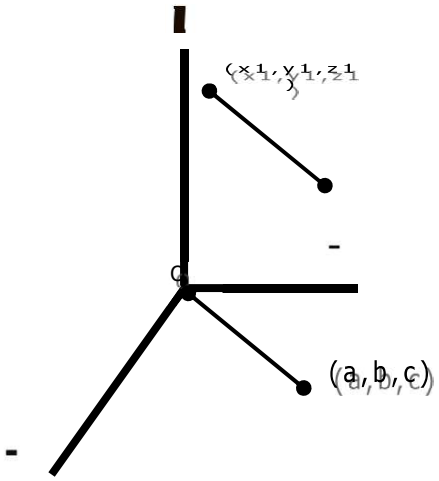
- Let a=(x2 – x1) & b=(y2 – y1) & c=(z2 – z1) then the equation of line becomes x=at + x1 & y=bt +y1 & z=ct + z1 the difference P2 – P1 = (x2 – x1) (y2 – y1) (z2 – z1) = (a, b, c) is the direction vector from P1 to P2 along the line through P1 and P2.

<u>Steps of rotation:</u>

Let (x1, y1, z1) be a point through which the rotation axis passes with (a, b, c) direction. A rotation of angle θ about an arbitrary axis is:

1. ***Translate the point(x1, y1, z1) to origin.***

$$\mathbf{Tr(-x1, -y1, -z1)=}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x1 & -y1 & -z1 & 1 \end{bmatrix}$$



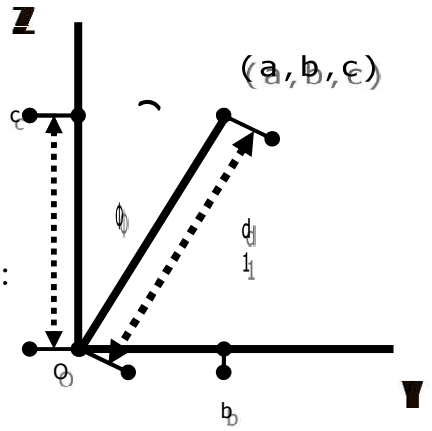
After this translation the direction vector (a, b, c) define the rotation axis as follows.

2. Rotate about the x-axis until the rotation axis corresponds to the z-axis.

This can be considering being a rotation about the origin. With the axis coming out of paper When the rotation axis is projected onto the x,z plane,

any point on it has x coordinate equal to zero. In particular a=0.

The point (0,b,c) is rotated **Φ** degree until the line corresponds to the z-axis. We have find the sin **Φ** and cos **Φ** we find that distance from the origin to (0,b,c) is : $\sqrt{b^2 + c^2}$= **d1**



Sin Φ= b/d1, Cos Φ= c/d1

Substituting these values into the x-axis rotation matrix we have:

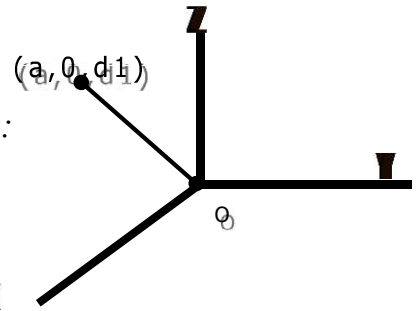
$$R(X, \Phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d1 & b/d1 & 0 \\ 0 & -b/d1 & c/d1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now the point(0,b,c)has been transformed to the point (0,0,d1) but since the rotation about the x-axis doesn't change the x coordinate value the point (a, b, c) is now at location (a, 0, d1).

3. ***Rotate about the y-axis until the rotation axis corresponds to the z-axis.***

Since (a, 0, d1) lies in the x, z plane we can visualize this as rotation about the origin with the y-axis coming out of the paper.

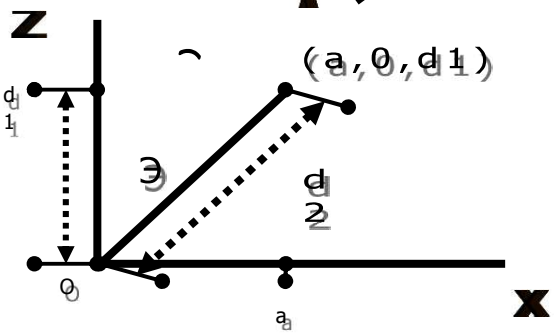
A rotation of angle **Э** in clockwise direction, we need to compute *sin Э, cos Э where:* $d2 = \sqrt{a^2 + (d1)^2} = \sqrt{a^2 + b^2 + c^2}$ *thus:*



sin Э = a/d2 ; cos Э = d1/d2

Substituting the value into y rotation matrix given:

$$R(y, Э) = \begin{bmatrix} d1/d2 & 0 & a/d2 & 0 \\ 0 & 1 & 0 & 0 \\ -a/d2 & 0 & d1/d2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



4. ***Rotate about the z-axis angle β.*** This require the Rz(***β***) matrix

$$R(Z, \beta) = \begin{bmatrix} cos(\beta) & sin(\beta) & 0 & 0 \\ -sin(\beta) & cos(\beta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5. ***Perform the inverse rotation of step (3) .*** requires Ry(-Э)

$$\mathbf{R(y, -Э)} = \begin{bmatrix} d1/d2 & 0 & -a/d2 & 0 \\ 0 & 1 & 0 & 0 \\ +a/d2 & 0 & d1/d2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6. ***Perform the inverse rotation of step (2).*** Requires Rx(-Φ)

$$\mathrm{R(X, -\Phi)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d1 & -b/d1 & 0 \\ 0 & +b/d1 & c/d1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

7. ***Perform the inverse translation of step (1).*** Require Tr (x1,y1,z1)

$$\mathbf{Tr(+x1, +y1, +z1)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ +x1 & +y1 & +z1 & 1 \end{bmatrix}$$

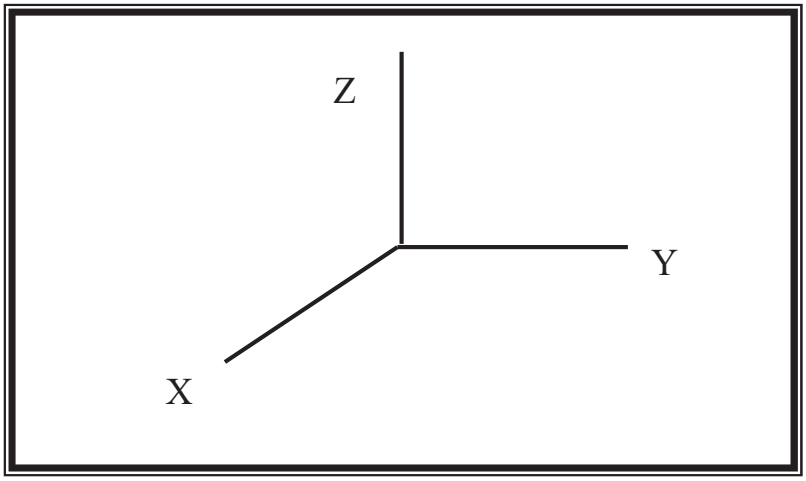
6- THREE – DIMENSIONAL TRANSFORMATIONS

The world composed of three – dimensional images so the object has height, width and depth. The computer uses a mathematical model to create the image.

6.1 Coordinate system:

A three- dimensional coordinate system can be view as an extension of the – two –-dimension coordinate system.

The third – dimension depth is represented by the **Z – axis** which is at right angle to the **X , Y** coordinate plane. A point can be described by triple **(X , Y , Z)** of coordinate values.



6.2 The Transformations:

a. Translation :-

A point (x , y , z) is translated to a new position (x_1 , y_1 , z_1) by move it dx units in the X – direction and by units in the Y – direction and dz units in Z – direction. Mathematically this can be represented as:-

$$X_1 = X + dx$$
$$Y_1 = Y + dy$$
$$Z_1 = Z + dz$$

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{bmatrix}$$

b. Scaling :-

Increasing the distance, between the points describing the object can make an object. In general, we can change the size of an object, or the entire image, by multiplying the distance between points by an enlargement or reduction factor. This factor is called the **scaling factor,** and the operation that the size is called **scaling**. If the scaling factor is greater than 1, the object is enlarge, if the factor is less than 1, the object is made smaller, a factor of 1 has no effect on the object. Whenever scaling is performed, there is one point that remains at the same location. This is called **fixed point** of the scaling transformation.

a) To scale an object from a origin point:-

We used the following matrix.

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b) To scale an object from fixed point (x_p , y_p , z_p), we perform the following three steps:

1. Translate the point (x_p , y_p , z_p) to the origin. Every point (x , y , z) is moved to a new point (x_p , y_p , z_p):

$$x_1 = x - x_p$$
$$y_1 = y - y_p$$
$$z_1 = z - z_p$$

2. Scale these translate points with the origin as the fixed points:

$$x_2 = x_1 \times S_x$$
$$y_2 = y_1 \times S_y$$
$$z_2 = z_1 \times S_z$$

3. Translate the origin back to the fixed point (x_p , y_p , z_p):

$$x_3 = x_2 + x_p$$
$$y_3 = y_2 + y_p$$
$$z_3 = z_2 + z_p$$

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_p & -y_p & -z_p & 1 \end{bmatrix} \times \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_p & y_p & z_p & 1 \end{bmatrix}$$

c. Rotation :-

1. Rotation about X – axis:

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

2. Rotation about Y – axis:

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

3. Rotation about Z – axis:

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

Example :

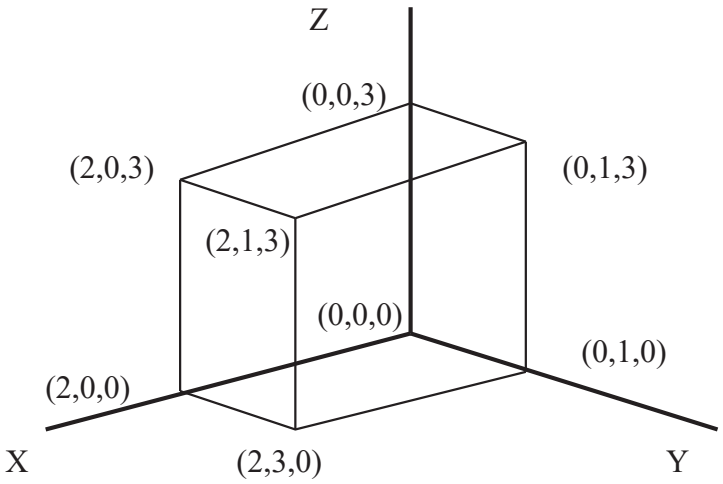
Draw the figure (0 , 0 , 0) , (0 , 1 , 0) ,(0 , 1 , 3) , (0 , 0 , 3) , (2 , 0 , 0) , (2 , 1 , 0), (2 , 0 , 3) , (2 , 1 , 3), and find: **Not: sin (90) = 1 , cos (90) = 0.**

a) Translate it to the point (0 , 3 , 0).

b) Scaling 4 times its size about the origin point.

c) Rotate its (90^o) about the Z – axis.

Solution:



a) Translate it to the point (0 , 3 , 0).

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 0 & 3 & 1 \\ 2 & 1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 1 \\ 0 & 3 & 0 & 1 \\ 0 & 4 & 3 & 1 \\ 0 & 3 & 3 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 6 & 0 & 1 \\ 2 & 3 & 3 & 1 \\ 2 & 4 & 3 & 1 \end{bmatrix}$$

b) Scaling 4 times its size.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 0 & 3 & 1 \\ 2 & 1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 4 & 0 & 1 \\ 0 & 4 & 12 & 1 \\ 0 & 0 & 12 & 1 \\ 8 & 0 & 0 & 1 \\ 8 & 12 & 0 & 1 \\ 8 & 0 & 12 & 1 \\ 8 & 4 & 12 & 1 \end{bmatrix}$$

c) Rotate its (90^o) about the Z – axis.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 0 & 3 & 1 \\ 2 & 1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(90) & \sin(90) & 0 & 0 \\ -\sin(90) & \cos(90) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & 2 & 0 & 1 \\ -3 & 2 & 0 & 1 \\ 0 & 2 & 3 & 1 \\ -1 & 2 & 3 & 1 \end{bmatrix}$$

H . W (11):

Draw the figure A (4 , 4 , 0) , B (-3 , 3 , 4) , C (-2 , 3 , 3), D (3 , -3 , 4) , E(3 , -2 , 3) and find:- **Not: sin (180) = 0 , cos (180) = -1**

a) Translate above shape to point (-3 , 4 , 3).

b) Scaling the figure, twice in X direction, three in Y direction and once in Z direction.

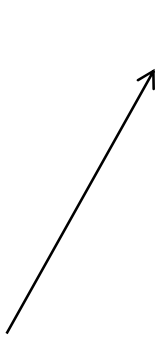
c) Rotate the figure (180^o) about X – axis.

Vectors:

A vector has a single direction and a length. A vector may be denoted [Dx, Dy] where Dx indicates how far to move along x-axis direction and Dy indicates how far to move along the y-axis.

These vector two types:

A) Free Vector: is shown by direction line segment whose length is a measure of its magnitude. Such as line AB, show figure 1 :



Anchor Vector:

Its starting position at the origin (0, 0) in particular direction, then we have a position vector. The position vector p=OP where p is specifying the position point with O is respected to the origin.

show in figure 2 below:

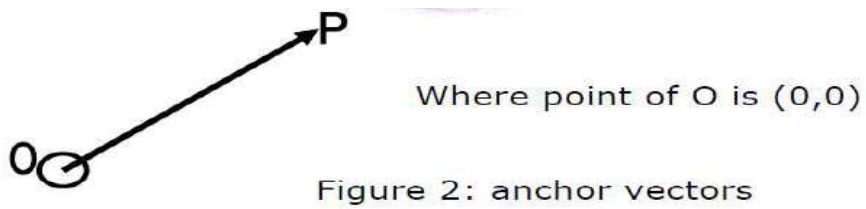


Figure 2: anchor vectors

Unit vector:

it is a vector whose size is "1", so it can be represented by an arrow of length 1. a unit vector in the direction of the X-axis is called i ,and a unit vector in direction of the Y-axis is called j.
Note: we can write vector in this lecture either point such as P(5,2) or vector row p(5 2) or component form OP=5i+2j.
We note that i=(1 0) and j=(0 1) see figure 3

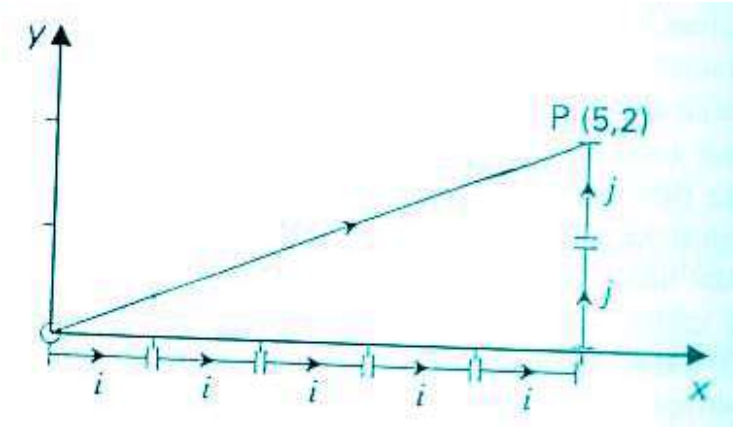


Figure 3: show that combine the unit vector i and j

measurement associated with vectors:

this following example of vector OP and OQ, we have shown above the there are different way to write OP & OQ either point or row vector or component form.
2.3.1 modulus of a vector: the modulus of a vector is given by the length of the arrow by using find length of line & term the modules of vector p is |OP|.

Example: if p(5,2) & Q(2,-4) in figure below to find modulus of two vector are:
|OP|= 5^2 +2^2 = 29 = 5.39
And |OQ| = 2^2+ (-4) ^2 = 20= 4.47

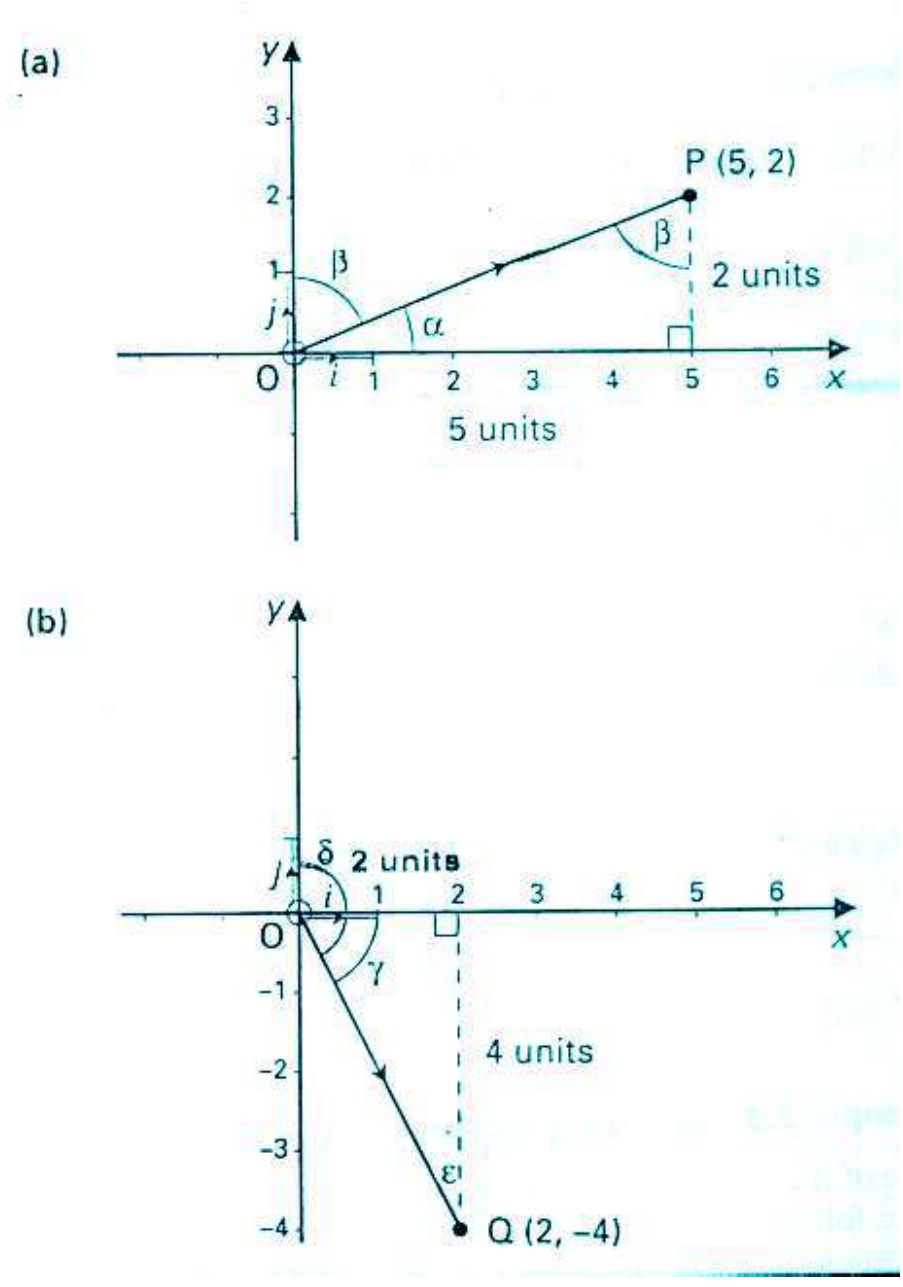


Figure 4: example vectors p,q

unit vectors: the unit vector in direction of OP is written OP, which is calculated as following : OP = vector OP/modulus of OP. in preview example then OP = OP/|OP|
OP/|OP| = (5i+2j)/5.39 = (5i/5.39) + (2j/5.39) = 0.93i+0.37j
Similarly we have OQ = OQ/|OQ| = (2i-4j)/4.47 = 0.45i-0.89j
2.3.3 Angles between vectors and axis

1. The angle between OP and i (X-axis) is ά, where tan ά =2/5=0.4 so ά =21.80.

2. The angle between OP and j (y-axis) is β, where β=90-21.80 = 68.20.

manipulation vectors:

The system of vectors, which we have introduced above, will be seen to give us the terminology and techniques for dealing with point, line, angle and surface too, as will be seen later. Moreover, as this system deals with geometrical quantities in numerical terms, it is extremely valuable for computer technology. To exploit their potential we must be able to manipulate vectors, so next we look at aspects of adding and subtracting vectors and "scaling" them by a number.
:adding vectors let see figure below two vector p, q

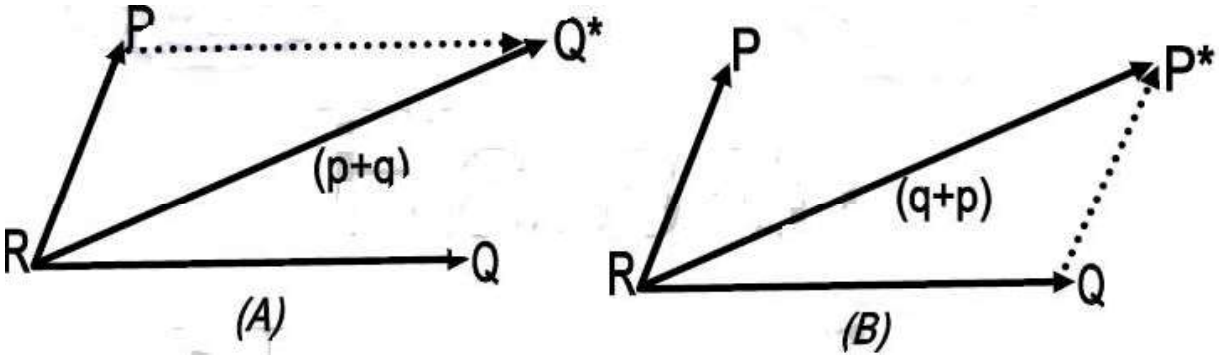
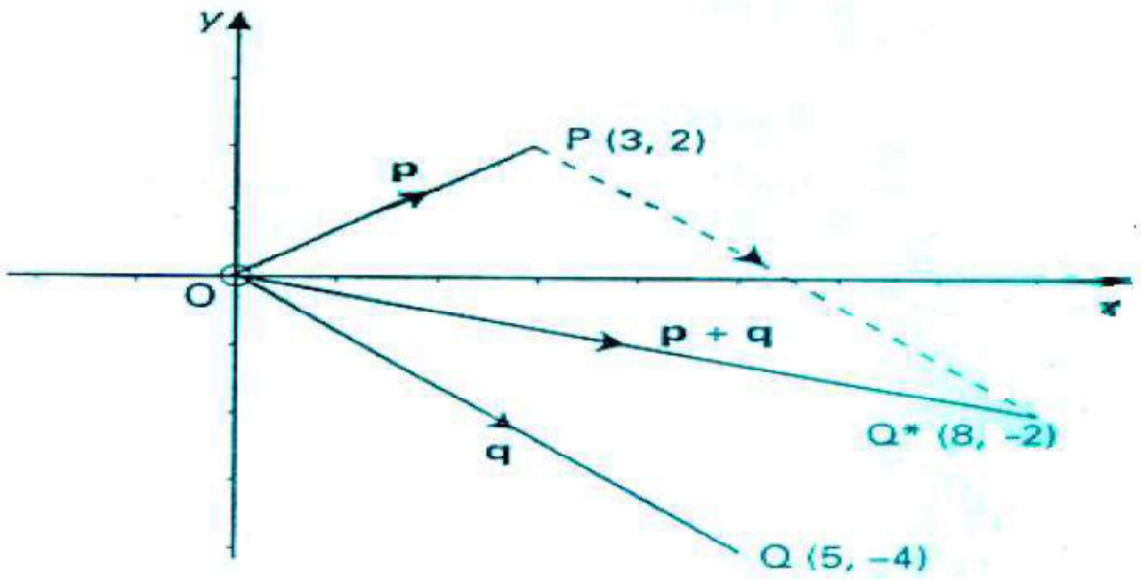


Figure 5 show adding two vectors (A) p+q, (B) q+p

Example: let vector p (3, 2) and Q (5,-4) find p+q and drowning.
Sol. / P=3i+2j and q=5i-4j
p+q= (3i+2j) + (5i-4j) =3i+2j+5i-4j=8i-2j (in component form)

p= (3 2) and q=(5 -4) then p+q =(3 2) + (5 -4)
= (3+5) (2-4)= (8 -2) (in row vector)
Show in figure 6 the result p+q below:



negative vectors and subtracting vectors:

-q is a negative vector, by which we mean a vector with the same magnitude as q but opposite direction, as shown figure 7. we are able calculate p-q by finding p+(-q): we reverse the direction of q and starting from Q, we then use triangle addition.

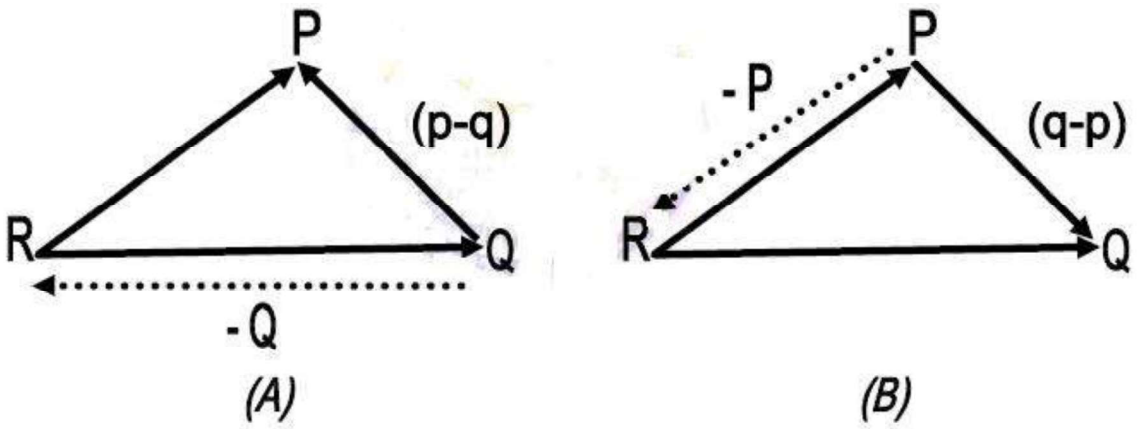


Figure 7 show drawing triangle in (A) p-q (B) q-p

Example: let p(3,2) and Q(5,-4) and drowning this vectors.

Sol. / in component form or row vector

(In component form)

p=3i+2j and q=5i-4j
p-q=(3i+2j)-(5i-4j)
=3i+2j-5j+4j
=-2i+6j

(In row vector)

p=(3 2) and q=(5 -4)
p-q=(3 2)-(5 -4)
= (3-5) (2+4)
= (-2 6)

Note: if we wish to determine the vector between any two point whose coordinate are known, then we use vector subtraction. Suppose we require the vector p(2,7) to Q(4,10): we note that direction matters, it is PQ that we want. First we identify the position vector OP=p and OQ=q, and then we use these to calculate the vector PQ using triangle addition in triangle OPQ: PQ=PO+OQ= -p+q = -(2 7)+(4 10)= (2 3). Similarly we can show that QP=(-2 -3), which is as expected since this is the negative of the vector PQ.

scaling Vectors:

we can scale any vectors by multiplying it by scalar number (just a number). To scale it by 3 "we make it three times bigger in the same direction, that is we multiply the vector by 3. that p is (4 3) as in figure 8 we have 3p=3(4 3)=(12 9), similarly a scale factor 1/2 in same direction (1/2)p=1/2 (4 3) = (2 1(1/2)).
A negative scale factor reverses the direction of a vector. When the scale factor is -1, than effect is just to reverse the vector, so that it has the same modulus but the opposite direction. If a negative scale factor is other -1, then alters the modulus of the vector as well. Thus referring to the same vector p in figure 8 : (-2)p=-2(4 3)=(-8 -6) the effects of these scalings are indicated in figure 8.

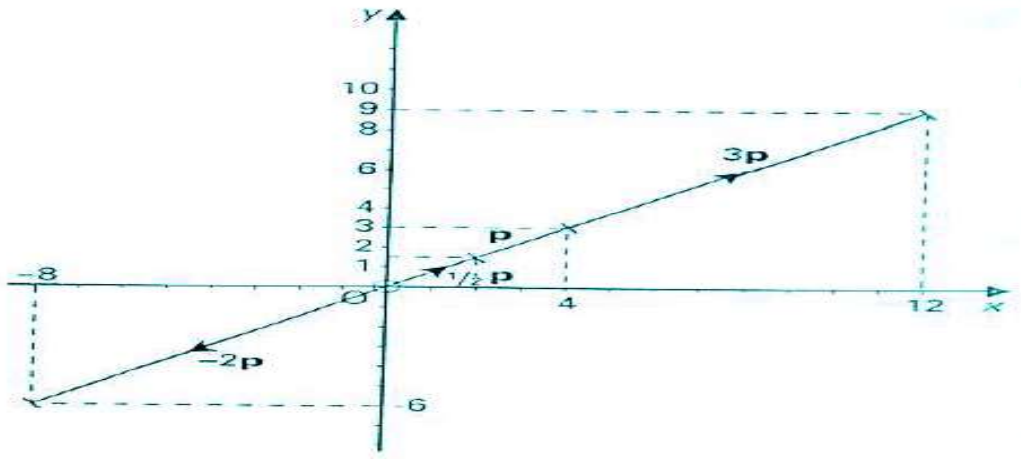


Figure 8: scaling vector.

multiplying vectors uses the "dot Product"

purpose of dot product {Direction Cosine} in computer graphics we shall use this product as a way of finding the angle between two vectors, and also of showing when two vectors are perpendicular. We define their dot product by $a.b=|a|\,|b|\cos\theta$;

where ɵ is $0<=\theta<=180$, and therefore to find angle between of two vectors is :

$\cos\theta = (a.b) / (|a|\,|b|)$

note: if dot product =zero implies either that at least one vector is the zero or that the vector are perpendicular.

We can easy calculating dot product a.b in 2D to two vectors a&b is: a.b= (a1 * b1) + (a2 * b2); where a=a1i + a2j & b= b1i + b2j

Example: calculate the angle between the vector a=3i+5j and b=2i+j as shown in figure 9 below:

Figure 9: example to find angle ɵ

Sol. / a . b = |a| |b| cos ɵ

|a|= 3^2 +5^2 = 34 And |b| = 2^2+1^2 = 5

a . b = (a1*b1) + (a2*b2) = (3*2) + (5*1) =11

Cos ɵ = (a. b)/ (|a| |b|) = 11/ (34 * 5) = 0.8437; then ɵ=32.47

Note: *then other multiplying vector is* ***"cross product"*** *this type is used in vector in 3D. These subject is useful in reach lecture 3D.*

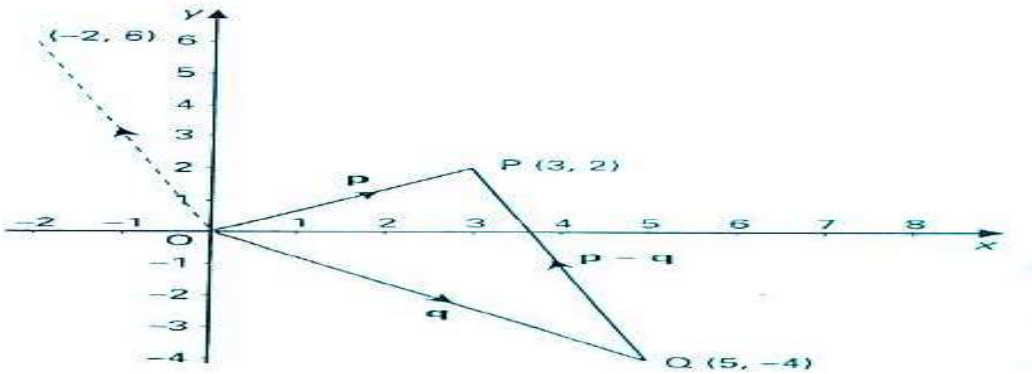


Figure 10: drowning of example in p-q