Computer Graphics

Third Class – Department of computer Science

By

Israa A. Alwan

2021-2022

Introduction to Computer Graphics

Graphics is defined as any sketch or a drawing or a special network that pictorially represents some meaningful information. **Computer Graphics** is used where a set of image needs to be manipulated or the creation of the image in the form of pixels and is drawn on the computer. Computer Graphics is the process of transforming and presenting the information in a visual form.

Definition of Computer Graphics:

It is the use of computers to create and manipulate pictures on a display device. It comprises of software techniques to create, store, modify, represents pictures.

Applications of Computer Graphics

- **Graphical User Interface (GUI):** It is a way of interacting with a computer using the icon, menu, and other visual, graphics by which user easily interacts.
- **Entertainment:** Computer graphics allow the user to make animated movies and games. Computer graphics are used to create scenes. Computer graphics are also used for special effects and animations.
- **Engineering Drawings:** Computer Graphics has also provided us the flexibility to make 3D models, house circuits and engineering drawings, etc. which is helpful for us.
- **Education and Training:** Computer graphics are also used to provide training to students with simulators. The students can learn about the machines without physically trying them.
- **Medical Imaging:** MRIs, CT scans, and other internal scans are possible because of computer graphics.
- **Flight Simulator:** Computer graphic is used to provide training to pilots of aircraft. The pilots give much time to a flight simulator on the ground instead of real airplanes.
- **Printing Technology:** Computer graphics are used in textile designing and flex printing.
- **Satellite Imaging:** Computer graphics are used to forecast the movement of the cloud and to predict the weather.
- **Cartography:** Computer graphics are used in map drawing.

Advantages of Computer graphics

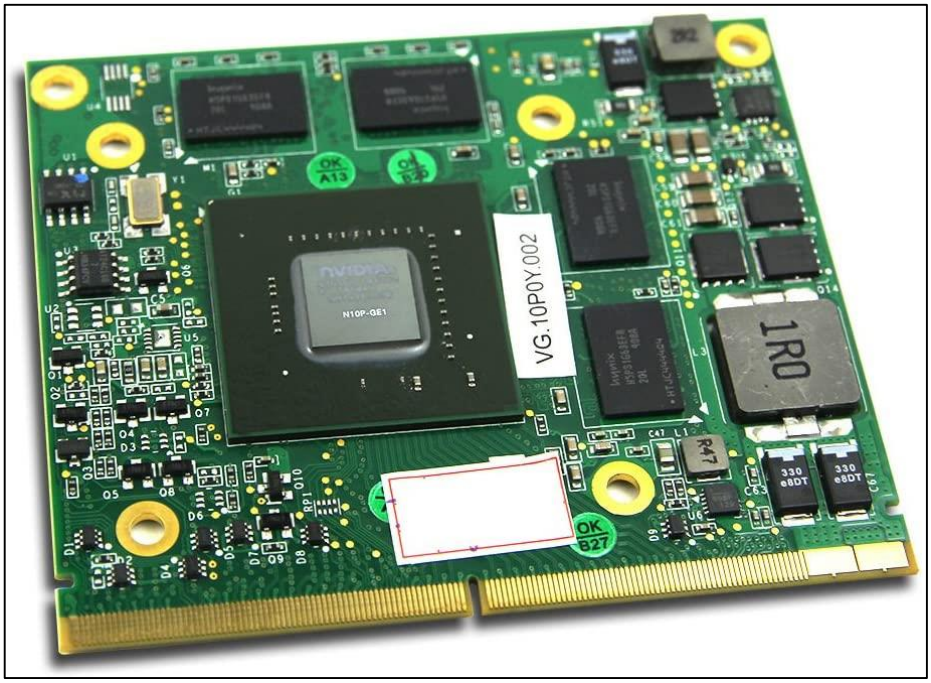
- Increase Productivity.
- Computer graphics give us tools for creating pictures of solid objects as well as of theoretical, engineered objects.
- The computer can store complex drawings and display complex pictures.

Disadvantages of Computer graphics

- Hardware characteristics and cost.
- Technical issues.

Display Adapter

A plug-in card is what your computer uses to convert data in your machine to useful images on your screen. Also commonly called a **"graphics card"** or **"video card"**. The display adapter determines the maximum resolution and number of colors that can be displayed, which the monitor must also be able to support. On most PCs, these graphics circuits are built into the motherboard's chipset. A separate plug-in card is required only to greatly enhance rendering for video games or other fast-motion graphics applications. The display adapter has several types, some of them are shown below:



Graphics Video Card.

Computer Display Standard	
CGA	**C**olor **G**raphics **A**dapter
EGA	**E**nhanced **C**olor **A**dapter
VGA	**V**ideo **G**raphic **A**rray
SVGA	**S**uper **V**ideo **G**raphic **A**rray
XGA	**E**xtended **V**ideo **G**raphic **A**rray

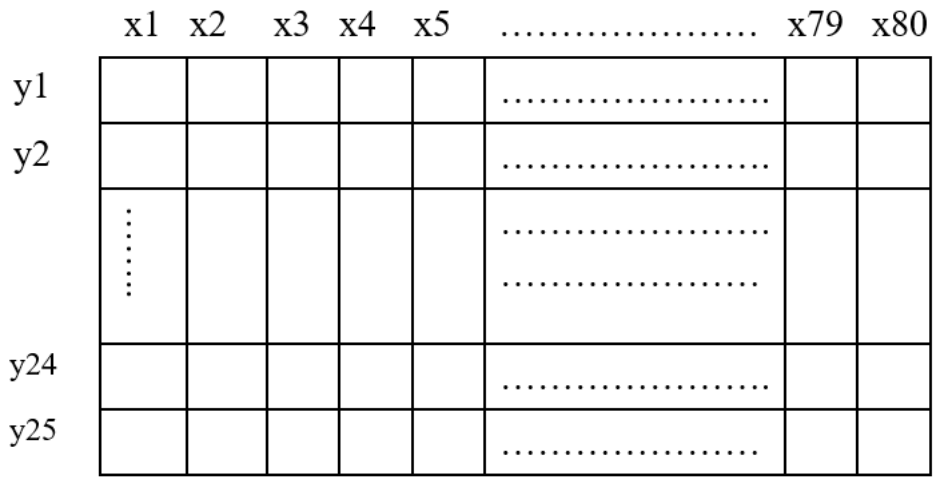
Note: display adapter, graphics card, display card, video adapter, video card, graphics adapter, graphics controller, VGA adapter and VGA card have all been terms for the plug-in board that creates the screen images.

PC Display Modes

A PC **graphics card** supports both **Text and Graphics modes:**

- **Text Mode**

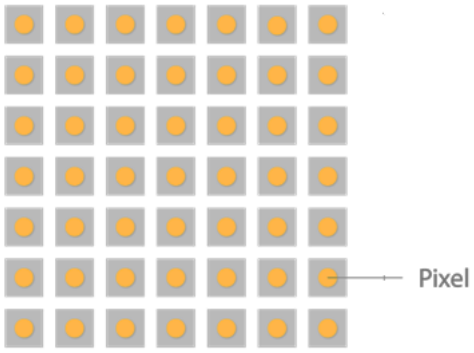
  Alternatively known as **character mode** or **alphanumeric mode**, **text mode** is a display mode divided into rows and columns of boxes showing only alphanumeric characters. Each box can contain one character. Basic unit is the **character cell**. Its screen, with coordinates (1 , 1 , 80 , 25).



Computer screen with Text Mode.

- **Graphics Modes**

  A way of displaying images on a computer screen or other graphics device such that the **basic unit is the pixel** (generates image using pixels). **Pixels**, **dots, or picture elements**, are the smallest **addressable** physical points on a display, as well as the base components. Pixels are therefore the building blocks of any image you see on your screen. Lines and characters on the screen are drawn pixel by pixel.

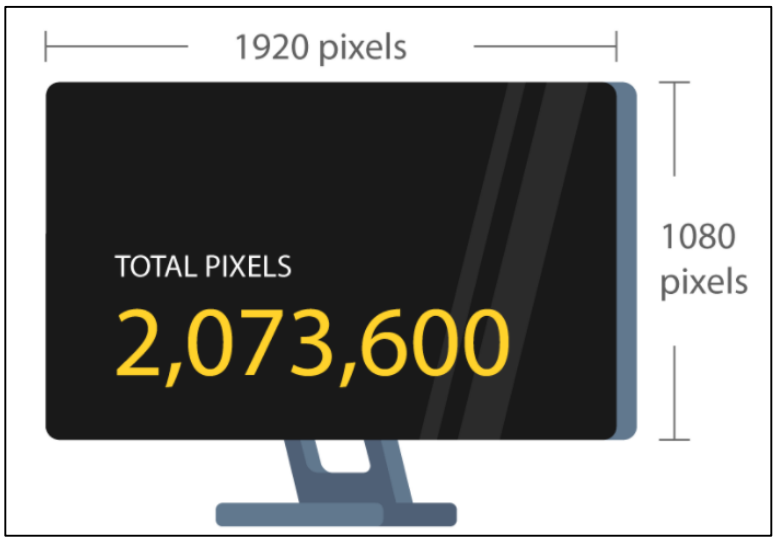


Computer screen with Graphics Mode.

Windows PCs boot up in text mode and switch to graphics mode. Many DOS applications supported both modes and switched between them based on the function selected by the user.

Resolution

The display resolution of a digital television, computer screen or display device is the number of distinct **pixels** in each dimension that can be displayed. It can be an ambiguous term especially as the displayed resolution is controlled by different factors in **cathode ray tube** (CRT) displays, **flat-panel displays** (including liquid-crystal displays) and **projection displays** using fixed picture-element (pixel) arrays. It is usually quoted as width × height, with the units in pixels: for example, 1920 × 1080 means the width is 1920 pixels and the height is 1080 pixels.

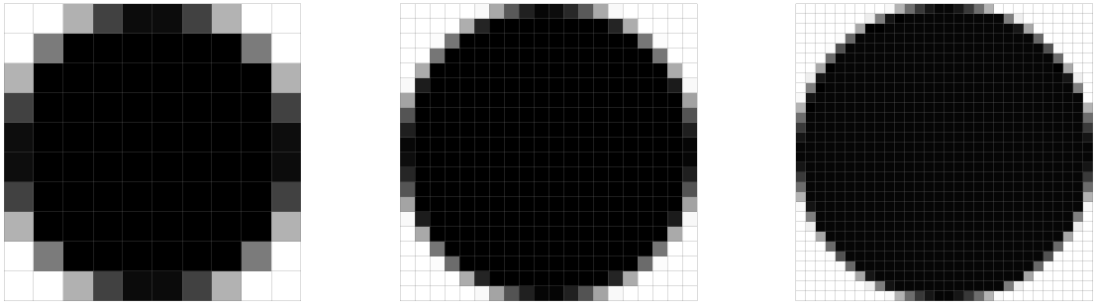


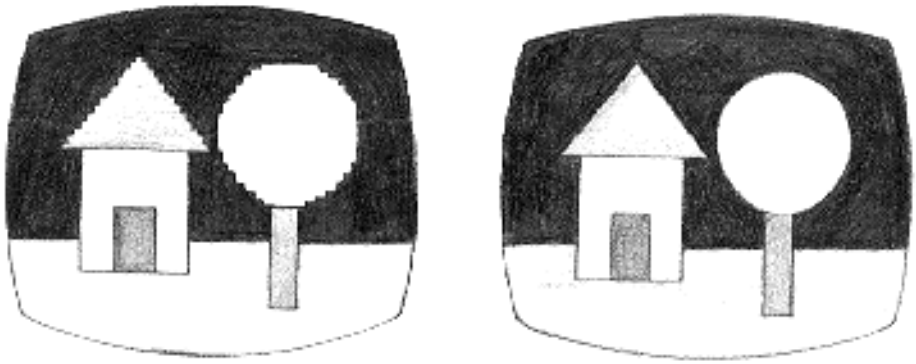
Screen Resolution.

Resolution of Computer Display Standard		
CGA	**C**olor **G**raphics **A**dapter	300 x 200
EGA	**E**nhanced **C**olor **A**dapter	320 x 200
VGA	**V**ideo **G**raphic **A**rray	640 x 480
SVGA	**S**uper **V**ideo **G**raphic **A**rray	800 x 600
XGA	**E**xtended **V**ideo **G**raphic **A**rray	1024 x 768

Note: Pixels and resolution are directly correlated and a higher resolution equals a higher number of pixels on a screen. To visualize this, we can think of pixels as puzzle pieces; each one makes up a small piece of a bigger picture. Moreover, the more pixels a screen has, the more detailed images can be. The more bits per pixel, the more different colors or shades of gray.

Note: In computer graphics, aliasing is the **stair- stepped** appearance of diagonal lines when there are not enough pixels on screen to represent them realistically. Also called “stair-stepping” and “jaggies”.





Example about low and high resolution.

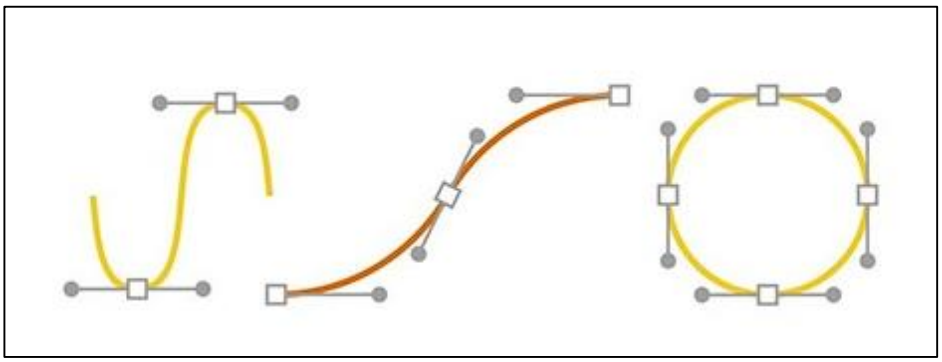
Note: A single graphics device can operate in a number of different graphics modes with different resolutions and color selections. We will deal with the **VGA adapter**. The graphics mode resolution is (640 x 480) and (16 colors).

Types of Computer Graphics

There are two types of computer graphics:

- **Vector Graphics**

  These graphics consist of anchored dots and connected by lines and curves, similar to the connect-the-dot activities you may have done as a kid. Also, vector graphics are made up of paths, each with a mathematical formula (vector) that tells the path how it is shaped. Because these graphics are not based on pixels, they are known as resolution independent, which makes them infinitely scalable. Their lines are sharp, without any loss in quality or detail, no matter what their size. These graphics are also device-independent, which means their quality doesn't depend on the number of dots available on a printer or the number of pixels on a screen. Because they consist of lines and anchor points, the size of the files are relatively small. This makes vector files the best format for graphic assets such as illustrations, icons and company logos, as the same file can be used for designs ranging from a mobile app to a large billboard without sacrificing quality or increasing file size.

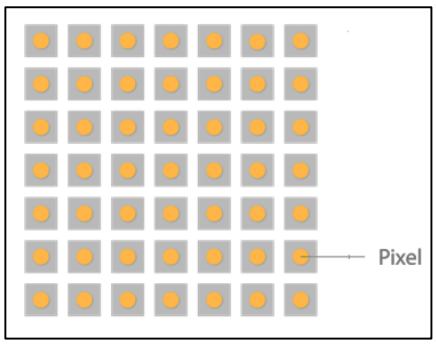


Vector shapes are made up of points and lines that create paths.

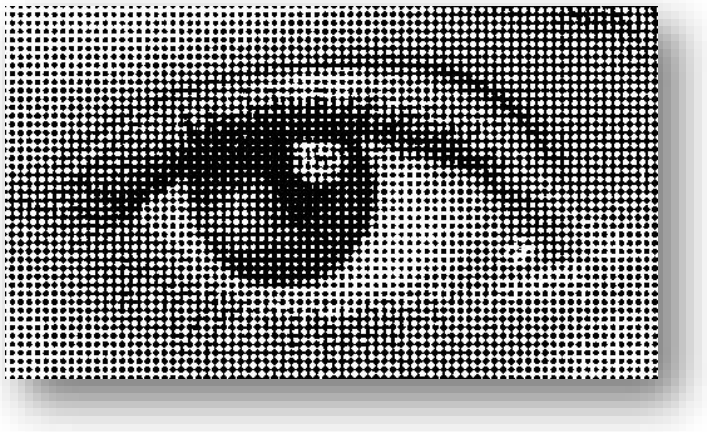
Vector image

- **Raster Graphics**

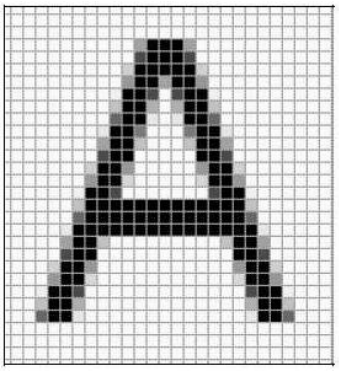
  Raster graphics, also called bitmap graphics, a type of digital image that uses tiny rectangular pixels, dots, or picture elements, arranged in a grid formation to represent an image. Raster is good for photographs. Raster graphics are great when creating rich and detailed images. Every pixel in a raster image can be a different color creating a complex image with all kinds of color and variations.



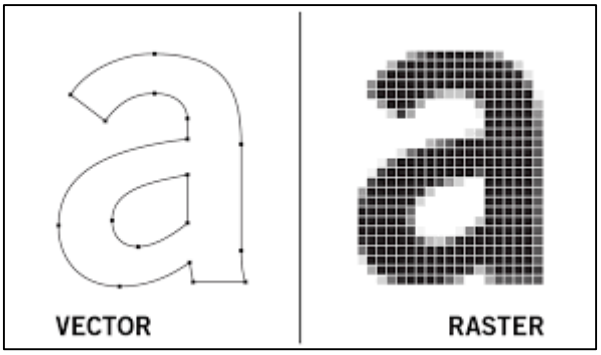
Computer screen with pixels is a raster



Computer screen with pixels is a raster

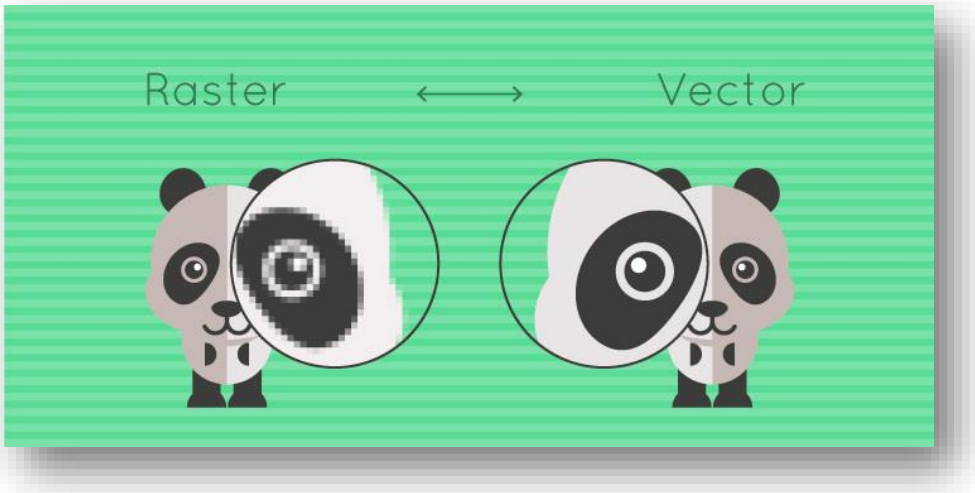


Computer screen with pixels is a raster



Raster image Vs Vector image

When the raster image is zoomed in or enlarged pixels appear like little squares on graph paper. Note the figure below. **We will deal with Raster Graphics**.



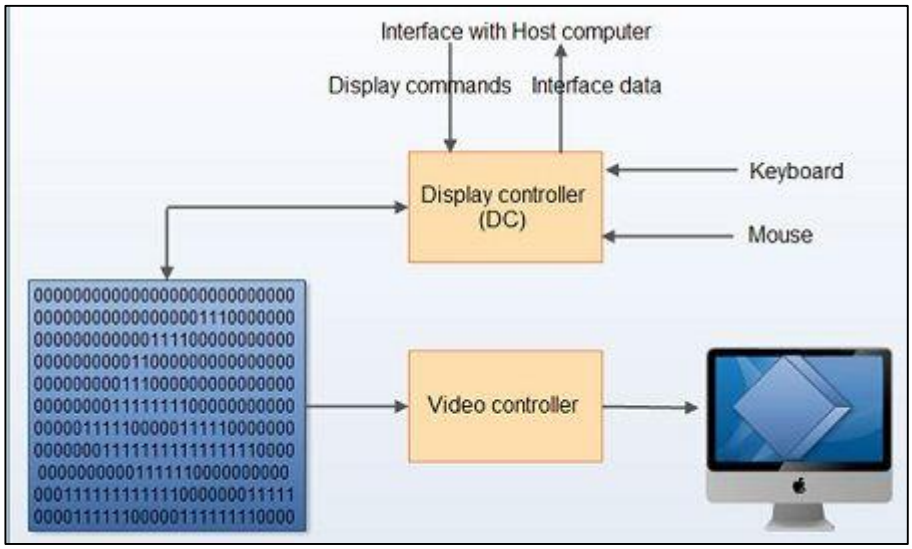
Raster image Vs Vector image

Note: It is known that Raster graphics is more popular than Vector graphics, but if your project requires scalable shapes and solid colors, vector is the best choice, but if your project requires complex color blends, raster is the preferred format.

Graphics Display System

It consists of four components:

1. **A display controller** that gets the inputs and commands from the user and determines the image to be displayed on the monitor. The display controller will divide the image into a number of pixels. This image which is to be displayed is stored in the frame buffer. The image will be stored as a matrix of intensity values.
2. **A digital memory** (frame buffer or Bitmap), in which the displayed Image is stored as a matrix of intensity values. The number of rows in the frame buffer array equal the number of raster lines on the display screen, and the number of columns in this array equals the number of pixels on each raster line.
3. **A monitor** (computer screen).
4. **Video controller** which is a simple interface that passes the contents of the frame buffer to the monitor. Inside the frame buffer the image is stored as a pattern of binary digital numbers, which represent a rectangular array of picture elements, or pixel. In the Simplest case where we wish to store only black and white images, we can represent black pixels by 0's in the frame buffer and white Pixels by 1's. The display controller simply reads each successive byte of data from the frame buffer and converts each 0 and 1 to the corresponding video signal. This signal is then fed to the monitor. If we wish to change the displayed picture all we need to do is to change of modify the frame buffer contents to represent the new pattern of pixels.



Graphics Display System.

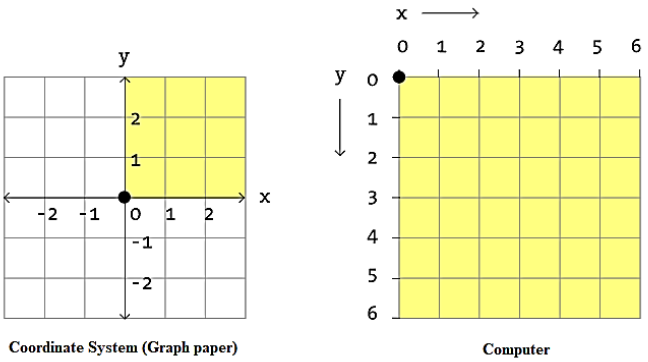
Note: A video controller, often referred to as a video or graphics card, is a key hardware component that allows computers to generate graphic information to any video display devices, such as a monitor or projector. Some modern computers do not include video cards, but rather have graphics processing units directly integrated into the computer's motherboard.

Scan Conversion Definition

It is a process of representing graphics objects a collection of pixels. The graphics objects are **continuous**. The pixels used are discrete. Each pixel can have either on or off state. The circuitry of the **video display device** of the computer is capable of converting binary values (0, 1) into a pixel on and pixel off information. 0 is represented by pixel off. 1 is represented using pixel on. Using this ability graphics computer represent picture having discrete dots. Any model of graphics can be reproduced with a dense matrix of dots or points. Most human beings think graphics objects as points, lines, circles, ellipses. For generating graphical object, many algorithms have been developed. The process of **scan conversion** is also called as **rasterization**. The algorithms implementation varies from one computer system to another computer system. Some algorithms are implemented using the software. Some are performed using hardware or firmware. Some are performed using various combinations of hardware, firmware, and software.

Coordinates of Computer Screen

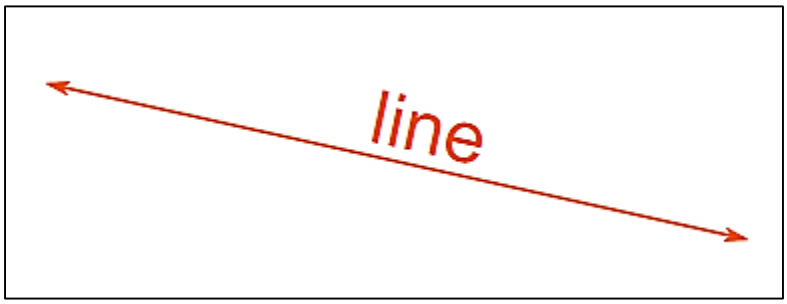
The graph paper ("Cartesian coordinate system") placed (0, 0) in the center with the y-axis pointing up and the x-axis pointing to the right (in the positive direction, negative down and to the left). The coordinate system for pixels in a computer window, however, is reversed along the y-axis. (0, 0) can be found at the top left with the positive direction to the right horizontally and down vertically.



Coordinates System (Graph Paper) Vs Coordinates of Computer Screen.

Line

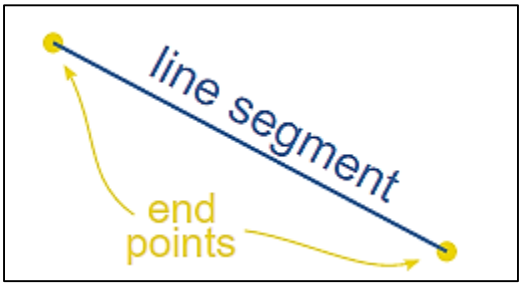
Line is straight (no bends), has no thickness, and extends in both directions without end (infinitely).



One end makes it a "Ray", and two ends makes it a "Line Segment".

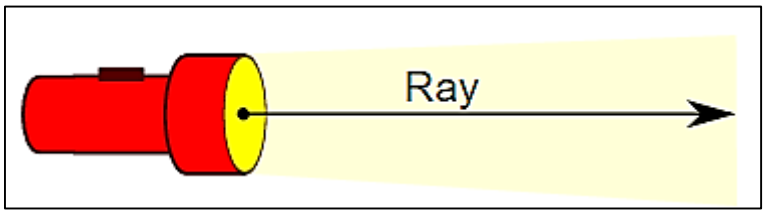
Line Segment

When it does have ends it is called a "Line Segment".



Ray

When it has just one end it is called a "Ray"

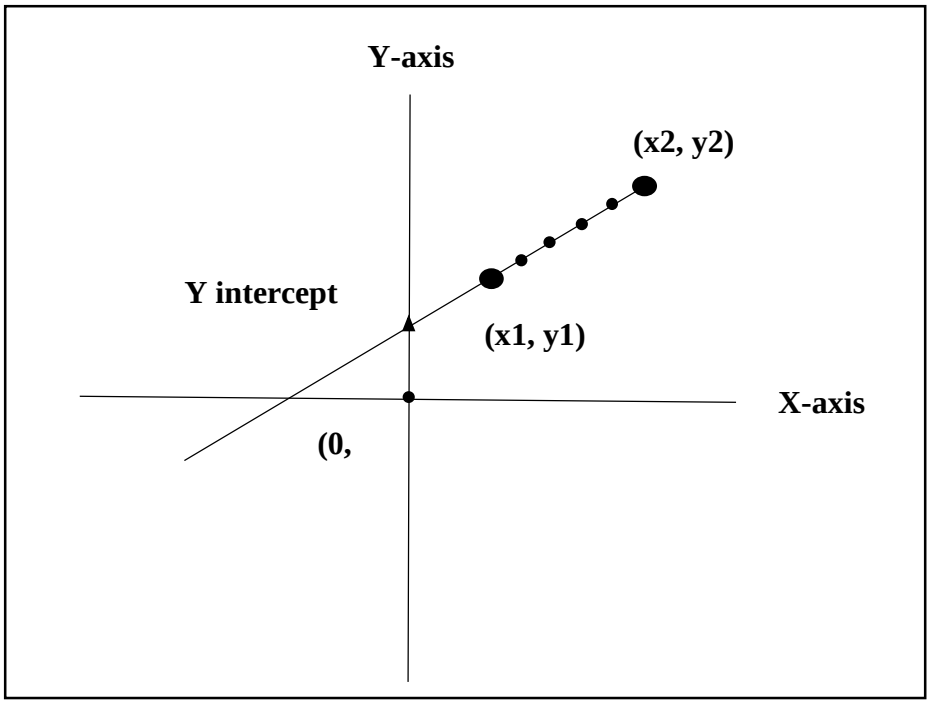


Equation of Straight line

The equation of a straight line is usually written as this way:

$$Y = mX + C$$

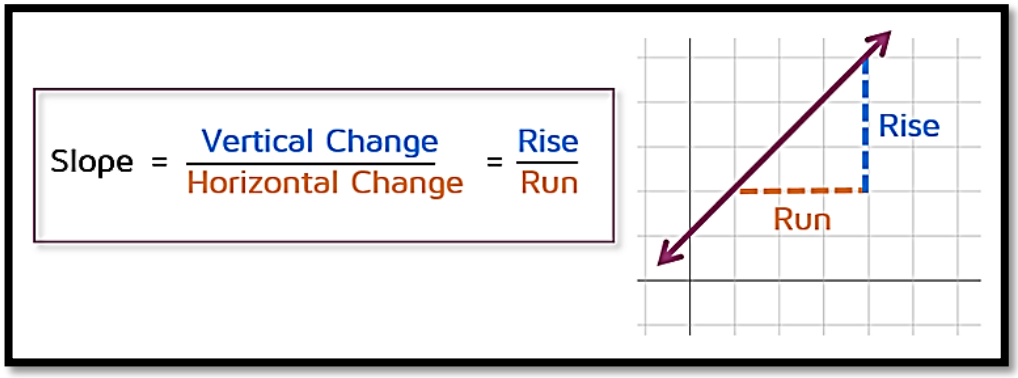
Where, **m** is the Gradient or Slope of a line, **C** is the intercepting point (line segment intercepts Y-axis when X=0), see the figure below:



If you know values of **m** and **c** you can find any point on line segment. There are different equations of a line but the above equation is specially used in computer graphics.

Slope

Slope measures the steepness of a line. **Slope** also tells us the direction of the line - if it goes up, down, or if it's horizontal or vertical. **Slope** is calculated as the ratio of the amount of vertical change to horizontal change.



m (slope)= $\frac{\Delta y}{\Delta x}$, where $\Delta x = x2 - x1$, $\Delta y = y2 - y1$

Ex: find the slope of a line, which has these two end points **(8, 8)** and **(20, 15)**.

$\Delta x = x2 - x1$

$\Delta x = 20 - 8$

$\underline{\boldsymbol{\Delta x = 12}}$

$\Delta y = y2 - y1$

$\Delta y = 15 - 8$

$\underline{\boldsymbol{\Delta y = 7}}$

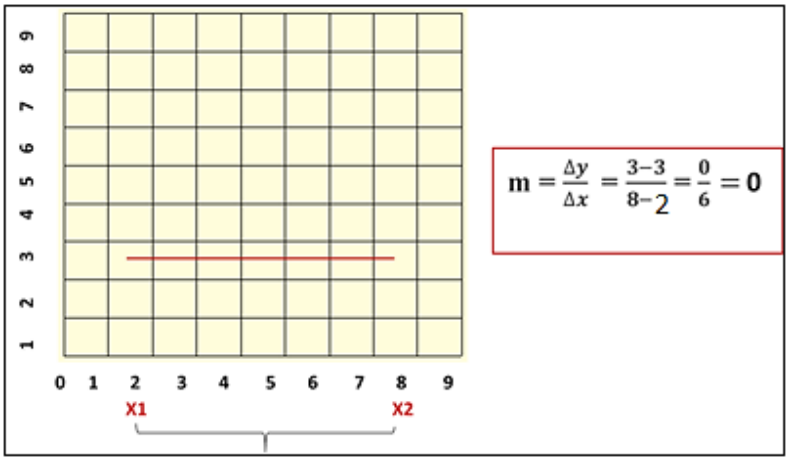
m (slope)= $\frac{\Delta y}{\Delta x}$

m = $\frac{\mathbf{7}}{\mathbf{12}}$ = 0.6

There are four different types of slope, depending on the direction of the line (zero, undefined, positive (m<1, m=1, m>1), negative).

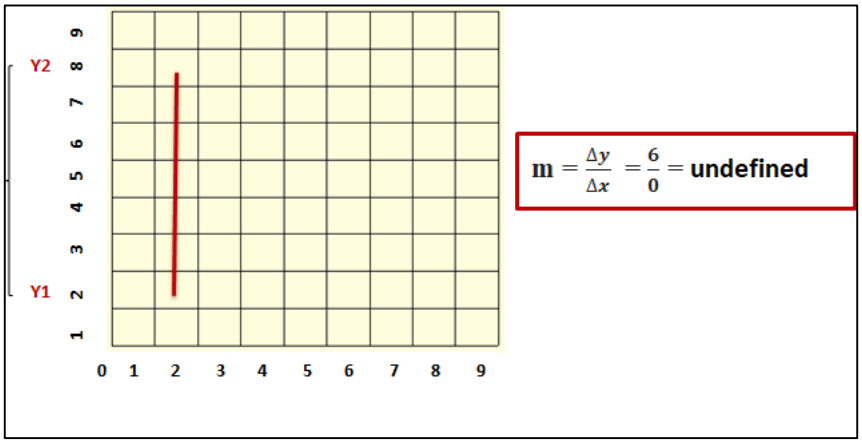
1- Slope of Zero

If the y-values are not changing as x increases, the line will have a slope of 0. Anytime the line is horizontal (flat from left to right), the slope is zero.



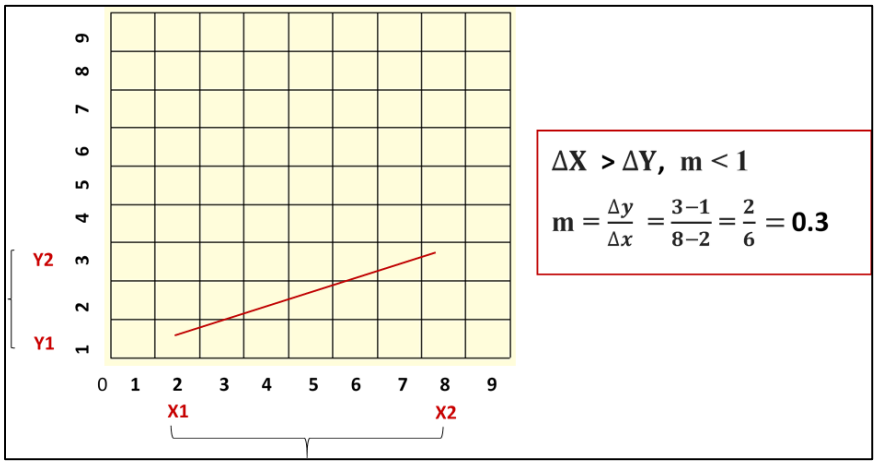
2- Undefined Slope

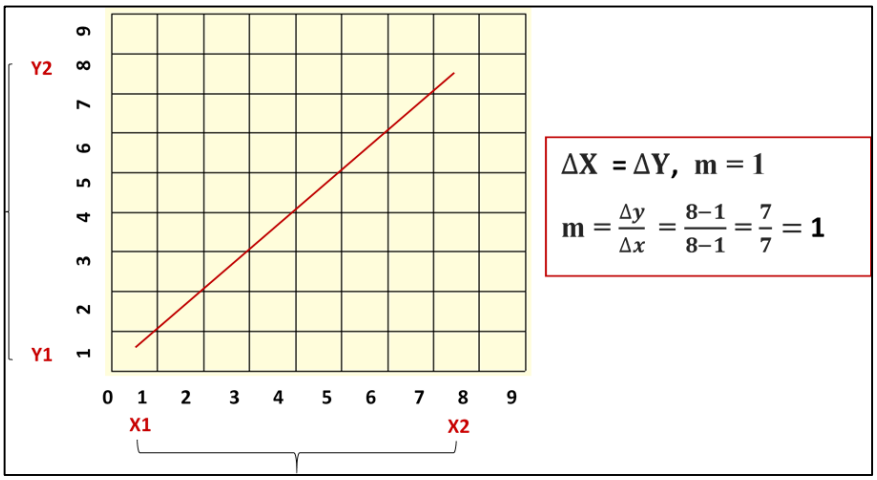
A vertical line has an undefined slope. In this situation, the y-values are changing, but the x-value always stays the same. If you look at the definition of slope, the amount of horizontal change is in the denominator of a fraction. In math, you can't have a 0 in the denominator. It doesn't make sense to divide by 0 so we say that the slope of a vertical line is undefined. There isn't a slope for these types of lines.

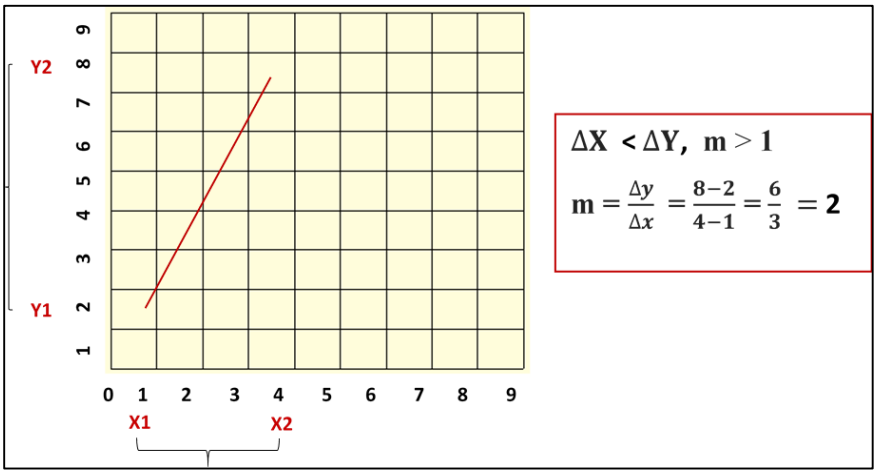


3- Positive slope (m<1, m=1, m>1)

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.

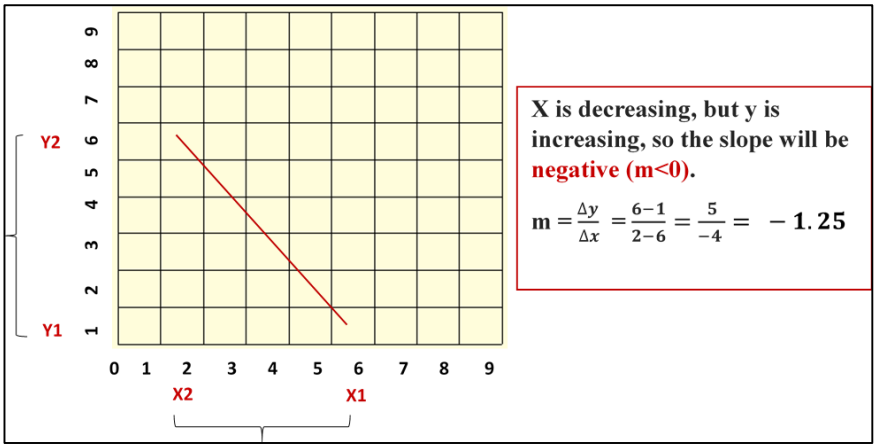






4- Negative slope

A negative slope means that two variables are negatively related; that is, when x increases, y decreases, and when x decreases, y increases.



Useful summery

The concept of slope is very useful, because it measures the relationship between two variables. 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. Graphically, a positive slope means that as a line on the line graph moves from left to right, the line rises.

A **negative slope** means that two variables are negatively related; that is, when *x* increases, *y* decreases, and when *x* decreases, *y* increases. Graphically, a negative slope means that as the line on the line graph moves from left to right, the line falls.

Slope and Angle of a line

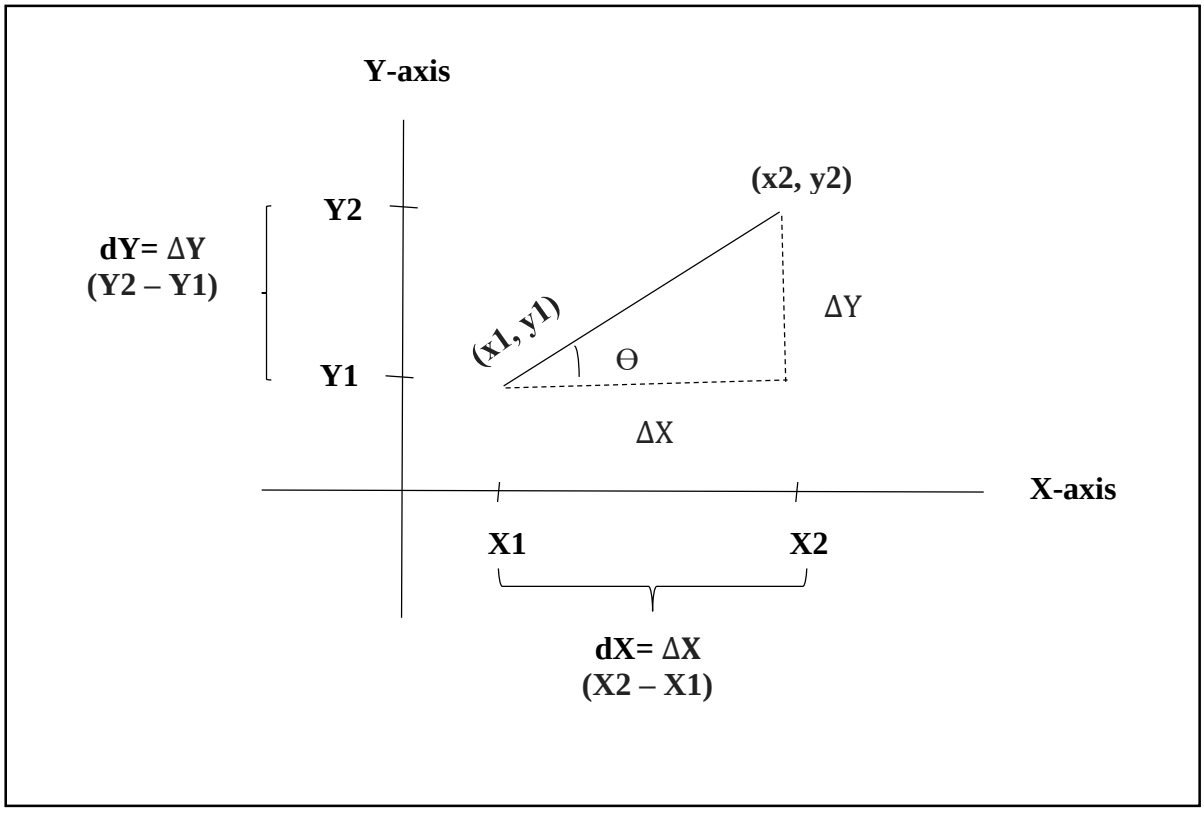
In the figure below, a line segment has two end-points (x1, y1) and (x2, y2). The dotted line that labeled ΔX is the difference between x2 and x1, and the dotted line that labeled ΔY is the difference between y2 and y1. These three sides make traingle with Θ angle. As we mentioned before, **Slope** is calculated as the ratio of the amount of vertical change to horizontal change.

$$\textbf{m (slope)} = \frac{\Delta y}{\Delta x}$$

ΔY is the opposite side of Θ angle and ΔX is the adjacent side of Θ angle. Tangent of an angle is the length of opposite side divided by the length of the adjacent side:

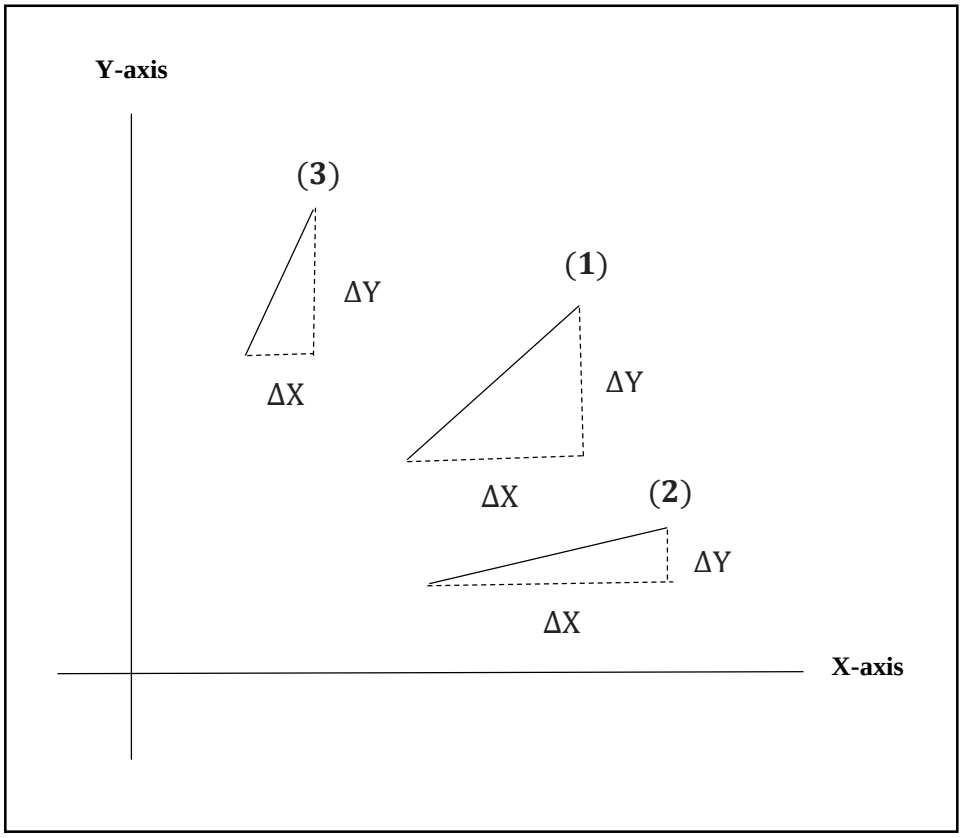
$$\textbf{Tan}(\Theta) = \frac{\textbf{opposite side}}{\textbf{adjacent side}} = \frac{\Delta y}{\Delta x} = \textbf{m (slope)}$$

So, **Tan(Θ) = m (slope)**.



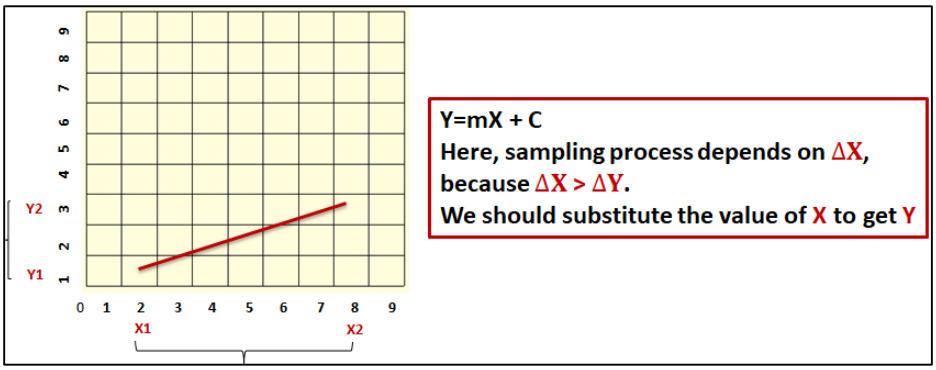
You know that Tan(45) = 1, so:

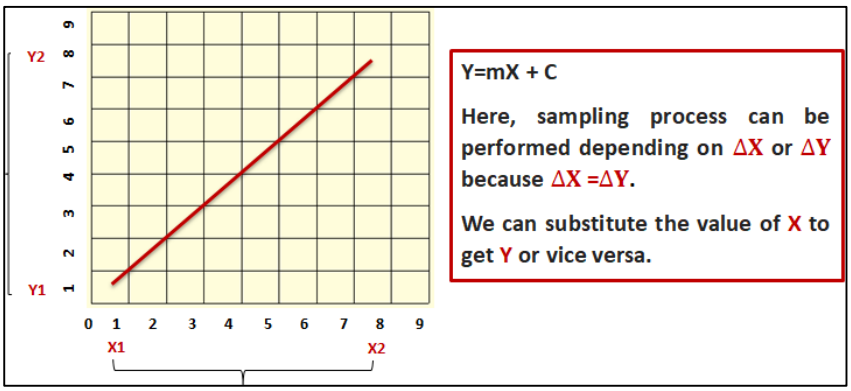
1- When a line sloped at 45 angle, then a slope will become equal to **one** (m =1). Where $\Delta Y = \Delta X$.
2- When a line sloped at an angle of less than 45 degree, then a slope will become less than **one** (m < 1). Where $\Delta Y < \Delta X$.
3- When a line sloped at an angle of greater than 45 degree, then a slope will become greater than **one** (m > 1). Where $\Delta Y > \Delta X$.

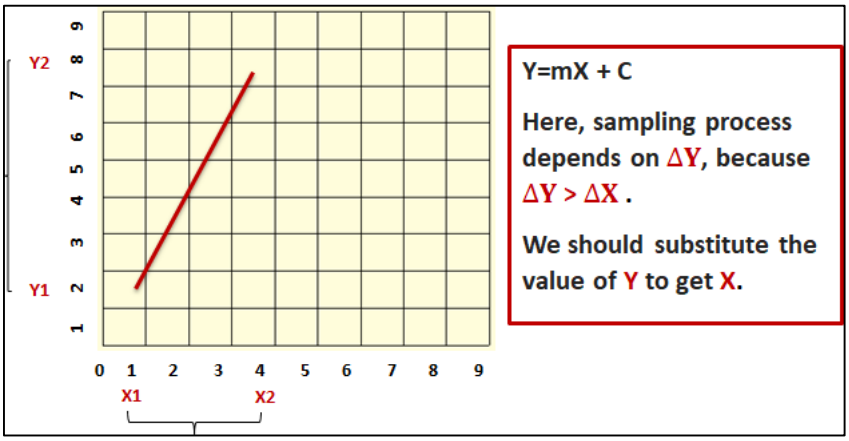


Sampling

Any line that we draw on the two dimensional coordinates system, must be represented by an equation. The line equation is Y=mX + C, where m is the slope of a line and C is the intercepting point of a line with y-axis when X=0. Sampling process means that, in line equation we substitute the value of X to get Y or vice versa. Sampling process of a line equation depends on **the greatest value** of **Δx** or **Δy**.







Line Generation Algorithms

In any 2-Dimensional plane if we connect two points (x1, y1) and (x2, y2), we get a line segment. But in the case of computer graphics we cannot directly join any two coordinate points, for that we should calculate intermediate point's coordinate and put a pixel for each intermediate point, of the desired color.

There are several algorithms for generating a straight line, but we will only deal with two algorithms, namely:

1- DDA algorithm.
2- Bresenham's line drawing algorithm.

DDA algorithm

The Digital Differential Analyzer (DDA) algorithm is an incremental scan-conversion method. Such an approach is characterized by performing calculations at each step using results from the preceding step.

In order to understand the DDA algorithm, let us consider the following mathematical examples:-

Ex1: By using the DDA algorithm, generate the points of the line which has two endpoints (2, 2) and (9, 2).

Sol:

$\Delta x = x2 - x1, \quad \Delta y = y2 - y1$

$\Delta x = 9 - 2 = 7, \quad \Delta y = 2 - 2 = 0$

Steps = 7 **(length of the line depends on the greatest value of Δx or Δy)**

$Xinc = \frac{\Delta x}{Steps} = \frac{7}{7} = 1, \quad Yinc = \frac{\Delta y}{Steps} = \frac{0}{7} = 0$

X	Y
2	2
3	2
4	2
5	2
6	2
7	2
8	2
9	2

Ex2: By using the DDA algorithm, generate the points of the line which has two endpoints (5, 4) and (12, 7).

Sol:

$\Delta x = x2 - x1, \quad \Delta y = y2 - y1$

$\Delta x = 12 - 5 = 7, \quad \Delta y = 7 - 4 = 3$

Steps = 7 **(length of the line depends on the greatest value of Δx or Δy)**

Xinc = $\frac{\Delta x}{Steps} = \frac{7}{7} = 1$, Yinc = $\frac{\Delta y}{Steps} = \frac{3}{7} = 0.4$

X	Y	Round(Y)
5	4	4
6	4.4	4
7	4.8	5
8	5.2	5
9	5.6	6
10	6	6
11	6.4	6
12	6.8	7

Ex3: By using the DDA algorithm, generate the points of the line which has two endpoints (17, 14) and (12, 9).

Sol:

$\Delta x = x2 - x1, \quad \Delta y = y2 - y1$

$\Delta x = 12 - 17 = -5, \quad \Delta y = 9 - 14 = -5$

Steps = 5 **(absolute value)**

Xinc = $\frac{\Delta x}{Steps} = \frac{-5}{5} = -1$, Yinc = $\frac{\Delta y}{Steps} = \frac{-5}{5} = -1$

X	Y
17	14
16	13
15	12
14	11
13	10
12	9

DDA Algorithm

Step1: Start Algorithm

Step2: Enter value of x_1, y_1, x_2, y_2.

Step3: Check coordinates values, if(x1<0 or x2<0 or y1<0 or y2<0) then print Error Message.

Step4: Calculate dx = x_2-x_1

Step5: Calculate dy = y_2-y_1

Step6: If ABS (dx) > ABS (dy)

Then steps = abs (dx)

Else steps = abs (dy)

Step7: xinc=dx/steps

yinc=dy/steps

Step8: Initial value for

x = x1

y = y1

Step9: Set pixels (round(x), round (y))

x = x + xinc

y = y + yinc

Step10: Repeat step 9 for **steps** times

Step11: End Algorithm

Ex4: Consider the line from **(0 , 0)** to **(4 , 8)**. Use the DDA algorithm to rasterize the line.

Sol:

$\Delta x = x2 - x1, \quad \Delta y = y2 - y1$

$\Delta x = 4 - 0 = |4| = 4, \quad \Delta y = 8 - 0 = |8| = 8$

Steps = 8 **(absolute value)**

$\text{Xinc} = \frac{\Delta x}{Steps} = \frac{4}{8} = 0.5, \quad \text{Yinc} = \frac{\Delta y}{Steps} = \frac{8}{8} = 1$

I	Plot	X	Y
		0	0
1	(0, 0)	0.5	1
2	(1, 1)	1	2
3	(1,2)	1.5	3
4	(2, 3)	2	4
5	(2, 4)	2.5	5
6	(3, 5)	3	6
7	(3, 6)	3.5	7
8	(4, 7)	4	8
9	Stop		

H.W: Consider the line from **(5 , 7)** to **(10 , 15)**. Use the DDA algorithm to rasterize the line.

Advantages:

1- It is simple and easy to implement algorithm.
2- It avoid using multiple operations which have high time complexities.
3- It is faster than the direct use of the line equation because it does not use any floating point multiplication and it calculates points on the line.

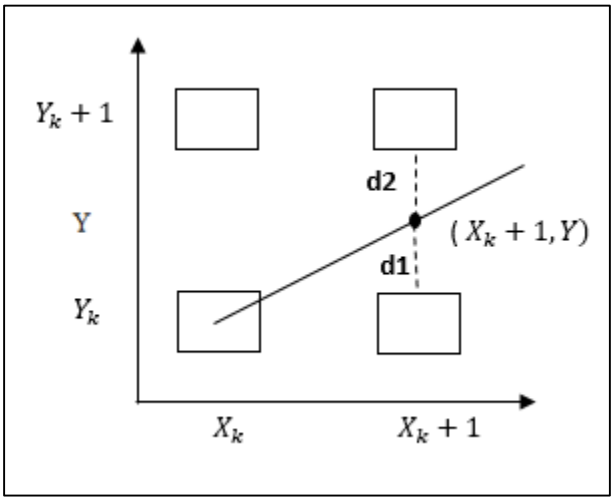
Disadvantages:

1- It deals with the rounding off operation and floating point arithmetic so it has high time complexity.
2- As it is orientation dependent, so it has poor endpoint accuracy.
3- Due to the limited precision in the floating point representation it produces cumulative error.

Bresenham’s Line drawing algorithm

Bresenham’s line algorithm uses only integer addition and subtraction and multiplication by 2, and we know that the computer can perform the operations of integer addition and subtraction very rapidly. The computer is also time-efficient when performing integer multiplication by powers of 2. Therefore, it is an efficient method for scan-converting straight lines.

Although developed originally for use with digital plotters, Bresenham's algorithm is equally suited for use with CRT raster devices. **The basic principle of Bresenham's line algorithm is to select the optimum raster locations to represent a straight line.** To accomplish this, the algorithm always increments either **x** or **y** by one unit depending on the slope of line. **The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel**. This distance is called **decision variable (v)** or **decision parameter (P) or** the **error (e)**. This is illustrated in the Fig. below.



As shown in the figure above, the line does not pass through all raster points (pixels). It passes through raster point (X_k,Y_k) and partially crosses between (X_k+1, Y_k) and (X_k+1, Y_k+1). Which one is the true? Therefore, we need to define a **decision parameter (p)**.

In mathematical a decision parameter is defined as:

P = d1 – d2

Let us define p = d1 – d2. Now if p > 0, then it implies that d1 > d2, i.e. the pixel above the line is closer to the true line. If d1 < d2 (i.e. p < 0) then we can say that the pixel below the line is closer to the true line. Thus by checking only the sign of decision parameter (p), it is possible to determine the better pixel to represent the line path.

The decision parameter (p) is initially set as:

p = 2 Δy - Δx

Where $\Delta y = y2 - y1$, and $\Delta x = x2 - x1$

Then according to value of **p** following actions are taken:

```
for i=1 to Δx
{
    Plot (x, y)
    x = x + 1
   if (p < 0)
              p = p + 2 Δy
   else
           {
              y = y + 1
              p = p + 2 Δy – 2 Δx
           }
 }
```

When p < 0, decision parameter **p** is initialized to p = p + 2 * Δy. When p >= 0 decision parameter **p** is initialized to **p = p + 2 Δy – 2 Δx.** **You should note that in both the cases x is incremented by 1.**

Conditions of Integer Bresenham's line drawing algorithm

Before writing the **Integer Bresenham's line drawing algorithm**, it is important to remember its conditions:

1. Our pixel coordinates go from left to right, and bottom to top, the way they do in the **first octant** in mathematics.
2. $X1 < X2$, and $Y1 \leq Y2$.
3. The slope of the line will be **$(0 \leq m < 1)$**, so we will be drawing the line from lower left to upper right; and, for any X value between X1 and X2, there will be exactly one pixel on.

Integer Bresenham's Line Drawing Algorithm

Step1: Read the line end points (X1, Y1) and (X2, Y2) such that:

[$(X1 < X2$ and $Y1 \leq Y2)$ and $(0 \leq m < 1)$ otherwise exit]

Step2: $\Delta X = X2 - X1$ and $\Delta Y = Y2 - Y1$

Step3: [Initialize starting point]

$X = X1$ and $Y = Y1$

step4: [Initialize value of decision parameter]

$P = 2 * \Delta Y - \Delta X$

Step5: for $i = 1$ to ΔX

```
{
    Plot(x , y)
    X = X + 1
    if (p < 0)
        p = p + 2 ΔY
```

else

{

Y = Y + 1

p = p + 2 ΔY – 2 ΔX

}

i = i + 1

}

Step6: Stop.

Ex: consider the line from **(0 , 0)** to **(5 , 3)**. Use the Integer Bresenham's algorithm to rasterize the line.

Sol:

ΔX = 5 , ΔY = 3 , P = 1

I	Plot	X	Y	P
		0	0	1
1	(0 , 0)	1	1	-3
2	(1, 1)	2		3
3	(2, 1)	3	2	-1
4	(3, 2)	4		5
5	(4, 2)	5	3	1
6	Stop			

H.W: consider the line from **(5 , 5)** to **(13 , 9)**. Use the Integer Bresenham's algorithm to rasterize the line.

Advantages

1. It involves only integer arithmetic, so it is simple.
2. It avoids the generation of duplicate points.
3. It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.
4. More accurate and efficient than DDA Algorithm.

Disadvantages

This algorithm is meant for basic line drawing only Initializing is not a part of Bresenham's line algorithm. Therefore, to draw smooth lines, you should want to look into a different algorithm.

Differentiate between DDA Algorithm and Bresenham's Line Algorithm:

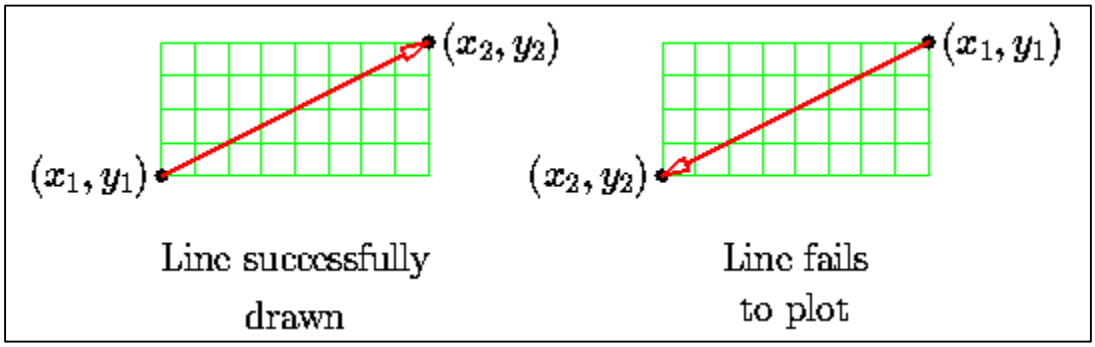
DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic.	1. Bresenham's Line Algorithm use fixed point, i.e., Integer Arithmetic
2. DDA Algorithms uses multiplication & division its operation.	2. Bresenham's Line Algorithm uses only subtraction and addition its operation.
3. DDA Algorithm is slowly than Bresenham's Line Algorithm in line drawing because it uses real arithmetic (Floating Point operation).	3. Bresenham's Algorithm is faster than DDA Algorithm in line because it involves only addition & subtraction in its calculation and uses only integer arithmetic.
4. DDA Algorithm is not accurate and efficient as Bresenham's Line Algorithm.	4. Bresenham's Line Algorithm is more accurate and efficient at DDA Algorithm.
5. DDA Algorithm can draw circle and curves but are not accurate as Bresenham's Line Algorithm.	5. Bresenham's Line Algorithm can draw circle and curves with more accurate than DDA Algorithm.

General Bresenham's Line drawing algorithm

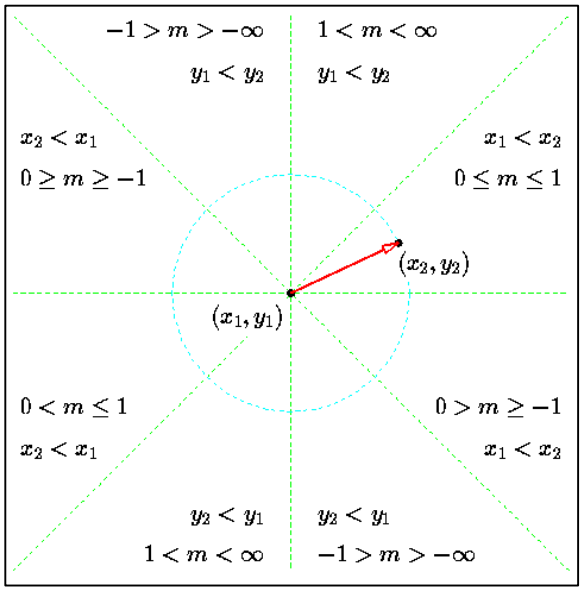
If we try out the C++ implementation of the Bresenham algorithm, we find it has some peculiar properties. As expected, it fails to plot lines with **negative slopes**. It also fails to plot lines of **positive slope greater than 1** (this is an interesting case).

More unusually, we find that **the order in which the endpoints are supplied to this routine is significant**, it will only work as long as **x1 is smaller than x2**.

In fact, if we have two line segments with the same endpoints, and the same slope, this routine may draw one of them successfully but fails to draw the other one.



Of course, this is not surprising really, when we consider that the function works by **incrementing** x. It does emphasise, however, that the routine is plotting *vectors*, direction is significant. Considering all the vectors from *(x1, y1)* to *(x2, y2)* we find that there are eight regions, (the ``octants'') and the basic Bresenham algorithm works in only one of them.



A full implementation of the Bresenham's algorithm must, of course, be able to handle all combinations of **slope** and endpoint **order**.

We can finalize everything. If we want to deal with positive or negative slope lines, we just adjust the step size to be +1 or -1. If we want to deal with slopes greater than 1 (or less the -1), we just interchange X and Y, and do our step increment (or decrement) using Y instead of X, etc.

General Bresenham's Line Drawing algorithm

Step1: Read the line end points (X1, Y1) and (X2,Y2).

Step2: $\Delta X = |X2 - X1|$ and $\Delta Y = |Y2 - Y1|$

Step3: s1 = sign(X2 - X1) and s2 = sign(Y2 - Y1)

Step4: [Initialize starting point]

X = X1 and Y = Y1

Step5: [Interchange ΔX and ΔY depending on the slop of the line]

if $\Delta Y > \Delta X$ then

temp = ΔX

$\Delta X = \Delta Y$

ΔY = temp

Interchange = 1

else

Interchange = 0

Step6: [Initialize value of decision variable or error]

p = 2 * $\Delta Y - \Delta X$

Step7: for i = 1 to ΔX

{

Plot(X , Y)

if Interchange = 1 then

Y = Y + s2

else

X = X + s1

if (p < 0)

p = p + 2 ΔY

else

{

if Interchange = 1 then

X = X+ s1

else

Y = Y + s2

p = p + 2 ΔY – 2 ΔX

}

i = i + 1

}//for

Step8: Stop.

Ex: consider the line from **(10 , 20)** to **(5 , 10)**. Use the General Bresenham's algorithm to rasterize the line.

Sol:

Line orientation is from right to left so:

$\Delta x = 5$, $\Delta y = 10$, $s1 = -1$, $s2 = -1$

Δy is greater than Δx so , $\Delta x = 10$, $\Delta y = 5$, Interchange = 1

I	**Plot**	**X**	**Y**	**P**
		10	20	0
1	(10, 20)	9	19	-10
2	(9, 19)		18	0
3	(9, 18)	8	17	-10
4	(8, 17)		16	0
5	(8, 16)	7	15	-10
6	(7, 15)		14	0
7	(7, 14)	6	13	-10
8	(6, 13)		12	0
9	(6, 12)	5	11	-10
10	(5, 11)		10	0
11	**Stop**			

Ex: consider the line from **(8, 7)** to **(3, 12)**. Use the General Bresenham's algorithm to rasterize the line.

Sol:

Line orientation is from right to left and from bottom to top so:

$\Delta x = 5$, $\Delta y = 5$, $s1 = -1$, $s2 = +1$, $m = -1$

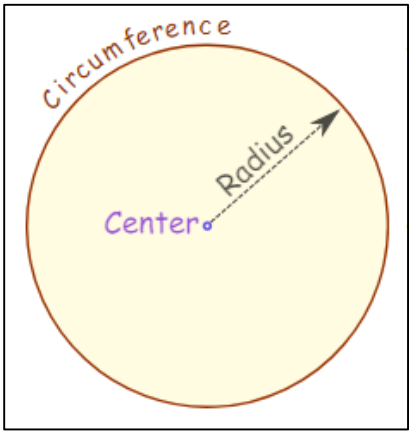
Δy is not greater than Δx so , Interchange = 0

I	Plot	X	Y	P
		8	7	5
1	(8, 7)	7	8	5
2	(7, 8)	6	9	5
3	(6, 9)	5	10	5
4	(5, 10)	4	11	5
5	(4, 11)	3	12	5
6	Stop			

H . W: consider the line from **(5 , 10)** to **(10 , 20)**. Use the General Bresenham's algorithm to rasterize the line.

Circle Drawing

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



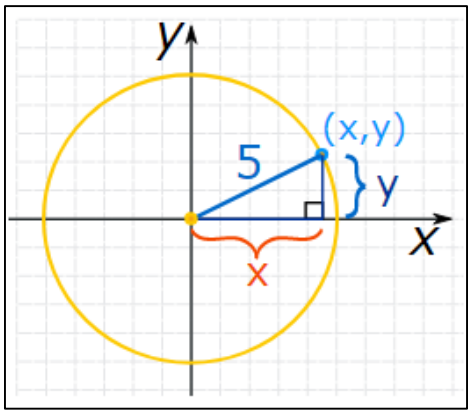
To draw a circle we need two things, **the coordinates of the center** and the **radius** of the circle.

Radius: The radius of a circle is the length of the line from the center to any point on its edge.

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.

The equation of circle is found using **Pythagoras**. See the following figure, when R=5.

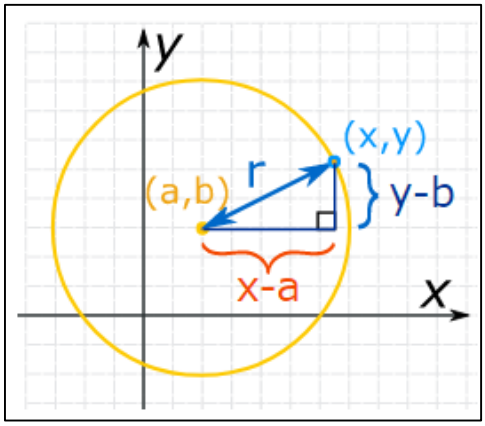


$x^2 + y^2 = 5^2$

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

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

See the figure below:



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

$$\boldsymbol{X^2 + Y^2 - R^2 = 0}$$

Above equation also called **circle function** f(x, y), where:

$$f(x, y) = \boldsymbol{X^2 + Y^2 - R^2}$$

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

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

Circle Generation Algorithm

It is not easy to display a continuous smooth arc on the computer screen as our computer screen is made of pixels organized in matrix form. So, to draw a circle on a computer screen we should always choose the nearest pixels from a printed pixel so as they could form an arc. There are two algorithm to do this:

1. Mid-Point circle drawing algorithm
2. Bresenham's circle drawing algorithm

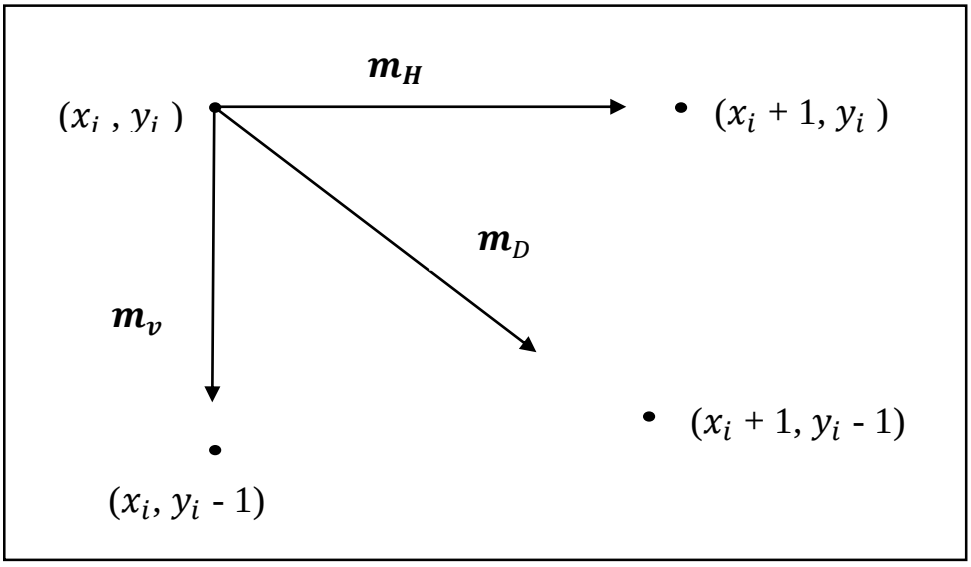
Before getting in the details of these algorithms let us, discuss the following concept:

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 represent the circle, horizontally to the right, diagonally downward to the right and vertically downward. These are labeled: $\boldsymbol{m_H}$, $\boldsymbol{m_D}$, $\boldsymbol{m_v}$, respectively in following figure. 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:

$$\boldsymbol{m_H} = \left| \ (\boldsymbol{x_i} + \boldsymbol{1})^{\boldsymbol{2}} + \ (\boldsymbol{y_i})^{\boldsymbol{2}} - \boldsymbol{R^2} \ \right|$$

$$\boldsymbol{m_D} = \left| \ (\boldsymbol{x_i} + \boldsymbol{1})^{\boldsymbol{2}} + \ (\boldsymbol{y_i} - \boldsymbol{1})^{\boldsymbol{2}} - \boldsymbol{R^2} \ \right|$$

$$\boldsymbol{m_v} = \left| \ (\boldsymbol{x_i})^{\boldsymbol{2}} + \ (\boldsymbol{y_i} - \boldsymbol{1})^{\boldsymbol{2}} - \boldsymbol{R^2} \ \right|$$



First quadrant pixel selection

Circle Generation Algorithm for Drawing One Quadrant of a Circle

Step1: start

Step2: X = XC

Step3: Y = YC + R

Step4: while (y >= 0)

Step5: plot (X, Y)

Step6: Eh = $\left| (X+1)^2 + Y^2 - R^2 \right|$

Step7: Ed = $\left| (X+1)^2 + (Y-1)^2 - R^2 \right|$

Step8: Ev = $\left| X^2 + (Y-1)^2 - R^2 \right|$

Step9: min = minimum (Eh, Ed, Ev)

Step10: if min = Eh then X = X + 1: goto step 13

Step11: if min = Ed then X = X + 1: Y = Y - 1: goto step 13

Step12: if min = Ev then Y = Y – 1

Step13: end while

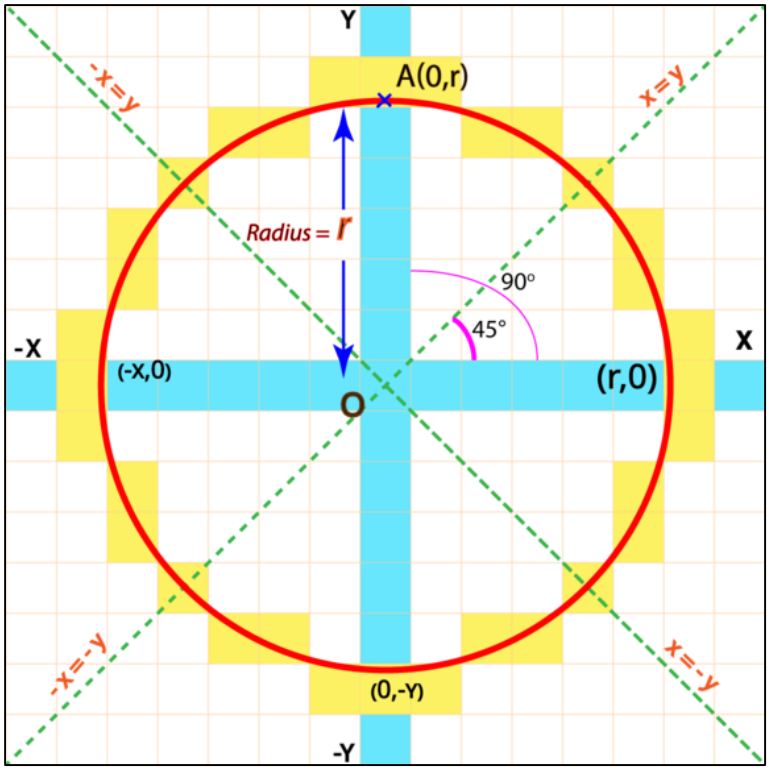
Step14: Stop

Ex: Draw a circle by using circle generation algorithm, when C = 0, R = 8.

X	Y	Plot	Eh	Ed	Ev	Min
0	8	(0, 8)	1	14	15	Eh=1
1		(1, 8)	4	11	14	Eh=4
2		(2, 8)	9	6	11	Ed=6
3	7	(3, 7)	1	12	19	Eh=1
4		(4, 7)	10	3	12	Ed=3
5	6	(5, 6)	8	3	14	Ed=3
6	5	(6, 5)	3	1	12	Ed=1
7	4	(7, 4)	16	9	6	Ev=6
7	3	(7, 3)	9	24	11	Ev=6
8	3	(8, 3)	21	21	4	Ev=4
8	2	(8, 2)	18	18	1	Ev=1
8	1	(8, 1)		17	0	Ev=0
8	0	(8, 0)				

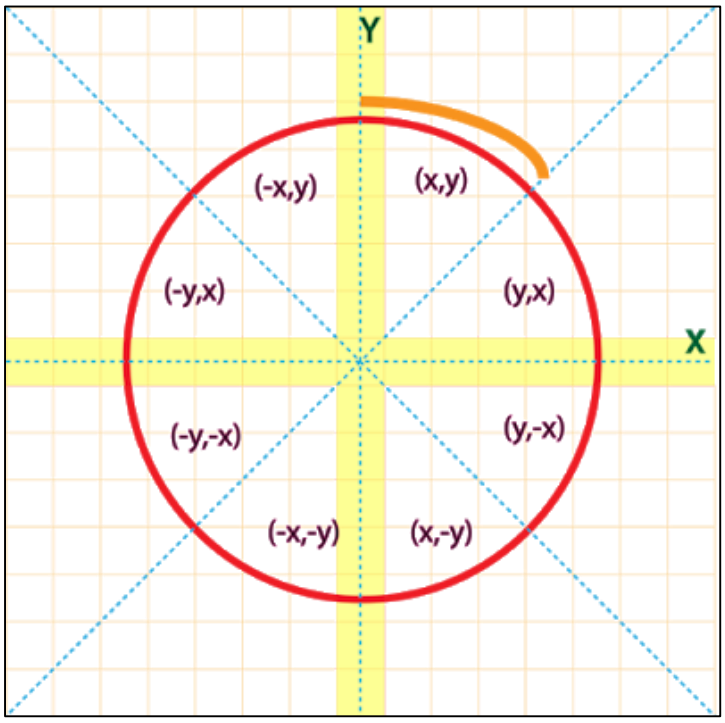
Basics of Bresenham's Circle Drawing Algorithm

Bresenham's circle drawing algorithm is used to determine **the next pixel** of screen to be illuminated while drawing a circle by determining **the closest nearby pixel**. As Circles are **symmetrical** so the values of y-intercept and x-intercept are **same if circle's Center coordinates are at Origin (0, 0)**, see the figure below.



Here, Radius = OA = r

Due to symmetrical property of Circle, we don't need to calculate all the pixels of all the octets and quadrants. We need to find the pixels of only **one octet**, rest we can conclude through using **(8-way symmetry),** see the following figure.

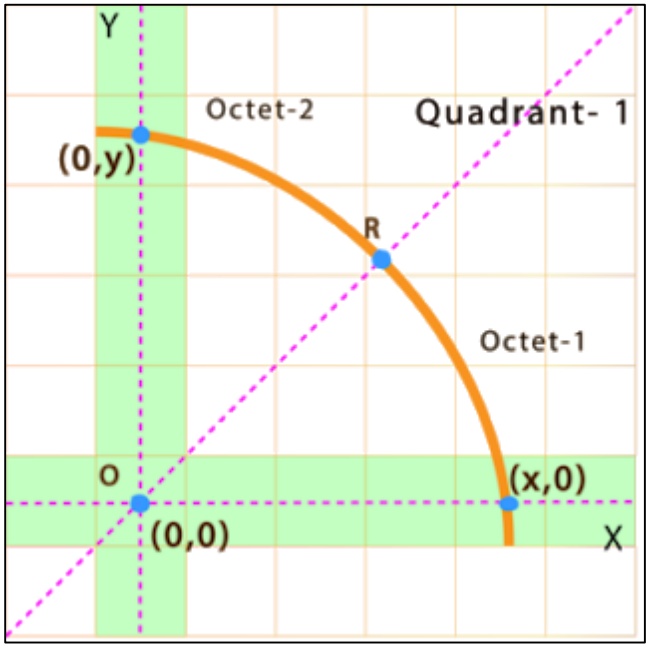


8-Way symmetry: Any circle follows 8-way symmetry. This means that for every point (x,y) 8 points can be plotted. These (x,y), (y,x), (-y,x), (-x,y), (-x,-y), (-y,-x), (y,-x), (x,-y).

For any point (x+a, y+b), points (x ± a, y ± b) and (y ± a, x ± b) also lie on the same circle. So, it is sufficient to compute only 1/8 of a circle, and all the other points can be computed from it.

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.

Let’s take the Octet-2 which is in quadrant-1. Here both x and y are positive here the initial pixel would be (0, y) coordinate, see the figure below.



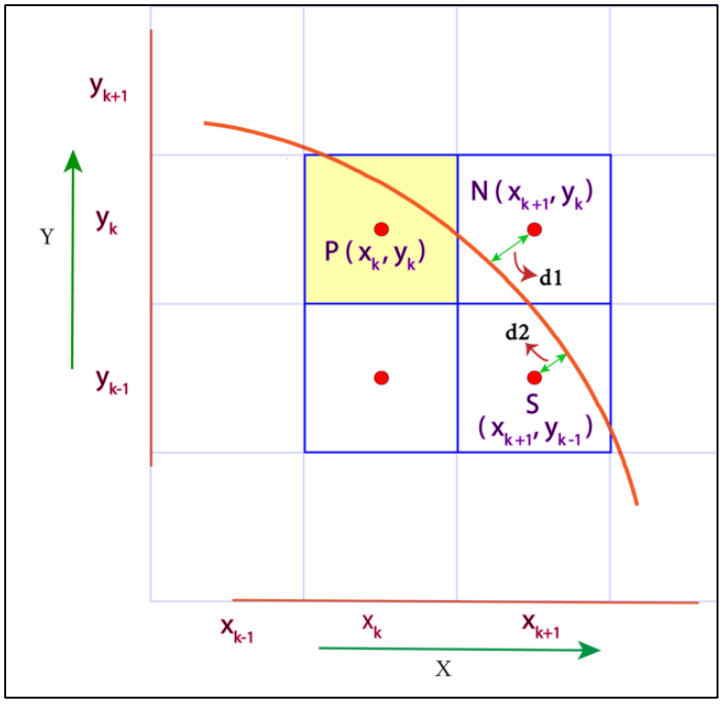
At point **R** both the value of both x and y coordinates would be same as R is at same distance of Both X and Y axis.

Derivation of Bresenham's circle algorithm

In the figure below, let's say our circle is at some random pixel **P** whose coordinates are **(x_k, y_k)**.

Now we need to find out our **next pixel**.

Note: This is **octet 2** so here, **x can never be decremented** as per properties of a circle **but y either needs to decremented or to be kept same**. **y is needed to be decided.**



Here it needs to decide whether go with N or S. For this, Bresenham's circle drawing algorithm will help us to decide by **calculating the difference between radius and the coordinates of the next pixels**.

The **shortest of d1 and d2** will help us decide our next pixel.

Note: $x_{k+1} = x_k + 1$ as x_{k+1} is the next consecutive pixel of x_k similarly $y_{k-1} = y_k - 1$.

Equation of Circle with Radius **r**:

$$(x - h)^2 + (y - k)^2 = r^2$$

When coordinates of center are at Origin i.e., (h=0, k=0)

$x^2 + y^2 = r^2$ (Pythagoras theorem)

Function of Circle Equation:

$F(C) = x^2 + y^2 - r^2$

Function of Circle at N:

$F(N) = (x_{k+1})^2 + (y_k)^2 - r^2$ (Positive)

Here the value of F(N) will be **positive** because N is **out-side the circle** that makes $(x_{k+1})^2 + (y_k)^2$ Greater than r^2.

Function of Circle at S:

$F(S) = (x_{k+1})^2 + (y_{k-1})^2 - r^2$ (Negative)

Here the value of F(S) will be **Negative** because S is **in-side the circle** that makes $(x_{k+1})^2 + (y_{k-1})^2$ Less than r^2.

Now we need a **decision parameter**, which help us **decide the next pixel**, say D_k and, $D_k = F(N)+F(S)$

Here either we will get the **positive** or **negative** value of D_k:

So if $D_k < 0$, that means the **negative F(S) is bigger than the positive F(N)**, that implies **Point N is closer to the circle than point S**. So, we will select **pixel N as our next pixel** (x_{k+1}, y_k).

and if $D_k > 0$, that means **positive F(N) is bigger and S is more closer as F(S) is smaller**. So, we will Select **S as our next pixel** (x_{k+1}, y_{k-1}).

Summery

$D_0 = 3-2r$ **//Initial value of decision parameter**

$$D_{k+1} = \begin{cases} D_k + 4x_k + 6 & \text{if } (D_k < 0) \\ D_k + 4(x_k - y_k) + 10 & \text{if } (D_k >= 0) \end{cases}$$

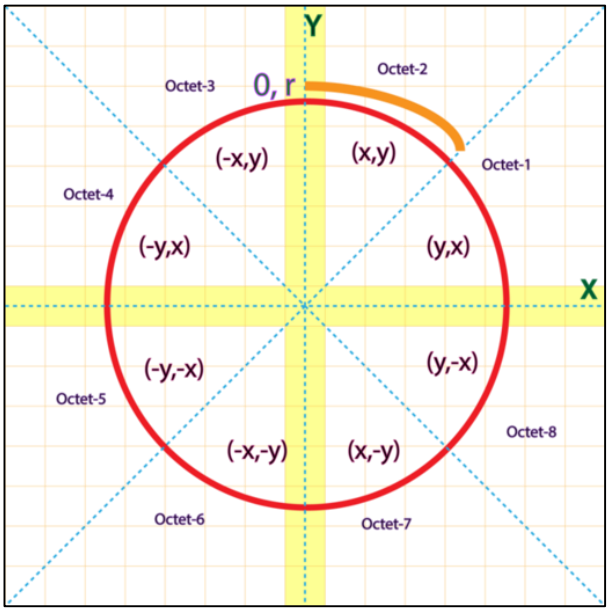
//next value of decision parameter

Bresenham's Circle Drawing Algorithm

A circle is made up of 8 Equal Octets so, we need to find only coordinates of any one octet rest we can conclude using that coordinates.

We took octet-2. Where X and Y will represent the pixel.

Let us make a **function Circle()** with parameters coordinates of Centre (Xc,Yc) and pixel point (X,Y) **that will plot the pixel on screen**.



We will find pixels assuming that Center is at Origin (0,0) then we will add the coordinates of center to corresponding X and Y while drawing circle on screen.

```
Circle (Xc,Yc,X,Y)
{
    Plot (Y+Xc , X+Yc)        ……Octet-1
    Plot (X+Xc , Y+Yc)        ……Octet-2
    Plot (-X+Xc , Y+Yc)       ……Octet-3
    Plot (-Y+Xc , X+Yc)       ……Octet-4
    Plot (-Y+Xc , -X+Yc)      ……Octet-5
    Plot (-X+Xc , -Y+Yc)      ……Octet-6
    Plot (X+Xc , -Y+Yc)       ……Octet-7
    Plot (Y+Xc , -X+Yc)       ……Octet-8
}
```

Each plot function is for different octet and will construct the circle while in loop.

Bresenham's circle drawing algorithm

Step 1: Start

Step 2: Get the Radius of Circle R and Coordinates of center of circle (Xc, Yc).

Step 3: X and Y are going to be plotted points so, Set X=0 and Y=R.

Step 4: Initialize decision Parameter

D = 3-2R

Step 5: while (X <= Y)

Step 6: Plot Circle (Xc, Yc, X, Y)

Step 7: Increment value of X

X=X+1

Step 8: if (D < 0) Then

D = D + 4X + 6

Else

Y=Y-1

D=D+4(X-Y) + 10

Step 9: End while.

Step 10: Stop.

Ex: Given the center point coordinates (Xc, Yc) as (0, 0) and radius as 8, generate all the points to form a circle.

Sol:

Step1: assign the starting point coordinates (X, Y) as:

X=0 , Y=R=8

Step2: Calculate the initial value of decision parameter D as:

D = 3 – 2 * R = 3 – 2 * 8 = -13

Step3: As D < 0, so case1 is satisfied. Thus,

X(new)= X (old) + 1 = 0 + 1 = 1

Y(new)= Y(old) = 8

D(new) = D(old) + 4* X+ 6 = -13 + (4*1) + 6 = -3

Step4: repeat Step3 while (X<= Y).

D_k	D_{k+1}	(X, Y)
		(0, 8)
-13	-3	(1, 8)
-3	11	(2, 8)
11	5	(3, 7)
5	7	(4, 6)
7		(5, 5)
Algorithm Terminates **These are all points for Octet-2**		

Algorithm calculates all the points of **octet-2** and terminates. Now, the points of **octet-1** are obtained using the mirror effect by swapping X and Y coordinates.

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)
These are all points for Quadrant-1.	

Now, the points for rest of the part are generated by following the signs of other quadrants.

The other points can also be generated by calculating each octet separately.

Here, all the points have been generated with respect to quadrant-1.

Quadrant-1 (X,Y)	Quadrant-2 (-X,Y)	Quadrant-3 (-X,-Y)	Quadrant-4 (X,-Y)
(0, 8)	(0, 8)	(0, -8)	(0, -8)
(1, 8)	(-1, 8)	(-1, -8)	(1, -8)
(2, 8)	(-2, 8)	(-2, -8)	(2, -8)
(3, 7)	(-3, 7)	(-3, -7)	(3, -7)
(4, 6)	(-4, 6)	(-4, -6)	(4, -6)
(5, 5)	(-5, 5)	(-5, -5)	(5, -5)
(6, 4)	(-6, 4)	(-6, -4)	(6, -4)
(7, 3)	(-7, 3)	(-7, -3)	(7, -3)
(8, 2)	(-8, 2)	(-8, -2)	(8, -2)
(8, 1)	(-8, 1)	(-8, -1)	(8, -1)
(8, 0)	(-8, 0)	(-8, 0)	(8, 0)
These are all points of the Circle.			

Ex: Given the center point coordinates (Xc, Yc) as (0, 0) and radius as 10, generate all the points to form a circle.

Sol:

Step1: assign the starting point coordinates (X, Y) as:
X=0 , Y=R=10
Step2: Calculate the initial value of decision parameter D as:
D = 3 – 2 * R = 3 – 2 * 10 = -17
Step3: As D < 0, so case1 is satisfied. Thus,
X(new)= X (old) + 1 = 0 + 1 = 1
Y(new)= Y(old) = 10
D(new) = D(old) + 4* X+ 6 = -17 + (4*1) + 6 = -7
Step4: repeat Step3 while (X<= Y).

D_k	D_{k+1}	(X, Y)
		(0, 10)
-17	-7	(1, 10)
-7	7	(2, 10)
7	-7	(3, 9)
-7	15	(4, 9)
15	13	(5, 8)
13	19	(6,7)
Algorithm Terminates **These are all points for Octet-2**		

Algorithm calculates all the points of **octet-2** and terminates. Now, the points of **octet-1** are obtained using the mirror effect by swapping X and Y coordinates.

Octet-2 Points	**Octet-1 Points**
(0, 10)	(7, 6)
(1, 10)	(8, 5)
(2, 10)	(9, 4)
(3, 9)	(9, 3)
(4, 9)	(10, 2)
(5, 8)	(10, 1)
(6,7)	(10,0)
These are all points for Quadrant-1.	

Now, the points for rest of the part are generated by following the signs of other quadrants.

The other points can also be generated by calculating each octet separately.

Here, all the points have been generated with respect to quadrant-1.

Quadrant-1 (X,Y)	Quadrant-2 (-X,Y)	Quadrant-3 (-X,-Y)	Quadrant-4 (X,-Y)
(0, 10)	(0, 10)	(0, -10)	(0, -10)
(1, 10)	(-1, 10)	(-1, -10)	(1, -10)
(2, 10)	(-2, 10)	(-2, -10)	(2, -10)
(3, 9)	(-3, 9)	(-3, -9)	(3, -9)
(4, 9)	(-4, 9)	(-4, -9)	(4, -9)
(5, 8)	(-5, 8)	(-5, -8)	(5, -8)
(6, 7)	(-6, 7)	(-6, -7)	(6, -7)
(7, 6)	(-7, 6)	(-7, -6)	(7, -6)
(8, 5)	(-8, 5)	(-8, -5)	(8, -5)
(9, 4)	(-9, 4)	(-9, -4)	(9, -4)
(9,3)	(-9, 3)	(-9, -3)	(9, -3)
(10,2)	(-10, 2)	(-10, -2)	(10, -2)
(10, 1)	(-10, 1)	(-10, -1)	(10, -1)
(10,0)	(-10,0)	(-10,0)	(10,0)

Ex: Given the center point coordinates (Xc, Yc) as (10, 10) and radius as 10, generate all the points to form a circle.

Sol:

Step1: assign the starting point coordinates (X, Y) as:
X=0 , Y= R =10
Step2: Calculate the initial value of decision parameter D as:
D = 3 – 2 * R = 3 – 2 * 10 = -17
Step3: As D < 0, so case1 is satisfied. Thus,
X(new)= X (old) + 1 = 0 + 1 = 1
Y(new)= Y(old) = 10
D(new) = D(old) + 4* X+ 6 = -17 + (4*1) + 6 = -7
Step4: Xplot = X + Xc = 1 + 10 = 11
Yplot = Y + Yc = 10 + 10 = 20
Step5: repeat Step3 and Step4 while (Xplot <= Yplot).

D_k	D_{k+1}	(X, Y)	(X_{plot}, Y_{plot})
		(0, 10)	(10, 20)
-17	-7	(1, 10)	(11, 20)
-7	7	(2, 10)	(12, 20)
7	-7	(3, 9)	(13, 19)
-7	15	(4, 9)	(14, 19)
15	13	(5, 8)	(15, 18)
13	19	(6, 7)	(16, 17)
Algorithm Terminates **These are all points for Octet-2.**			

Algorithm calculates all the points of **octet-2** and terminates. Now, the points of **octet-1** are obtained using the mirror effect by swapping X and Y coordinates.

Octet-2 Points	**Octet-1 Points**
(10, 20)	(17, 16)
(11, 20)	(18, 15)
(12, 20)	(19, 14)
(13, 19)	(19, 13)
(14, 19)	(20, 12)
(15, 18)	(20, 11)
(16, 17)	(20, 10)
These are all points for Quadrant-1.	

Now, the points for rest of the part are generated by following the signs of other quadrants. The other points can also be generated by calculating each octet separately.

The following table, all the points have been generated with respect to quadrant-1.

Quadrant-1 (X,Y)	Quadrant-2 (-X,Y)	Quadrant-3 (-X,-Y)	Quadrant-4 (X,-Y)
(10, 20)	(10, 20)	(10, 0)	(10, 0)
(11, 20)	(9, 20)	(9, 0)	(11, 0)
(12, 20)	(8, 20)	(8, 0)	(12, 0)
(13, 19)	(7, 19)	(7, 1)	(13, 1)
(14, 19)	(6, 19)	(6, 1)	(14, 1)
(15, 18)	(5, 18)	(5, 2)	(15, 2)
(16, 17)	(4, 17)	(4, 3)	(16, 3)
(17, 16)	(3, 16)	(3, 4)	(17, 4)
(18, 15)	(2, 15)	(2, 5)	(18, 5)
(19, 14)	(1, 14)	(1, 6)	(19, 6)
(19, 13)	(1, 13)	(1, 7)	(19, 7)
(20, 12)	(0, 12)	(0, 8)	(20, 8)
(20, 11)	(0, 11)	(0, 9)	(20, 9)
(20, 10)	(0, 10)	(0, 10)	(20, 10)
These are all points of the Circle.			

Advantages of Bresenham's Circle Drawing Algorithm

The advantages of Bresenham's Circle Drawing Algorithm are-

- The entire algorithm is based on the simple equation of circle $X^2 + Y^2 = R^2$.
- It is easy to implement.

Disadvantages of Bresenham's Circle Drawing Algorithm

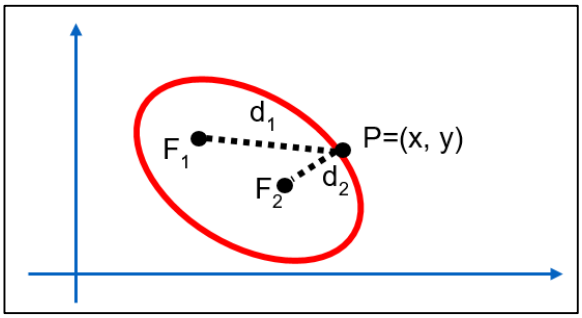
The disadvantages of Bresenham's Circle Drawing Algorithm are-

- Accuracy of the generating points is an issue in this algorithm.
- This algorithm suffers when used to generate complex and high graphical images.
- There is no significant enhancement with respect to performance.

Ellipse Generation Algorithm

Ellipse is a modified circle whose radius varies from a maximum value in one direction to a minimum value in the perpendicular direction (Ellipse is an elongated circle).

Ellipse is also defined as the geometric figure which is the set of all points on a plane whose distance from two fixed points **known as the foci** remains a constant.

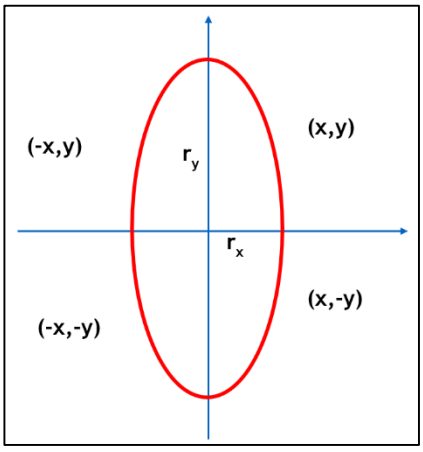


Foci of the ellipse.

Note: Foci of the ellipse, is the distance from any point on the ellipse to two fixed position. The sum of these two distances is the same value for all points on the ellipse.

Note: Ellipse consists of two axes: major and minor axes where the major axis is the longest diameter and minor axis is the shortest diameter.

Note: Unlike circle, the ellipse has **four-way symmetry property** which means that only the quadrants are symmetric **while the octants are not**. Here, we will calculate the points for one quadrant while the points for the remaining three can be calculated using the former points.



Four-way symmetry.

Note: Ellipse equation centered at the origin is**:**

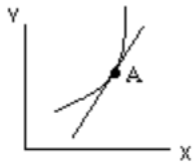
$$r_y^2 \cdot x^2 + r_x^2 \cdot y^2 = r_y^2 \cdot r_x^2$$

Determining the Slope of a Curve at the Point of Tangency

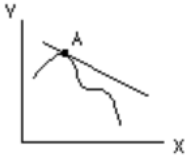
One of the differences between the slope of a straight line and the slope of a curve is that **the slope of a straight line is constant**, while the slope of a curve **changes from point to point**. It changes as you move along it. For this reason, we measure the slope of a curve at just one point. For example, instead of measuring the slope as the change between any two points (between A and B or B and C), we measure the slope of the curve at a single point (at A or C). We can do this by using the **Tangent line**.

A tangent is a straight line that touches a curve at a single point and does not cross through it. The point where the curve and the tangent meet is called **the point of tangency**. Both of the figures below show a tangent line to the curve.

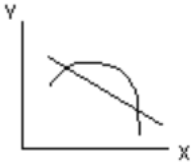
This curve has a tangent line to the curve with point A being the point of tangency. In this case, the slope of the tangent line is positive.



This curve has a tangent line to the curve with point A being the point of tangency. In this case, the slope of the tangent line is negative.



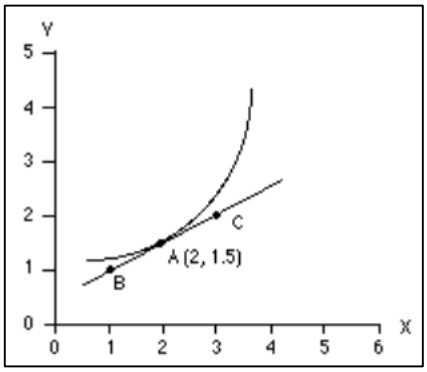
The line on this graph crosses the curve in two places. This line is **not** tangent to the curve.



The slope of a curve at a point **is equal to the slope of the straight line that is tangent to the curve at that point.**

Example

What is the slope of the curve at point A?



Sol:

The slope of the curve at point A is equal to the slope of the straight line BC. By finding the slope of the straight line BC, we have found the slope of the curve at point A.

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

dx = x2 – x1 = 3 -1 = 2

dy = y2 – y1= 2 – 1 = 1

slope = dy/dx

The slope at point A is 1/2, or 0.5.

This is the slope of the curve only at point A. To find the slope of the curve at any other point, we would need to draw a tangent line at that point and then determine the slope of that tangent line.

Mid-Point Ellipse drawing Algorithm:

In computer graphics, the **mid-point ellipse algorithm** is an incremental method of drawing an ellipse. This method is modified from Bresenham's algorithm used in the circle generation. The **mid-point ellipse drawing algorithm** is used to calculate all the perimeter points of an ellipse. **In this algorithm**, the mid-point between the two pixels is **calculated** which helps in calculating the decision parameter. The value of the decision parameter determines whether the mid-point lies inside, outside, or on the ellipse boundary and the then position of the mid-point helps in drawing the ellipse.

Now, for understanding the proper working of the algorithm, let us consider the elliptical curve in the first quadrant.

As mentioned before, the Ellipse equation is:

$$r_y^2 \cdot x^2 + r_x^2 \cdot y^2 = r_y^2 \cdot r_x^2$$

For a given point $P(x_i, y_i)$, the quantity f(x, y) (Ellipse function)

$$f(x, y) = r_y^2 \cdot x^2 + r_x^2 \cdot y^2 - r_y^2 \cdot r_x^2$$

is a measure telling where P lies in relation to the true ellipse:

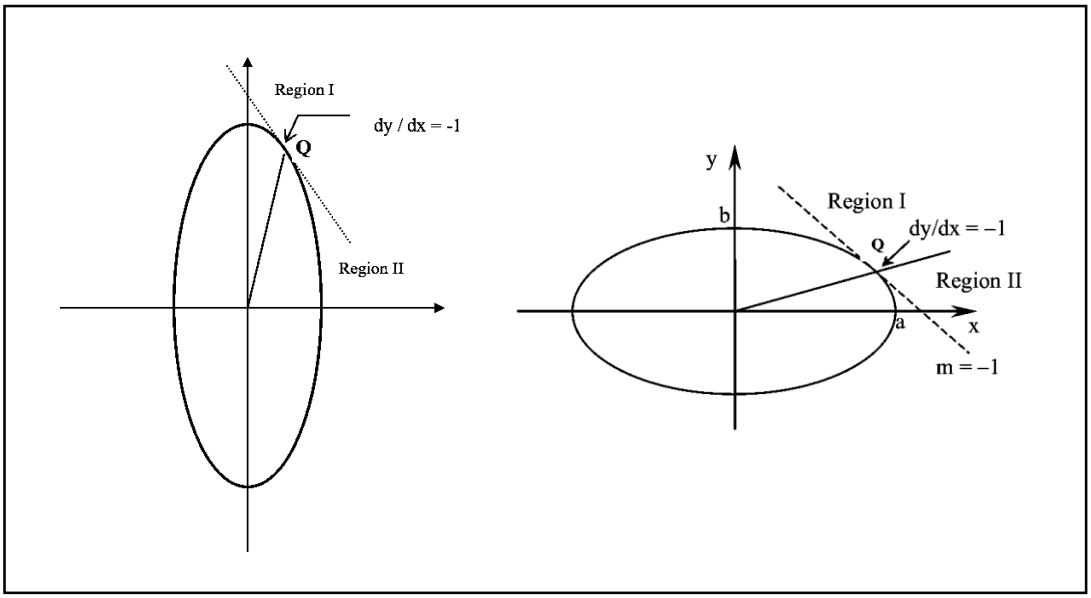
f(x, y) < 0 then (x, y) is inside the ellipse.

f(x, y) > 0 then (x, y) is outside the ellipse.

f(x, y) = 0 then (x, y) is on the ellipse

In the ellipse each quadrant is divided into two regions: I and II respectively. In region I, the slope on the curve is greater than –1 while in region II less than –1. At point Q the slope of the curve is -1, where the slope of an ellipse is given by:

$m = dy/dx = - (fx / fy) = - (2 r_y^2 x / 2 r_x^2 y) = -1$, where fx & fy are partial derivatives of f(x, y).



The starting point for region I will be (0, r) and for region II, the initial points will become the ending points of region I, where the slope becomes less than -1. That's why we need to keep in check the value of slope while plotting the points for region I to know whether we have reached region II or not.

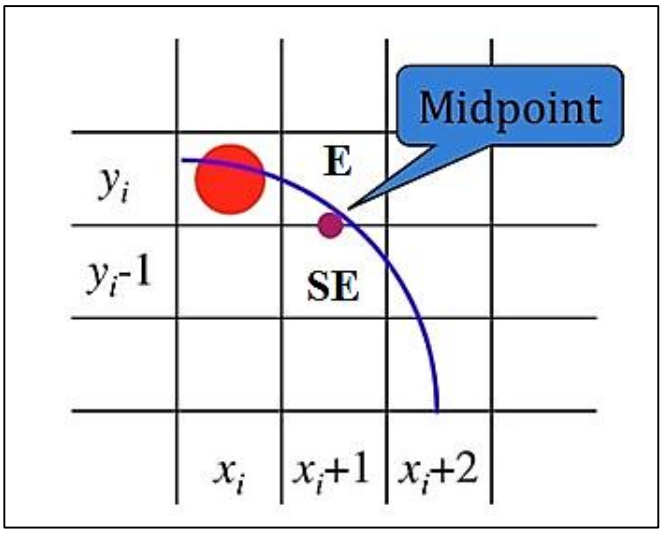
Derivation for Region I

In region I (dy/dx > –1), and X is always incremented but Y is not (Y_i or Y_i – 1). Let us consider two pixels: one which is outside **(E)** and the other which is inside **(SE)** respectively.

Let their coordinates be:

For E; (X_i +1, Y_i)

For SE; (X_i +1, Y_i - 1)



Region I.

Value for their mid-point will be:

$M_P = [((X_i +1 + X_i +1) / 2) , ((Y_i + Y_i – 1) / 2)]$

$= (X_i +1 , Y_i – 1 / 2)$

To define the decision parameter the M_P will be put in ellipse function f(x, y):

$$d1_i = r_y^2 (X_i + 1)^2 + r_x^2 (Y_i - 1 / 2)^2 - r_x^2 r_y^2$$

Successive parameter can be defined as:

$$d1_{i+1} = r_y^2 (X_{i+1} + 1)^2 + r_x^2 (Y_{i+1} - 1 / 2)^2 - r_x^2 r_y^2$$

After a number of derivation steps:

If d1 < 0 the midpoint is inside the ellipse then y_i is closer ($Y_i + 1 = Y_i$) (choose point **E**).

$d1 = d1 + 2\ r_y^2\ x + r_y^2$, with $2\ r_y^2\ x = 2\ r_y^2\ x + 2\ r_y^2$

We know that the $dx = 2\ r_y^2\ x$, so the above equations can be rewritten as the following:

$d1 = d1 + dx + r_y^2$

$dx = dx + 2\ r_y^2$

If d1>= 0 the midpoint is outside or on the ellipse then $y_i - 1$ is closer ($Y_i + 1 = Y_i - 1$) (choose point **SE**).

$d1 = d1 + 2\ r_y^2\ x + r_y^2 - 2\ r_x^2\ y$, with $2\ r_x^2\ y = 2\ r_x^2\ y - 2\ r_x^2$

We know that the $dy = 2\ r_x^2\ y$, so the above equations can be rewritten as the following:

$d1 = d1 + dx - dy + r_y^2$

$dy = dy - 2\ r_x^2$

The initial value of the decision parameter is:

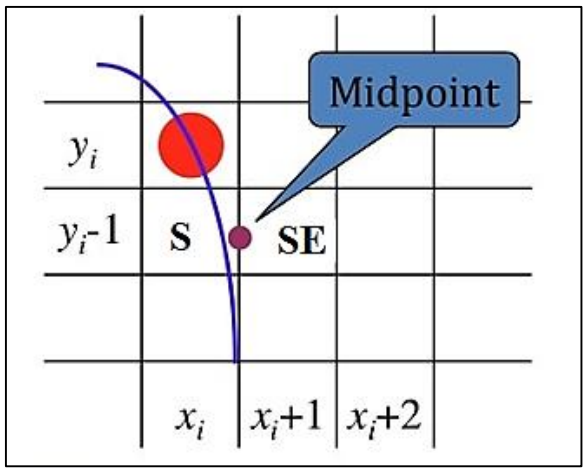
$d1 = r_y^2 - r_x^2\ r_y + ¼\ r_x^2$ (by putting the initial point $(0, r_y)$ in ellipse function f(x, y))

Derivation for Region II

In region II (dy/dx < –1), all calculations are similar to that in region I except that Y is always decremented but X is not (X_i or $X_i + 1$). Let us consider two pixels: one which is inside **(S)** and the other which is outside **(SE)** respectively. Let their coordinates be:

For S; (X_i , $Y_i - 1$)

For SE; (X_i +1, Y_i - 1)



Region II.

Value for their mid-point will be:

$M_P = [((X_i + 1 + X_i) / 2) , ((Y_i - 1 + Y_i - 1) / 2)]$

$= (X_i + 1 / 2, Y_i - 1)$

To define the decision parameter the M_P will be put in ellipse function f(x, y):

$d2_i = r_y^2 (X_i + 1/2)^2 + r_x^2 (Y_i - 1)^2 - r_x^2 r_y^2$

Successive parameter can be defined as:

$d2_{i+1} = r_y^2 (X_{i+1} + 1/2)^2 + r_x^2 (Y_{i+1} - 1)^2 - r_x^2 r_y^2$

After a number of derivation steps:

If d1> 0 the midpoint is outside the ellipse then x_i is closer ($x_i + 1 = x_i$) (choose point **S**).

$d2 = d2 - 2 r_x^2 y + r_x^2$, with $2 r_x^2 y = 2 r_x^2 y - 2 r_x^2$

We know that the $dy = 2 r_x^2 y$, so the above equations can be rewritten as the following:

$d2 = d2 + r_x^2 - dy$

$dy = dy - 2 r_x^2$

If d2<= 0 the midpoint is inside the ellipse then $x_i + 1$ is closer ($x_i + 1 = x_i + 1$) (choose point **SE**).

$d2 = d2 + 2 r_y^2 x - 2 r_x^2 y + r_x^2$, with $2 r_y^2 x = 2 r_y^2 x + 2 r_y^2$

$2 r_x^2 y = 2 r_x^2 y - 2 r_x^2$

We know that the $dx = 2 r_y^2 x$, so the above equations can be rewritten as the following:

$d2 = d2 + dx - dy + r_x^2$

$dx = dx + 2 r_y^2$

$dy = dy - 2 r_x^2$

The initial value of the decision parameter is:

$d2 = r_y^2 (x + 1/2)^2 + r_x^2 (y - 1)^2 - r_x^2 r_y^2$ (by putting the last point(X_i, Y_i) of Region I in ellipse function f(x, y))

The algorithm described above shows how to obtain the pixel coordinates in the first quadrant only. The ellipse centre is assumed to be at the origin. In actual implementation, the pixel coordinates in other quadrants can be simply obtained by use of the symmetric characteristics of an ellipse. For a pixel (x, y) in the first quadrant, the corresponding pixels in other three quadrants are (x, –y), (–x, y) and (–x, –y) respectively. If the centre is at (XC, YC), all calculated coordinate (x, y) should be adjusted by adding the offset (XC, YC). For easy implementation, a function PlotEllipse() is defined as follows:

PlotEllipse (XC, YC, X, Y)

putpixel (XC+X, YC+Y)

putpixel (XC+X, YC–Y)

putpixel (XC–X, YC+Y)

putpixel (XC–X, YC–Y)

end PlotEllipse

Mid-Point Ellipse drawing Algorithm

Step 1: Start

Step 2: Take input radius along x axis (r_x) and y axis (r_y) and obtain center of ellipse (Xc, Yc).

Step 3: Initialize the initial point of **Region1** (we assume ellipse to be centered at origin) as:

$X=0, Y=r_y$

Step4: Obtain the initial decision parameter for **Region1** as:

$$d1 = r_y^2 + 0.25 * r_x^2 - r_x^2 * r_y$$

Step5: initialize the partial derivatives (dx and dy) as:

$dx = 2\, r_y^2\, x$

$dy = 2\, r_x^2\, y$

Step6: Repeat steps while (dx < dy)

Plot (X+Xc, Y+Yc)

Plot (-X+Xc, Y+Yc)

Plot (-X+Xc, -Y+Yc)

Plot (X+Xc, -Y+Yc)

X = X+1

if (d1 < 0)

{

dx = dx + $2\, r_y^2$

d1 = d1 + dx + r_y^2

}

else

{

y=y-1

dx = dx + 2 r_y^2

dy = dy - 2 r_x^2

d1 = d1 + dx – dy + r_y^2

}

Step 7: When dx ≥ dy, plot **Region 2**:

Step 8: Obtain the initial decision parameter for **Region2** (using the last point of region1) as:

d2= $r_y^2 (x + 0.5)^2 + r_x^2 (y - 1)^2 - r_x^2 r_y^2$

Step 9: while (y>=0)

Y = Y – 1

if (d2 > 0)

{

dy = dy - 2 r_x^2

d2 = d2 + r_x^2 – dy

}

else

{

X = X + 1

dx = dx + 2 r_y^2

dy = dy - 2 r_x^2

d2 = d2 + dx – dy + r_x^2

}

Plot (X+Xc, Y+Yc)

Plot (-X+Xc, Y+Yc)

Plot (-X+Xc, -Y+Yc)

Plot (X+Xc, -Y+Yc)

Step 10: Stop

Ex: Given input ellipse parameter r_x=8 and r_y=6, determine pixel positions **along the ellipse path in the first quadrant** using the midpoint ellipse algorithm.

Sol:

r_x=8 , r_y = 6

Region 1

$(X_0, Y_0) = (Xc, r_y) = (0, 6)$

$dx = 2\, r_y^2\, x = 0$ (with increment $2\, r_y^2 = 72$)

$dy = 2\, r_x^2\, y = 2\, r_x^2\, r_y = 2*64*6 = 768$ (with increment $-2\, r_x^2 = -128$)

$d1 = r_y^2 + 0.25 r_x^2 - r_x^2 * r_y$

$= 36 + 16 - 64*6 = -332$

I	d1	(x, y)	dx	dy
		(0, 6)	0	768
1	-332	(1,6)	72	768
2	-224	(2,6)	144	768
3	-44	(3,6)	216	768
4	208	(4,5)	288	640
5	-108	(5,5)	360	640
6	288	(6,4)	432	52
7	244	(7,3)	504	384

Move out region 1 since dx > dy

Region 2

$(X_0, Y_0) = (7, 3)$ (last position in Region 1)

Last value of dx = 504 (with increment 2 r_y^2 = 72)

Last value of dy = 384 (with increment - 2 r_x^2 = -128)

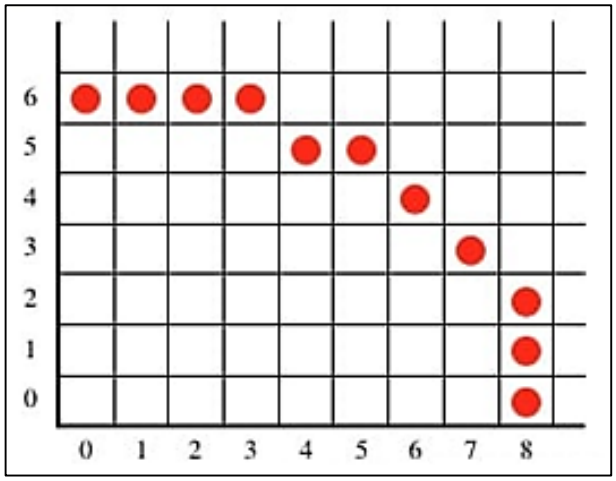
d2= $r_y^2 (x + 1/2)^2 + r_x^2 (y - 1)^2 - r_x^2 r_y^2$

= $36(7 + 0.5)^2 + 64 (3 - 1)^2$ - 64*36

= 36(56.25) + 64(4) – 64*36 = -23

I	d2	(x, y)	dx	dy
1	-23	(8,2)	576	256
2	407	(8,1)	576	128
3	343	(8,0)	--	--

Stop at Y = 0



Note: If the ellipse center is not (0, 0), the point can be found as you learned in the circle algorithm.

H. W: Can the Mid-Point Ellipse drawing Algorithm draw a circle?

2D Geometric Transformation

Geometric Transformation means changing some graphics into something else by applying rules, or it is the geometrical changes of an object from a current state to modified state.

There are two ways or types of 2D Transformation:

1- Object Transformation: Alter the coordinates descriptions of an object, while the Coordinate system unchanged.
2- Coordinate transformation: Produce a different coordinate system.

Why the Transformation is needed?

To manipulate the initially created object and to display the modified object without having to redraw it.

2D Geometric Transformation (Object Transformation)

We can have various types of object transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.

Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

In 2D object, transformation is to change:

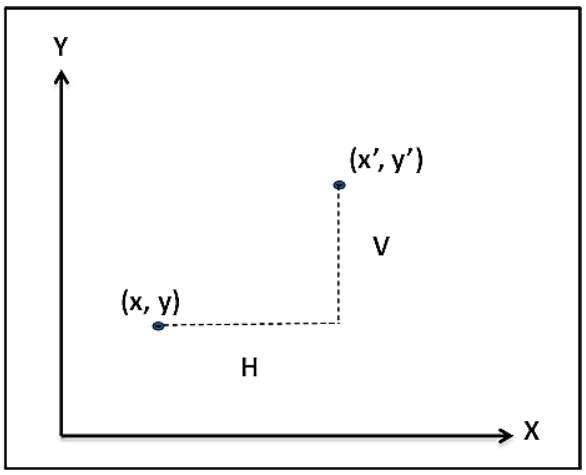
- Object's Position (Translation).
- Object's Orientation (Rotation).
- Object's Size (Scaling).

Basic Transformations

- Object's Reflection (Mirror Image).
- Object's shapes (shear).

Translation

A translation moves all points in an object along the same straight-line path to new positions on the screen. You can translate a point in 2D by adding translation coordinate (H, V) to the original coordinate X,Y to get the new coordinate X′,Y′. Where, H is added to X and V is added to Y. see the figure below.



Horizontal and Vertical displacement.

A translation can also be interpreted as the addition of a constant vector to every point, or as shifting the origin of the coordinate system.

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

Note: Translation coordinates (H, V) are also called “Translation or Shift Vector” or “Translation Factor”.

Note: The H and V represent the Horizontal and Vertical displacement or distance that point has moved, So:

1- If H is **Positive**, the point moves to the **Right**.

2- If H is **Negative**, the point moves to the **Left**.

3- If V is **Positive**, the point moves to the **Up**.

4- If V is **Negative**, the point moves to the **Down**.

We can apply Translation on following objects-

- Point (pixel).
- Line.
- Rectangle.
- Polygon.
- Square.
- Circle.

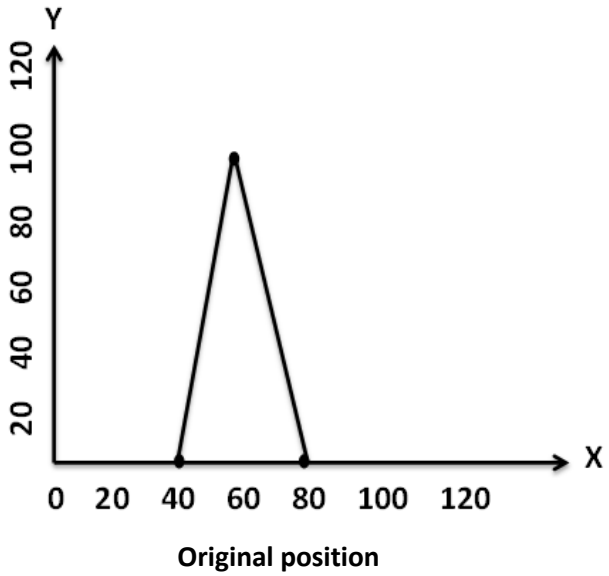
Note: By translation, we can move any object from one to another place without changing the shape, size, or orientation of the object. So translation is called **rigid transformation**.

Ex: Given a triangle with its three vertices (40, 0), (80, 0), (60, 100). Apply translation with 120 units to the right and 20 units up.

Sol:

Old coordinates of the triangle = (40, 0), (80, 0), (60, 100).

Translation coordinate = (H, V) = (120, 20)



Original position

Let the new vertix (Xnew, Ynew)

for the coordinates (40, 0)

Xnew= Xold + H = 40 + 120 =160

Ynew= Yold + V = 0 + 20 = 20

Thus, new coordinates for (40, 0) is (160, 20)

for the coordinates (80, 0)

Xnew= Xold + H = 80 + 120 = 200

Ynew= Yold + V = 0 + 20 = 20

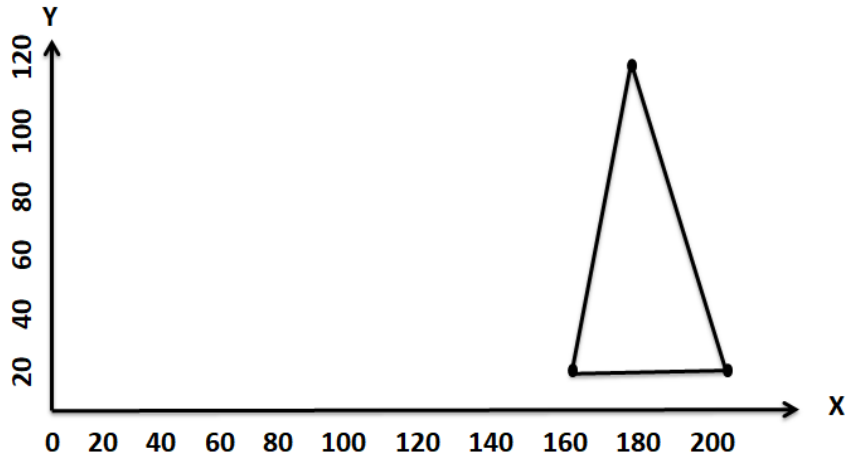
Thus, new coordinates for (80, 0) is (200, 20)

for the coordinates (60, 100)

Xnew= Xold + H = 60 + 120 =180

Ynew= Yold + V = 100 + 20 = 120

Thus, new coordinates for (60, 100) is (180, 20)



Position after Translation

H. W

1- Write a function to translate any object (up, down, right, left).

2- Write a program to draw a polygon and using translated function to translate it in any direction

3- Consider a triangle defined by its three vertices (20, 0), (60, 0), (40, 100) be translated 20 units to the Left.

4- Consider a triangle defined by its three vertices (40, 0), (100, 0), (60, 100) be translated 40 units to the Left and 40 units down.

Homogeneous Coordinate Representation

The previous Translation is also shown in the form of 3 x 3 matrix:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{pmatrix}$$

Translation Matrix

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{pmatrix}$$

Note: for translating an object, each vertex must multiply by translation matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{pmatrix} = \begin{pmatrix} X1' & Y1' & 1 \\ X2' & Y2' & 1 \\ X3' & Y3' & 1 \end{pmatrix}$$

Ex: Given a traingle with its three vertices (40, 100), (20, 0), (60, 0). Apply translation with 20 units to the right using matrix representation.

Sol:

Translation matrix is

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \mathbf{20} & \mathbf{0} & 1 \end{pmatrix}$$

$$\begin{pmatrix} 40 & 100 & 1 \\ 20 & 0 & 1 \\ 60 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 20 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 60 & 100 & 1 \\ 40 & 0 & 1 \\ 80 & 0 & 1 \end{pmatrix}$$

Old vertices **New vertices**

40*1 + 100*0 + 1*20 = 60

40*0 + 100*1 + 1*0 = 100

40*0 + 100*0 + 1*1 = 1

20*1 + 0*0 + 1*20 = 40

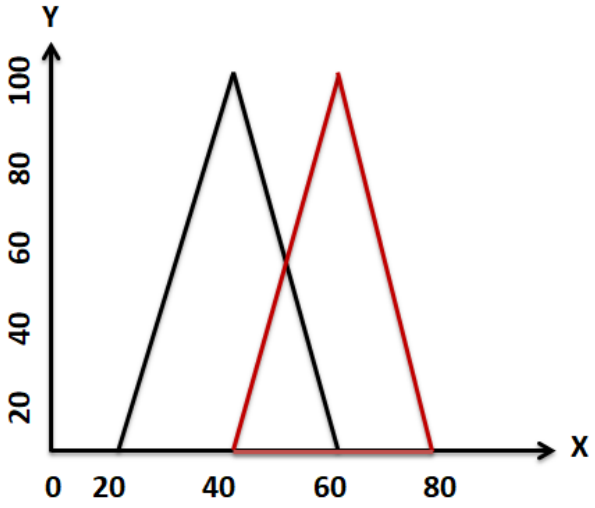
20*0 + 0*1 + 1* 0 = 0

20*0 + 0*0 + 1*1 = 1

60*1 + 0*0 + 1*20 = 80

60*0 + 0*1 + 1*0 = 0

60*0 + 0*0 + 1*1 = 1

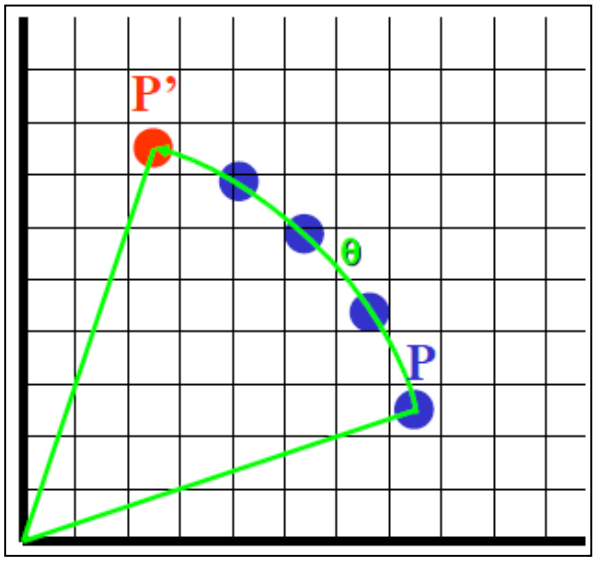


Triangle before and after translation

H. W: Given a triangle with its three vertices (40, 0), (100, 0), (60, 100). Apply translation with 40 units to the Left and 40 units down, using matrix representation.

Rotation

Rotation is a process of changing the angle of the object. Also, it is defined as a process of repositioning all points in an object along a circular path in the plane centered at the pivot point. Or rotation means that, all points of the object are transformed to new positions by rotating the points through a specified rotation angle about the rotation axis (in 2D, rotation pivot or pivot point).



A point Rotation

Where, P is original position. P' is final position or position after rotation and θ is angle of rotation.

The Rotation of any object depends upon the two things:

1- **Rotation Point**: It is also called the **Pivot point**. Where, the pivot point may be the **origin (0, 0)** or **any arbitrary point**.

2- **Rotation Angle**: It is denoted by **Theta (θ)**.

Note: a point located at the origin does not change its place; therefore, Rotation is relative to the origin.

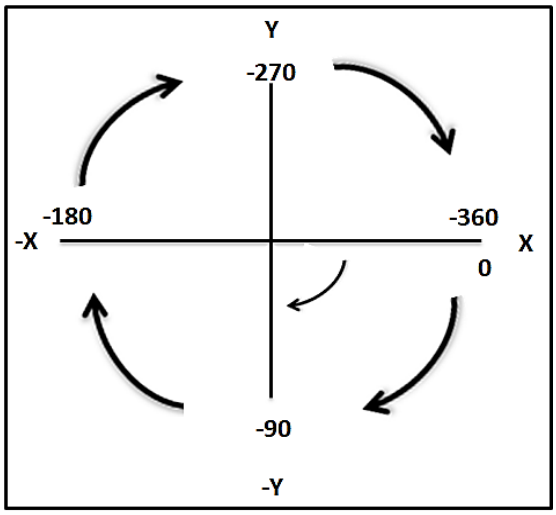
Note: An unwanted translation is a byproduct of rotation. So, the location of a rotated object can be controlled by choosing the location of a point (pivot point) with respect to which the rotation is performed.

Types of Rotation

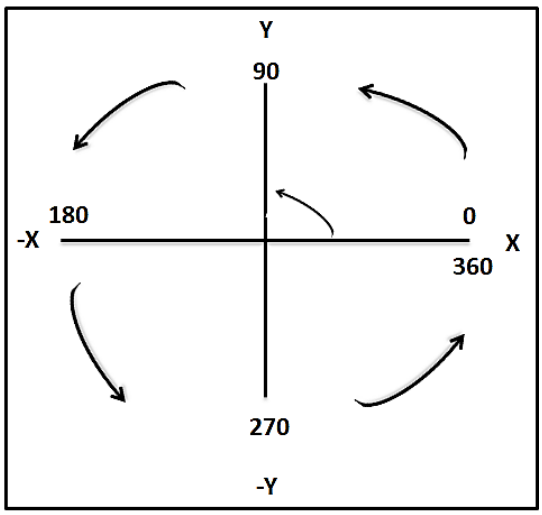
We can rotate an object in two ways or directions:-

1- **Clockwise**: An object rotates clockwise if the value of the Rotation angle is negative (-).

2- **Counter-Clockwise (Anti-Clockwise)**: An object rotates anti-clockwise if the value of the Rotation angle is positive (+).



Clockwise (angle is negative)



Counter clockwise (angle is positive)

Note: Rotation can be applied on the following objects:

- **Straight Line:** Straight Line is rotated by the endpoints with the same angle and redrawing the line between new endpoints.
- **Polygon:** Polygon is rotated by shifting every vertex using the same rotational angle.
- **Curved Lines:** Curved Lines are rotated by repositioning of all points and drawing of the curve at new positions.
- **Circle:** It can be obtained by center position by the specified angle.
- **Ellipse**: Its rotation can be obtained by rotating major and minor axis of an ellipse by the desired angle.

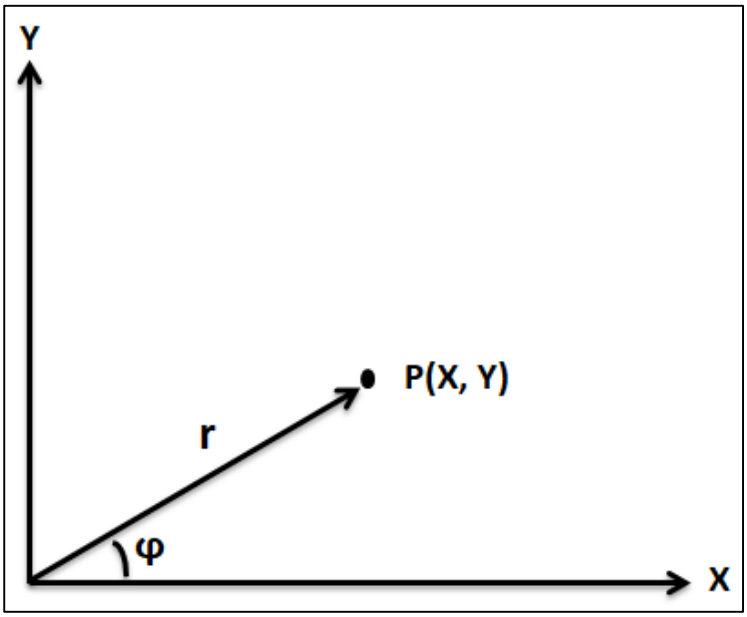
Note: By rotation, we can rotate an object about a fixed point without changing its size and shape (rigid transformation). But the orientation is changed.

Rotation Rules

Any point (X, Y) can be represented by:

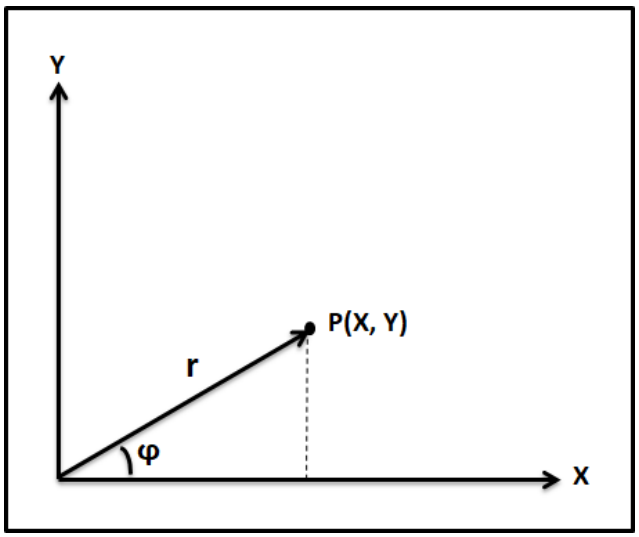
1- **Its radial distance (r)**, where, r is the Euclidean distance from the origin to the point(X, Y) its self.

2- **Its angle (ϕ)** of X-Axis.



X and Y can be calculated as following:

Note the figure below, after connecting the dotted straight line, we have a triangle. According to trigonometry, (r) is the **hypotenuse**, X-axis is the **adjacent side** to (ϕ), and dotted straight line (Y-axis) is the **opposite side** to (ϕ).



According to the trigonometric rules:

$\text{Cos}(\phi) = \frac{\text{Adj}}{\text{hyp}}$

$\text{Cos}(\phi) = \frac{X}{r}$

$\therefore\ X = r * \text{Cos}(\phi)$

$\text{Sin}(\phi) = \frac{opp}{hyp}$

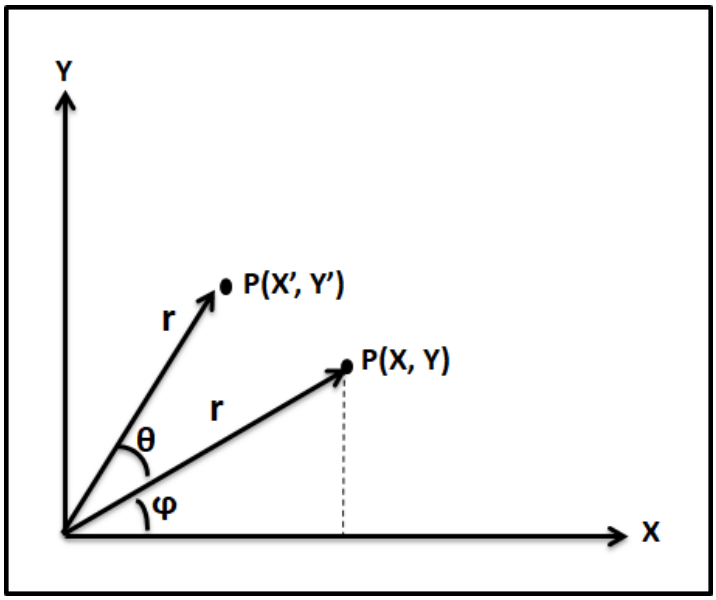
$\text{Sin}(\phi) = \frac{Y}{r}$

$\therefore\ Y = r * \text{Sin}(\phi)$

$$\left.\begin{aligned} X &= r * \text{Cos}(\phi) \\ Y &= r * \text{Sin}(\phi) \end{aligned}\right\} \quad \textbf{..... (1)}$$

If p(x, y) is rotated at an angle (θ) that is counterclockwise direction. The transformed point (X', Y') is represented as:

$$\left.\begin{aligned} X' &= r * \text{Cos}(\phi+\theta) \\ Y' &= r * \text{Sin}(\phi+\theta) \end{aligned}\right\} \quad \textbf{..... (2)}$$



By using the trigonometric rules, equation (2) becomes:

$$X' = r * \cos(\phi) * \cos(\theta) - r * \sin(\phi) * \sin(\theta)$$
$$Y' = r * \sin(\phi) * \cos(\theta) + r * \cos(\phi) * \sin(\theta) \quad \text{..... (3)}$$

From the definition of X and Y, equation (3) becomes:

$$X' = X * \cos(\theta) - Y * \sin(\theta)$$
$$Y' = Y * \cos(\theta) + X * \sin(\theta) \quad \text{..... (4)}$$

Note: So, **equation (4)** is the final formula that is used to rotate a point P(X, Y) counterclockwise (positive angle) **about the origin**.

To rotate a point P(X, Y) through a **clockwise** angle (θ) about the origin of the coordinates system , we must rewrite equation (4) as following to get equation(5):

$$X' = X * \cos(\theta) + Y * \sin(\theta)$$
$$Y' = - X * \sin(\theta) + Y * \cos(\theta) \quad \text{..... (5)}$$

Note: Cos (θ) = Cos (-θ), i.e. Cos (45) = 0.707, Cos (-45) = 0.707

Sin (-θ) = - Sin (θ), i.e. Sin (45) = 0.707, Sin (-45) = -0.707

Note: So, **equation (5)** is the final formula that is used to rotate a point P(X, Y) clockwise (negative angle) **about the origin**.

If we want to rotate an object at an angle (θ) about a pivot point (X_p, Y_p) other than the origin, we must perform the following three steps:

Step1: Translate

Translate the pivot point (X_p, Y_p) to the origin. Every point(x, y) defining the object is translated to a new point (X', Y') where:

$$X' = X - X_p$$
$$Y' = Y - Y_p$$

Step2: Rotate

Use these translated points (X', Y'), θ degree about the origin to obtain new point(X", Y") where:

$X'' = X' * \cos(\theta) - Y, * \sin(\theta)$

$Y'' = Y' * \cos(\theta) + X' * \sin(\theta)$

Step3: Translate

Translate the center of rotation **<u>back</u>** to the pivot point (X_p, Y_p).

$X''' = X'' + X_p$

$Y''' = Y'' + Y_p$

Note: The previous three steps can be performed in one step:

$X''' = (X - X_p) * \cos(\theta) - (Y - Y_p) * \sin(\theta) + X_p$

$Y''' = (Y - Y_p) * \cos(\theta) + (X - X_p) * \sin(\theta) + Y_p$

Ex: Rotate a line whose endpoints are (3, 4) and (12, 15) about origin through a 45° counterclockwise direction.

Sol:

Rotation angle: +45

Pivot point: origin (0,0)

$X' = X * \cos(\theta) - Y * \sin(\theta)$

$Y' = Y * \cos(\theta) + X * \sin(\theta)$

$\cos(45°) = 0.707$

$\sin(45°) = 0.707$

For (3, 4)

$X' = 3 * 0.707 - 4 * 0.707 = -0.707$

$Y' = 4 * 0.707 + 3 * 0.707 = 4.949$

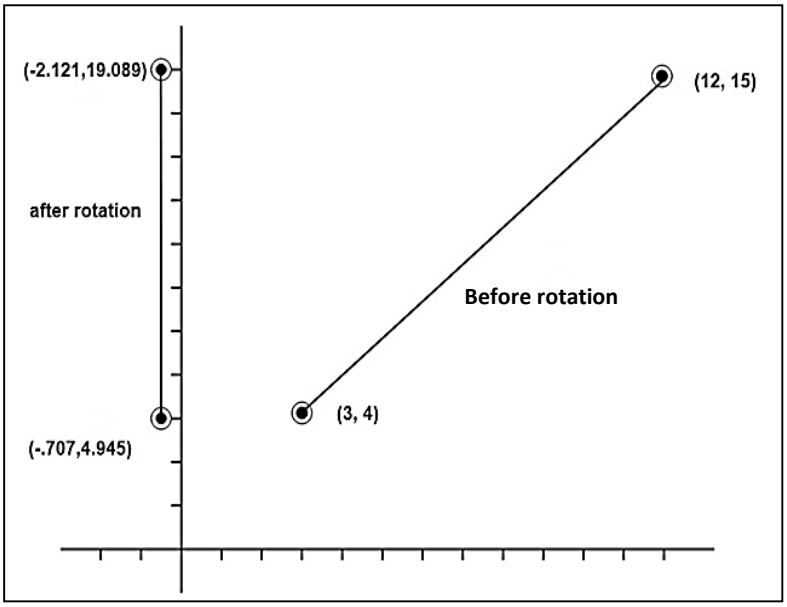
<u>The point (3, 4) after rotation will be (-0.707, 4.949)</u>

For (12, 5)

$X' = 12 * 0.707 - 15 * 0.707 = -2.121$

$Y' = 15 * 0.707 + 12 * 0.707 = 19.089$

<u>The point (12, 5) after rotation will be (-2.121, 19.089)</u>



Ex: Rotate a line whose endpoints are (3, 4) and (12, 15) about a pivot point (3, 4) through a 45° counterclockwise direction.

Sol:

Rotation angle: +45

Pivot point (X_p, Y_p): (3, 4)

Cos (45 °) = 0.707

Sin (45 °) = 0.707

1- Translate the line to the origin

$X' = X - X_p$

$Y' = Y - Y_p$

For (3, 4)

X' = 3 – 3 = 0

Y' = 4 – 4 = 0

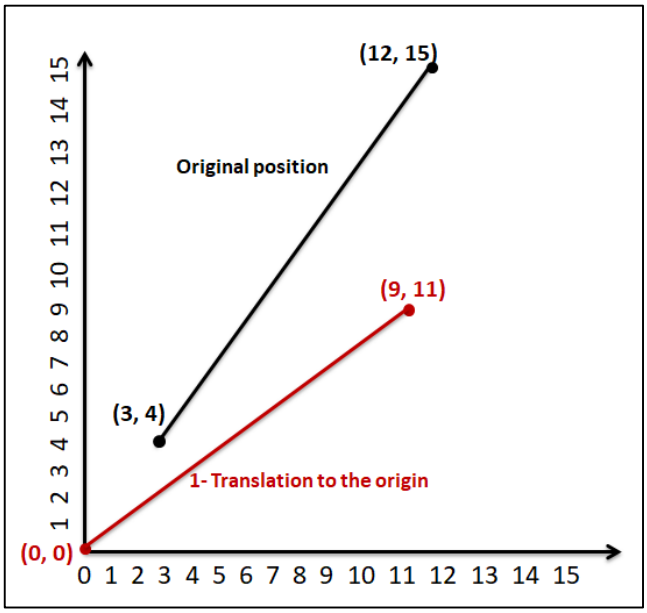
point (3, 4) after translation will be (0, 0)

For (12, 15)

X' = 12 – 3 = 9

Y' = 15 – 4 = 11

point (12, 15) after translation will be (9, 11)



2- Rotate the translated line about the origin.

Rotate angle: +45°

Pivot point: (0, 0)

X"= X' * Cos (θ) – Y' * Sin (θ)

Y"= Y' * Cos (θ) + X' * Sin (θ)

For (0, 0)

X" = 0 * 0.707 – 0 * 0.707 = 0

Y" = 0 * 0.707 + 0 * 0.707 = 0

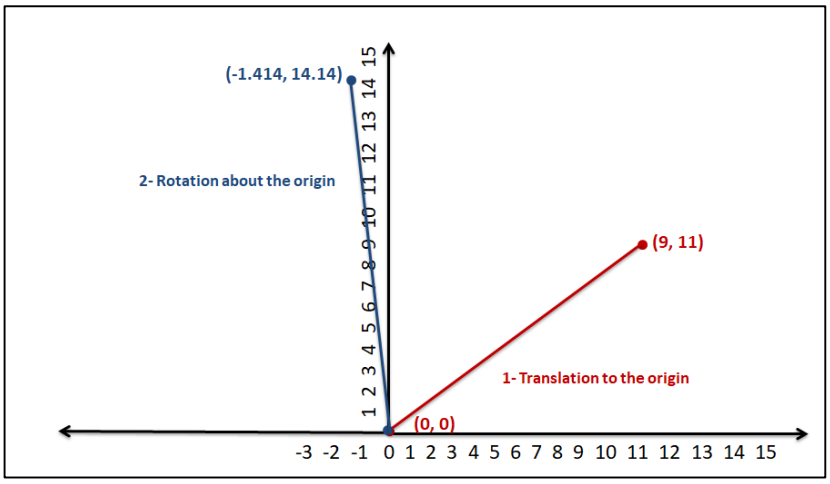
point (0, 0) after rotation about origin will be (0, 0)

For (9, 11)

X" = 9 * 0.707 – 11 * 0.707 = -1.414

Y" = 11 * 0.707 + 9 * 0.707 = 14.14

point (9, 11) after rotation about origin will be (-1.414, 14.14)



3- Translate back the rotated line.

$X''' = X'' + X_p$

$Y''' = Y'' + Y_p$

For (0, 0)

$X''' = 0 + 3 = 3$

$Y''' = 0 + 4 = 4$

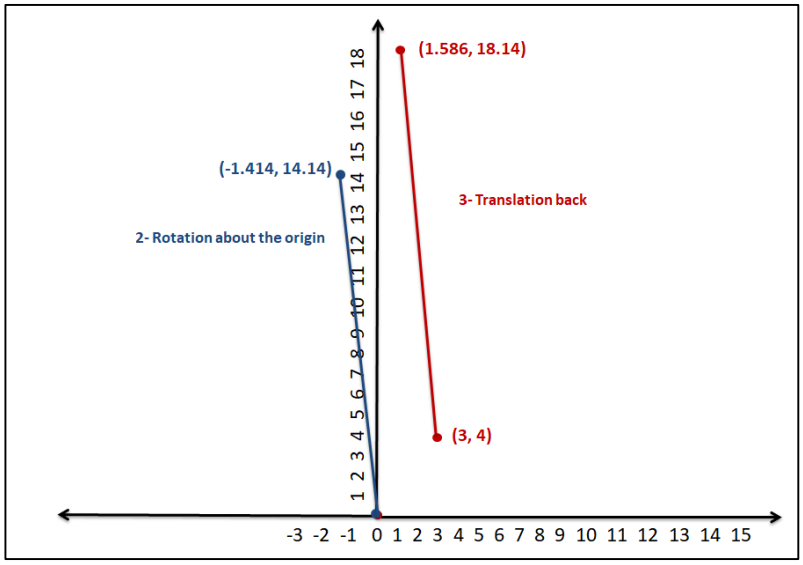
point (0, 0) after translation back will be (3, 4)

For (-1.414, 14.14)

$X''' = -1.414 + 3 = 1.586$

$Y''' = 14.14 + 4 = 18.14$

point (-1.414, 14.14) after translation back will be (1.586, 18.14)



H. W:

1. Rotate the triangle (10, 0), (30, 0), (50, 80) 45° counterclockwise about the origin.
2. Rotate the triangle (7, 8), (4, 4), (10, 5) 90° counterclockwise about the point (7, 8).
3. Rotate the above triangle 90° clockwise about the point (4, 4).
4. Write an equation to rotate any picture clockwise about the pivot point.
5. Write a program which rotates a polygon.
   - **a-** Counter clockwise about the origin.
   - **b-** Counter clockwise about a pivot point.

Homogeneous Coordinate Representation

The previous rotation is also shown in the form of 3 x 3 matrix:

1- Counterclockwise direction:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} \mathbf{Cos(\theta)} & \mathbf{Sin(\theta)} & \mathbf{0} \\ \mathbf{-Sin(\theta)} & \mathbf{Cos(\theta)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

2- Clockwise direction:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} \mathbf{Cos(\theta)} & \mathbf{-Sin(\theta)} & \mathbf{0} \\ \mathbf{Sin(\theta)} & \mathbf{Cos(\theta)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

Note: for rotation an object, each vertex must multiply by rotation matrix.

$$\begin{pmatrix} X_1 & Y_1 & 1 \\ X_2 & Y_2 & 1 \\ X_3 & Y_3 & 1 \end{pmatrix} * \begin{pmatrix} \text{Cos}(\theta) & \text{Sin}(\theta) & 0 \\ -\text{Sin}(\theta) & \text{Cos}(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: Rotate a line whose endpoints are (3, 4) and (12, 15) about origin through a 45° counterclockwise direction, using matrix representation.

Sol:

Rotation angle: +45

Pivot point: (0, 0)

Cos (45 °) = 0.707

Sin (45 °) = 0.707

$$\begin{pmatrix} \text{Cos}(\theta) & \text{Sin}(\theta) & 0 \\ -\text{Sin}(\theta) & \text{Cos}(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Counterclockwise rotation matrix

$$\begin{pmatrix} 3 & 4 & 1 \\ 12 & 15 & 1 \end{pmatrix} * \begin{pmatrix} 0.707 & 0.707 & 0 \\ -0.707 & 0.707 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -0.707 & 4.949 & 1 \\ -2.121 & 19.089 & 1 \end{pmatrix}$$

Ex: Rotate a line whose endpoints are (3, 4) and (12, 15) about a pivot point (3, 4) through a 45° counterclockwise direction, using matrix representation.

Sol:

Rotation angle: +45

Pivot point (X_p, Y_p): (3, 4)

Cos (45 °) = 0.707

Sin (45 °) = 0.707

$$\begin{pmatrix} \mathbf{Cos(\theta)} & \mathbf{Sin(\theta)} & \mathbf{0} \\ \mathbf{-Sin(\theta)} & \mathbf{Cos(\theta)} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

Counterclockwise rotation matrix

1- Translate the line to the origin.

$$\begin{pmatrix} \mathbf{3} & \mathbf{4} & \mathbf{1} \\ \mathbf{12} & \mathbf{15} & \mathbf{1} \end{pmatrix} * \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{-3} & \mathbf{-4} & \mathbf{1} \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{9} & \mathbf{11} & \mathbf{1} \end{pmatrix}$$

2- Rotate the translated line about the origin.

$$\begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{9} & \mathbf{11} & \mathbf{1} \end{pmatrix} * \begin{pmatrix} \mathbf{0.707} & \mathbf{0.707} & \mathbf{0} \\ \mathbf{-0.707} & \mathbf{0.707} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{-1.414} & \mathbf{14.14} & \mathbf{1} \end{pmatrix}$$

3- Translate back the rotated line.

$$\begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{-1.414} & \mathbf{14.14} & \mathbf{1} \end{pmatrix} * \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{3} & \mathbf{4} & \mathbf{1} \end{pmatrix} = \begin{pmatrix} \mathbf{3} & \mathbf{4} & \mathbf{1} \\ \mathbf{1.586} & \mathbf{18.14} & \mathbf{1} \end{pmatrix}$$

H. W:

1- Rotate the triangle (10, 0), (30, 0), (50, 80) 45° counterclockwise about the origin, using matrix representation.

2- Rotate the triangle (7, 8), (4, 4), (10, 5) 90° counterclockwise about the point (7, 8), using matrix representation.

3- Rotate the above triangle 90° clockwise about the point (4, 4), using matrix representation.

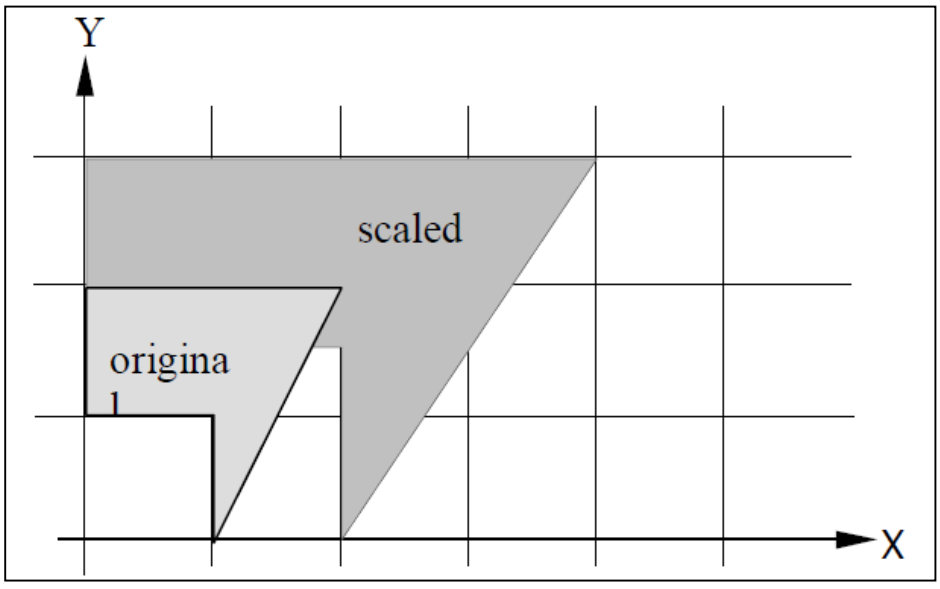
Scaling

A scaling transformation is used to alter or change the size of objects. In the scaling process, we either compress or expand the dimension of the object. Scaling operation can be achieved by multiplying each vertex coordinate (x, y) of the polygon by scaling factor to produce the transformed coordinates as (x', y'). There are two scaling factors, **Sx** in x direction and **Sy** in y-direction.

Generally, scaling factor is (Sx, Sy), but scaling equations are:

$$\left.\begin{aligned} X' &= X * Sx \\ Y' &= Y * Sy \end{aligned}\right\} \text{ Scaling about the origin}$$

Note: a point located at the origin does not change its place; therefore, Scaling is relative to the origin (as rotation), see the figure below.



Note: An unwanted translation is a byproduct of scaling. So, the location of a scaled object can be controlled by choosing the location of a point (fixed point) with respect to which the scaling is performed.

Ex: Scale a triangle whose its vertices are (4,4), (7, 8), (10, 5) by Sx = 2 and Sy = 2, about the origin point.

Sol:

Old coordinates of the triangle = (4,4), (7, 8), (10, 5).

Scaling factor (Sx, Sy) = (2, 2)

pivot point (Xp, Yp) = (0,0)

For (4, 4)

Xnew = Xold * Sx = 4 * 2 = 8

Ynew = Yold * Sy = 4 * 2 = 8

point (4, 4) after scaling about origin will be (8, 8)

For (7, 8)

Xnew = Xold * Sx = 7 * 2 = 14

Ynew = Yold * Sy = 8 * 2 = 16

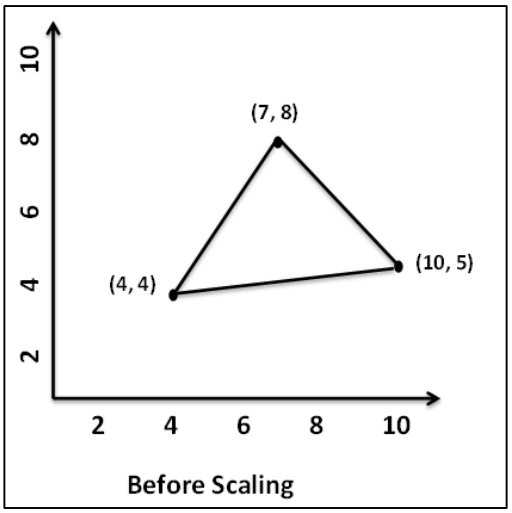
point (7, 8) after scaling about origin will be (14, 16)

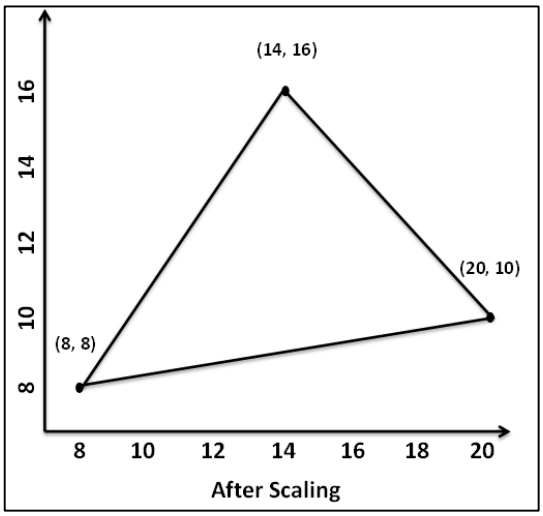
For (10, 5)

Xnew = Xold * Sx = 10 * 2 = 20

Ynew = Yold * Sy = 5 * 2 = 10

point (10, 5) after scaling about origin will be (20, 10)



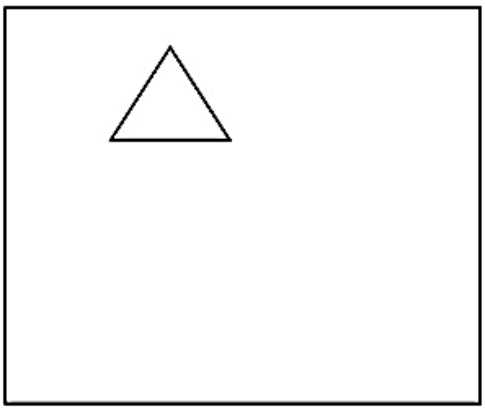


Scaling Factors Effects

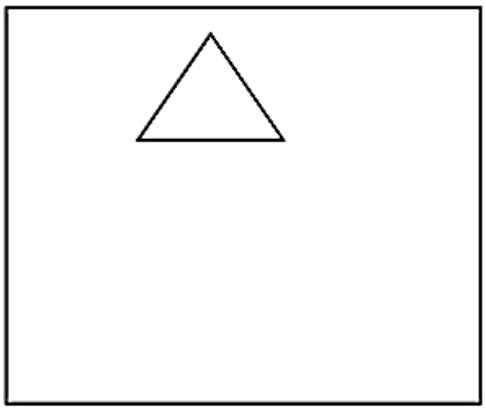
Scaling factors affect size as following:

- If Sx = Sy , then a scaling process will be (uniform scaling).
- If Sx ≠ Sy , then a scaling process will be (non uniform scaling) or (differential scaling). Object's shape will elongate or distort.
- If Sx , Sy < 1, size is reduced (shrink), object moves closer to origin.
- If Sx , Sy > 1, size is increased, object moves further from origin.
- If Sx = Sy = 1, size does not change.
- If the picture to be enlarged to twice its original size then Sx = Sy =2, so on three times, four times ….etc.

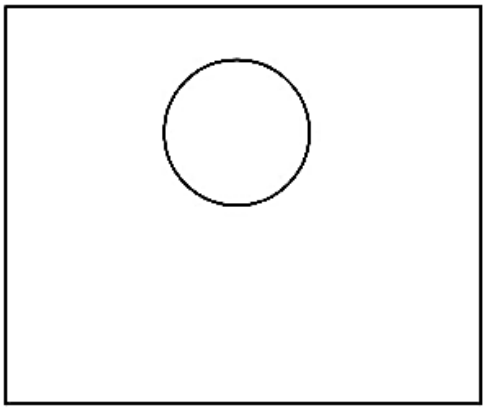
See the following figures:



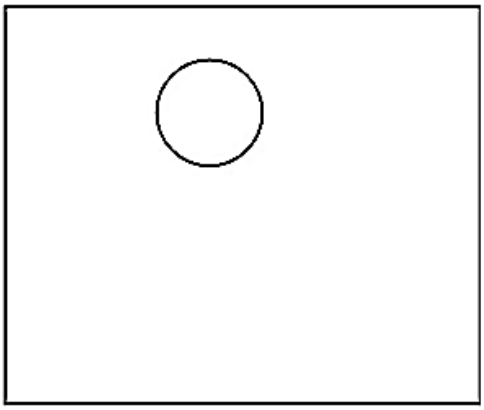
Original Image



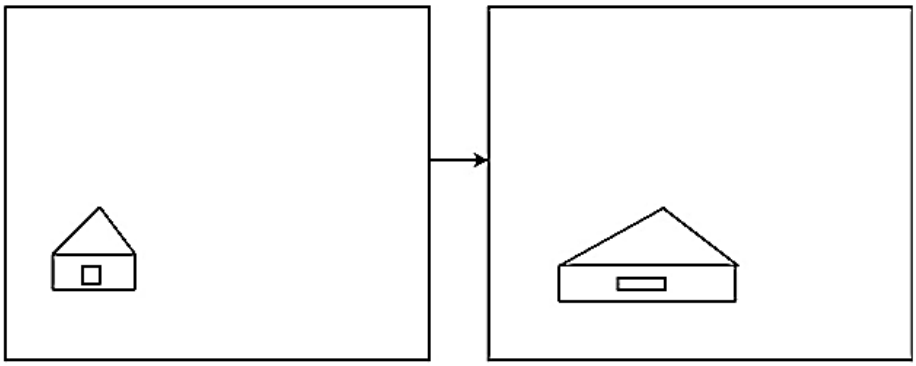
Enlarged Image (Twice times Sx=Sy=2)



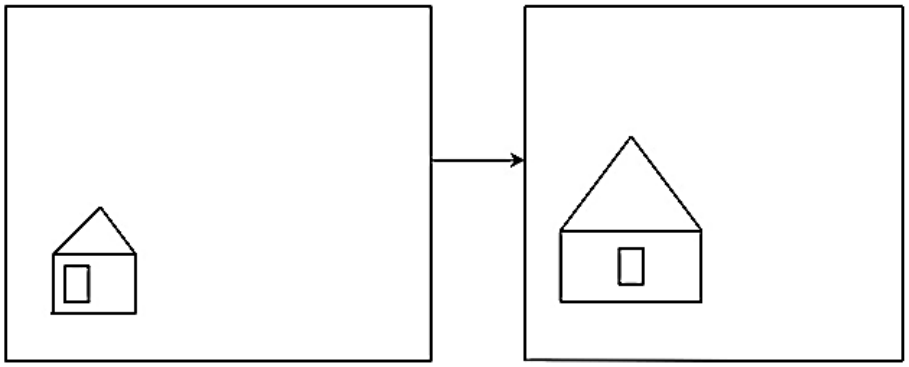
Original Image



Reduction Image (Sx,Sy<1)



Original Object Object after scaling in X direction (Sy=1)



Original Object Object after scaling in Y direction (Sx=1)

Scaling Relative to a Pivot or Fixed Point

To scale an object about a pivot or fixed point (Xp, Yp), we must perform the following three steps:

Step1: Translate

All coordinates of object are translated so that the fixed point is moved to the coordinate origin.

$X' = X - X_p$

$Y' = Y - Y_p$

Step2: Scaling

The translated coordinates are scaled with respect to the origin.

$X'' = X' * Sx$

$Y'' = Y' * Sy$

Step3: Translate

The coordinates are translated back so that the fixed point is returned to its original position.

$X''' = X'' + X_p$

$Y''' = Y'' + Y_p$

Note: The previous three steps can be performed in a one step:

$X'''= (X - X_p) * Sx + X_p$

$Y'''= (Y - Y_p) * Sy + Y_p$

Ex: Scale a triangle whose its vertices are (4,4), (7, 8), (10, 5) by Sx = 2 and Sy = 2, about the fixed point (4, 4).

Sol:

Old coordinates of the triangle = (4,4), (7, 8), (10, 5).

Scaling factor (Sx, Sy) = (2, 2)

pivot or fixed point (Xp, Yp) = (4, 4)

For (4, 4)

Xnew = (Xold – Xp) * Sx + Xp = (4 – 4) * 2 + 4 = 4

Ynew = (Yold – Yp) * Sy + Yp = (4 – 4) * 2 + 4 = 4

the new coordinates are (4, 4)

For (7, 8)

Xnew = (Xold – Xp) * Sx + Xp = (7 – 4) * 2 + 4 = 10

Ynew = (Yold – Yp) * Sy + Yp = (8 – 4) * 2 + 4 = 12

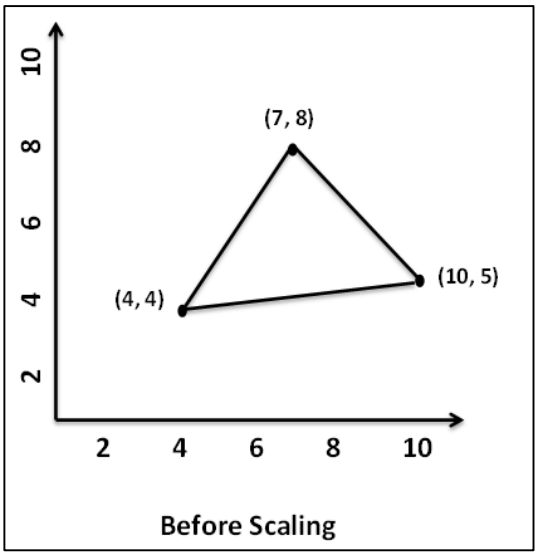
the new coordinates are (10, 12)

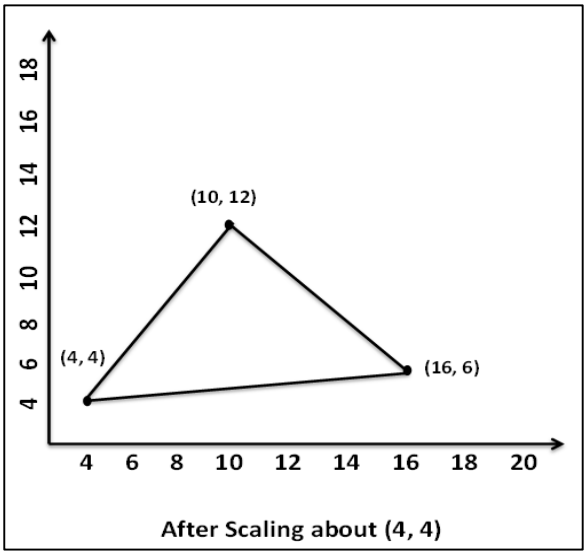
For (10, 5)

Xnew = (Xold – Xp) * Sx + Xp = (10 – 4) * 2 + 4 = 16

Ynew = (Yold – Yp) * Sy + Yp = (5 – 4) * 2 + 4 = 6

the new coordinates are (16, 6)





Note: By scaling, we can expand or compress an object without changing its shape or orientation, so scaling is called non-rigid transformation.

H. W

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

2- Magnify the above triangle to ½ its size.

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

4- Write a scale function to magnify any image.

5- Write an equation to magnify any picture without translation.

6- Consider an object defined by its vertices (10, 10), (20, 10), (20, 20), (10, 20) being scaled twice to the X and Y direction about the point (15, 15).

Homogeneous Coordinate Representation

The previous scaling is also shown in the form of 3 x 3 matrix:

$$\begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Scaling Matrix

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note: for scaling an object, each vertex must multiply by scaling matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

EX: Magnify the triangle (0, 0), (8, 10), (12, 4), to 4 times its size, about the origin point, using matrix representation.

$$\begin{pmatrix} 0 & 0 & 1 \\ 8 & 10 & 1 \\ 12 & 4 & 1 \end{pmatrix} * \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 32 & 40 & 1 \\ 48 & 16 & 1 \end{pmatrix}$$

EX: Magnify the triangle (0, -3), (-6, -7), (6, -7), to 3 times its size, about the point (0, -3), using matrix representation.

Sol:

Scaling factor (Sx, Sy) = (3, 3)

pivot point (Xp,Yp) = (0, -3)

1- Translate the triangle to the origin.

$$\begin{pmatrix} 0 & -3 & 1 \\ -6 & -7 & 1 \\ 6 & -7 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & +3 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ -6 & -4 & 1 \\ 6 & -4 & 1 \end{pmatrix}$$

Translation matrix

2- Scale the Translated triangle about origin.

$$\begin{pmatrix} 0 & 0 & 1 \\ -6 & -4 & 1 \\ 6 & -4 & 1 \end{pmatrix} * \begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ -18 & -12 & 1 \\ 18 & -12 & 1 \end{pmatrix}$$

Scaling matrix

3- Translate back the scaled triangle.

$$\begin{pmatrix} 0 & 0 & 1 \\ -18 & -12 & 1 \\ 18 & -12 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -3 & 1 \\ -18 & -15 & 1 \\ 18 & -15 & 1 \end{pmatrix}$$

H. W

1- Magnify the triangle (0, 0), (8, 10), (12, 4), to 4 times its size, about the origin point, using matrix representation.

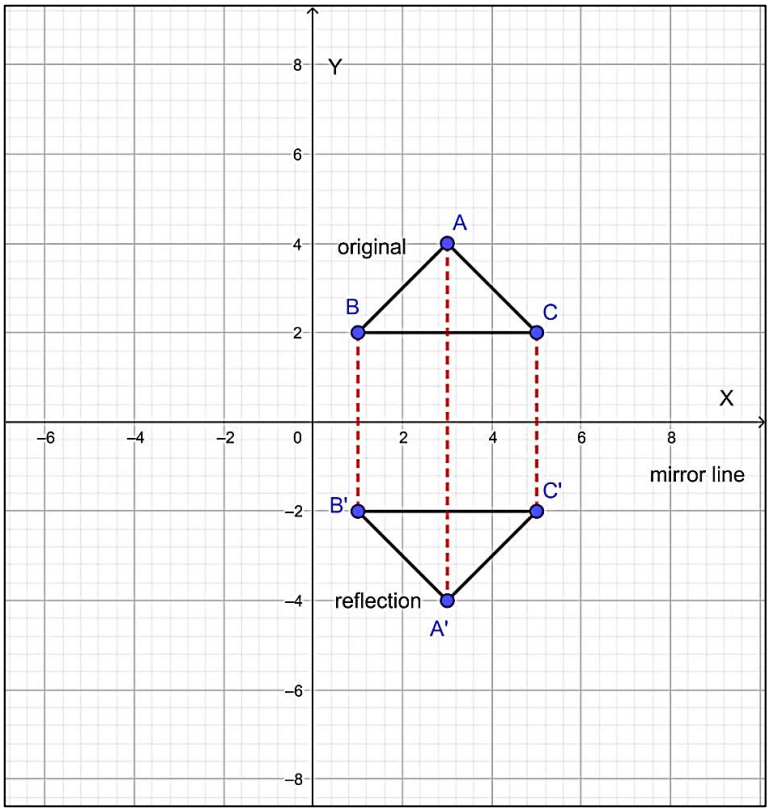
2- Magnify the above triangle to ½ its size, using matrix representation.

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

4- Consider an object defined by its vertices (10, 10), (20, 10), (20, 20), (10, 20) being scaled twice to the X and Y direction about the point (15, 15), using matrix representation.

Reflection Transformation

The Reflection is a mirror image of the original object. All the points of the original object are **reflected** or **flipped** on a line called the **axis of reflection**, **line of reflection**, **axis of symmetry**, or **mirror line**.



Properties of reflection

- Reflection is a kind of rotation where the angle of rotation is 180 degree.
- A reflection is defined by the mirror line. In the above diagram, X-axis is the mirror line. If a mirror line is vertical or horizontal, all the points on the mirror line will not change. These points are said to be **invariant**.
- The reflected object is always formed on the other side of mirror.
- Under reflection, the shape and size of an image is exactly the same as the original figure. This type of transformation is called **isometric transformation or rigid transformation**.
- The orientation is **laterally inverted**, that is they are facing opposite directions. The objects materialize as if they are mirror images, with right and left inverted.

- Corresponding parts of the figures (original object and its reflection) are the same distance from the line of reflection.

Types of Reflection

There are two types of reflection:

- **Reflection about a line**: in this type the line of reflection is the perpendicular bisector of the line joining any point and its image. Also, Lettering the vertices of the objects in a reflection requires changing the sequence of the letters (such as from clockwise to anticlockwise or vice versa). There are four types:

  1- Reflection about the X-axis or line Y=0

  2- Reflection about the Y-axis or line X=0

  3- Reflection about line Y = X

  4- Reflection about line y = -X

- **Reflection about a point:** A point reflection occurs when an object is created around a single point called **point of reflection**. For every point in the object, there is another point found directly opposite it on the other side of the center therefore; the point of reflection becomes the midpoint of the segment joining the point with its reflected image. Lettering sequence remains the same. The **most frequently used point is the origin (0, 0)**.

Reflection about X-Axis or line Y=0

This transformation keeps X values same but flips (changes the sign) y values of coordinate positions.

We can represent the Reflection along X-axis by following equations:

$X' = X$

$Y' = -Y$

Note: In the second equation, the negative sign does not mean that Y must be a negative value, but rather that the value of Y is **negated**. Meaning if the value of Y is negative it will become positive and vice versa. **So on for all of the reflection equations**.

Ex: Reflect a triangle whose its vertices are A (-2, 2), B (1, 5), C (3, 3) about X-axis.

Sol:

For A (-2,2)

X' = X = -2

Y' = -Y = -2

point A (-2, 2) after reflection will be A'(-2, -2)

For B (1, 5)

X' = X = 1

Y' = -Y = -5

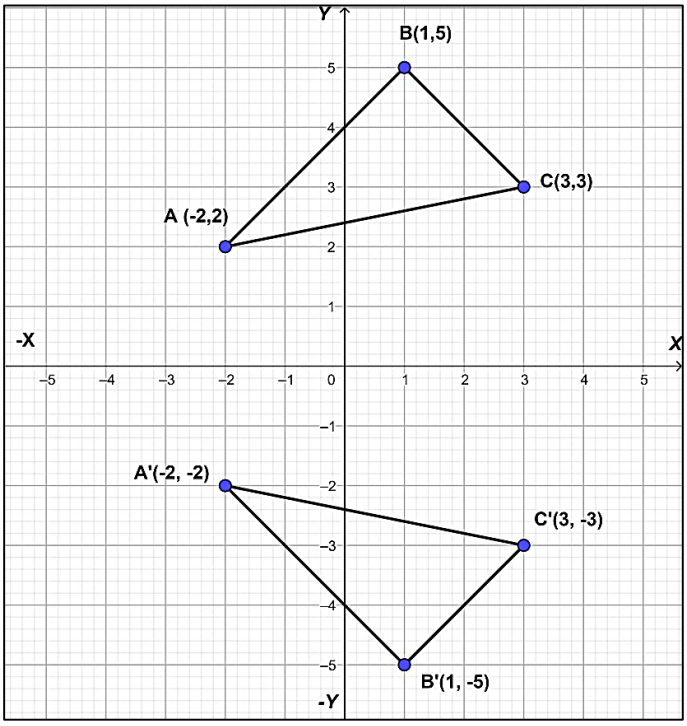
point B (1, 5) after reflection will be B'(1, -5)

For C (3, 3)

X' = X = 3

Y' = -Y = -3

point C (3, 3) after reflection will be C'(3, -3)



Note: In above figure, lettering direction of original triangle is clockwise while in its reflected image is anticlockwise.

We can also represent Reflection about **X-axis** in the form of matrix:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

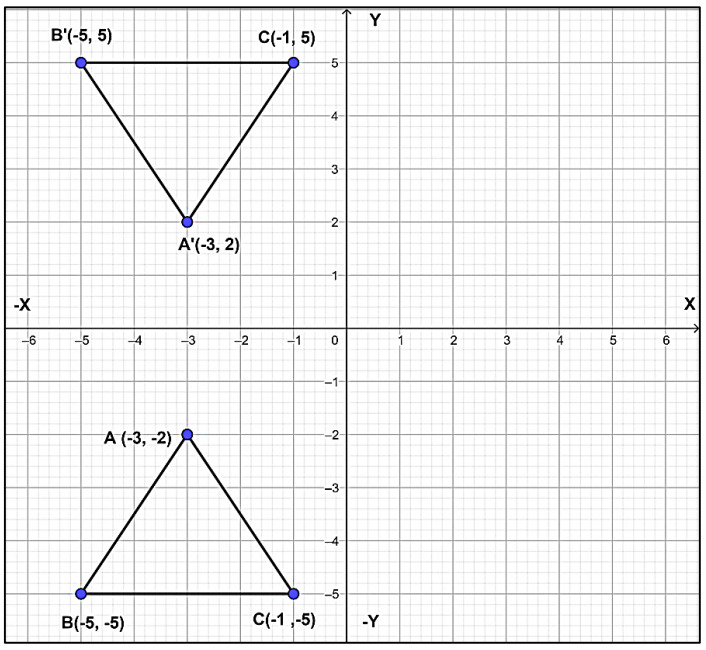
Note: for reflection an object, each vertex must multiply by reflection matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: Reflect a triangle whose its vertices are A(-3, -2), B(-5, -5), C(-1, -5) about X-axis using matrix.

Sol:

$$\begin{pmatrix} -3 & -2 & 1 \\ -5 & -5 & 1 \\ -1 & -5 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -3 & 2 & 1 \\ -5 & 5 & 1 \\ -1 & 5 & 1 \end{pmatrix}$$



Note: In above figure, lettering direction of original triangle is anticlockwise while in its reflected image is clockwise.

Reflection about Y-Axis or line X=0

This transformation keeps Y values same but flips (changes the sign) X values of co-ordinate positions.

We can represent the Reflection along Y-axis by following equations:

X' = -X

Y' = Y

Ex: Reflect a triangle whose its vertices are A (2, 3), B (6, 2), C (5, -1) about Y-axis.

Sol:

For A (2,3)

X' = -X = -2

Y' = Y = 3

point A (2, 3) after reflection will be A' (-2, 3)

For B (6,2)

X' = -X = -6

Y' = Y = 2

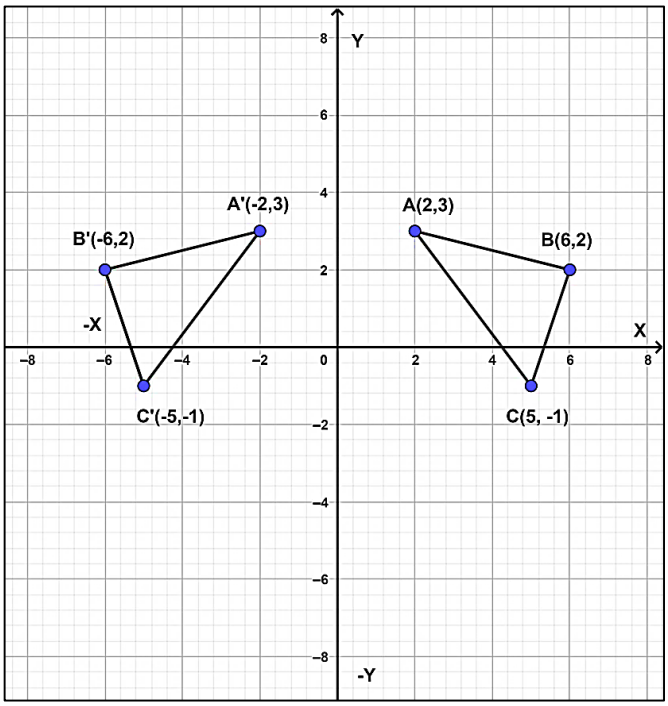
point B (6, 2) after reflection will be B'(-6, 2)

For C (5, -1)

X' = -X = -5

Y' = Y = -1

point C (5, -1) after reflection will be C'(-5, -1)



Note: In above figure, lettering direction of original triangle is clockwise while in its reflected image is anticlockwise.

We can also represent Reflection about **Y-axis** in the form of matrix:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

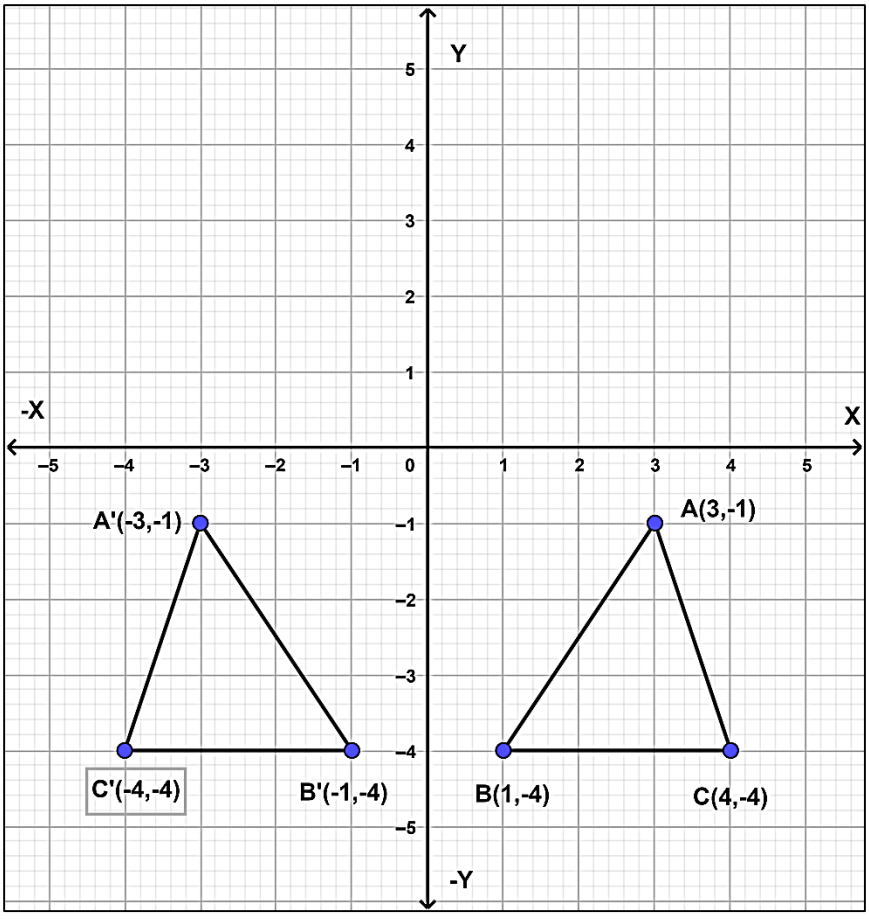
Note: for reflection an object, each vertex must multiply by reflection matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: Reflect a triangle whose its vertices are A(3, -1), B(1, -4), C(4, -4) about Y-axis using matrix.

Sol:

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



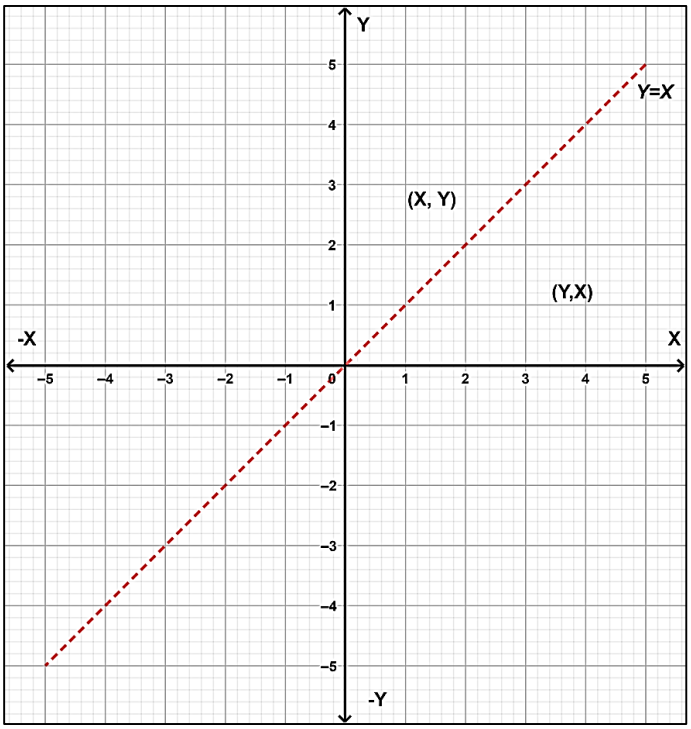
Note: In above figure, lettering direction of original triangle is anticlockwise while in its reflected image is clockwise.

Reflection about the line Y=X

This transformation interchanges X and Y values of coordinate positions (the *x*-coordinate and *y*-coordinate change positions), see the figure below. We can represent the Reflection along the line Y=X by the following equations:

X' = Y

Y' = X



Ex: Reflect a triangle whose its vertices are A(-3, -2), B(-1, 0), C(-4, 1) about the line Y= X.

Sol:

For A (-3, -2)

X' = Y = -2

Y' = X = -3

point A (-3, -2) after reflection will be A'(-2, -3)

For B (-1, 0)

X' = Y = 0

Y' = X = -1

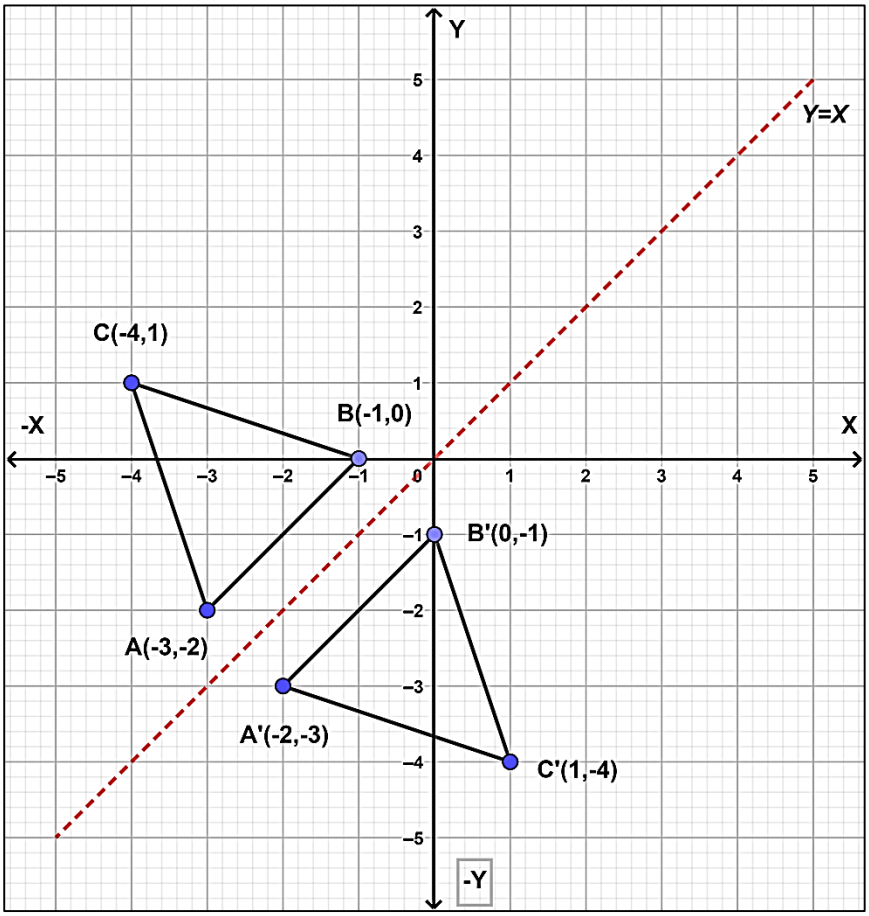
point B (-1, 0) after reflection will be B'(0, -1)

For C (-4, 1)

X' = Y = 1

Y' = X = -4

point C (-4, 1) after reflection will be C'(1, -4)



Note: In above figure, lettering direction of original triangle is anticlockwise while in its reflected image is clockwise.

We can also represent Reflection about the line Y=X in the form of matrix:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

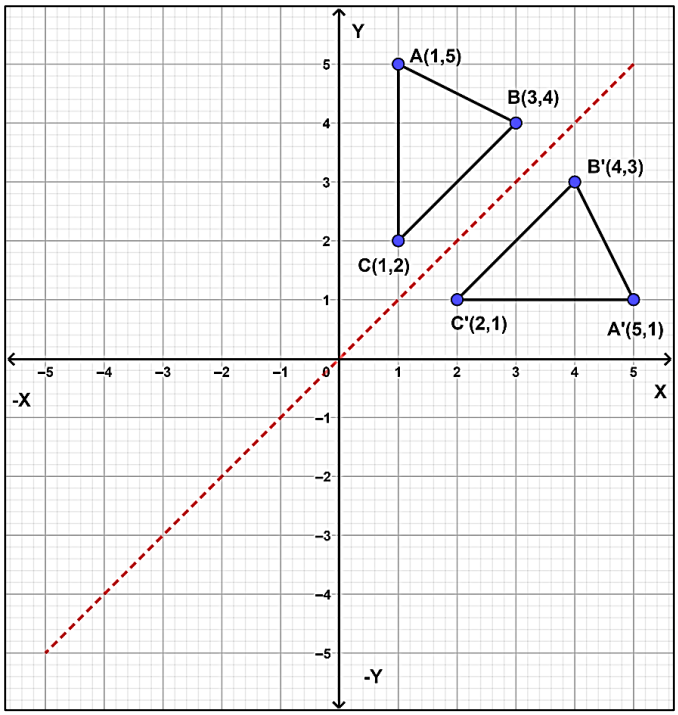
Note: for reflection an object, each vertex must multiply by reflection matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: Reflect a triangle whose its vertices are A (1, 5), B (3, 4), C (1, 2) about the line Y= X using matrix.

Sol:

$$\begin{pmatrix} 1 & 5 & 1 \\ 3 & 4 & 1 \\ 1 & 2 & 1 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 1 & 1 \\ 4 & 3 & 1 \\ 2 & 1 & 1 \end{pmatrix}$$



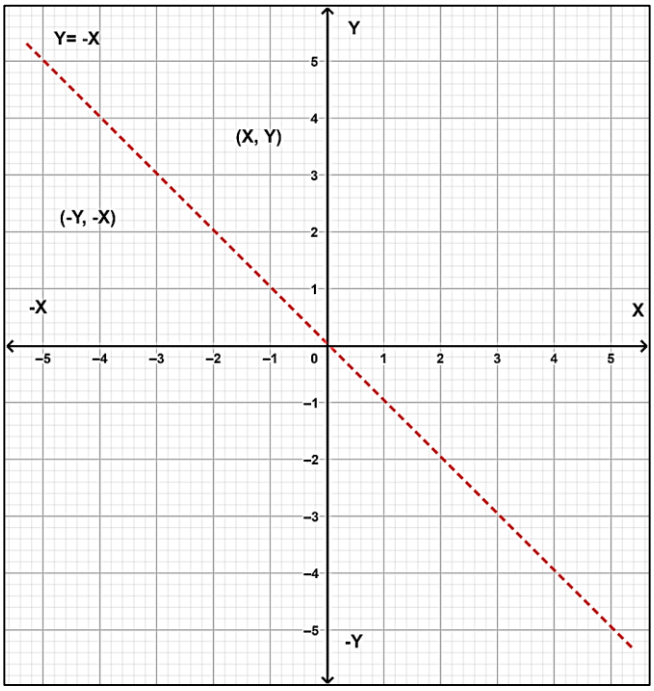
Note: In above figure, lettering direction of original triangle is clockwise while in its reflected image is anticlockwise.

Reflection about the line Y= -X

This transformation interchanges X and Y values of coordinate positions. (The *x*-coordinate and *y*-coordinate change positions and signs). See the figure below. We can represent the Reflection along the line Y= -X by the following equations:

X' = -Y

Y' = -X



Ex: Reflect a triangle whose its vertices are A (0, 5), B (2, 3), C (-2, 3) about the line Y= -X.

Sol:

For A (0, 5)

X' = -Y = -5

Y' = -X = 0

point A (0, 5) after reflection will be A'(-5, 0)

For B (2, 3)

X' = -Y = -3

Y' = -X = -2

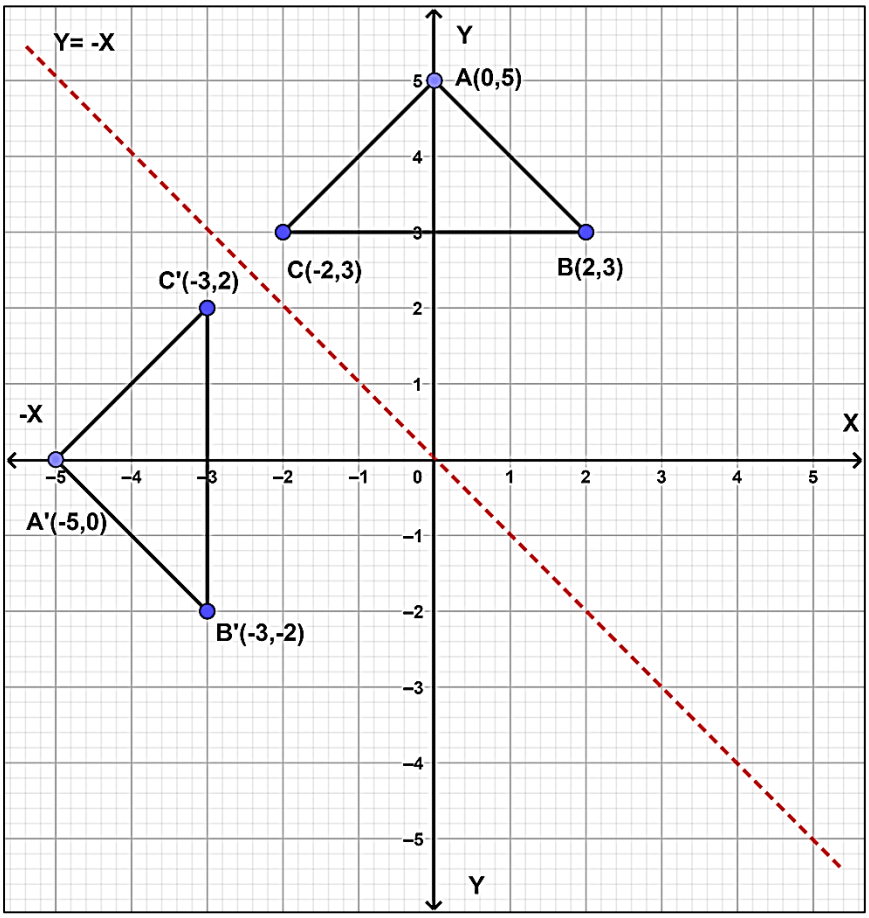
point B (2, 3) after reflection will be B'(-3, -2)

For C (-2, 3)

X' = -Y = -3

Y' = -X = - (-2) = 2

point C (-2, 3) after reflection will be C'(-3, 2)



Note: In above figure, lettering direction of original triangle is clockwise while in its reflected image is anticlockwise.

We can also represent Reflection about the line Y= -X in the form of matrix:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

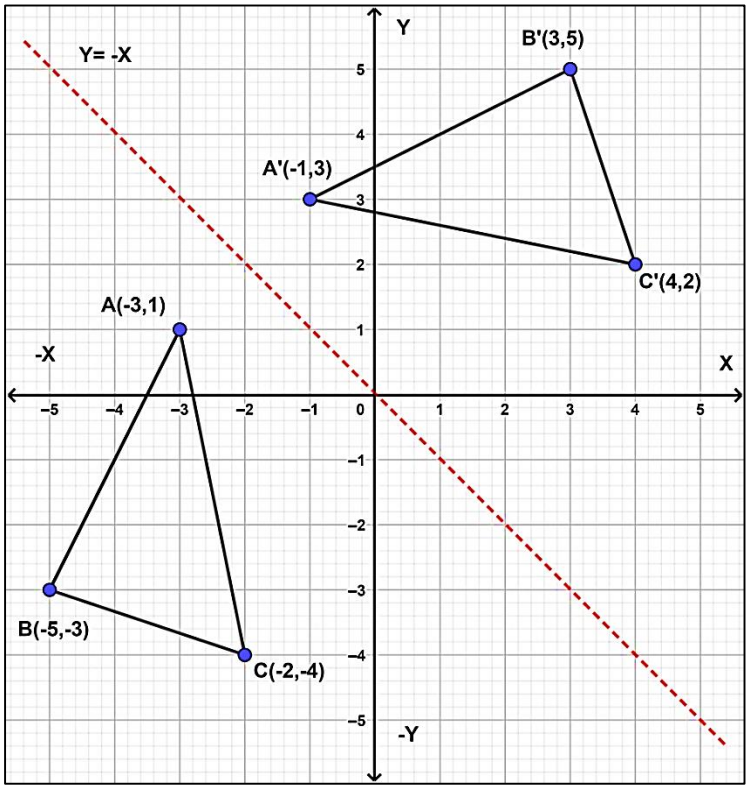
Note: for reflection an object, each vertex must multiply by reflection matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: Reflect a triangle whose its vertices are A (-3, 1), B (-5, -3), C (-2, -4) about the line Y= -X using matrix.

Sol:

$$\begin{pmatrix} -3 & 1 & 1 \\ -5 & -3 & 1 \\ -2 & -4 & 1 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 3 & 1 \\ 3 & 5 & 1 \\ 4 & 2 & 1 \end{pmatrix}$$



Reflection about the origin (0, 0)

This transformation flips (changes the signs) X and Y values of coordinate positions.

We can represent the Reflection about the origin by the following equations:

X' = -X

Y' = -Y

Ex: Reflect a triangle whose its vertices are A (1, 5), B (1, 2), C (5, 2) about the origin.

Sol:

For A (1, 5)

X' = -X **= -1**

Y' = -Y = -5

point A (1, 5) after reflection will be A'(-1, -5)

For B (1, 2)

X' = -X **= -1**

Y' = -Y = -2

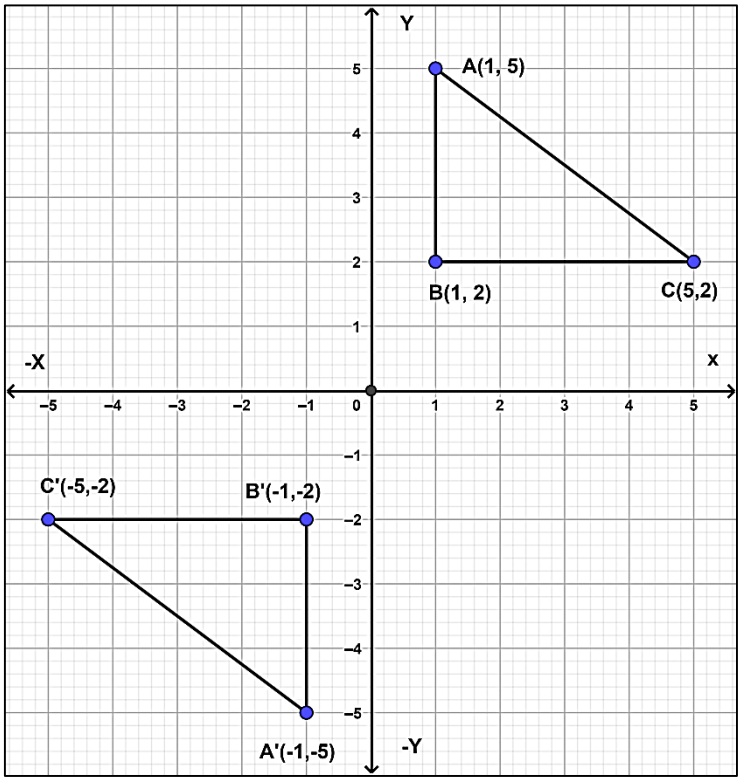
point B (1, 2) after reflection will be B'(-1, -2)

For C (5, 2)

X' = -X **= -5**

Y' = -Y = -2

point C (5, 2) after reflection will be C'(-5, -2)



Note: In above figure, lettering direction remains the same (anticlockwise).

Note: the coordinates of triangle A'B'C' are similar to the coordinates as triangle ABC, **but the signs have been changed**.

We can also represent Reflection about the origin in the form of matrix:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

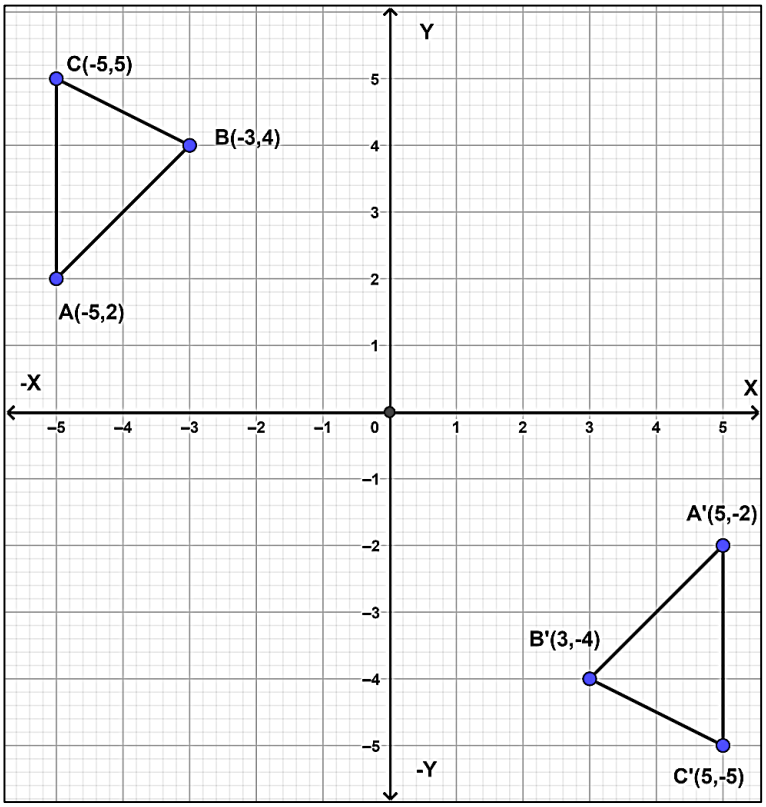
Note: for reflection an object, each vertex must multiply by reflection matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: Reflect a triangle whose its vertices are A (-5, 2), B (-3, 4), C (-5, 5) about the origin using matrix.

Sol:

$$\begin{pmatrix} -5 & 2 & 1 \\ -3 & 4 & 1 \\ -5 & 5 & 1 \end{pmatrix} * \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 5 & -2 & 1 \\ 3 & -4 & 1 \\ 5 & -5 & 1 \end{pmatrix}$$



Note: In above figure, lettering direction remains the same (anticlockwise).

Note: the coordinates of triangle A'B'C' are similar to the coordinates as triangle ABC, **but the signs have been changed**.

H. W: 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 using reflection equations and draw the result.

Finally don't forget: Reflections are FLIPS!!

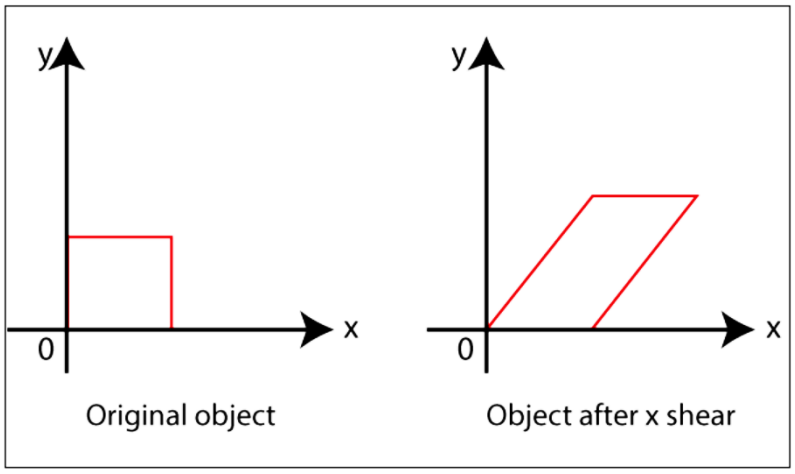
Shear Transformation

Shearing also known **skewing** is a transformation that slants the shape of an object. It distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other (It appears like somebody has dragged the object).

Shearing deals with changing the shape and size of the 2D object along x-axis and y-axis. We can denote shearing with **'Shx'** and **'Shy'**. These **'Shx'** and **'Shy'** are called **"Shearing factor".** The shear can be in one direction or in two directions. So we can perform shearing on the object in three ways:

1- **Shearing along x-axis**: In this, we can preserve the y coordinate and only change the x coordinate. It is also called "Horizontal Shearing" or "X-Shearing".
We can represent X-Shearing by the following equations:

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



Horizontal or X-Shearing

According to the above equations and above figure, in X-shearing 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 (**Any point above the X-axis is shiffted to the right (increasing x) if Shx > 0 , and to the left (decreasing x) if Shx < 0). Points below the x-axis move in the opposite direction, while points on the axis stay fixed.

We can represent X-shearing in the form of matrix:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} 1 & 0 & 0 \\ Shx & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note: for shearing an object, each vertex must multiply by X-shearing matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ Shx & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: A Triangle with P (2, 2), Q (0, 0) and R (2, 0). Apply Shearing factor 2 on X-axis. Find out the new coordinates of the triangle?

Sol:

The coordinates of triangle = P (2, 2), Q (0, 0), R (2, 0)

Shearing Factor for X-axis, **Shx** = 2

For P (2, 2)

X' = X + Shx* Y

X' = 2 + 2 x 2 = 6

Y' = Y = 2

The New Coordinates = (6, 2)

For Q (0, 0)

X' = X + Shx* Y

X' = 0 + 2 x 0 = 0

Y' = Y = 0

The New Coordinates = (0, 0)

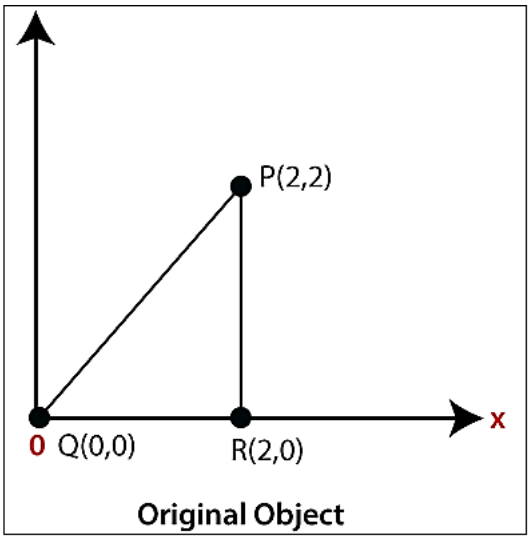
For R (2, 0)

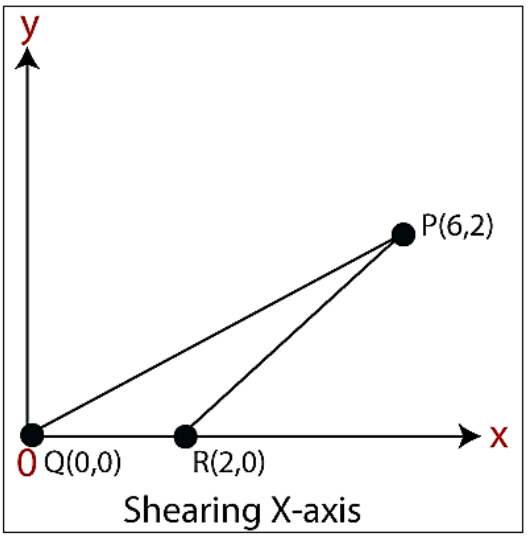
X' = X + Shx* Y

X' = 2 + 2 x 0 = 2

Y' = Y = 0

The New Coordinates = (2, 0)





Ex: A Triangle with P (2, 2), Q (0, 0) and R (2, 0). Apply Shearing factor 2 on X-axis. Find out the new coordinates of the triangle using matrix representation?

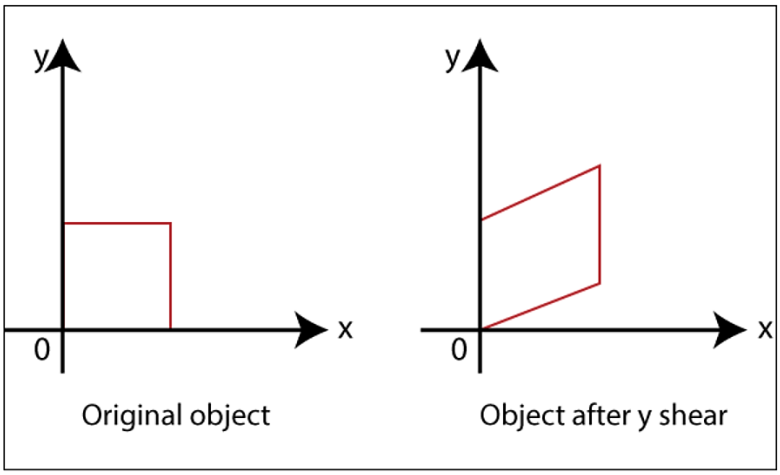
$$\begin{pmatrix} 2 & 2 & 1 \\ 0 & 0 & 1 \\ 2 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 6 & 2 & 1 \\ 0 & 0 & 1 \\ 2 & 0 & 1 \end{pmatrix}$$

2- **Shearing along y-axis**: In this, we can preserve the x coordinate and only change the y coordinate. It is also called "Vertical Shearing."

We can represent Vertical Shearing by the following equation:

$$X' = X$$

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



Vertical or Y-Shearing

We can represent Y-shearing in the form of matrix:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} 1 & Shy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note: for shearing an object, each vertex must multiply by Y-shearing matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} 1 & Shy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: A Triangle with P (2, 2), Q (0, 0) and R (2, 0). Apply Shearing factor 2 on Y-axis. Find out the new coordinates of the triangle?

Sol:

The coordinates of triangle = P (2, 2), Q (0, 0), R (2, 0)

Shearing Factor for Y-axis, **Shy** = 2

For P (2, 2)

X' = X = 2

Y' = Y + Shy* X

Y' = 2 + 2*2 = 6

The New Coordinates = (2, 6)

For Q (0, 0)

X' = X = 0

Y' = Y + Shy* X

Y' = 0 + 2*0 = 0

The New Coordinates = (0, 0)

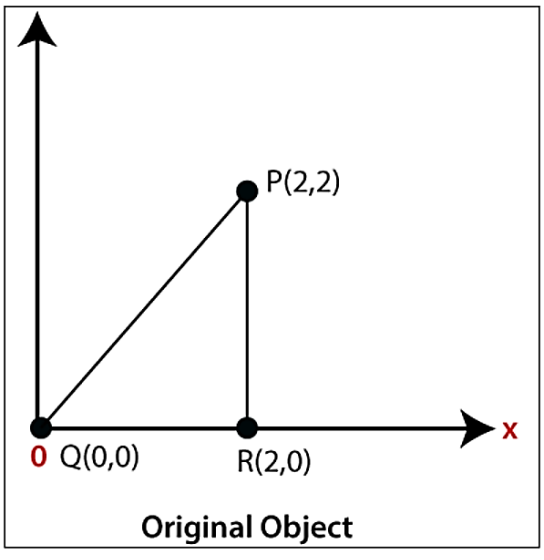
For R (2, 0)

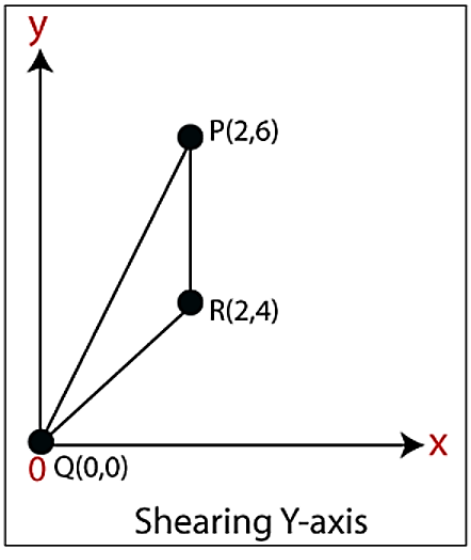
X' = X = 2

Y' = Y + Shy* X

Y' = 0 + 2*2 = 4

The New Coordinates = (2, 4)



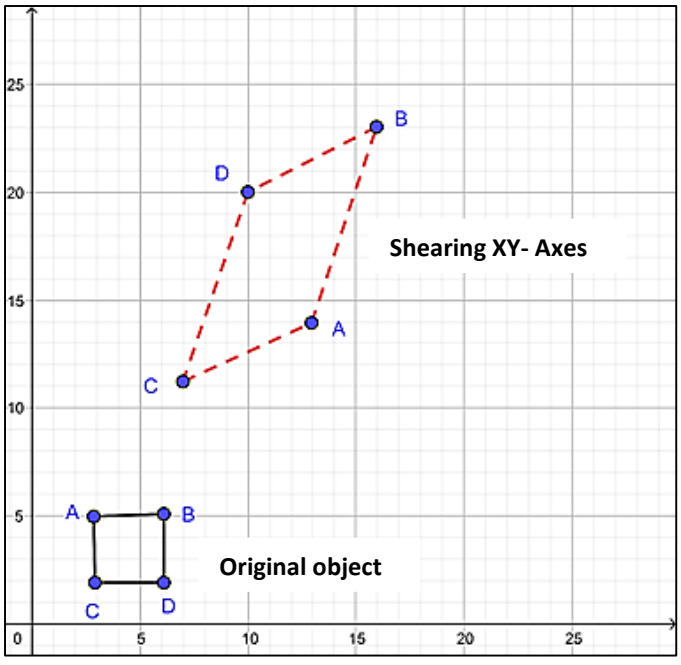


Ex: A Triangle with P (2, 2), Q (0, 0) and R (2, 0). Apply Shearing factor 2 on Y-axis. Find out the new coordinates of the triangle using matrix representation?

$$\begin{pmatrix} 2 & 2 & 1 \\ 0 & 0 & 1 \\ 2 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 6 & 1 \\ 0 & 0 & 1 \\ 2 & 4 & 1 \end{pmatrix}$$

3- **Shearing along XY-Axes:** Here layers will be slided in both x as well as y direction. The sliding will be in horizontal as well as vertical direction. The shape of the object will be distorted. The matrix of shear in both directions is given by:

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



We can represent XY-shearing in the form of matrix:

$$[X' \quad Y' \quad 1] = [X \quad Y \quad 1] * \begin{pmatrix} 1 & Shy & 0 \\ Shx & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note: for shearing an object, each vertex must multiply by XY-shearing matrix.

$$\begin{pmatrix} X1 & Y1 & 1 \\ X2 & Y2 & 1 \\ X3 & Y3 & 1 \end{pmatrix} * \begin{pmatrix} 1 & Shy & 0 \\ Shx & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} X_1' & Y_1' & 1 \\ X_2' & Y_2' & 1 \\ X_3' & Y_3' & 1 \end{pmatrix}$$

Ex: A rectangle with A (3, 5), B (6, 5), C (3, 2) and D (6, 2). Find out the new coordinates of the rectangle along XY-axes? **(**Shearing factor 2 on X-axis and 3 on Y-axis).

Sol:

The coordinates of the rectangle = A (3, 5), B (6, 5), C (3, 2), D (6, 2)

Shearing Factor for X-axis, **Shx** = 2

Shearing Factor for Y-axis, **Shy** = 3

For A (3, 5)

X' = X + Shx* Y

X' = 3 + 2 * 5 = 13

Y' = Y + Shy* X

Y' = 5 + 3 * 3 = 14

<u>The New Coordinates = (13, 14)</u>

For B (6, 5)

X' = X + Shx* Y

X' = 6 + 2 * 5 = 16

Y' = Y + Shy* X

Y' = 5 + 3 * 6 = 23

The New Coordinates = (16, 23)

For C (3, 2)

X' = X + Shx* Y

X' = 3 + 2 * 2 = 7

Y' = Y + Shy* X

Y' = 2 + 3 * 3 = 11

The New Coordinates = (7, 11)

For D (6, 2)

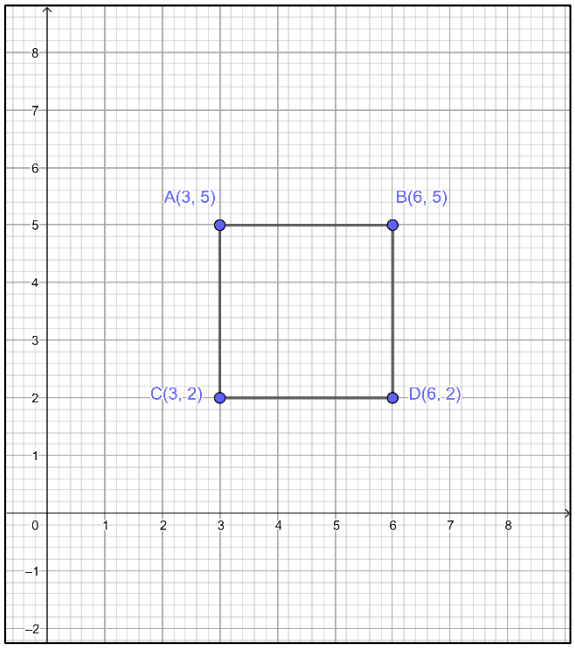
X' = X + Shx* Y

X' = 6 + 2 * 2 = 10

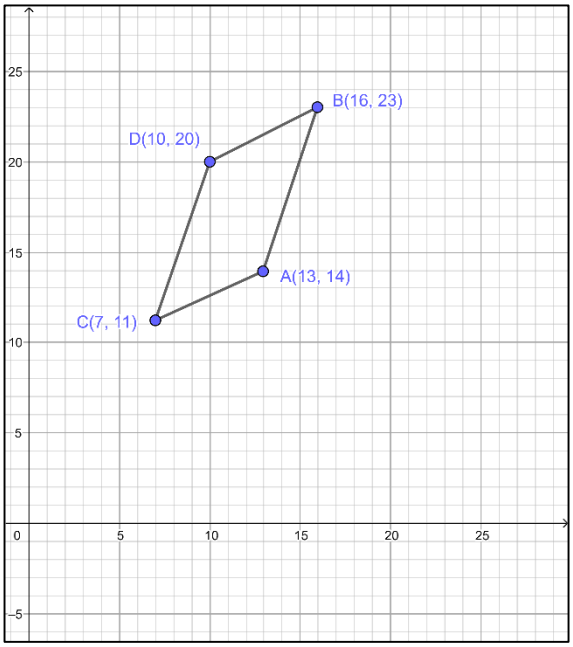
Y' = Y + Shy* X

Y' = 2 + 3 * 6 = 20

The New Coordinates = (10, 20)



Original object



Shearing XY- Axes

Ex: A rectangle with A (3, 5), B (6, 5), C (3, 2) and D (6, 2). Find out the new coordinates of the rectangle along XY-axes using matrix representation? **(**Shearing factor 2 on X-axis and 3 on Y-axis).

Sol:

The coordinates of the rectangle = A (3, 5), B (6, 5), C (3, 2), D (6, 2)

Shearing Factor for X-axis, **Shx** = 2

Shearing Factor for Y-axis, **Shy** = 3

$$\begin{pmatrix} 3 & 5 & 1 \\ 6 & 5 & 1 \\ 3 & 2 & 1 \\ 6 & 2 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 3 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 13 & 14 & 1 \\ 16 & 23 & 1 \\ 7 & 11 & 1 \\ 10 & 20 & 1 \end{pmatrix}$$

Very Useful Summery

A shearing transform. Although shears can in fact be **built up out of rotation and scaling if necessary**. A shear will "tilt" objects. A horizontal shear will tilt things towards the left (for negative shear) or right (for positive shear). A vertical shear tilts them up or down.

H. W: A rectangle with A (3, 5), B (6, 5), C (3, 2) and D (6, 2). Find out the new coordinates of the rectangle along X-axis? **(**Shearing factor -2 on X-axis).

H. W: A rectangle with A (-8, 7), B (-4, 7), C (-8, 3) and D (-4, 3). Find out the new coordinates of the rectangle along Y-axis? **(**Shearing factor 2 on Y-axis).

H. W: Find out the new coordinates of the object along x-axis, y-axis, xy- axes. (Applying shear factor 4 on X-axis and 1 on Y-axis). The coordinates of the triangle are A (1, 1), B (0, 0), and C (1, 0).

Difference between scaling and shearing

With scaling you can resize a geometry or a group of geometries **uniformly or non-uniformly**. This gives you the ability to resize different components of your model relative to other components. Non-uniform scaling gives you ability to create more interesting shapes than were possible before – e.g. Ellipses and Ellipsoids.

Shearing is another useful transform. It comes handy when you want to **skew** an object sideways while still keeping the surfaces in perpendicular direction flat.