# Abstract

**Computer Graphics** is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. In other words, we can say that computer graphics is a rendering tool for the generation and manipulation of images. In this book we us MATLAB language because MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. There are two basic ways to create graphs in MATLAB. Either you can use the MATLAB GUI plotting tools to interactively create or you can use the command interface by entering MATLAB graphics commands in the Command Window.

**<u>This Book involve Four Chapters</u>**

Chapter One **Basics of Computer Graphics**: In This Chapter we defined some basic terminology and discuss the Advantages and Disadvantages of Computer Graphics, Basics of raster display and display screen.

Chapter Two **Graphics Primitives:** This Chapter reviews various methods (algorithms) to plotting point, drawing (Line, Circle, Ellipse, Arc and Sector).

Chapter Three **2D Transformation & Viewing**: In this chapter, we describe how we can matrix multiplication to accomplish change in vector such as Translation, Scaling, Rotation, Reflection and Shear for 2D; we also discuss how these transformations operate differently on locations (points), also describe the methods for 2DViewing, Clipping and windowing.

Chapter Four **3D Concepts & Object Representation:** This Chapter describe 3D Modeling Transformations **(**Translation, Scaling, Rotation**,** Reflection and Shear ) for 3D

# Contents

# CHAPTER ONE

# BASICS OF COMPUTER GRAPHICS

# *Chapter One*

# *Basics of Computer Graphics*

- **Introduction of Computer Graphics.**
- **Definition of Computer Graphics.**
- **Advantages and Disadvantages of Computer Graphics.**
- **Why Computer Graphics Used?**
- **Applications of Computer Graphics.**
- **Elements of Pictures Created in Computer Graphics.**
- **Raster Graphics.**
- **Display Screens.**
- **Screen Clarity.**

# Chapter One

# Basics of Computer Graphics

## 1.1 Introduction of Computer Graphics

Computer graphics necessitates the use of technology. The Process transforms and presents information in a visual form. The role of computer graphics insensible. In today life, computer graphics has now become a common element in user interfaces, T.V commercial motion pictures.

Computer Graphics is the creation of pictures with the help of a computer. The product of the computer graphics is a picture it may be a business graph, drawing, and engineering.

In computer graphics, two or three-dimensional pictures can be created that are used for research. Many hardware devices algorithm has been developing for improving the speed of picture generation with the passes of time.

It includes the creation storage of models and image of objects. These models for various fields like engineering, mathematical and so on.

Today computer graphics is entirely different from the earlier one. It is an interactive user can control the structure of an object of various input devices such as (keyboard, mouse, joystick,..etc).

## 1.2 Definition of Computer Graphics

**Computer Graphics** is an art of drawing pictures, lines, charts, etc..... using computers with the help of programming computer graphics image is made up of number of pixels. **Pixel** is an abbreviation of (picture element) is the smallest addressable graphical unit represented on the computer screen.

Computer graphics also are essential to scientific visualization, a discipline that uses images and colors to model complex phenomena such as air currents and electric fields, and to computer-aided engineering and design, in which objects are drawn and analyzed in computer programs.

**"Computer graphics"** also refers to the tools used to make such pictures. There are both **hardware** and **software** tools.

***Hardware*** tools include video monitors and printers that display graphics, as well as input devices like a mouse or trackball that let a user point to items and draw figures. The computer itself, of course, is a hardware tool, along with its special circuitry to facilitate graphical display or image capture.

As for ***software*** tools, familiar with the usual ones: the operating system, editor, compiler, and debugger that are found in any programming environment. For graphics, there must also be a collection of "graphics routines" that produce the pictures themselves.

## 1.3 Advantages and Disadvantages of Computer Graphics

The main advantages of computer graphics are as follows: -

1. **Usability:** Graphical techniques offer more flexible and options compared to other traditional methods in design. One can make changes and undo them without tampering with the whole design. It is also possible to view a model from different angles by rotating it along various axes. One can also perfect on minute details of a design by magnifying it to see them clearly.

2. **Research and Product Development:** Graphical representation software contributes much in research. Models can be presented in three dimensions giving researchers a broader picture of how natural phenomena operate. In engineering, presentation of models in three-dimensional manner enables engineers to identify weaknesses in structures and areas of possible improvement.

**3. Time:** Computer aided design on the other hand involves designing of a graphical presentation of a virtual model. Tests are then done on the model using special software.

This saves not only time, but also other resources that would have been used in testing the real structure, hence the cost of production.

**4. Design:** Advertising is an important aspect in the business world. Customers respond to product or service depending on how it is presented to them.

The main **disadvantages** of computer graphics are as follows:-

**1. Complexity:** A majority of complex graphical system applications require prior training before use.

Some of the graphics applications are so complex that they need an expert to install and customize the settings.

**2. Limitations:** Like all other computerized systems, graphical system lack the intelligence of understanding real world conditions and principles like the purpose of the structure it is designing.

The designer has to figure out a way of obtaining the relevant results while maintaining the objective of the design process.

## 1.4 Why Computer Graphics Used?

Computer graphics is responsible for displaying art and image data effectively and meaningfully to the consumer. It is also used for processing image data received from the physical world, such as photo and video content.

Suppose a shoe manufacturing company want to show the sale of shoes for five years. For this vast amount of information is to store. So a lot of time and memory will be needed. This method will be tough to understand by a common person. In this situation graphics is a better alternative.

Graphics tools are charts and graphs. Using graphs, data can be represented in pictorial form. A picture can be understood easily just with a single look.

Interactive computer graphics work using the concept of two-way communication between computer users.

The computer will receive signals from the input device, and the picture is modified accordingly. Picture will be changed quickly when we apply command figure 1.1 show computer graphics Used.

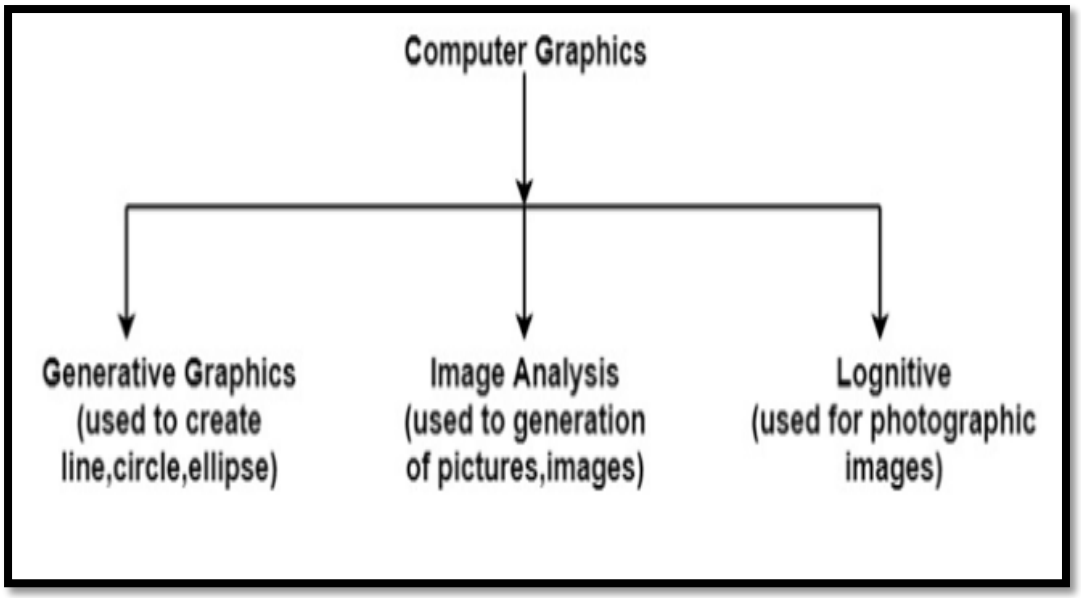



**Figure 1.1 Computer Graphics Used.**

## 1.5 Applications of Computer Graphics

Computer graphics deals with creation, manipulation and storage of different type of images and objects, some of the applications of computer graphics are:

**1. Computer Art:** Using computer graphics we can create one and commercial art which include animation packages, paint packages. These packages provide facilities for designing object shapes and specifying object motion. Cartoon drawing, paintings, logo design can also be done.

**2. Computer Aided Drawing:** Designing of buildings, automobile, aircraft is done with the help of computer-aided drawing, this helps in providing minute details to the drawing and producing more accurate and sharp drawings with better specifications.

**3. Presentation Graphics:** For the preparation of reports or summarizing the financial, statistical, mathematical, scientific, economic data for research reports, managerial reports, moreover, creation of bar graphs, pie charts.

**4. Education:** Computer generated models are extremely useful for teaching huge number of concepts and fundamentals in an easy to understand and learn manner. Using computer graphics many educational models can be created through which more interest can be generated among the students regarding the subject.

**5. Training:** Specialized system for training like simulators can be used for training the candidates in a way that can be grasped in a short span of time with better understanding. Creation of training modules using computer graphics is simple and very useful.

**6. Animation software:** Enables you to chain and sequence a series of images to simulate movement. Each image is like a frame in a movie. It can be defined as a simulation of movement created by displaying a series of pictures, or frames. A cartoon on television is one example of animation.

**7. Image Processing**: Various kinds of photographs or images require editing in order to be used in different places. Processing of existing images into refined ones for better interpretation is one of the many applications.

**8. Graphical User Interface:** The use of pictures, images, icons, pop-up menus, graphical objects help in creating a user-friendly environment where working is easy and pleasant, using computer graphics we can create such an atmosphere where everything can be automated and anyone can get the desired action performed in an easy fashion.

**9. Virtual –Reality Environments:** a more recent application of computer graphics is in the creation of Virtual –Reality Environments in which a user can interact with the objects in a three-dimensional scene. Figure 1.2 show the Virtual –Reality

**Figure 1.2 Virtual -Reality**

**10. Entertainment:** Television production, motion picture ,and music videos routinely use computer graphics method. These are some of the applications of computer graphics due to which it's popularity has increased to a huge extend and will keep on increasing with the progress in technology.

## 1.6 Elements of Pictures Created in Computer Graphics

What makes up a computer drawn picture? The basic objects out of which such pictures are composed are called output primitives. One useful categorization of these is:

1. Polylines
2. Text
3. Filled Regions
4. Raster images

### 1. Polylines

A polyline is a connected sequence of straight lines. When there are several lines in a polyline, each one is called an edge, and two adjacent lines meet at a vertex.

In Figure 1.3 we see an Example of Edge and Vertex.

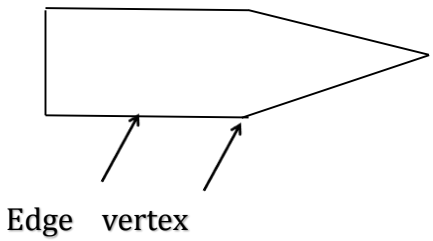



**Figure 1.3 Example of Edge and Vertex.**

A polyline need not form a closed figure, but if the first and last points are connected by an edge, the polyline is a polygon. The Figure 1.4 show Examples of polygons.

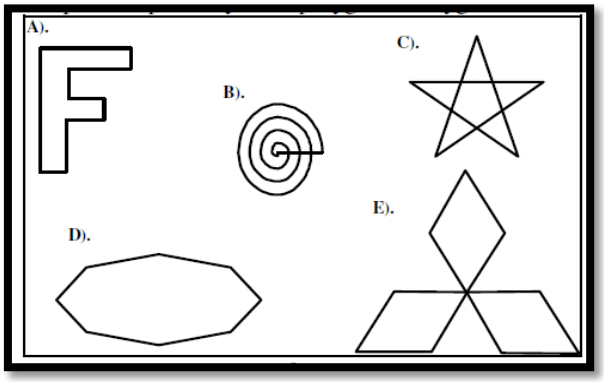



**Figure 1.4 Examples of polygons.**

For instance, the polyline shown in Figure 1.5 is given by the sequence (2, 4), (2, 11), (6, 14), (12, 11),(12, 4), .... (what are the remaining vertices in this polyline?).

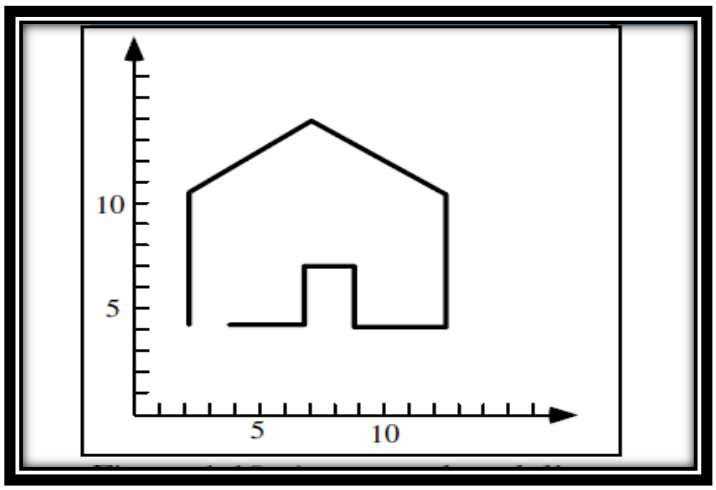



**Figure 1.5 An Example Polyline.**

## Attributes of Lines and Polylines

The attributes of a graphic primitive are the characteristics that affect how it appears, and some important attributes of a polyline are the color and thickness of its edges, the manner in which the edges are dashed, and the manner in which thick edges blend together at their endpoints.

Typically, all of the edges of a polyline are given the same attributes. Figure 1.6 show the line type.

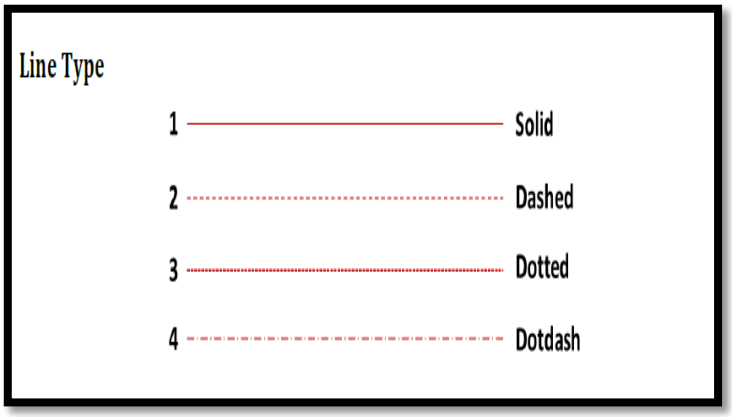



**Figure 1.6 The Line Type.**

## 2. Text

Some graphics devices have two distinct display modes, a text mode and a graphics mode.

The text mode is used for simple input/output of characters to control the operating system or edit the code in a program.

Text displayed in this mode uses a built-in character generator. The character generator is capable of drawing alphabetic, numeric, and punctuation characters, and some selection of special symbols such as ð and ⨁.

Compare with the graphic mode, the PC's text mode is easy to use. Displaying information on the screen is a simple as placing ASCII char in specific memory location.

The text screen divided into 80 column and 25 rows. The graphic mode requires a completely different orientation instead of character; you have pixels, the smallest picture element on your computer display.

Today most screens can display text and graphics.

**Attributes of Text**

There are many text attributes, the most important of which are typeface, color, size, spacing, and Orientation.

A font is a specific set of character shapes (a typeface) in a particular style and size.

## 3. Filled Regions

The filled region (sometimes called “fill area”) primitive is a shape filled with some color or pattern. The boundary of a filled region is often a polygon.

In Figure 1.7 see Examples of filled Polygons and Figure 1.8 show Different Style of Area Filling

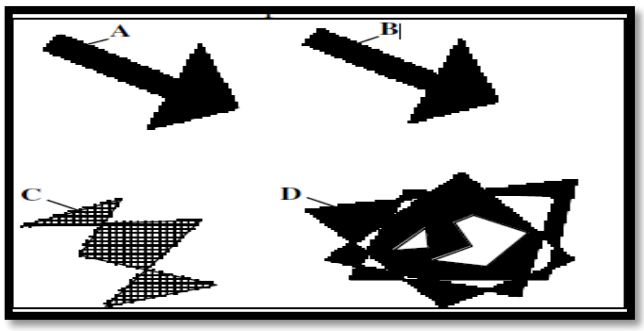



**Figure 1.7 Examples of Filled Polygons.**

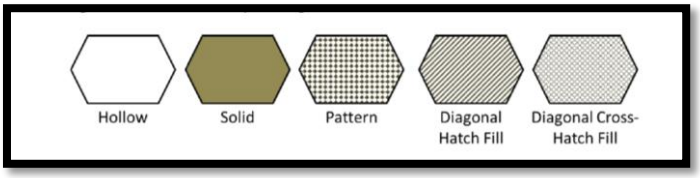



**Figure 1.8 Different Style of Area Filling**

## 4. Raster Image

A raster image, It is made up of many small “cells”, in different shades of gray, The individual cells are often called “pixels” (short for “picture elements”). Normally your eye can’t see the individual cells, it blends them together and synthesizes an overall picture.

A raster image is stored in a computer as an array of numerical values. This array is thought of as being rectangular, with a certain number of rows and a certain number of columns.

Each numerical value represents the value of the pixel stored there. The array as a whole is often called a “pixel map”. The term “bitmap” is also used. A simple figure represented as a bitmap.

Figure 1.9a shows a simple example where a figure is represented by a 17 by 19 array (17 rows by 19columns) of cells in three shades of gray.

Suppose the three gray levels are encoded as the values 1, 2, and 7. Figure 1.9b shows the numerical values of the pixel map for the upper-left 6 by 8 portion of the image.

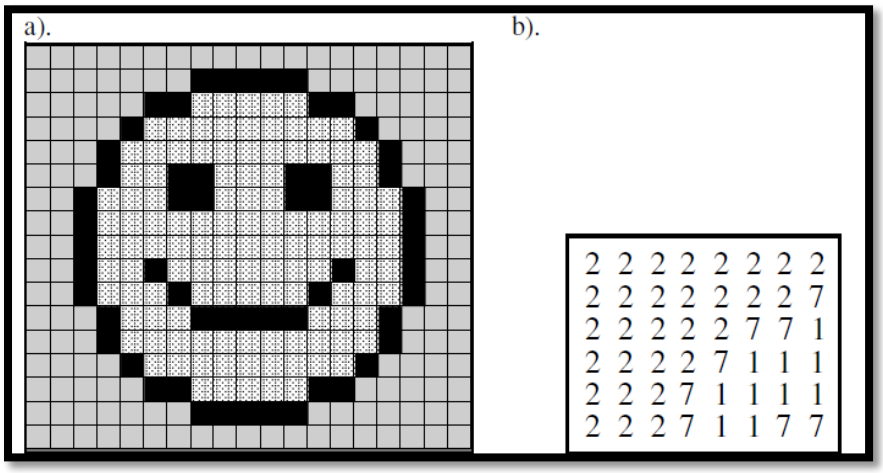



**Figure 1.9 A simple figure represented as a bitmap.**

## How are Raster Images Created?

The three principal sources are:

### 1. Hand designed images

A designer figures out what values are needed for each cell, and types them into memory. Sometimes a paint program can be used to help automate this: the designer can draw and manipulate various graphical shapes, viewing what has been made so far. When satisfied, the designer stores the result in a file.

### 2. Computed Images

An algorithm used to "render" a scene, which might be modeled abstractly in computer memory. As a simple example, a scene might consist of a single yellow smooth sphere illuminate by a light source that emanates orange light.

## 3. Scanned Images.

A photograph or television image can digitized as described above. In effect, a grid is placed over the original image, and at each grid point, the digitizer reads into memory the "closest" color in its repertoire. The bitmap is then stored in a file for later use.

## 1.7 Raster Graphics

An image that is presented on the computer screen is made up of pixels. The screen consists of a rectangular grid of pixels, arranged in rows and columns. The pixels are small enough that they are not easy to see individually.

In fact, for many very high-resolution displays, they become essentially invisible. At a given time, each pixel can show only one color.

Most screens these days use 24-bit color, where a color can be specified by three 8-bit numbers, giving the levels of red, green, and blue in the color.

Any color that can be shown on the screen is made up of some combination of these three “primary” colors. As we can see from Figure 1. 10 Graphical System.

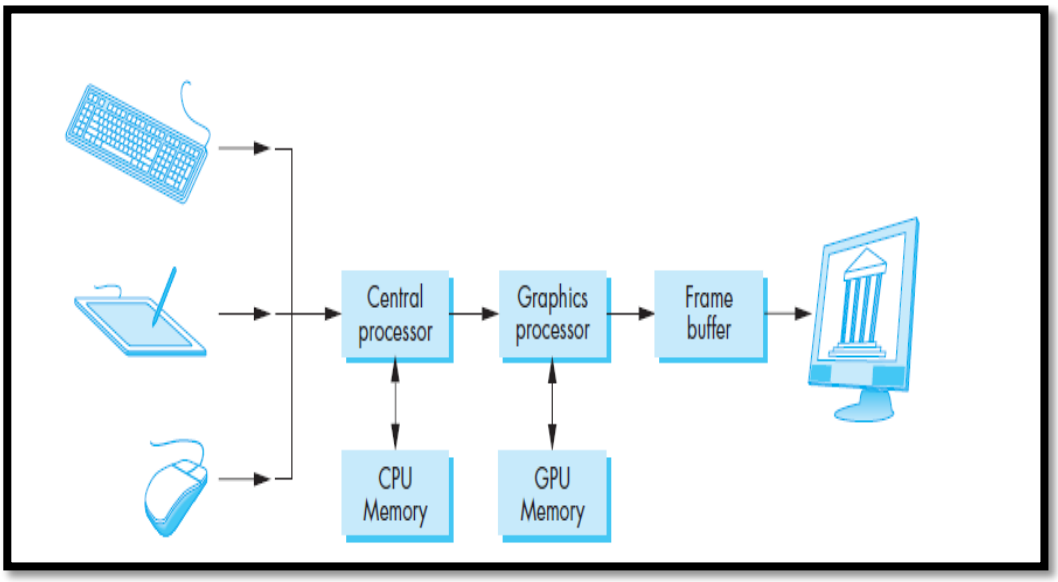



**Figure 1. 10 Graphical System.**

In any case, the color values for all the pixels on the screen are stored in a large block of memory known as a frame buffer. Changing the image on the screen requires changing color values that are stored in the frame buffer.

The screen is redrawn many times per second, so that almost immediately after the color values are changed in the frame buffer, the colors of the pixels on the screen will be changed to match, and the displayed image will change.

A computer screen used in this way is the basic model of raster graphics. The term “raster” technically refers to the mechanism used on older vacuum tube computer monitors: An electron beam would move along the rows of pixels, making them glow.

The beam was moved across the screen by powerful magnets that would deflect the path of the electrons.

The stronger the beam, the brighter the glow of the pixel, so the brightness of the pixels could be controlled by modulating the intensity of the electron beam.

The color values stored in the frame buffer were used to determine the intensity of the electron beam. (For a color screen, each pixel had a red dot, a green dot, and a blue dot, which were separately illuminated by the beam.)

A modern flat-screen computer monitor is not a raster in the same sense. There is no moving electron beam. The mechanism that controls the colors of the pixels is different for different types of screen. But the screen is still made up of pixels, and the color values for all the pixels are still stored in a frame buffer.

The idea of an image consisting of a grid of pixels, with numerical color values for each pixel, defines raster graphics.

## 1.8 Display Screens

Display screens are output devices that show programming instructions and data as they are being and information after it is processed, display screens are either CRT (Cathode-Ray-Tube) or Flat-panel display.

### 1. The Cathode Ray Tube (CRT) Display:

It contains tube of glass, with a big end represents the screen coated inside by phosphor layer, while the other end essentially contains the electronic gun and Deflection Coils. Figure 1. 11 show Cathode Ray Tube (CRT).

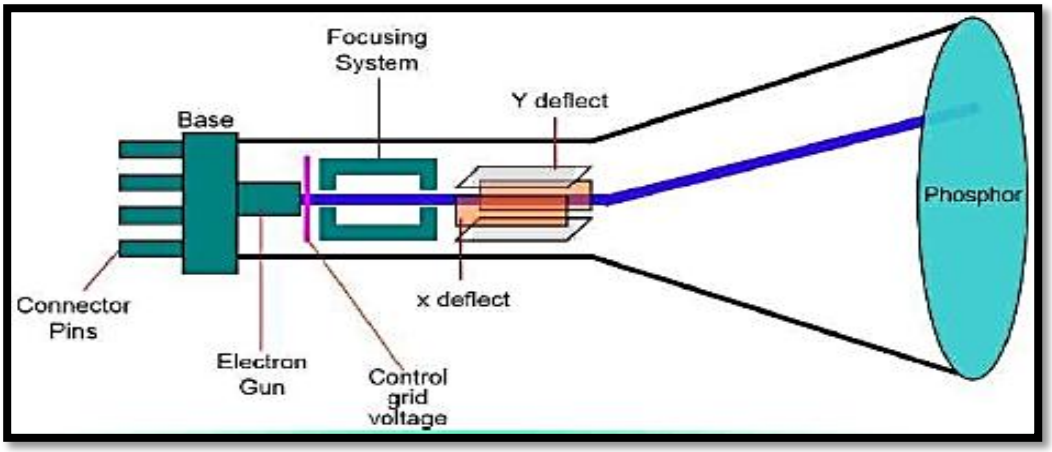



**Figure 1. 11 Cathode Ray Tube(CRT).**

## The Electronic gun contains parts as follows:

1. Heating Element
2. Cathode
3. Control Grid
4. Acceleration Anode
5. Focusing Grid.

CRT based on the physical concept that the phosphor produce a light if its strike with Electrons have velocity and moment that affects the phosphor electron to speed and free to give the light.

CRT It is an evacuated glass tube with an inner side phosphor coated screen and the CRT works as follow:

**1.** The electron gun emits a beam of electrons (cathode rays).

**2.** The electron beam passes through focusing and deflection systems that direct it towards specified positions on the phosphor-coated screen.

**3.** When the beam hits the screen, the phosphor emits a small spot of light at each position contacted by the electron beam.

**4.** It redraws the picture by directing the electron beam back over the same screen points quickly.

There are two techniques used for producing images on the CRT screen:

## A. Vector scan/Random Scan Display

It directly traces out only the desired lines on CRT based on equation stored at the computer memory. To draw a line between point p1 & p2 we directly drive the beam deflection circuitry, which focus beam directly from point p1 to p2. In Figure 1.12 we see the Architecture of a vector display.

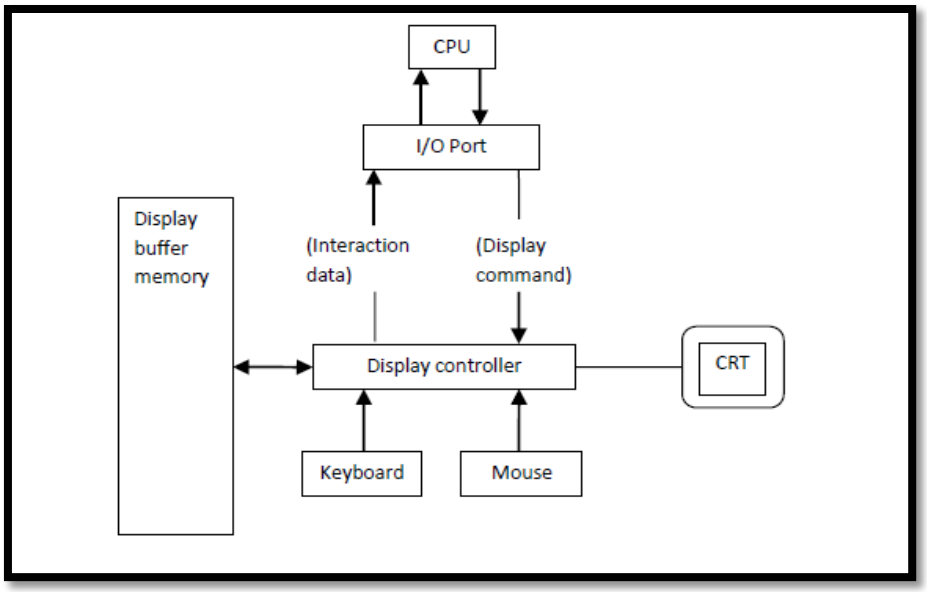



**Figure 1.12 Architecture of a vector display**

## B. Raster Scan Display

The display image is stored in the form of 1's and 0's in the refresh buffer. The video controller reads this refresh buffer and produces the actual image on screen. It will scan one line at a time from top to bottom & then back to the top. Figure 1.13 show the Architecture of a raster display.

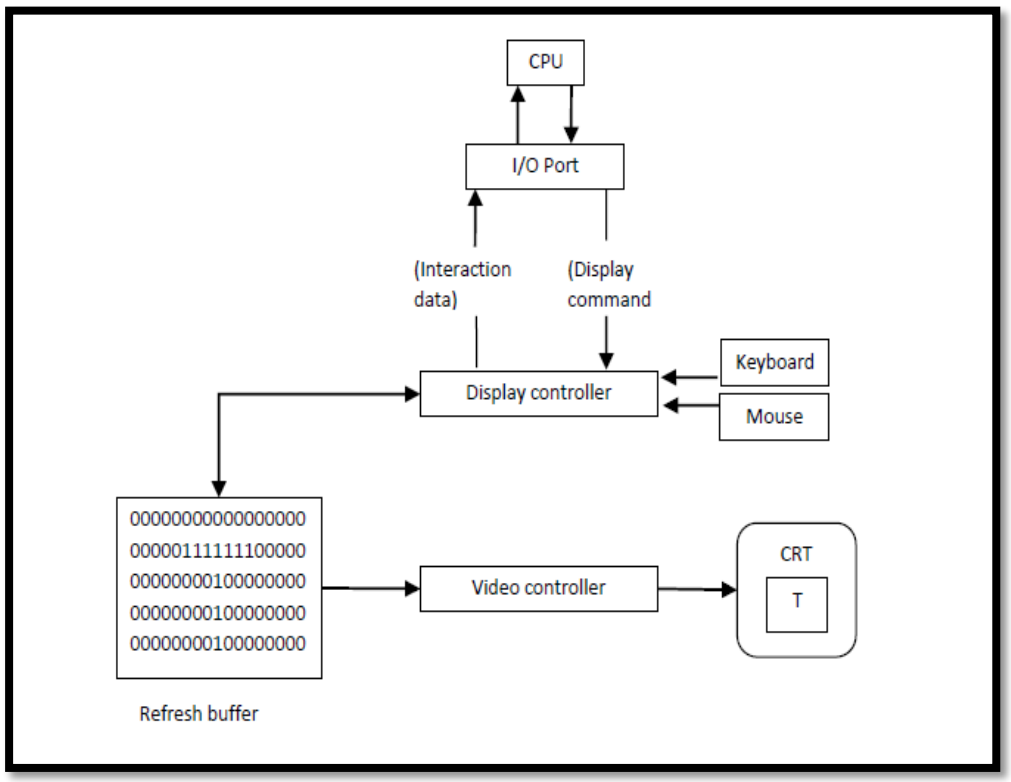



**Figure 1.13 Architecture of Raster Display**

Frame Buffer is a special area of memory dedicated to graphics only used in raster image.

The Frame buffer holds set of intensity values for all the screen points.

These values are retrieved from frame buffer and display on screen one row at a time. Additional bits are required when color and intensity variations can be displayed up to 24-bits per pixel are included in true color display systems.

For monochrome system with one bit per pixel the frame buffer is commonly called a Bitmap. And for systems with multiple bits per pixel, the frame buffer is often referred as a Pix map.

**Advantages of CRT display**

1. Low cost.
2. Low weight.
3. High response and high resolution.
4. High flame and wide view angle.

5. Flat screen with clear image, clear colors and high resolution.

6. Authenticated technique and easy, expensive repairs.

**Disadvantages of CRT display**

1. The view angle is relatively narrow.

2. Big size, huge weight with the screen size.

3. Energy consumed.

4. Electromagnetic rays surrounds the screen.

5.Most of the CRT screen works with analogue signal and rarely with digital one.

# Different between Raster Scan and Random Scan System

| Base of Difference | Raster Scan System | Random Scan System |
|---|---|---|
| Electron Beam | The electron beam is swept across the screen, one row at a time, from top to bottom. | The electron beam is directed only to the parts of screen where a picture is to be drawn. |
| Resolution | Its resolution is poor because raster system in contrast produces zigzag lines that are plotted as discrete point sets. | Its resolution is good because this system produces smooth lines drawings because CRT beam directly follows the line path. |
| Picture Definition | Picture definition is stored as a set of intensity values for all screen points, called pixels in a refresh buffer area. | Picture definition is stored as a set of line drawing instructions in a display file. |
| Realistic Display | The capability of this system to store intensity values for pixel makes it well suited for the realistic display of scenes contain shadow and color pattern. | These systems are designed for line-drawing and can't display realistic shaded scenes. |
| Draw an Image | Screen points/pixels are used to draw an image. | Mathematical functions are used to draw an image. |

## 2. The Flat Panel Display

Compared to CRT displays, flat panel displays are much thinner, weightless, and consuming less power. Thus, they are better for portable computers. Flat panel displays are made up of two plates of glass with a substance between them, which is activating in different ways.

Flat panel displays are distinguished in two ways:

1. By the substance between the plates of glass.
2. By the arrangement of the transistors in the screens.

Two common types of technology used in flat panel display screens are:

### A. Liquid Crystal display (LCD)

It consists of a substance called liquid crystal, the molecules of which line up in a way that alter their optical properties.

As a result, light usually backlighting behind the screen is blocked or allowed through to create an image.

There are some advantages and disadvantages of LCD :-

**Advantages of LCD:**

1. Low power consumption.
2. Small size.
3. Low cost.

**Disadvantages of LCD:**

1. LCD are temperature dependent (0-70C).
2. LCD do not emit light; as a result, the image has very little contrast.
3. LCDs have no color capability.
4. The resolution is not as good as that of a CRT.

## B. Gas Plasma Display

It is like a neon bulb, in which the display uses a gas that emits light in the presence of an electric current. That is, the technology uses neon gas and electrodes above and below the gas.

Figure 1.14 show Generic flat panel display.

When electric current passes between the electrodes, the gas glows. Although gas plasma technology has better resolution than LCD technology, it is more expensive and thus is not used as often as a LCD.

On the other hand, LCDs are not practical for screens larger than 20 inches and so are not practical for TV size screen.

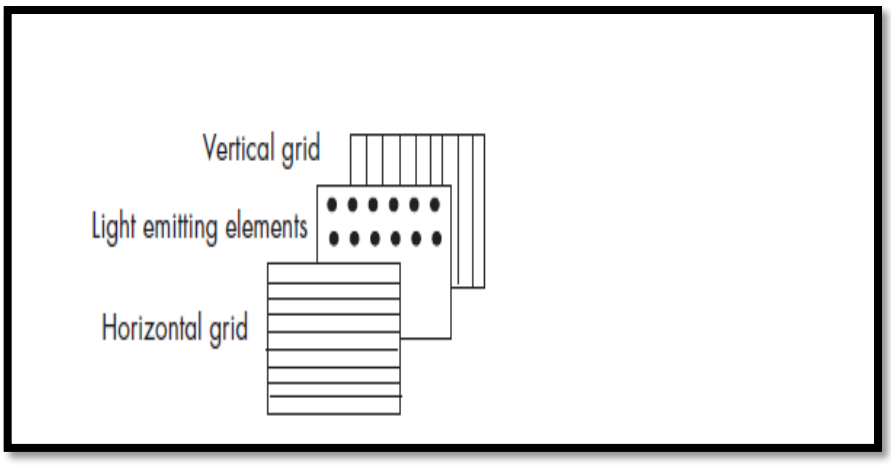



**Figure 1.14 Generic Flat Panel Display**

There are some advantages and disadvantages of plasma panel display:

**Advantages of Plasma Panel Display:**

1. High resolution.
2. Large screen size is also possible.
3. Less volume.
4. Less weight.
5. Flicker free display.

**Disadvantages of Plasma Panel Display:**

1. Poor resolution.
2. Wiring requirement anode and the cathode is complex.
3. Its addressing is also complex.

## 1.9 Screen Clarity

The screen clarity depends on three qualities:

### 1 .Resolution:

Resolution is the numbers of pixels in digital image with N*M (N horizontal pixels× M vertical pixels), the number of pixels give the capacity to display the details in original image.

Resolution is expressed in terms of the formula each pixel can be assigned a color or particular shade of gray. A screen with 640× 480 pixels multiplied together equals 307200 pixels.

This screen will be less clear and sharp than a screen with 800× 600 (equals 480000) 0r 1024×768 (equals 786432) pixels.

Resolution depend on:

1. Special resolution (number of pixels).
2. Brightness resolution (values of pixels).

## 2 .Dot Pitch:

It is the amount of space between the center of adjacent pixels, the closer the dots, the crisper the image. For crisp images, dot pitch should be less than 0.31 millimeter.

## 3. Refresh Rate:

It is the number of times per second that the pixels are recharged so that their glow remains bright. In general, displays are refreshed 45 to 100 times per second.

# CHAPTER TWO

# GRAPHICS PRIMITIVES

# *Chapter Two*

## *Graphics Primitives*

- **Introduction Graphics Primitives**
- **Point Plotting Techniques.**
- **Qualities of Good Line Drawing Algorithms.**
  - **Horizontal, Vertical and Diagonal Lines.**
  - **Line Generation Algorithms (Arbitrary Slope Line)**
  - **The Digital Differential Analyzer (DDA).**
  - **Bresenham's Algorithm.**
- **Generation of Circles Algorithms.**
  - **Direct (implicit) algorithm**
  - **Parametric (polar) algorithm**
  - **The Symmetry of circle:**
  - **Incremental Algorithm**
  - **Bresenham's circle algorithm**
  - **Midpoint Circle Algorithm**
- **Ellipses Drawing**
  - **Polar representation of an ellipse**
  - **Incremental method to drawing of ellipse**
- **Arc and Sector**
- **Scan converting a rectangle**

# Chapter Two

# Graphics Primitives

## 2.1 Introduction Graphics Primitives

You will be introduced the concept of writing pictures on the screen as a set of points. Any picture can be thought of as a combination of points. The idea is to identify the points, which form the part of the picture one is trying to draw and by a suitable technique display these points. To do this, the screen is supposed to be made up of a number of pixels (picture cells), each pixel corresponding to a point. Those pixels, which form a part of the picture being drawn, are made to light up so that the picture is visible on the screen. The trick is to switch on the laser beam (of the CRT) when it is passing over the pixel and switch it off when it is passing over a pixel that does not form a part of the picture. This chapter tells us about the techniques of identifying those pixels that should from the part of the picture and the various difficulties that one will have to encounter in the process.

Once the pixels are identified, that hardware takes over the question of actually drawing the pictures.

## 2.2 Point Plotting Techniques

In order to draw a picture on a raster display, we must determine the corresponding points in the frame buffer that make up the picture. To perform this task, we must write scan conversion point plotting algorithms.

Both the frame buffer and the display screen are given a two-dimensional coordinates system with the origin at the lower left corner.

To create a two-dimensional image, each point in the image is assigned a color. A point in 2D can be identified by a pair of numerical coordinates. Colors can also be specified numerically. However, the assignment of numbers to points or colors is somewhat arbitrary.

So we need studying coordinate systems, which associate numbers to points, and color models, which associate numbers to colors.

## 2.2.1 Pixel Coordinates

A digital image is made up of rows and columns of pixels. A pixel in such an image can be specified by saying which column and which row contains it. In terms of coordinates, a pair of integers giving the column number and the row number can identify a pixel. For example, the pixel with coordinates (3, 5) would lie in column number 3 and row number 5 see Figure 2.1.

Conventionally, columns are numbered from left to right, starting with zero. Most graphics systems, including the ones we will study in this chapter, number rows from top to bottom, starting from zero.

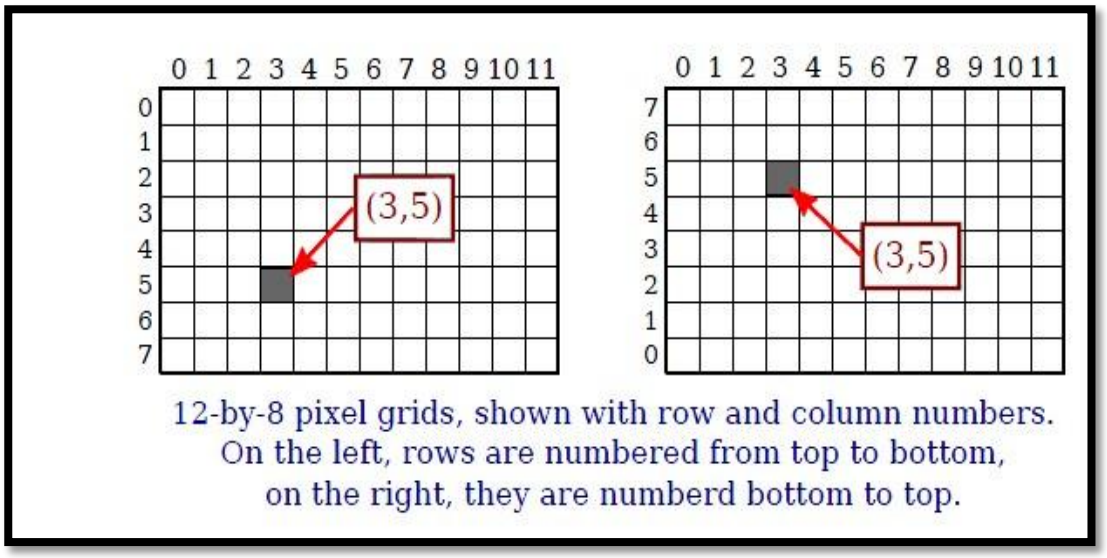



**Figure 2.1 Coordinate on Graphics System**

The drawing on the screen starts from top to down, and from left to right. The pixel coordinates on the screen (VGA 640x480) is shown in figure 2.2 below:

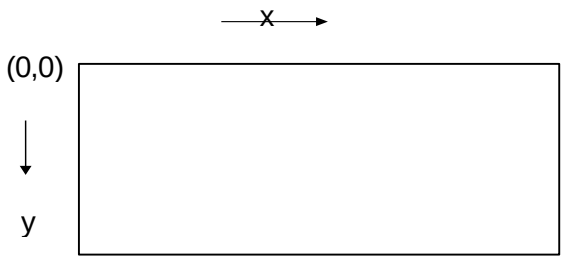



**Figure 2.2 Coordinates on the Drawing Screen**

## 2.2.2 Scan Conversion

The process of representing continuous graphics objects as a collection of discrete pixels called **Scan conversion**. Figure 2.3 suggests how an image is created and displayed.

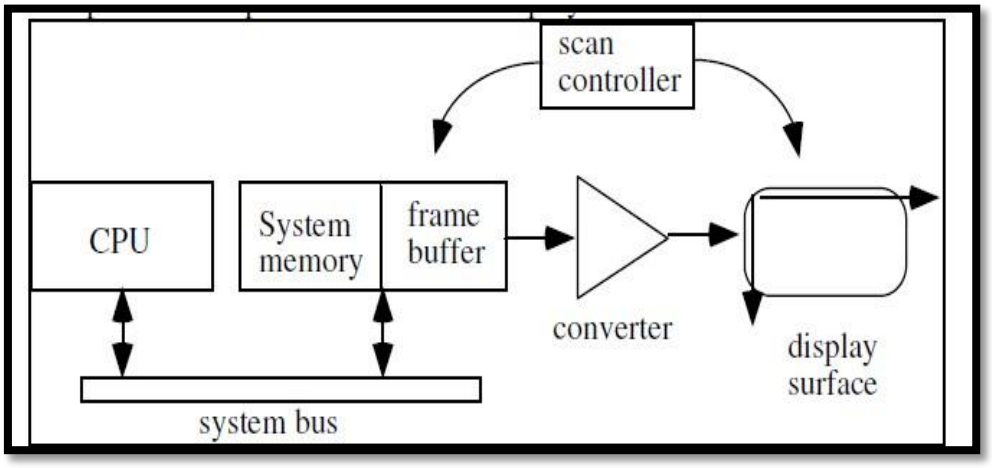



**Figure 2.3 Block diagram of a computer with Raster Display.**

A line connects two points. It is a basic element in graphics. To draw a line, you need two points between which you can draw a line. In the following three algorithms, we refer the one point of line as X0, Y0 and the second point of line as X1, Y1.

In fact, the closer the points to one another, we see better pictures (see the example below) Figure 2.4.

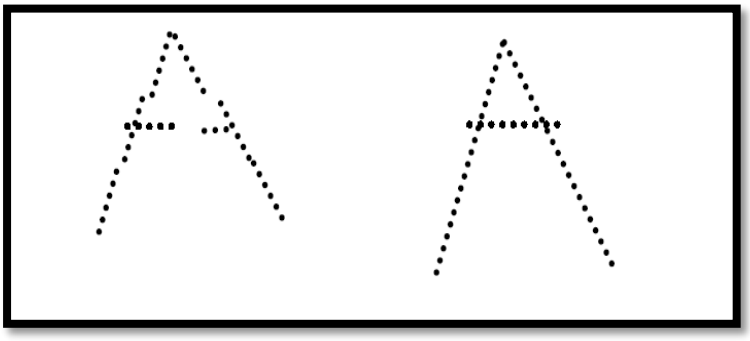


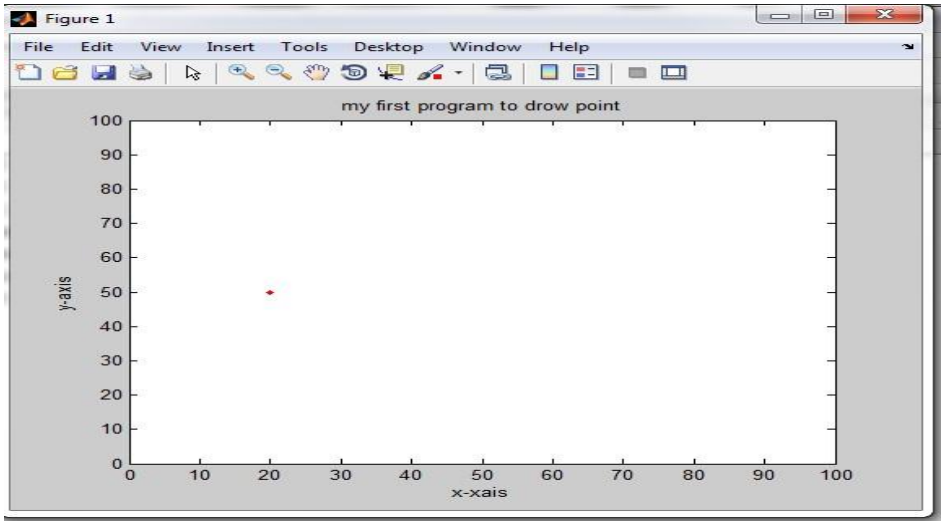
**Figure 2.4 Examples of point plotted pictures**

In the above figure 2.4 both pictures indicate A, but in the second picture, the points are closer and hence it appears more like A than the first. How many points are there per unit area of the screen indicate what is known as the "**resolution**" of the monitor. Higher the resolution, we get more number of points and hence better quality pictures can be displayed (As a corollary, such high-resolution monitors are costlier).

# Program1 :Matlab program to draw the point(20,50)?

```
clc
clear all
close all
x=20;
y=50;
axis([0 100 0 100])
 plot(x,y,'.r')
xlabel('x-axis')
ylabel('y-axis')
title('my first program to draw point')
```

# The output of this program is show below:



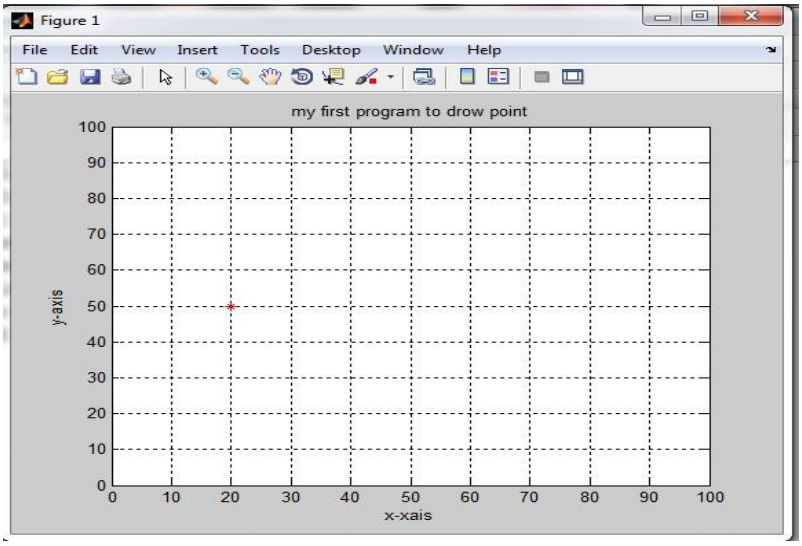
# Program 2: Matlab program to draw any point?

```
clc
clear all
close all
x=input('enter the value of x');
y=input('enter      the value of y');
axis([0 100 0 100])
grid on
plot(x,y,'*r')
xlabel('x-axis')
ylabel('y-axis')
 title('my first program to draw point')
```

## The output of program is:

enter the value of x 20

enter the value of y 50



## 2.3 Qualities of Good Line Drawing Algorithms

Before we start looking at a few basic line drawing algorithms, we see what the conditions that they should satisfy are. While the same picture can be drawn using several algorithms, some are more desirable than others, because they provide as features that enable us to draw better "quality" pictures. A few of the commonly expected qualities are as follows:

**i. Lines should appear straight:** Often straight lines drawn by the point plotting algorithms do not appear very straight.
The reason is not far to be searched for. Any point plotting algorithm will give a series of points (x,y) for various values of x and y. In a general case, the values of x and y need not be integers, they can be any real numbers. But on the screen, pixel values are only integers.

So, what do we do? The easy solution is to round off. If two points (successive points) are given say as (6.6, 15.4) and (7.4, 16) when rounded off 6.6 becomes 7 and 15.4 become 15.

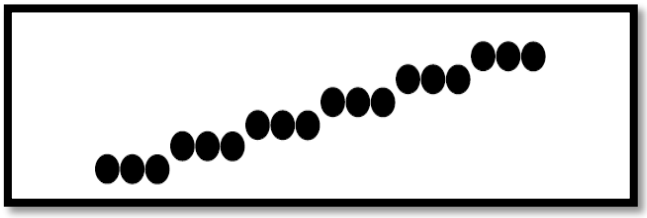


**Figure 2.5 Jagged Lines**

Note that the slope of the line has changed - A series of such changes between successive points make the lines look as shown in the figure 2.5 above (Jagged lines).

**ii. Lines should terminate accurately:** The cause is still the same as in plotting point technique. Because of inaccuracies and approximations, the lines do not terminate accurately. Either they stop short of the point at which they should end or extend beyond the points result? Intersections and joints do not form correctly. Look at the examples below Figure 2.6.

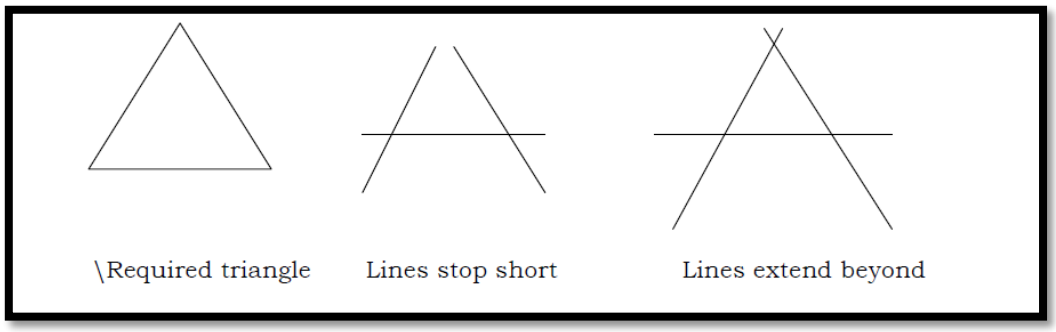



**Figure 2.6 Example to Draw Triangle**

**iii. Maintain constant intensity:** The pictures are drawn with illumination i.e. a number of points along the line are illuminated. As long as the intensity of these points is uniform, we have a pleasing picture to look at.

This can be done if the points to be illuminated are equidistance from one another. However, because of the inaccuracies in the algorithm, we often end up with either dots that are too close or a bit further from each other.

Obviously two points, close to each other, when illuminated, make the points look brighter. The result is a line that is brighter in save parts and not so bright in others. The result will be a line that looks jagged and non-uniform.

**iv. Lines should be drawn rapidly:** This is especially the case in interactive graphics, wherein lines are drawn in real time. While there may not be many problems with straight lines, complex figures may need longer computations before the next pants are identified. Hence, the picture is drawn in bits and pieces, which may appear unpleasant or even irritating at times.

Having seen some of the requirements of algorithms, we now see a few practical algorithms to draw simple figures like straight lines or circles.

## 2.4 Line Drawing Algorithms (Straight Line)

A line connects two points. It is a basic element in graphics. To draw a line, you need two points between which you can draw a line.

In the following **three algorithms**, we refer the one point of line as Xstart,Ystart, and the second point of line as Xend, Yend.

## 2.4.1 Horizontal Lines

The screen coordinates of the points on a horizontal line are obtained by keeping the value of (y) constant and repeatedly incrementing the (x) value by one unit as in algorithm (1).

**Algorithm of Horizontal Lines (1)**

Input: Xstart, Xend, Yspecified.

Output: Horizontal line.

```
{
  for x= Xstart to Xend
  plot (x, yspecified , color);
}
```

If Xstart > Xend then replace Xend by Xstart and vice versa in for loop at algorithm (1).

**Example 1:** Trace the line where the endpoints are (1, 5), (5, 5) by Horizontal line, and draw the line in screen coordinate.

**Solution:**

Xstart=1, Yspecified=5,

Xend =5, Yspecified=5

X1=1, y1=5

X2=2, y2=5

X3=3, y=5

X4=4, y=5

X5=5, y=5

We generate point from (1,5) to (5,5) as show in table 2.1 Below:

| X | Y | Plot in screen (x, y) |
|---|---|---|
| 1 | 5 | (1,5) |
| 2 | 5 | (2,5) |
| 3 | 5 | (3,5) |
| 4 | 5 | (4,5) |
| 5 | 5 | (5,5) |

**Table 2.1 Point Generation using Horizontal Algorithm**

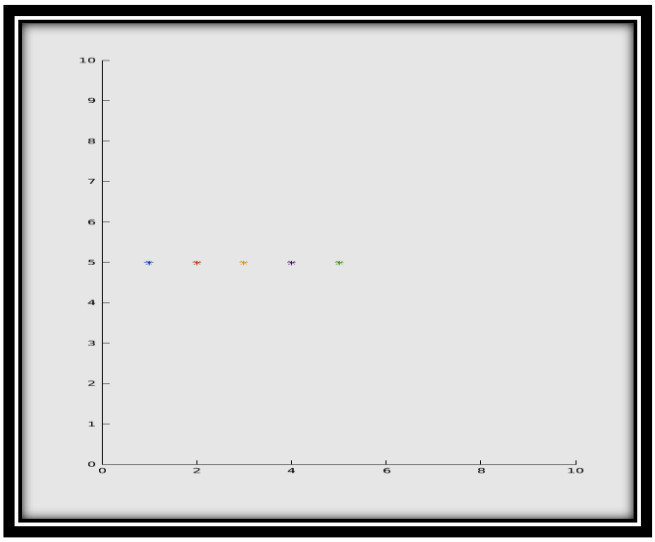



**Figure 2.7 Drawing line using Horizontal Line Algorithm**

# Program 3: Drawing Horizontal line using MATLAB programing Language.

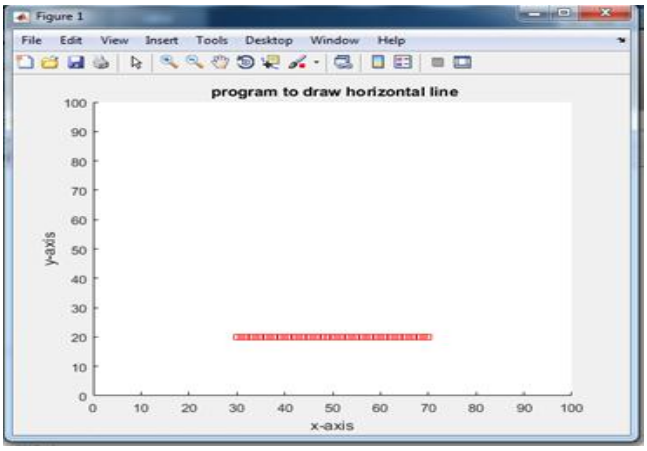
```
% Horizontal line
 clc;
 clear all;
 close all
x1=input('Enter x-start value: ');
x2=input('Enter x-end value : ');
if x1>x2
 m=x1;
 x1=x2;
 x2=m;
end
y=input('Enter y value : ');

axis([0 100 0 100])
hold on
for x=x1:x2
  plot (x,y,'ro')
end
xlabel('x-axis')
ylabel('y-axis')
title(' program to draw horizontal line')
```

## The output of program

Enter x-start value: 30

Enter x-end value: 70

Enter y value: 20




## 2.4.2 Vertical Lines

The screen coordinates of the points on a vertical line are obtained by keeping the value of (x) constant and repeatedly incrementing the (y) value by one unit as in algorithm (2).

**Algorithm of Vertical Lines (2):**

Input: Ystart,Yend, Xspecified.

Output: Vertical line.

```
{
for y= Ystart to Yend
plot (xspecified,y,color);
 }
```

If Ystart > Yend then replace Yend by Ystart and vice versa in for loop at algorithm (2).

**Example 2:** Trace the line where the endpoints are (5, 1), (5, 5) by Horizontal line, and draw the line in screen coordinate.

**Solution:**

xspecified=5,Ystart=1,

xspecified=5,Yend =5

X1=5, y1=1

X2=5, y2=2

X3=5, y=3

X4=5, y=4

X5=5, y=5

We generate point from (5,1) to (5,5) as show in table 2.2 Below:

| X | Y | Plot in screen (x, y) |
|---|---|---|
| 5 | 1 | (5,1) |
| 5 | 2 | (5,2) |
| 5 | 3 | (5,3) |
| 5 | 4 | (5,4) |
| 5 | 5 | (5,5) |

**Table 2.2 Point Generation using Vertical Algorithm**

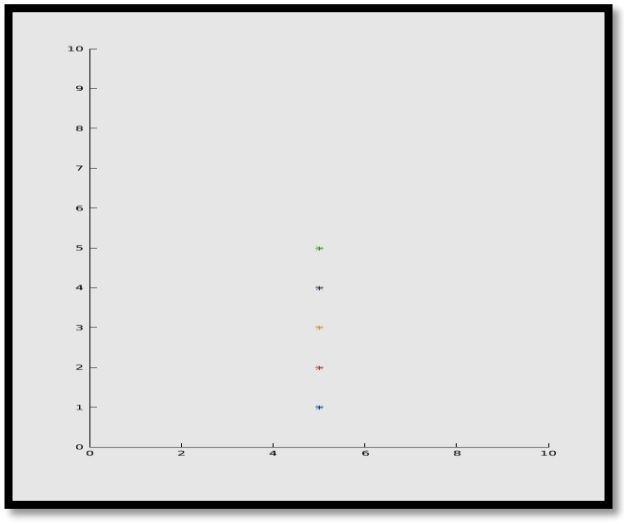
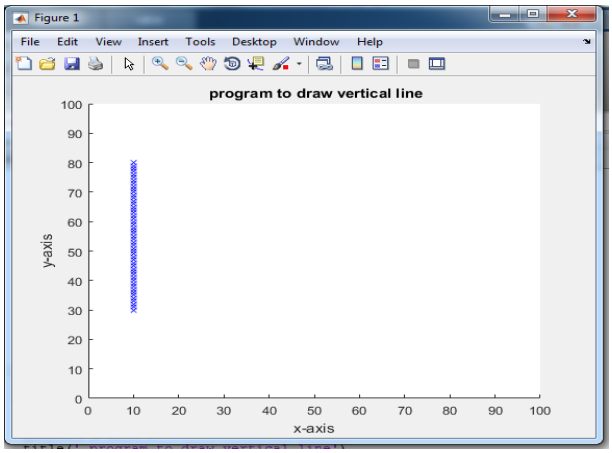



**Figure 2.8 Drawing line using Vertical Line Algorithm**

# Program 4: Drawing Vertical line using MATLAB

```
% Vertical line
clc;
clear all;
close all;
 y1=input('Enter ystart value: ');
 y2=input('Enter y-end value : '); x=input('Enter
 x value : ');
 if y1>y2
  m=y1;
  y1=y2;
  y2=m;
 end
 axis ([0 100 0 100])
 hold on
 for y=y1:y2
    plot (x,y,'bx')
 end
  xlabel('x-axis')
  ylabel('y-axis')
  title(' program to draw vertical line')
```

**The output of program:**




Enter ystart value: 30

Enter y-end value: 80

Enter x value: 10

## 2.4.3 Diagonal Lines

To draw a diagonal line with a slope equal to (+1), we need only repeatedly increment by one unit both x and y values from the start to end pixels as shown in algorithm (3).

**Algorithm of Diagonal Lines (3):**

Input: Xstart,Ystart, Xend, Yend. i=0

Output: Diagonal line.

```
{
while (xstart + i) ≤ Xend )
        {
        plot (xstart + i, ystart + i, color);
        i= i+1;
        }
}
```

To draw a line with slop equals to (-1), replace (ystart+i) by (ystart-i) in algorithm (3).

**Example 3:** Trace the line where the endpoints are (1, 1), (5, 5) by Horizontal line, and draw the line in screen coordinate.

**Solution:**

Xstart=1,Ystart=1,

Xend=5,Yend =5

X1=1, y1=1

X2=2, y2=2

X3=3, y=3

X4=4, y=4

X5=5, y=5

We generate point from (1,1) to (5,5) as show in table 2.1 Below:

| X | Y | Plot in screen (x, y) |
|---|---|---|
| 1 | 1 | (1,1) |
| 2 | 2 | (2,2) |
| 3 | 3 | (3,3) |
| 4 | 4 | (4,4) |
| 5 | 5 | (5,5) |

**Table 2.3 Point Generation using Diagonal Algorithm**

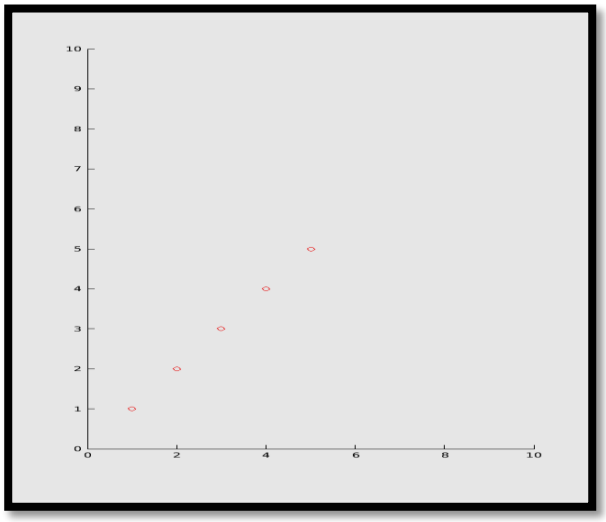



**Figure 2.9 Drawing line using Diagonal Line Algorithm**

# Program 5: program in MATLAB to draw Diagonal line.

```
%Diagonal line
clc ;
clear all;
close all
x1=input('Enter x-start value: ');
y1=input('Enter y-start value: ');
x2=input('Enter x-end value : '); y2=input('Enter y-end value : ');
i=0;
axis ([0 100 0 100])
hold on
 while (x1+i)<=x2
plot(x1+i,y1+i,'b*')
  i=i+1;
end
 xlabel('x-axis')
 ylabel('y-axis')
 title(' program to draw diagonal line')
```

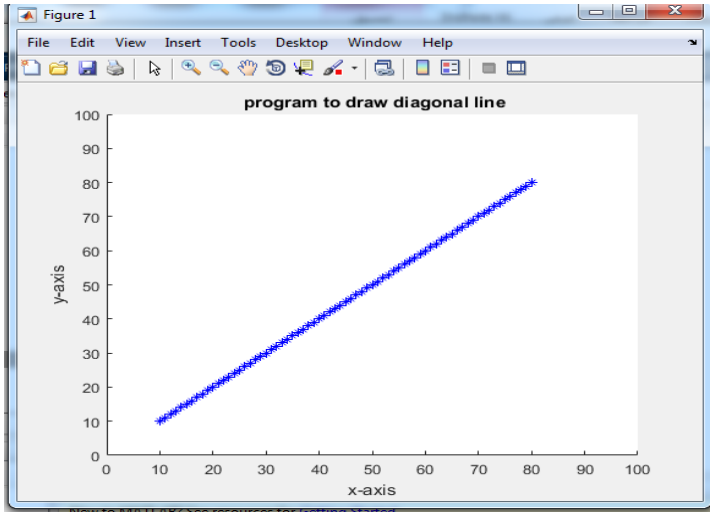
## The output of program:

Enter x-start value:10

Enter y-start value:10

Enter x-end value : 80

Enter y-end value :8



## 2. 5 Line Generation Algorithms (Arbitrary Slope Line)

Drawing lines with arbitrary slope creates several problems, such as:

**1.** The display screen can be illuminated only at pixels locations; therefore, a raster scan display has a staircase effect that only approximates the actual line as shown in figure 2.10. Although it may not be possible to choose pixels that lie on the actual line, we want to turn on pixels lying closest to it. For example, below figure, the pixel at location B is a better choice than the one at location A.

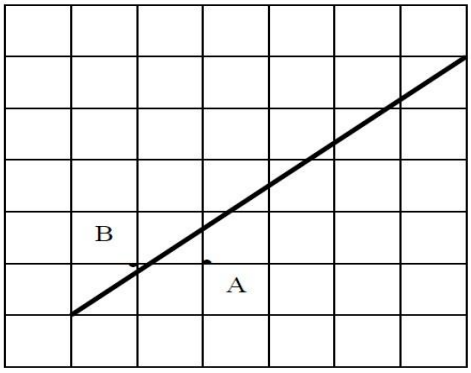



**Figure 2.10 approximate of pixel A and pixel B**

**2.** Determining the closest (best) pixels is not easy.

Different algorithms calculate different pixels to make up the approximating line.

The choice of algorithm depends on:

1. The speed of line generation.
2. The appearance of the line.

Therefore, to understand these criteria had better let's look at several different line generating algorithms.

### 2.5.1 Direct Method

In this method, we learn how to draw a line between two points by drawing a group of pixels using the command plot (x , y , color), with substituting in straight line equation:

$$\mathbf{Y = m \times X + b}$$

Where (m) is the slope and (b) is a constant which represents the clipping from y-axis (y-intercept).

$$m = \frac{yend - ystart}{xend - xstart}$$ **,b=ystart – m * xstart**

**Note:** start may be 1, and end may be 2.

Direct method for drawing lines can be shown in algorithm (4).

**Algorithm of Direct method (4):**

```
Input: Xstart,Ystart, Xend, Yend.
Output: Arbitrary line.
{
        dx=Xend-Xstart
        dy=Yend-Ystart
       m=dy/dx
        b= Ystart – m * Xstart ;
       for x= Xstart to Xend  step sign(dx)
            {
                y=m *x + b;
             plot(x,y,color);
    }
}
```

Direct method is clarified in algorithm (4), assuming that, Sign is a function returns (-1,0,+1) as it's argument is (<0,=0,>0).

**Example 4:** Trace the line where the endpoints are (1, 5), (7, 2) by Direct method, and draw the line in screen coordinate.

**Solution:**

x1=1,y1=5

x2=7,y2=2

**dx** = x2-x1

**=**7-1=**6**;

**dy**=y2-y1

**=**2-5= **-3**;

**m**= dy/dx

**=**-3/6= -1/2 =**-0.5**;

**b**=y1-m*x1

**=**5- (-0.5)*1=**5.5 ;**

We generate point from (1,5) to (7,2) as show in table 2.4 Below:

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

**Table 2. 4 Point Generation using Direct Algorithm**

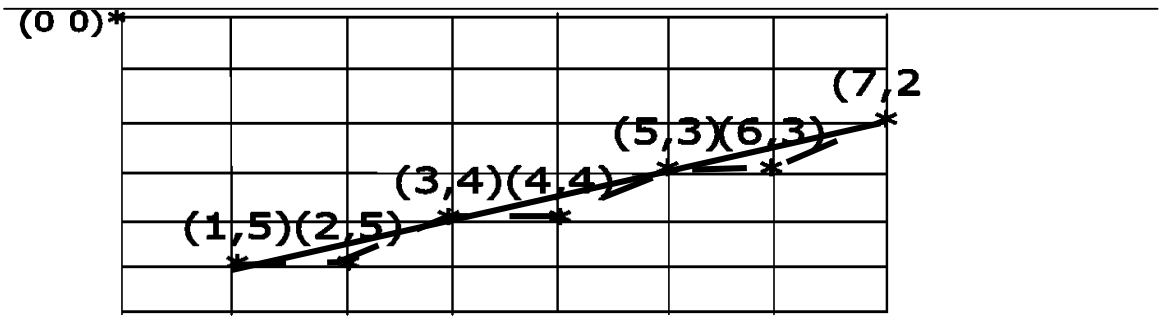
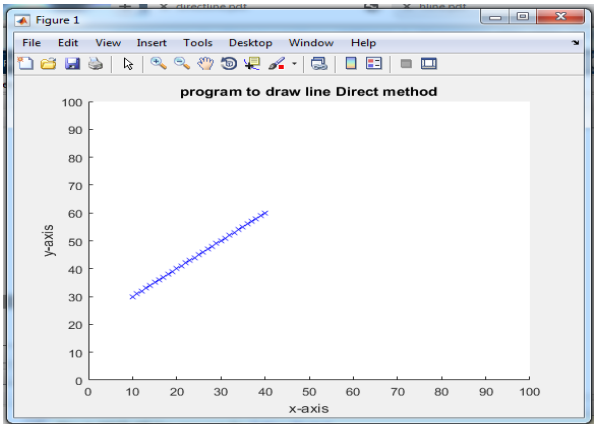



**Figure 2.11 Drawing line using Direct Line Algorithm**

# Program 6: Program in MATLAB to draw Arbitrary line using Direct method.

```
%Direct method for Arbitrary line

clc;
close all;
clear all
x1=input('Enter x-start value: ');
y1=input('Enter y-start value: ');
x2=input('Enter x-end value : ');
y2=input('Enter y-end value : ');
axis([0 100 0 100]);
hold on
dx=x2-x1;
dy=y2-y1;
m=dy/dx;
b= y1-m * x1 ;
i=0;
 for x=x1:sign(dx):x2
  y= m *x + b ;
  plot (x , y , 'bx');
  i=i+1;   end
 xlabel('x-axis')
 ylabel('y-axis')
 title(' program to draw line Direct method')
```

**The output of program:**




Enter x-start value: 10

Enter y-start value: 30

Enter x-end value : 40

Enter y-end value : 60

## 2.5.2 The Digital Differential Analyzer (DDA)

The ***Digital Differential*** **Analyzer (DDA)** is a scan-conversion line algorithm and is the simple line generation algorithm based on calculating either dy or dx. A line is sampled at unit intervals in one coordinate and corresponding integer values nearest the line path are determined for other coordinates, DDA Algorithm for drawing lines can be shown in algorithm (5).

**Algorithm of DDA (5):**

Consider one point of the line as (X1,Y1) and the second point of the line as (X2,Y2).

```
Input: x1,x2,y1,y2.
        i=0;
Output: Arbitrary line
{    If (abs(x2-x1)≥abs(y2-y1))

          length= abs( x2-x1);

      else

          length= abs( y2-y1);

      dx=(x2 -x1)/length;

      dy=(y2-y1)/length;

      x=x1;

      y=y1;

      while (i≤length)

         {

            plot(round(x),round(y),color);

            x=x+  dx  ;

            y=y+  dy ;

            i=i+1 ;

}
```

## Advantage of DDA algorithm:

1. The DDA algorithm is a faster method for calculating pixel positions than the direct
2. DDA is simple algorithm.

## Disadvantage of DDA algorithm:

1. Floating-point arithmetic is time consuming.
2. Poor end point accuracy.

**Example 5**: Trace the line where the end points between (23,33), (29,40) by DDA method.

## Solution:

x1=23,y1=33

x2=29,y2=40

Find dx=x2-x1

=29-23=6

dy=y2-y1

=40-33=7

If dy>dx then length=dy (choice the large dx or dy)

Length=7

Xinc=dx / length

=6/7 =0.857

Yinc=dy / length

=7/7=1.

We generate point from (23,33) to (29,40) as show in table 2.5 Below:

| i | X | Y | Plot (X) | Plot (y) | Plot(x,y) |
|---|---|---|---|---|---|
| 0 | 23 | 33 | 23 | 33 | (23,33) |
| 1 | 23.857 | 34 | 24 | 34 | (24,34) |
| 2 | 24.714 | 35 | 25 | 35 | (25,35) |
| 3 | 25.571 | 36 | 26 | 36 | (26,36) |
| 4 | 26. 429 | 37 | 26 | 37 | (26,37) |
| 5 | 27.286 | 38 | 27 | 38 | (27,38) |
| 6 | 28.143 | 39 | 28 | 39 | (28,39) |
| 7 | 29 | 40 | 29 | 40 | (29,40) |

**Table 2.5 Point Generation using DDA Algorithm**

The line is drawing as show in below figure 2.12

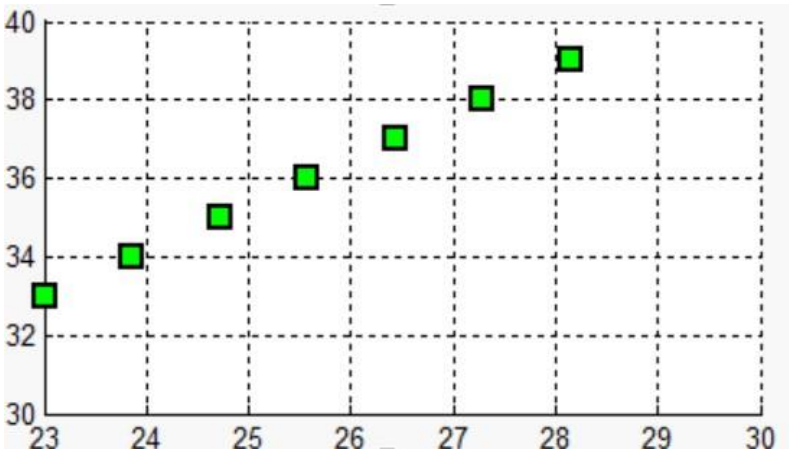



**Figure 2.12 Drawing Line using DDA Algorithm**

**NOTE**:

The DDA algorithm is faster than the direct use of the line equation since it calculates points on the line without any floating point multiplication. However, a floating-point addition is still needed in determining each successive point. Furthermore, cumulative error due to limited precision in the floating-point representation may cause calculated points to drift away from their true position when the line relatively long.

**Example 6**: Trace the line where the end points (5,5) , (10,9) by DDA algorithm.

**Solution:**

x1=5,y1=5

x2=10,y2=9

**dx=(x2-x1)**

=(10-5)=5

**dy=(y2-y1)**

=(9-5)=4

dx>dy

then dx= Length=5

**xinc=dx/length**

=5/5=1

**yinc=dy/length**

=4/5=0.8

We generate point from (5,5) to (10,9) as show in table 2.6 Below:

| i | x | Y | plot (x) | plot(y) | Plot(x,y) |
|---|---|---|---|---|---|
| 0 | 5 | 5 | 5 | 5 | (5,5) |
| 1 | 6 | 5.8 | 6 | 6 | (6,6) |
| 2 | 7 | 6.6 | 7 | 7 | (7,7) |
| 3 | 8 | 7.4 | 8 | 7 | (8,7) |
| 4 | 9 | 8.2 | 9 | 8 | (9,8) |
| 5 | 10 | 9 | 10 | 9 | (10,9) |

**Table 2.6 Point Generation using DDA Algorithm**

The line is drawing as show in below Figure 2.13

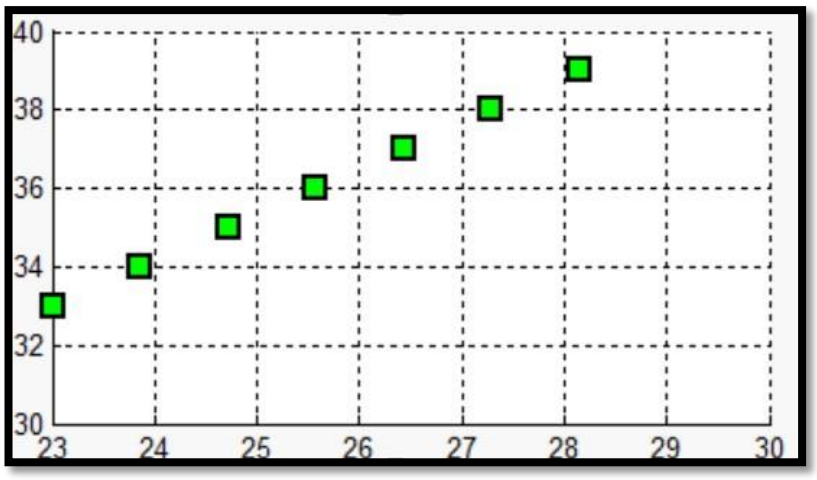
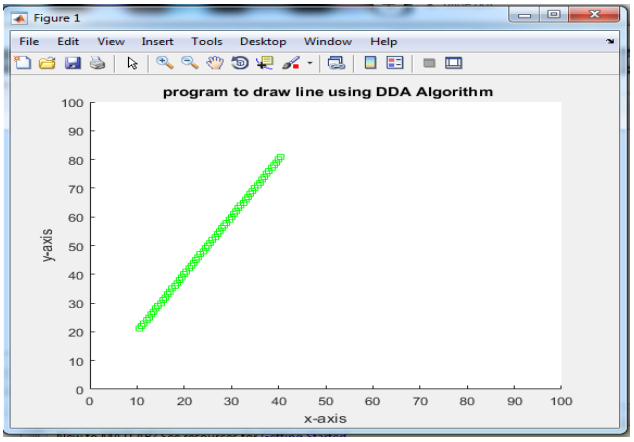



**Figure 2.13 Drawing line using DDA Algorithm**

# Program 7: program in Matlab to draw Arbitrary line using DDA algorithm.

```
%DDA algorithm-Arbitrary line

clc;
close all;
clear all
x1=input('Enter x-start value: ');
y1=input('Enter y-start value: ');
x2=input('Enter x-end value : ');
y2=input('Enter y-end value : ');
i=0;
dx=(x2-x1);
dy=(y2-y1);
if (abs(dx)>=abs(dy))
  length=abs(dx);
else
  length=abs(dy);
end
axis([0 100 0 100]);
hold on
xinc=dx/length;
yinc=dy/length;
x=x1; y=y1;
  while (i<=length)
  x=x+xinc;
          y=y+yinc;
          plot(x,y,'gs')
           i=i+1;

end
xlabel('x-axis')
ylabel('y-axis')
title(' program to draw line using DDA Algorithm')
```

**The output of program:**




Enter x-start value: 10

Enter y-start value:20

Enter x-end value : 40

Enter y-end value :80

## 2.5.3 Bresenham's Algorithm

This algorithm is designed on a very interesting feature of the DDA. An accurate and efficient raster line-generating algorithm, developed by Bresenham, scan converts lines using only incrementa1 integer calculations that can be adapted to display circles and other curves.

Figures 2.14 and 2.15 illustrate sections of a display screen where straight line segments are to be drawn. The vertical axes show-scan-line positions, and the horizontal axes identify pixel columns.

Sampling at unit ***x*** intervals in these examples, we need to decide which of two possible pixel positions is closer to the line path at each sample step. Starting from the left endpoint shown in Figure2.10**,** we need to determine at the next sample position whether to plot the pixel at position (11, 11) or the one at (11, 12). Similarly, Figure 2.15 shows-a negative slope-line path starting from the left endpoint at pixel position (50, 50). In this one, do we select the next pixel position as (51,50) or as (51,49)? These questions are answered with Bresenham's line algorithm by testing the sign of an integer parameter, whose value is proportional to the difference between the separations of the two pixel positions from the actual line path. To illustrate Bresenham's approach, we- first consider the scan-conversion process for lines with positive slope less than 1.

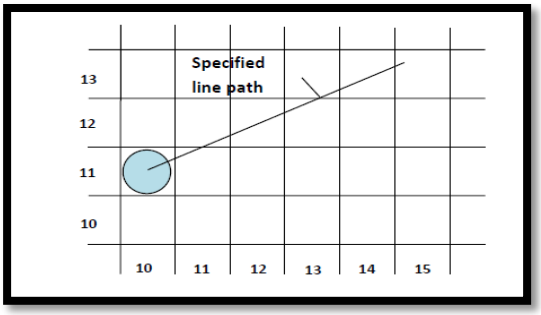



**Figure 2.14 Section of a display screen where a straight line segment 1s to be plotted, starting from the pixel at column 10 on scan Line 11**

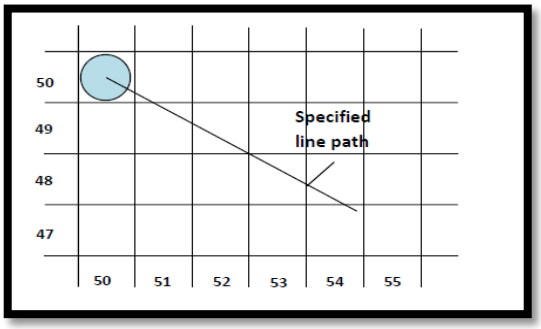



**Figure 2.15 Section of a display screen where a negative slope line segment 1s to be plotted, starting from the pixel at column 50 on scan line 50.**

Designed so that each iteration changes one of the coordinate's values by ± 1.

The other coordinate may or may not change depending on the value of an error term maintained by the algorithm.

The error term records the distance measured perpendicular to the axis of greatest movement between the exact path of the line and the actual dots generated.

**Algorithm of Bresenham's line (6).**

```
Input: x1,x2,y1,y2.
Output: Arbitrary line
{  x=x1 ;y=y1; dx=x   2-x1; dy=y2  -y1;
    e=dy/dx  -0.5  ;
    for i= 0 to abs(dx)
       { plot (x,y,color);
         While (e>=0)
               {
                 if y1>y2
                     { y=y -1;e=e -1;
                  else
                      y=y+1; e=e-1;
                     }
               }
          If x1>x2
           { x=x -1;
          else
             x=x+1;
            }
        e=e+dy/dx
       }
}
```

The steps of the algorithm are self-explanatory. After plotting each point, find the error involved, if it is greater than Zero, then in the next step, the next incremental point is to be plotted and error by error-1; else error remains the same and the point will not be incremented. In either case, the other coordinate will be incremented (In this case, it is presented that x - coordinate is

uniformly incremented at each stage, while y coordinate is either incremented or retained as such depending on the value of error).

The **advantages** of Bresenham Line Drawing Algorithm are-

1. It is easy to implement.
2. It is fast and incremental.
3. It executes fast but less faster than DDA Algorithm.
4. The points generated by this algorithm are more accurate than DDA Algorithm.
5. It uses fixed points only.

The **disadvantages** of Bresenham Line Drawing Algorithm are-

1. Though it improves the accuracy of generated points but still the resulted line is not smooth.
2. This algorithm is for the basic line drawing.
3. It can not handle diminishing jaggies.

**Example 8:** trace the line where the end points (50, 65), (59, 68) by Bresenham's algorithm.

**Solution:**

x1=50, y1=65

x2=59, y2=68

**dx= x2-x1**

59-50= 9

**dy = y2-y1**

= 68-65= 3

**m= dy/dx**

=3/9 = 0.333

**e=dy/dx-0.5**

=0.333 – 0.5

= -0.167

We generate point from (50,65) to (59,68) as show in table 2.8 Below:

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

**Table 2.8 point generation using Bresenham's Algorithm**

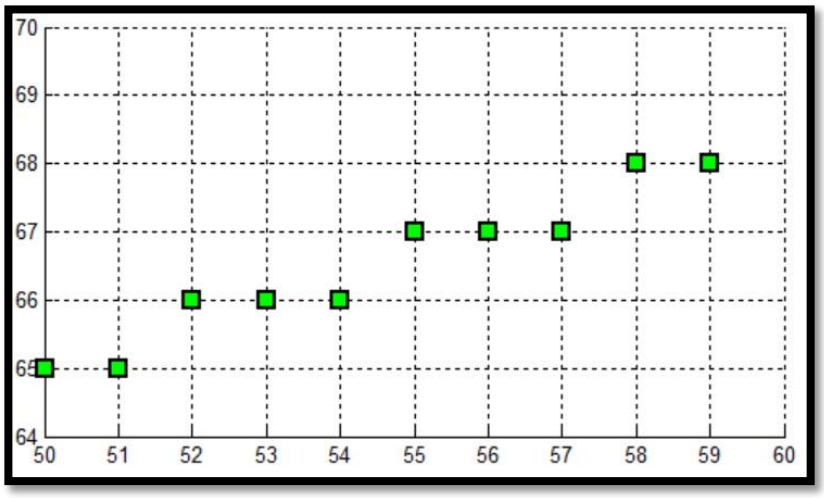



**Figure 2.16 Drawing line using Bresenham's Algorithm**

**Example 9:** Trace the line where the end points (7,3) , (15,7) by Bresenham's algorithm.

**Solution:**

**dx=(x2-x1)**

=(15-7)=8

**dy=(y2-y1)**

=(7-3)=4

**m=dy/dx**

=4/8=0.5

**e=m-0.5**

=0.5-0.5 =0

We generate point from (7,3) to (15,7) as show in table 2.9 Below:

| i | X | Y | e |
|---|---|---|---|
| 0 | 7 | 3 | 0 -1+m |
| 1 | 8 | 4 | -0.5 +m |
| 2 | 9 | 4 | 0 -1+m |
| 3 | 10 | 5 | -0.5 +m |
| 4 | 11 | 5 | 0 -1+m |
| 5 | 12 | 6 | -0.5 +m |
| 6 | 13 | 6 | 0 -1+m |
| 7 | 14 | 7 | -0.5 +m |
| 8 | 15 | 7 | 0 -1+m |

**Table 2.9 point generation using Bresenham's Algorithm**

A plot of the pixels generated along this line path is shown in figure (2.17)

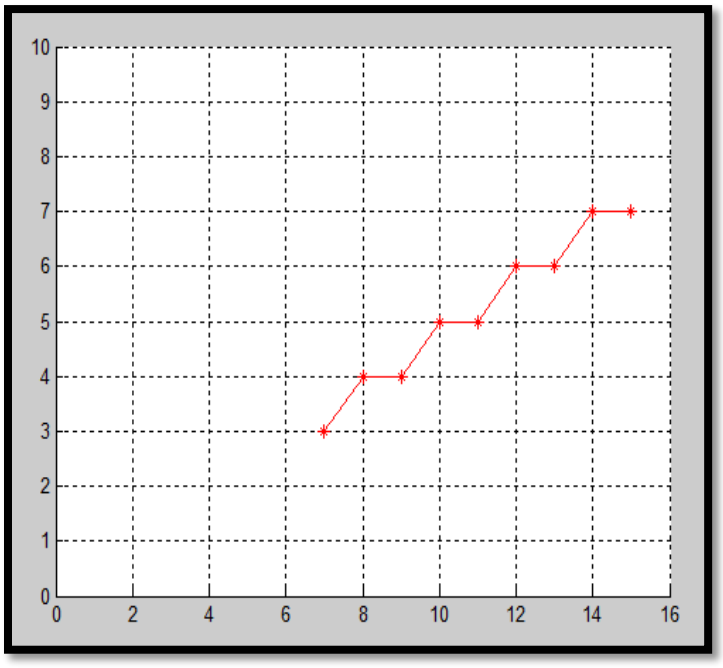



**Figure 2.17 Drawing line using Bresenham's Algorithm**

# Program 8: MATLAB program to draw Arbitrary line using Bresenham's method

```
% Bresenham's method- Arbitrary line
clc;
close all;
clear all;
x1=input('Enter x-start value: ');
y1=input('Enter y-start value: ');
x2=input('Enter x-end value : ');
y2=input('Enter y-end value : ');
x=x1;    y=y1;
dx=x2-x1;
dy=y2-y1;
e=(dy/dx)-0.5;
axis([0 100 0 100]); hold on
for i=0:abs(dx)
  plot(x,y,'gs');
  while (e>=0)
     if y1>y2
  y=y-1;
  e=e-1;
 else
  y=y+1;
  e=e-1;
 end
 if x1>x2
  x=x-1;
 else
 x=x+1;
 end
   e=e+(dy/dx);
 end
xlabel('x-axis')
ylabel('y-axis')
title(' program to draw line Bresenham's method ')
```

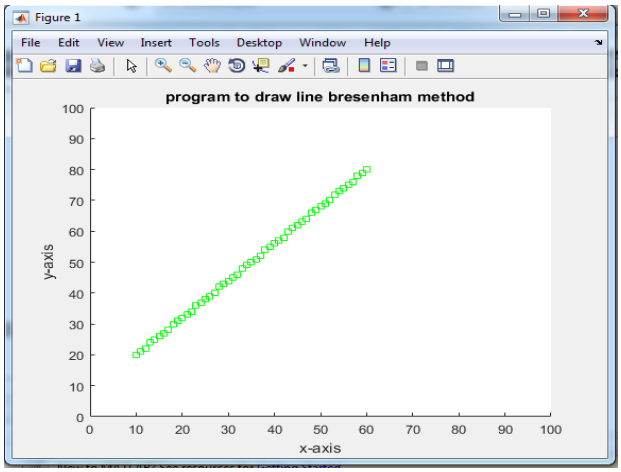
# The output of program:

Enter x-start value:10

Enter y-start value: 20

Enter x-end value : 60

Enter y-end value :80




## 2.6 Generation of Circles Algorithms

The circle is a special kind of curves. The circle is a closed curve with same starting and ending point. Circles are probably the most used curves in elementary graphics.

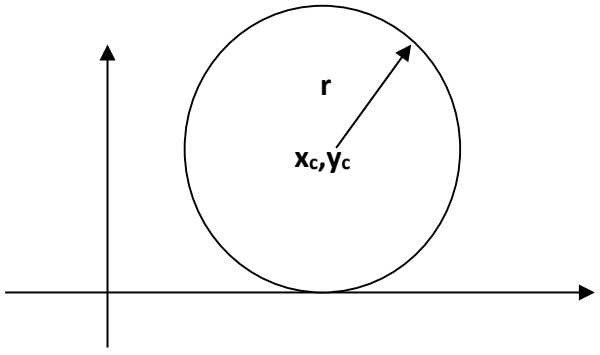



**Figure 2.18 Circle with center coordinates (xc, *yc*) and radius (r)**

A circle is specified by the coordinates of its center ( , $yc$) and its radius (r)

- The circle equation is : $(x-xc)^2 + (y-yc)^2 = r^2$ ……. (1)
- If the center of the circle is at the origin (0,0) then the equation is :

$$x^2+ y^2=r^2 \text{ …… (2)}$$

Solving equation (1) for y :

$$y=yc\pm\sqrt{r^2 + (x - xc)^2}$$

Note: To draw a circle increment the x values by one unit from – r to +r and use the above equation to solve for the two y values at each step.

### 2.6.1 Direct (implicit) Algorithm

In this method the first pixel of circle is at left side as equation x=xc-r y=yc

to draw the circle we can increment x from – r to +r or from 0 to 2r by one unit at each step and solving for y

$y=yc\pm\sqrt{r^2 + (x - xc)^2}$ , x=x+1

This method of drawing a circle is inefficient because:

1. We are not taking advantages of the symmetry of the circle.

2. The amount of processing time required to perform the squaring and square root operations repeatedly.
3. X values are equally spaced (they differ by one unit ) the y values are not. The circle is denes and flat near the y-axis and has large gaps and is steep near the x-axis.

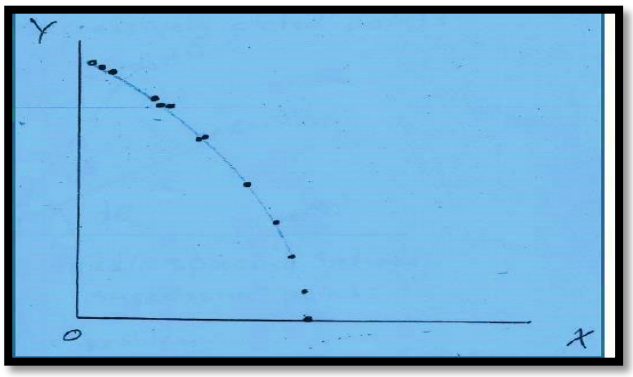



**Figure 2.19 value of x and y**

**Algorithm of Direct (implicit) Generation Circle (7)**

Input : xc ,yc , r.

Output : Circle

{ x=xc-r;

for i= 0 to 2*r

{ y=yc+ $\sqrt{r^2 - (x - x_c)2}$

plot (x, integer (y) ,color)

y=yc- $\sqrt{r^2 - (x - x_c)2}$

plot (x, integer (y),color)

x=x+1;

}

}

**Example 10**: Find the point of a circle where $xc$=10, $yc$= 10 and r=5 using direct algorithm?

**Solution:**

Xc=10

Yc=10

X=xc-r

X=10-5=5

For i=0:2*r

Y=yc+sqrt((r^2)-(x-xc)^2)

Plot(x,round(y),'.y')

Y=yc-sqrt((r^2)-(x-xc)^2)

Plot(x,round(y),'.y')

X=x+1 ;End

We generate points to draw circle using direct algorithm, see table 2.10.

| X | Y | Round(y) | Y | Round(y) | Plot(X,Y) |
|---|---|---|---|---|---|
| 5 | 10 | 10 | 10 | 10 | (5,10),(5,10) |
| 6 | 13 | 13 | 7 | 7 | (6,13),(6,7) |
| 7 | 14 | 14 | 6 | 6 | (7,14),(7,6) |
| 8 | 14.5 | 15 | 5.4 | 5 | (8,15),(8,5) |
| 9 | 14.8 | 15 | 5.1 | 5 | (9,15),(9,5) |
| 10 | 15 | 15 | 5 | 5 | (10,15),(10,5) |
| 11 | 14.8 | 15 | 5.1 | 5 | (11,15),(11,5) |
| 12 | 14.5 | 15 | 5.4 | 5 | (12,15),(12,5) |
| 13 | 14 | 14 | 6 | 6 | (13,14),(13,6) |
| 14 | 13 | 13 | 7 | 7 | (14,13),(14,7) |
| 15 | 10 | 10 | 10 | 10 | (15,10),(15,10) |

**Table 2.10 Points Generation to Draw Circle using Direct Algorithm**

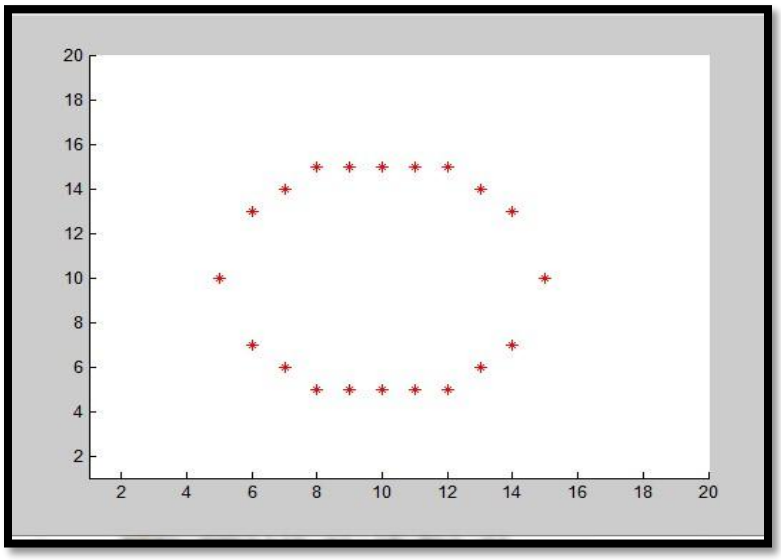



**Figure 2.20 Drawing circle using Direct Algorithm**

# Program 9 : MATLAB program to draw circle using Direct Method

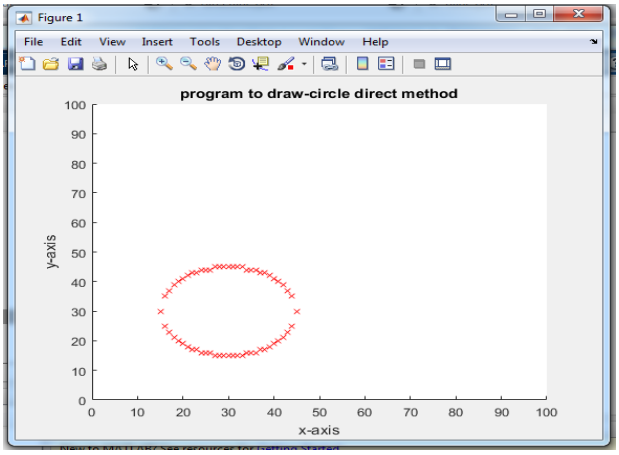
```
clc;
clear all;
close all
xc=input('enter x-value :  ');
yc=input('enter y-value :  ');
r=input('enter radius -value :  ');
x=xc-r;
axis ([0 100 0 100])
hold on
 for i=0:2*r
   y=yc+sqrt(power(r,2)-power((x-xc),2));
   plot(x, round(y),'rx')
   y=yc-sqrt(power(r,2)-power((x-xc),2));
   plot(x, round(y),'rx')
   x=x+1;
end
 xlabel('x-axis')
 ylabel('y-axis')
 title(' program to draw -circle direct method ')
```

## The output of program:

enter x-value : 30

enter y-value : 30

enter radius -value: 15




### 2.6.2 Parametric (polar) Algorithm

One method of eliminating the problem of plotting points evenly spaced around the circle is to use polar representation of a circle:

$$x=xc+r\cos\Theta,$$

$$y = y+r \sin\Theta.$$

Where: $\Theta \rightarrow$ is measured in radians from 0 to $2\pi$ arc length= r x $\Theta$, r=radius (constant) in this method we depend on angles to draw the circle, since it propose the first angle th=0, and end angle is two_pi (360).

The change in angle (dth) must be small value dth=1/r.

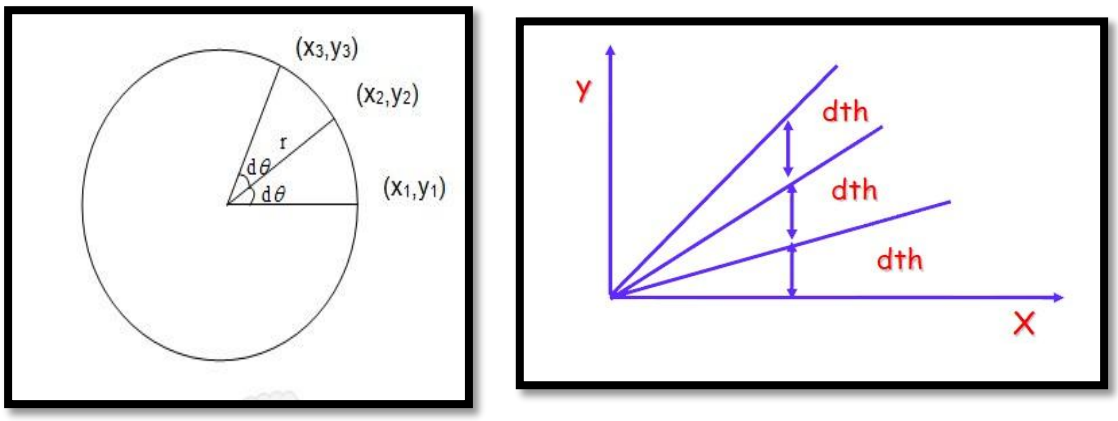



**Figure 2.21 show the angle Ɵ and dth**

The algorithm is shown below:

**Algorithm of Parametric (Polar) Generation Circle(8)**

Input : xc ,yc , r.

Output : Circle

```
{ th=0;
dth=1/r;
while (th<=2*pi)
{ x = xc + r *cos (th)
y = yc + r sin(th)
plot (integer(x),integer(y),color)
th = th + dth; }
}
```

Note: the algorithm use cos & sin operation and do not take the advantage of symmetric in circle

**Example 11:**Find the point of a circle where $xc$=10, $yc$= 10 and r=5 using polar algorithm ?

**Solution:**

Th=0

Dth=1/r=1/5

While th <=2*pi X=xc+r*cos(th)

Y=yc+r*sin(th)

Plot(round(x),round(y),'.k')

Th=th+dth

End

Below the table 2.11 show the generation points to draw circle using polar algorithm .

| X | Round(x) | Y | Round(y) | Th | plot(x,y) |
|---|---|---|---|---|---|
| 15 | 15 | 10 | 10 | 0.2 | (15,10) |
| 14.9 | 15 | 10.9 | 11 | 0.4 | (15,11) |
| 14.6 | 15 | 11.9 | 12 | 0.6 | (15,12) |
| 14.1 | 14 | 12.8 | 13 | 0.8 | (14,13) |
| 13.4 | 13 | 13.5 | 14 | 1 | (13,14) |
| 12.7 | 13 | 14.2 | 14 | 1.2 | (13,14) |
| 11.8 | 12 | 14.6 | 15 | 1.4 | (12,15) |
| : | : | : | : | : | : |
| : | : | : | : | : | : |
| 14.9 | 15 | 9.5 | 10 | 6.4 | (15,10) |

**Table 2.11 Generation Points to Draw Circle using Polar Algorithm**

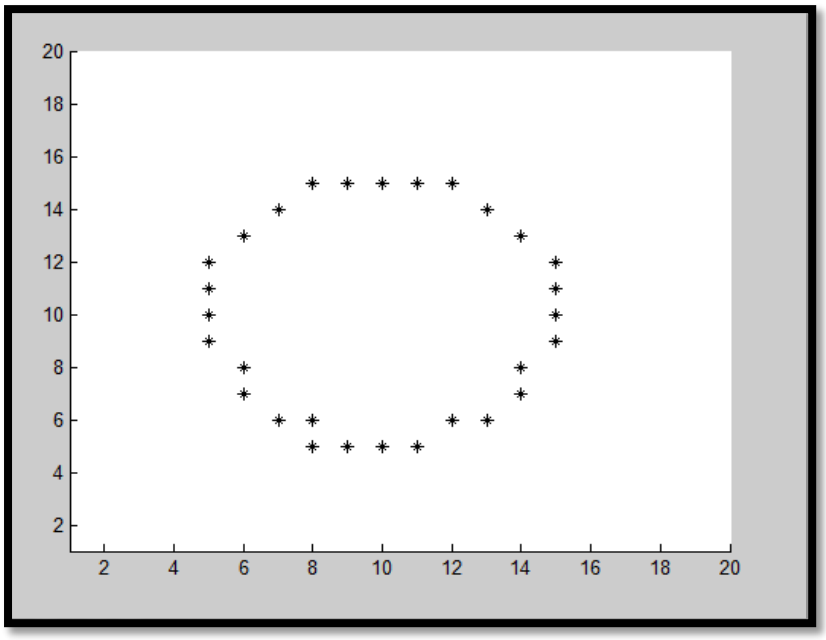



**Figure 2.22 Drawing circle using polar algorithm**

## Program 10 : MATLAB program to draw circle using parametric (polar) algorithm.

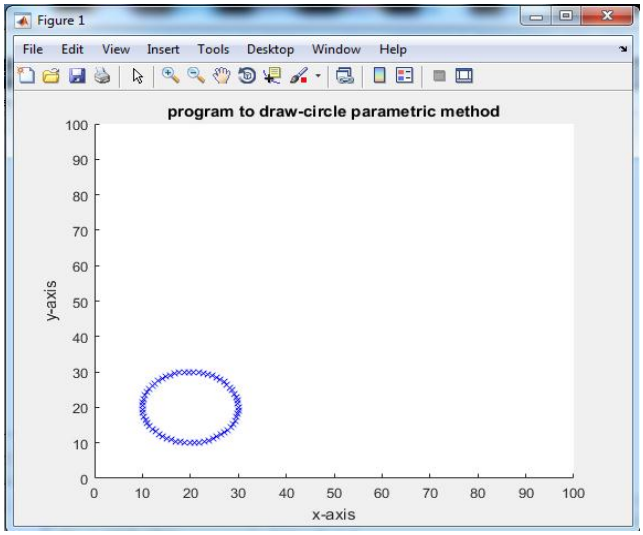
```
%parametric method-circle
clc;   clear all;   close all
xc=input('enter x-value : ');
yc=input('enter y-value : ');
r=input('enter radius -value : ');
th=0;       dth=1/r;
axis ([0 100 0 100])
hold on
while th<=2*pi
x=xc+r*cos(th);
y=yc+r*sin(th);
th=th+dth
plot(x,y,'--bx')
        end

xlabel('x-axis')
ylabel('y-axis')
title(' program to draw- circle parametric method ')
```

**The output of program:**

enter x-value :20

enter y-value :20

enter radius -value :10




## 2.7 The Symmetry of Circle

A circle is a geometric figure, which is round, and can be divided into 360 degrees. A circle is a symmetrical figure, which follows 8-way symmetry.

**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. Therefore, it is sufficient to compute only 1/8 of a circle, and all the other points can be computed from it.

**Therefore,** Computation can be reduced by considering the symmetry of circle.

Circle sections in adjacent octants within one quadrant are symmetric with respect to the 45' line dividing the two octants figure 2. 20 Show the symmetric of circle.

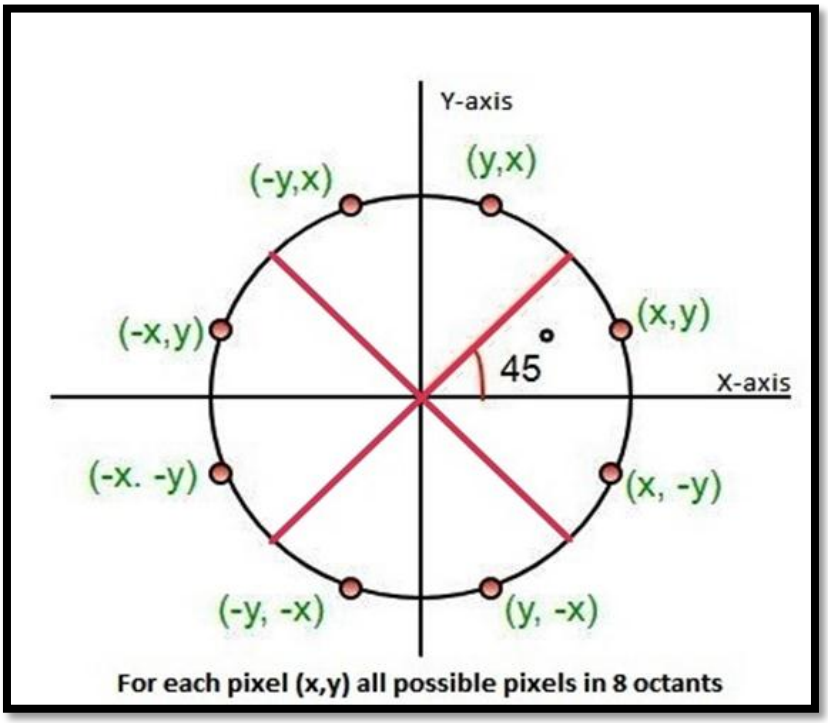



**Figure 2.20 Symmetric of Circle.**

## 2.7.1 Incremental Algorithm

One method to eliminating the problem of using polar representation to draw a circle is that repeated calculation of values from cosθ and sinθ that consumes of significant amount of processing time. we can speed up the computation by using incremental method that calculate the points on a circle from the coordinates of the previous calculated point, this technique requires only an initial calculation of the sin and cos.

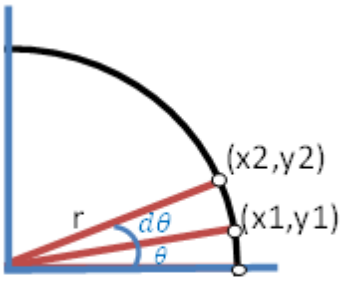
This method proposed the center of circle at origin point (0,0), so the first pixel in the circle is (0, r).

Two consecutive points (x1, y1) and (x2, y2) on a circle are related by:

$x1=r*(\theta)$ …. (1) $y1=r*(\theta)$ …. (2)

$x2=r*(\theta+d\theta)$ …. (3) $y2=r*(\theta+d\theta)$ …. (4)

$d\theta$: is a fixed angular step size.




$$cos(A+B)=cosA\ cosB-sinA\ sinB$$

$$sin(A+B)=sinA\ cosB+cosAsinB$$

using trigonometry, we get:

$$x2=r*(\theta)*cos(d\theta)-r*sin(\theta)*sin(d\theta) \ldots. (5)$$

$$y2=r*(\theta)*cos(d\theta)+r*cos(\theta)*sin(d\theta) \ldots. (6)$$

Substituting x1 and y1 at last two equations we get:

$$x2=x1*(d\theta)-y1*sin(d\theta) \ldots. (7)$$

$$y2=y1*(d\theta)+x1*sin(d\theta) \ldots. (8)$$

These are the incremental equation we want to generate a circle starting the circle at x1=0 , y1=r and fixed-angle increment dθ. we can compute all points on the circle by calculating cos dθ and sin dθ only once.

**Algorithm of Incremental Generation Circle (9)**

```
Input: xc, yc, r.
Output: circle
{
        dth=1/r, x=0, y=r
        ct=cos(dth), st=sin(dth)
        while (x<=y)
        {
                plot (round (xc+x), round (yc+y), color)
                plot (round (xc+x), round (yc-y), color)
                plot (round (xc-x), round (yc+y), color)
                plot (round (xc-x), round (yc-y), color)
                plot (round (xc+y), round (yc+x), color)
                plot (round (xc+y), round (yc-x), color)
                plot (round (xc-y), round (yc+x), color)
                plot (round (xc-y), round (yc-x), color)

                xtemp = x;
                x = x *ct - y * st;
                y =xtemp *st+ y * ct;
        }
}
```

# Program 11: MATLAB program to draw circle using Incremental algorithm

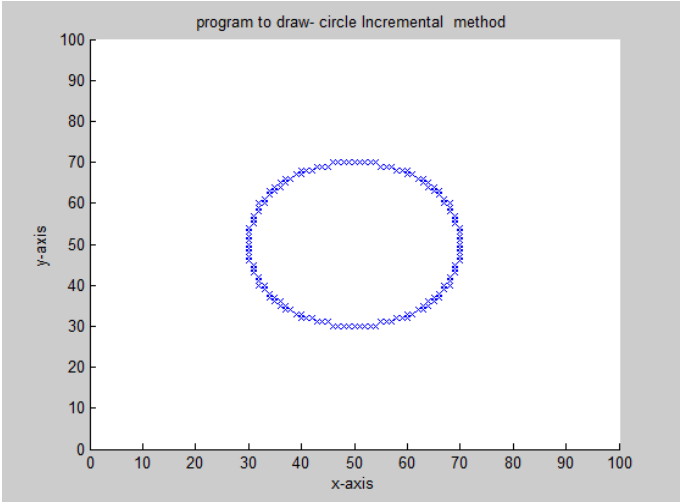
```
%Incremental method-circle
clc;clear all;close all
xc=input ('enter x-value:  ');
yc=input ('enter y-value:  ');
r=input ('enter radius -value:  ');
axis ([0 100 0 100])
hold on
d=1/r;
x=0;
y=r;
cd =cos(d);
sd=sin(d);
while (x<=y)
   plot(round(xc+x),round(yc+y),'bx')
   plot(round(xc+x),round(yc-y),'bx')
   plot(round(xc-x),round(yc+y),'bx')
   plot(round(xc-x),round(yc-y),'bx')
   plot(round(xc+y),round(yc+x),'bx')
   plot(round(xc+y),round(yc-x),'bx')
   plot(round(xc-y),round(yc+x),'bx')
   plot(round(xc-y),round(yc-x),'bx')
   xtemp=x;
   x=x*cd-y*sd;
   y=y*cd+xtemp*sd;
end
xlabel('x-axis')
ylabel('y-axis')
title(' program to draw- circle symmetric method  ')
```

**The output of program:**

enter x-value :50

enter y-value :50

enter radius -value :20




## 2.7.2 Bresenham's Circle Algorithm:

The values of a circle centered at the origin are computed in a 45 sector from x=0 to x=y the remaining seven sectors are obtained from the eight-point symmetry of the circle.

They value for this sector decrease as the x values increase if (0, r) is the starting point of the algorithm, then as x increase by one unit the y value either remains the same or is decrease by one unit. If (x, y) is a pixel on the circle, the next pixel is either A or B.

A :( x+1, y); to the right of previous point

B: (x+1, y-1); down and to the right of the previous point.

The algorithm proceeds to choose Pixel A or B by finding and comparing the distance from each pixel to the point on the circle that has x value of (x+1) these distances measure how far from the

circle each pixel is the pixel with the smallest distances the best approximation on the circle. The square of the distance of pixel A from the center of the circle is:

$$(x+1)^2 + y^2.$$

The difference between this squared distance and the squared distance to the closest point on the circle is: ***d(A)= (x+1) ^2 +y^2 -r^2***

for pixel B the distance is: ***d(B)= (x+1) ^2 + (y-1) ^2 – r^2***.

$$d \begin{cases} < 0 \; if \; (x, y) \; is \; inside \; th \; e \; circle \; boundary \\ = 0 \; if \; (x, y) \; is \; on \; th \; e \; circle \; boundary \\ > 0 \; if \; (x, y) \; is \; outside \; th \; e \; circle \; boundary \end{cases}$$

A point lies on the circle if its d value equals 0, in order to determine which pixel A or B has the smallest d value we examine the sum

**S=d(A)+d(B);**

- There are three cases to be consider:

1.If the circle goes between A and B then

d(A) > 0 and d(B) <0

If abs (d(A)) > abs (d(B)) → choose B , such that S> 0

If abs (d(A)) <= abs(d(B)) → choose A , such that S<= 0

2.If the circle goes through or above A then

d(A)<=0 and d(B)<0 → choose A, such that S<0

3.If the circle goes through B or below then

d(A) > 0 and d(B) >=0 → choose B , such that S>0

figure 2.21 show closest point on the circle.

Rule: if S>0 we choose pixel B otherwise we choose pixel A.

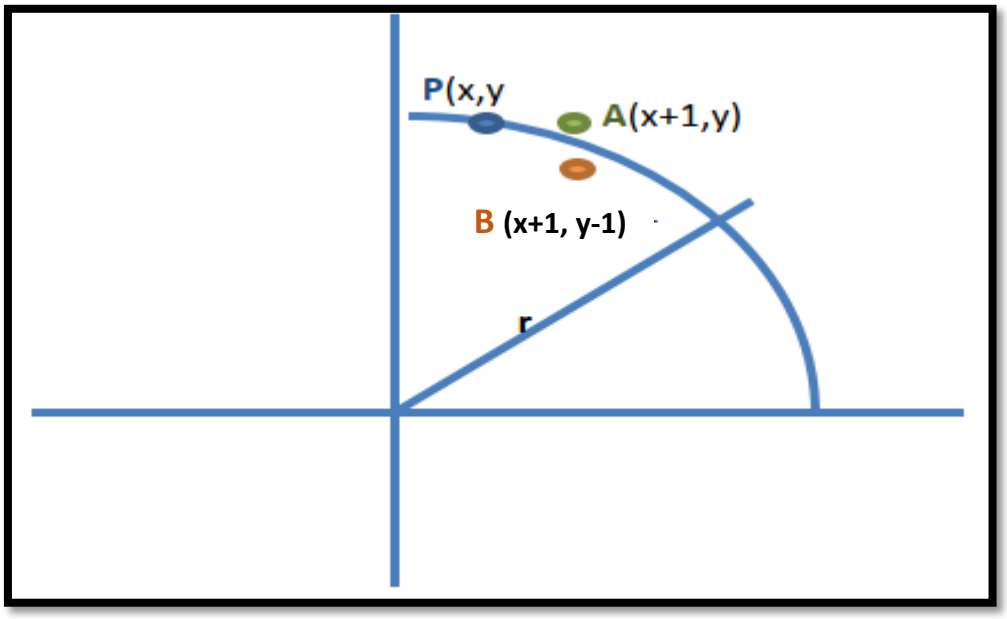



**Figure 2.21 Closest Point on the Circle**

**Algorithm of Bresenham Circle (10)**

```
Input : xc,yc,r.
Output : circle
{
        x=0; y=r;
        While(x<=y)
        {
            plot (round (xc+x) , round (yc+y) ,color)
            plot (round (xc+x) , round (yc-y) ,color )
            plot (round (xc-x) , round (yc+y) ,color )
            plot (round (xc-x) , round (yc-y) ,color )
            plot (round (xc+y) , round (yc+x) ,color )
            plot (round (xc+y) , round (yc-x) ,color )
            plot (round (xc-y) , round (yc+x) ,color )
            plot (round (xc-y) , round (yc-x) ,color )
            da= (x+1)^2+(y)^2-(r)^2
            db= (x+1)^2+ (y-1)^2 -(r)^2
            s= da + db
            if (s>0)
              { y=y-1  ; }
            x=x+1;
        }
}
```

The **advantages** of Bresenham Circle Drawing Algorithm are-

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

The **disadvantages** of Bresenham Circle Drawing Algorithm are-

1. Like Mid Point Algorithm, accuracy of the generating points is an issue in this algorithm.
2. This algorithm suffers when used to generate complex and high graphical images.
3. There is no significant enhancement with respect to performance.

**Example 12:** Trace the Bresenham algorithm to generate six points of the circle centered at (300,150) with a radius equal to 9 unit.

**Solution:**

da = (x+1) ^2 + y^2 – R^2 , db= (x+1) ^2 + (y-1) ^2 – R^2, R^2=81

| x | y | x+xc | y+yc | d(A) | d(B) | s |
|---|---|---|---|---|---|---|
| 0 | 9 | 300 | 159 | 1 | -16 | -15 |
| 1 | 9 | 301 | 159 | 4 | -13 | -9 |
| 2 | 9 | 302 | 159 | 9 | -8 | 1 |
| 3 | 8 | 303 | 158 | -1 | -16 | -17 |
| 4 | 8 | 304 | 158 | 8 | -7 | 1 |
| 5 | 7 | 305 | 157 | 4 | -9 | -5 |

**Example 13** :Trace the Bresenham algorithm to generate six points of the circle centered at (120,100) with a radius equal to 8 unit.

**Solution:**

xc=120 , yc=100 , r=8 , x=0 , y=r=8 ;

da = (x+1) ^2 + y^2 – r^2 , db= (x+1) ^2 + (y-1) ^2 – r^2

r^2=64 ; s=d(A)+d(B)

| x | y | x+xc | y+yc | d(A) | d(B) | s |
|---|---|---|---|---|---|---|
| 0 | 8 | 120 | 108 | 1 | -14 | -13 |
| 1 | 8 | 121 | 108 | 4 | -11 | -7 |
| 2 | 8 | 122 | 108 | 9 | -6 | 3 |
| 3 | 7 | 123 | 107 | 1 | -12 | -11 |
| 4 | 7 | 124 | 107 | 10 | -3 | 7 |
| 5 | 6 | 125 | 106 | 8 | -3 | 5 |

# Program 12: MATLAB program to draw circle using Bresenham method

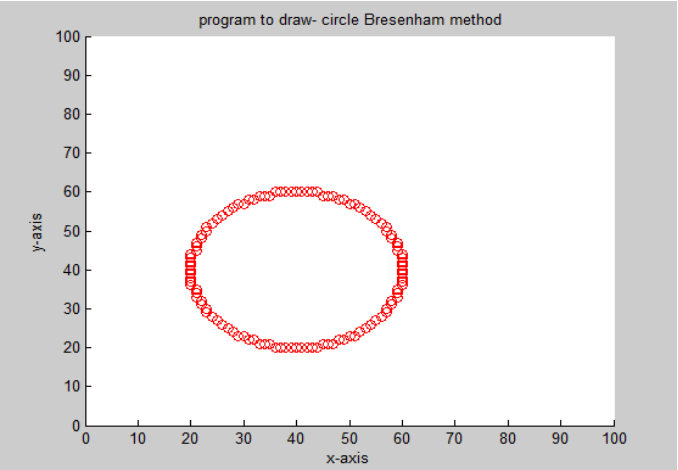
```
%Bresenham method-circle
clc;
clear all;
close all;
xc=input('enter xc-value :  ');
yc=input('enter yc-value :  ');
r=input('enter radius -value :  ');
axis ([0 100 0 100]);
hold on
x=0;y=r;
while(x<=y)
      plot(round(xc+x),round(yc+y),'ro')
      plot(round(xc+x),round(yc-y),'ro')
      plot(round(xc-x),round(yc+y),'ro')
      plot(round(xc-x),round(yc-y),'ro')
      plot(round(xc+y),round(yc+x),'ro')
      plot(round(xc+y),round(yc-x),'ro')
      plot(round(xc-y),round(yc+x),'ro')
      plot(round(xc-y),round(yc-x),'ro')
      da= (x+1)^2+(y)^2-(r)^2;
      db= (x+1)^2+ (y-1)^2 -(r)^2;
      s= da + db;
      if (s>0)
         y=y-1;
      end
      x=x+1;
end
xlabel('x-axis')
ylabel('y-axis')
title(' program to draw- circle Bresenham method')
```

## The output of program:

enter x-value :40

enter y-value :40

enter radius -value :20




## 2.7.3 Midpoint Circle Algorithm:

Bresenham's line algorithm for raster displays is adapted to circle generation by setting up decision parameters (P) for finding the closest pixel to the circumference at each step.

For a given radius r and screen center position ( xc,yc), we can first set up our algorithm to calculate pixel positions around a circle path centered at the coordinate origin (0,0). So first pixel is (0,r)

each calculated position (x,y) is moved to its proper screen position by adding xc to x and yc to y.

• we compute the first octant pixels from x=0 to x=y.

• Positions for the other seven octants are then obtained by symmetry,

•Similarly to the case with lines, there is an incremental algorithm for drawing circles –the mid-point circle algorithm.

•In the mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points .

• we can take unit steps in the positive x direction over octant and use a decision parameter to determine which of the two possible y positions is closer to the circle path at each step.

$$r^2 = x^2 + y^2$$
$$P = (x+1)^2 + (y-1)^2 - r^2$$
$$p = 2(1-r)$$

• Any point ( x , y) on the boundary of the circle with radius r satisfies the equation p = 0.

• If the point is in the interior of the circle, P is negative value.

• if the point is outside the circle, P is positive.

$$P\begin{cases} < 0 \ if \ (x,y) \ is \ inside \ the \ circle \ boundary \\ = 0 \ if \ (x,y) \quad is \ on \ the \ circle \ boundary \\ > 0 \ if \ (x,y) \ is \ outside \ the \ circle \ boundary \end{cases}$$

• We need to determine whether the pixel at position (x+ 1, y) or the one at position (x+1,y-1) is closer to the circle. So If p<0,the point is inside the circle and the pixel (x+1,y) is closer to the circle boundary.

Otherwise, the point is outside or on the circle boundary and we select the pixel (x+1,y-1 ).

$$P\begin{cases} < 0 \quad (x+1,y) \\ \qquad p = p + 2x + 1 \\ \geq 0 \ (x+1,y-1) \\ \qquad p = p + 2(x-y) + 1 \end{cases}$$

**Algorithm of Midpoint Circle (11)**

```
Input : xc,yc,r.

Output : circle

{
   x=0; y=r;
 p=2*(1-r);
   while (x<y)

 {

       plot (round (xc+x) , round (yc+y) ,color)

       plot (round (xc+x) , round (yc-y) ,color)

       plot (round (xc-x) , round (yc+y) ,color )

       plot (round (xc-x) , round (yc-y) ,color)

       plot (round (xc+y) , round (yc+x) ,color )

       plot (round (xc+y) , round (yc-x) ,color )

       plot (round (xc-y) , round (yc+x) ,color )

       plot (round (xc-y) , round (yc-x) ,color )

       x=x+1;

       if p<0

       p=p+(2*x)+1;
       else
              y=y-1;
       p=p+2*(x-y)+1;
       end

    }

 }
```

## Advantages Mid-point Circle Algorithm:

1. The mid-point circle algorithm is an efficient algorithm in drawing a circle.
2. The implementation of the algorithm is easy from the programmer's point of view.

## Disadvantage Mid-point Circle Algorithm:

1. The time consumption of this algorithm is high.

**Example 14** : Trace the Midpoint algorithm to generate six points of the circle centered at (120,130) with a radius equal to 7 unit.

## Solution:

p=2(1-r)=2(1-7)=-12

If p<0 p=p+2*x+1

If p>=0 p=p+2(x-y)+1

| X | y | P | x+xc | y+yc |
|---|---|---|---|---|
| 0 | 7 | -12 | 120 | 137 |
| 1 | 7 | -9 | 121 | 137 |
| 2 | 7 | -4 | 122 | 137 |
| 3 | 7 | 3 | 123 | 137 |
| 4 | 6 | 0 | 124 | 136 |
| 5 | 5 | 1 | 125 | 135 |

# Program 13: MATLAB program to draw circle using Midpoint-Method Circle

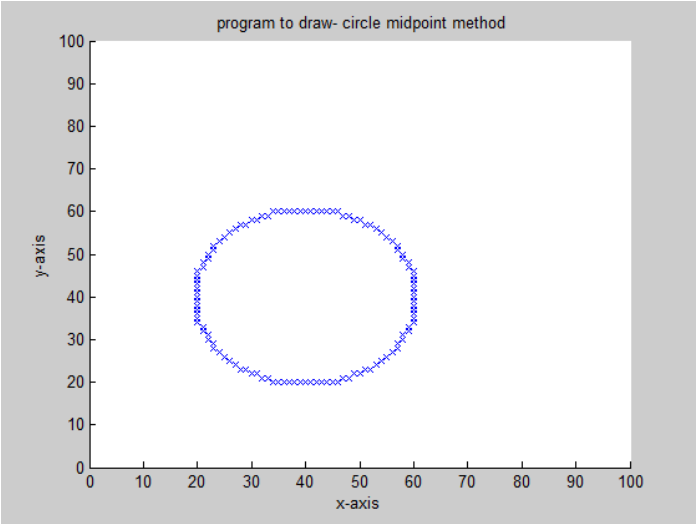
```
%midpoint circle
clc;
clear all;
close all;
xc=input('enter xc-value :  ');
yc=input('enter yc-value :  ');
r=input('enter radius -value :  ');
axis ([0 100 0 100]);
k=0; hold on
x = 0 ; y = r ; p = 2*(1-r);
while x <= y
      plot(round(xc+x),round(yc+y),'bx')
      plot(round(xc+x),round(yc-y),'bx')
      plot(round(xc-x),round(yc+y),'bx')
      plot(round(xc-x),round(yc-y),'bx')
      plot(round(xc+y),round(yc+x),'bx')
      plot(round(xc+y),round(yc-x),'bx')
      plot(round(xc-y),round(yc+x),'bx')
      plot(round(xc-y),round(yc-x),'bx')
      x = x + 1;
      if    p < 0
      p = p + 2 * x + 1;
      else
      y = y - 1 ;
      p = p + 2 * (x - y) + 1;
      end
end
xlabel('x-axis')
ylabel('y-axis')
title(' program to draw- circle midpoint method')
```

**The output of program :**

enter x-value :40

enter y-value :40

enter radius -value :20




## 2.8 Ellipses Drawing

An ellipse is a set of points such that the sum of the distances from two fixed positions is the same for all points. It is a variation of a circle. Stretching a circle in one direction produce an ellipse.

Following algorithms for ellipse drawing which stretched in the x or y direction. Figure 2.22 show an ellipse with center point (xc,yc).

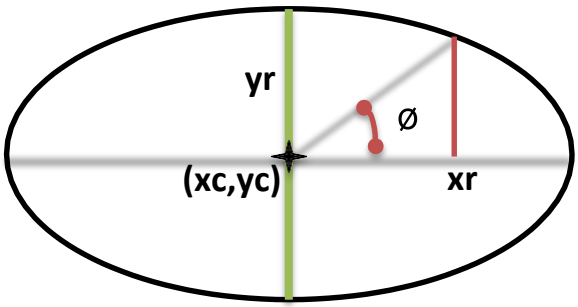



**Figure 2.22 Ellipse**

The values of (xr) and (yr) effect the shape of the ellipse.

- If yr > xr the ellipse is longer in the y direction
- If yr < xr the ellipse is longer in the x direction.
- If xr = yr the equation produces a circle.
- The ellipse can be drawn using four-points symmetry. figure 2.23 show symmetry of an ellipse.

If (x,y) lies on the ellipse so do the points (-x,y), (x,-y) and (-x,-y).

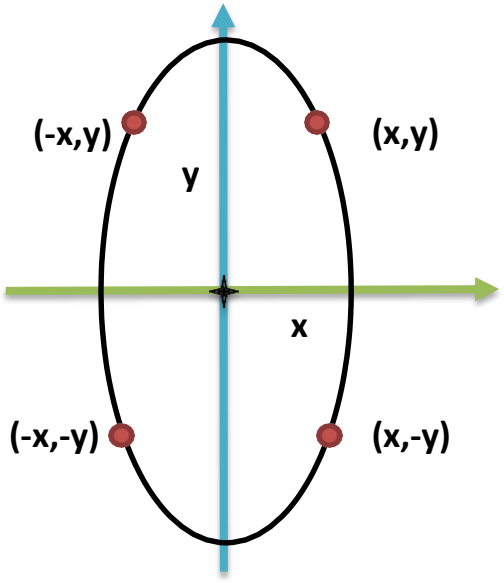



**Figure 2.23 Symmetry of an Ellipse**

There are some algorithms to draw and representation of an ellipse as follow:

## 2.8.1 The Polar Representation of An Ellipse

The polar equation for an ellipse centered at (xc, yc) and xr is the radius on the x-axis and yr is the radius on the y-axis.

The polar equation for this type of ellipse centered at ($xc$, $yc$) are:

x= $xc$+xr*cos($\theta$)
y=$yc$ +yr*sin($\theta$) ..... (1)

$\theta$ values between 0 and 2π radius

**Algorithm of Polar representation of an Ellipse (12)**

```
Input : xc,yc,xr,yr.
Output: Ellipse .
  {
        dth=1/ ((xr +yr)/2) ;
        th=0 ;
       pi= 3.1416;
       While (th<=2*pi)
          {
           x= xr*cos(th) ; y= yr*sin(th) ;
            plot (integer (xc+x), integer(yc +y),color);
            plot (integer (xc-x), integer (yc+y), color);
            plot (integer (xc+x), integer (yc-y), color);
            plot (integer (xc-x), integer (yc-y), color) ;
            th=th+dth ;
   }
   }
```

**Example 15 :**Trace the algorithm that use the polar representation to generate six points of the ellipse centered at (300, 150) with xr=10, yr=5.

**Solution:**

dth=1/((xr+yr)/2) ; 2*pi=6.2832

=1/((10+5)/2)

=0.133

| x=xr*cos(th) | Plot(x) | y=yr*sin(th) | Plot(y) | x+xc | y+yc | th |
|---|---|---|---|---|---|---|
| 10 | 10 | 0 | 0 | 310 | 150 | 0 |
| 9.9 | 10 | 0.6 | 1 | 310 | 151 | 0.1333 |
| 9.6 | 10 | 1.3 | 1 | 310 | 151 | 0.2667 |
| 9.2 | 9 | 1.9 | 2 | 309 | 152 | 0.4000 |
| 8.6 | 9 | 2.5 | 3 | 309 | 153 | 0.5333 |
| 7.8 | 8 | 3.0 | 3 | 308 | 153 | 0.6667 |

## 2.8.2 Incremental method to drawing of Ellipse

Similar of discuss in the circle, but differ for equations:

The following incremental equations for an ellipse are derived from equations (1)

$$Xn+1 = Xn \cos(\Delta\Theta) - (Xr/Yr) * Yn \sin(\Delta\Theta) \ldots\ldots (2)$$

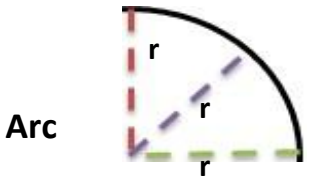
$$Yn+1 = Yn \cos(\Delta\Theta) + (Yr/Xr) * Xn \sin(\Delta\Theta) \ldots\ldots (3)$$

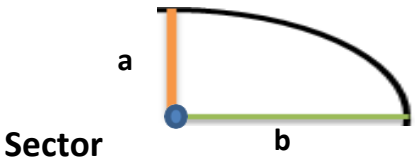
This algorithm is modify the polar representation (leave that of students to write this algorithm)

## 2.9 Arc and Sector

**Arc:** is part of circle but is needed start angle & end angle and distance of center of arc to surrounding is same as.

**Sector**: is part of ellipse but contains two line; line1 from center to start angle & other line from center to end angle & distance between center of sector to surrounding is different as.



## 2.9.1 Scan-converting ARCs and Sectors

1. ARC can be generated by:

### A.Polynomial method

Value of x is varied from $x1$ to $x2$ and the values of y are found be evaluating the expression $\sqrt{r2+\sqrt{x2}}$. An arc would be appearing to be nothing more than portions of circles.

### B. Trigonometric method

The starting value is set equal to $\theta 1$ and the ending value is set equal to $\theta 2$. The rest of the steps are similar to those used when scan-converting a circle except that symmetry is not used.

2. Sectors

A sector is scan-converting an arc and then scan-converting two-lines from the center of the Arc to the endpoints of the Arc.

**Example 16:** Assume that a sector whose center is at point (h,k) is to be scan-converted.

- First scan-convert an Arc from $\theta 1$ to $\theta 2$ .
- Next a line would be scan-converted from (h,k) to (r cos($\theta 1$)+h, r sin($\theta 1$)+k).
- A second line would be scan-converted from (h,k) to (r cos($\theta 2$)+h, r sin($\theta 2$)+k).

## 2.10 Scan-converting a Rectangle

Rectangle sides are parallel to the coordinates axes may be constructed if the locations of two vertices are known. The remaining corner points are then derived. if we have the following rectangle in figure 2.24.

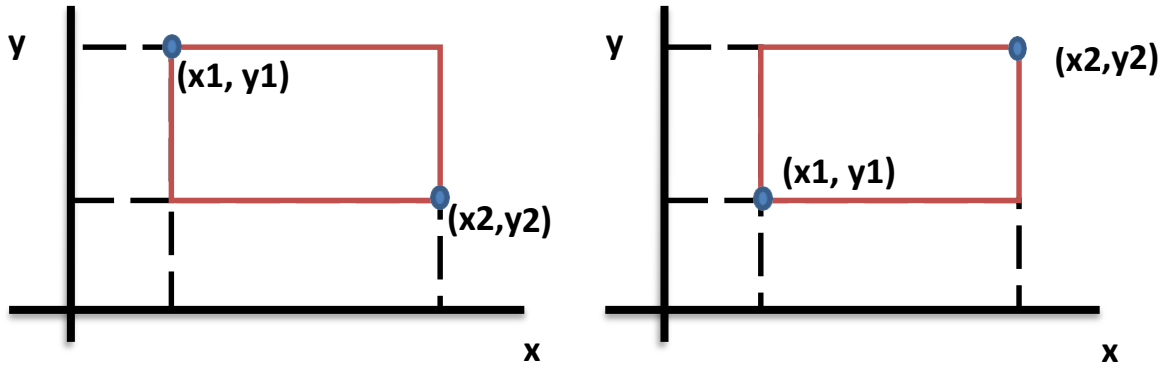



**Figure 2.24 rectangle with two vertices**

The lines would be drawn as follows:

plot ([$x1$ $x1$] , [$y1$ $y2$] ,color)
plot ([$x1$ $x2$] , [$y2$ $y2$] ,color)
plot ([$x2$ $x2$] , [$y2$ $y1$] ,color)

**OR**

Line ([$x1$ $x1$] , [$y1$ $y2$] , color)
Line ([$x1$ $x2$] , [$y2$ $y2$] , color)
Line ([$x2$ $x2$] , [$y2$ $y1$] , color)

# CHAPTER THREE

# 2D TRANSFORMATION & VIEWING

# *Chapter Three*

## *2D Transformation & Viewing*

- **Introduction To Transformation**
- **Types of Transformation**
  - **Translation**
  - **Scaling**
  - **Rotation**
  - **Reflection**
  - **Shear**
- **Matrix Representation of Transformation**
- **2D Viewing**
- **Window to Viewport Transformation ( Mapping)**
- **Window to Viewport Transformation N**
- **Clipping and windowing**
  - **Clipping window**
  - **Point clipping**
  - **Line clipping**
  - **The Cohen–Sutherland algorithm**
  - **Intersection points**

# Chapter Three

# 2D Transformation & Viewing

## 3.1 Introducton To Transformation

One of the most common and important tasks in computer graphics is to transform (changing) the coordinates (position, orientation, size and shape) of either object within the graphical scene or the camera that is viewing the scene. It is also frequently necessary to transform coordinates from one coordinate system to another, (e.g. world coordinates to viewpoint coordinates to screen coordinates). All of these transformations can be efficiently and sufficiently handled using some simple matrix representations, which we will see can be particularly useful for combining multiple transformations into a single composite transform matrix.

The **advantage** of used the transformation:

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

## 3.2 Types of Transformations

Three basic types of transformations that can perform in two dimensions:

a. Translation (shift OR move).
b. Scaling.
c. Rotation

These basic **transformations** can also be combined to obtain more complex transformations.

Some package provides few additional transformations which are useful in certain application. Two such transformation are **Reflection** and **Shear**.

### 3.2.1 Translation

Translation is a transformation that moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate $(tx, ty)$ to the original coordinate (X, Y) to get the new coordinate (X', Y').Figure 3.1 show the translation and **Mathematically** this can be represented as:

$$X' = X + tx \quad \& \quad Y' = Y + ty$$

**Note**: Using coordinate system the translating factor are

If $tx$ >0 then point moves to the right.

If $tx$ <0 then point moves to the left.

If $ty$ >0 then point moves to the up.

If $ty$ <0 then point moves to the down.

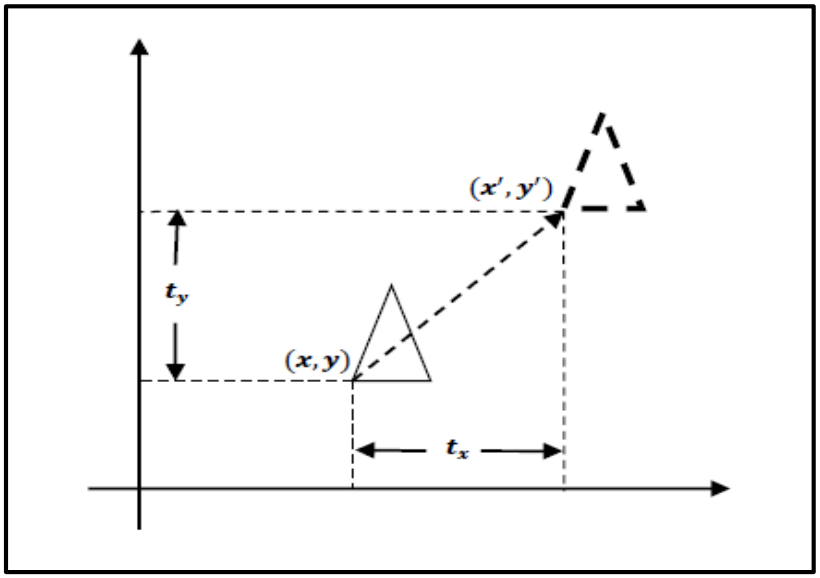



**Figure 3.1 Translation**

**Example 1**: Translate the point **A(10,10)**, 2 unit in x direction and 1 unit in y direction? (using **mathematical equation**)

**Solution**

X =10, Y=10,

$tx$=2 , $ty$=1

**X'= X+ $tx$**

=10+2=12

**Y'= Y + $ty$**

=10+1=11 ,

the coordinate after translation is **A'(12,11)**.

# Program 1: Matlab program to Translate point.

```
% 2D-Translation Transformation for One Point
clc;
clear all;
close all
% enter x-value & y-value for point
x=input ('enter x-value ');
y=input ('enter y-value ');
%enter Translating factor tx & ty
tx=input ('enter Tx: ');
ty=input ('enter Ty: ');
x1=x+tx;
y1=y+ty;
axis ([0 100 0 100]);
hold on
plot (x, y,'r*','markersize',10)
plot (x1, y1,'bo','markersize',10)
legend ('Before Translation', 'After Translation')
xlabel('X-axis')
ylabel('y-axis')
title('2D-Translation-point')
```

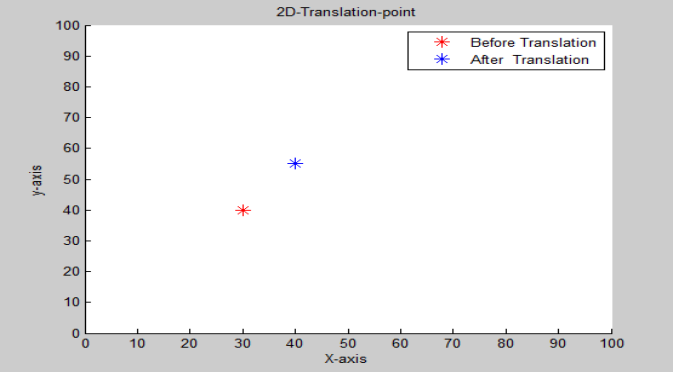
## The Output of program:

enter x-value 30

enter y-value 40

enter Tx: 10

enter Ty: 15



## 3.2.2 Scaling

Scaling is a transformation used to change the size of an object. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object by the scaling factor ( S ) to get the desired result.

**Notes:**

If the scaling factor (S < 1 ) ; then we can reduce the size of the object.

If the scaling factor (S > 1 ) ; then we can increase the size of the object.

If the scaling factor (S = 1 ) ; then no change.

Let us assume that the original coordinates are (x, y), the scaling factors are ($S_x$, $S_y$), and the produced coordinates are (x', y'). This can be **mathematically** represented as shown below :

$$x' = x \cdot Sx \quad \& \quad y' = y \cdot Sy$$

If $S_x = S_y \rightarrow$ No change in the shape the object $(uniform\ scaling)$

If $S_x \neq S_y$ change in the shape the object (distortion *in the original object*)

The scaling process is shown in figure 3.2 as the following.

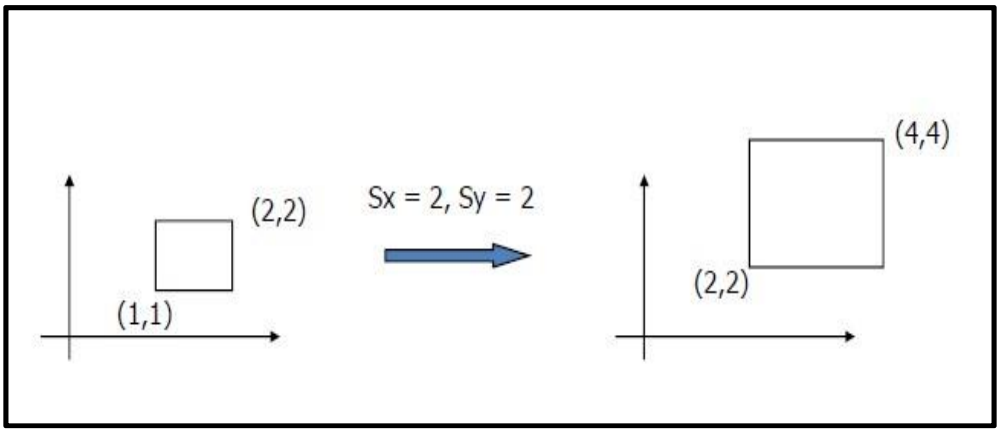



**Figure 3.2 Scaling**

Whenever the scaling process is performed there is one point still no change (same position), this point is called fixed point of the scaling transformation.

There are **two types of scaling** depending on fixed point:

1. If **the fixed point at the origin**, then the point (x,y) can be scaled by a scale factor $S_x, S_y$ in the x-axis and y-axis direction respectively to the new point $(x', y')$.

$$\mathbf{x' = x \cdot S_x \quad \& \quad y' = y \cdot S_y}$$

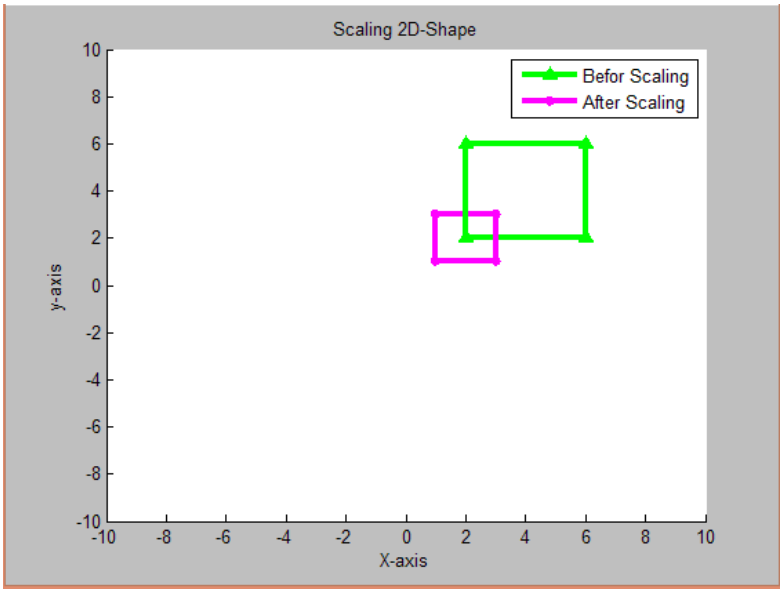
**Example 2:** consider square with left -bottom corner at (2,2) and height-top corner at (6,6) apply transformation which makes its size half.

**Solution**

As we want size half so value scale factor are Sx= 0.5, Sy =0.5 ,and the coordinates of square are A(2,2),B(6,2),C(6,6),D(2,6).

| **A(2,2)** | **B(6,2)** | **C(6,6)** | **D(2,6)** |
|---|---|---|---|
| $\mathbf{x' = x \cdot S_x}$ | $\mathbf{x' = x \cdot S_x}$ | $\mathbf{x' = x \cdot S_x}$ | $\mathbf{x' = x \cdot S_x}$ |
| = 2*0.5=1 | = 6*0.5=3 | = 6*0.5=3 | = 2*0.5=1 |
| $\mathbf{y' = y \cdot S_y}$ | $\mathbf{y' = y \cdot S_y}$ | $\mathbf{y' = y \cdot S_y}$ | $\mathbf{y' = y \cdot S_y}$ |
| =2*0.5=1 | =2*0.5=1 | =6*0.5=3 | =6*0.5=3 |
| A'**(1,1)** | B'**(3,1)** | C'**(3,3)** | D'**(1,3)** |

The final coordinates of square are A'**(1,1)** ,B'**(3,1),**C'**(3,3),**D'**(1,3).**




## 2. If fixed point is Arbitrary Point

Arbitrary Point is a point that is based a random choice or personal rather than a reason or system and figure 3.3 show the types of Arbitrary Point.

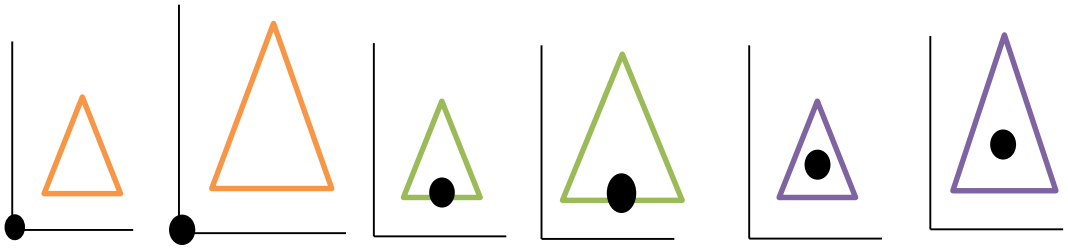


**Figure 3.3 Types of Arbitrary Point**

The **scaling** is performed with respect to any point ($x_f$, $y_f$) as fixed point according to the **three steps**:

**1.** We must first **translate** the object so that the fixed point is coincide with the origin as follows: every object point (x, y) is moved to the new position ($x'$, $y'$) such as:

$$x' = x - x_f , \qquad y' = y - y_f$$

**2.** We then **scaled** these translated points by scale factors $S_x$ and $S_y$ ,so that :

$$x'' = x' . S_x , \qquad y'' = y' . S_y$$

**3.** Then perform the inverse of the original translation to translate (**move)** the fixed point back to its original position.

$$x''' = x'' + x_f , \qquad y''' = y'' + y_f$$

These three steps can be combined in the following equation that scales a point (xf, yf).

*the final equations of scaling object about arbitory point are*

$$\boldsymbol{x''' = (x - x_f) . S_x + x_f , \qquad y''' = (y - y_f) . S_y + y_f}$$

**Example 3**: Consider a tringle defined by its three vertices (1,0), (4,0), (3,2) been scaled 3 units to the Sx and 3 units to the Sy with respect to a fixed point (3,0) . Find the new coordinates of this tringle after Scaling.

**Solution :**

Sx=3 ; Sy=3 ; xf=3 ; yf=0 ;

P1=(1,0)

1- x'=x-xf ; x'=1-3 ; x'=-2 ; y'=y-yf ; y'=0-0 ; y'=0;

2- x''=x'*Sx ; x''=-2*3 ;x''=-6 ; y''=y'*Sy ; y''=0*3 ; y''=0;

3- x'''=x''+xf ; x'''=-6+3; x'''=-3 ; y'''=y''+yf ;y'''=0+0; y'''=0;

P1=(1,0) → P'1(-3,0)

So, the new coordinated of the tringle are :

OR

**Solution :**

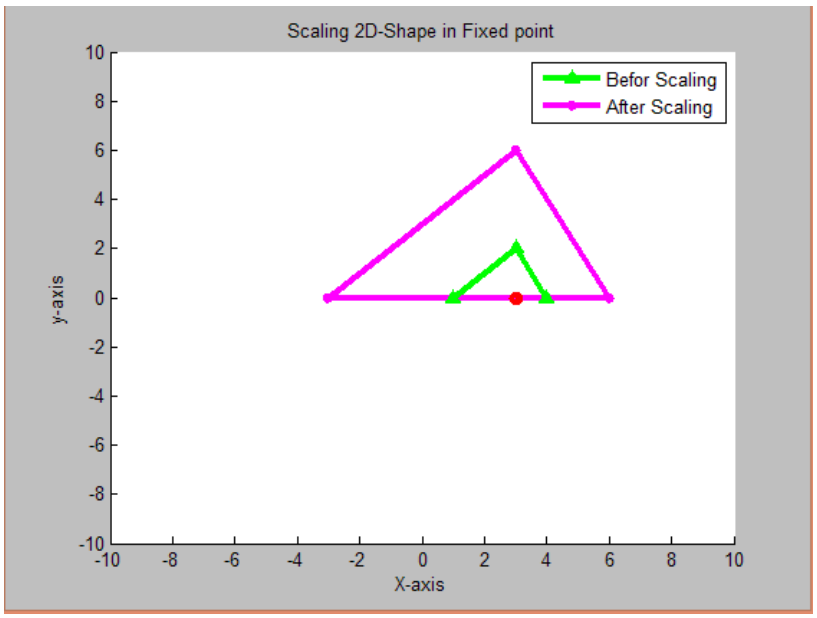
Sx=3 ; Sy=3 ; xf=3 ; yf=0 ;

P1=(1,0)

1- x'''= (x-xf )* Sx +xf ; x'''=(1-3)*3+3; x'''=-3 ; y'''= (y-yf )*Sy+yf ;y'''=(0-0)*3+0; y'''=0;

P1=(1,0) → P'1(-3,0)

| p(x,y) | P'(x',y') |
|---|---|
| (1,0) | (-3,0) |
| (4,0) | (6,0) |
| (3,2) | (3,6) |



# Program 2: Matlab program Scaling Transformation for 2D shape.

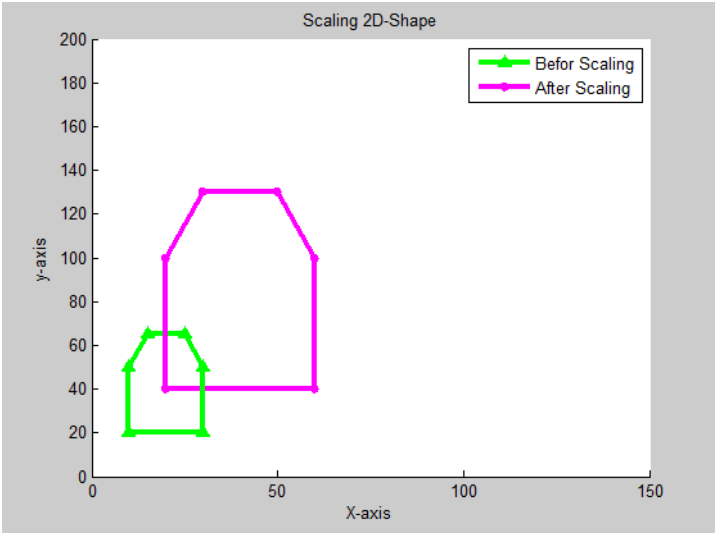
```
% SCALING TRANSFORMATION PROGRAM for 2D-shape
clc; clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of Shape vertices: ');
%enter SCALING factor sx & sy
sx=input ('enter Sx: ');
sy=input ('enter Sy: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
x(i)=input ('enter x-value: ');
y(i)=input ('enter y-value: ');
x1(i)=x(i)*sx;
y1(i)=y(i)*sy;
end
axis ([0 150 0 200]);
hold on
for i=1: g-1
plot([x(i) x(i+1)], [y(i) y(i+1)], ...
'g^-','LineWidth',3,'markersize',5)
plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...
'm*-','LineWidth',3,'markersize',5)
end
plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...
'g^-','LineWidth',3,'markersize',5)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...
'm*-','LineWidth',3,'markersize',5)

Legend ('Before Scaling', 'After Scaling')
xlabel('X-axis')
ylabel('y-axis')
title ('Scaling 2D-Shape')
```

# The Output of program:

enter no. of Shape vertices: 6

enter Sx:  2

enter Sy: 2




enter x-value 10

enter y-value 20

enter x-value 30

enter y-value 20

enter x-value 30

enter y-value 50

enter x-value 25

enter y-value 65

enter x-value 15

enter y-value 65

enter x-value 10

enter y-value 50

## 3.2.3 Rotation

Rotation is a transformation that used to reposition the object along the circular path in the XY -plane. You can rotate an object about the origin or about a pivot point.

It is possible to rotate one or more objects or the entire image about any point in world space in either negative oriented a (clockwise) where angle is negative oriented or positive oriented (Anti-clockwise) where angle is positive.

There are **two types of Rotate** :

### 1. Rotate about the origin

Any point (x,y) can be represented by its radial distance (r) from the origin and its angle ∅ of the x-axis.

**x= r*cos (∅)**

**y=r*sin(∅)** **…….. (1)**

If (x,y) is rotated an angle $\theta$ in the Anti-clockwise direction. The transformed point $(\bar{x}, \bar{y})$ is represented as:

$$\bar{x} = r * \cos(\emptyset + \theta)$$

$$\bar{y} = r * \sin(\emptyset + \theta) \quad \textbf{……. (2)}$$

The Figure 3.4 show the Rotate about the origin.

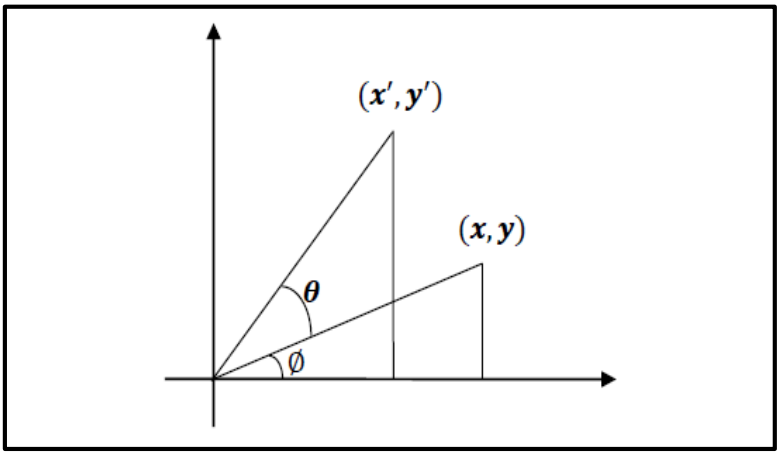



**Figure 3.4 Rotate about the Origin**

Using the laws of sines and cosines we get:

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

$$y = x\sin(\theta) + y\cos(\theta) \quad \ldots\ldots\ldots\ldots(3)$$

Equation (3) are the transformation that rotate a point an angle ($\theta$ ) about the origin in the ***Anti-clockwise*** direction.

To rotate an object each point defining that object must be transformed using equation (3). The object is then drawn using the list of transformed points.

To rotate in ***clockwise*** change the angle $\theta$ to - $\theta$ where:

$$\cos(-\theta) = \cos(\theta)$$

$$\sin(-\theta) = -\sin(\theta)$$

So to rotate a point (x,y) through a ***clockwise*** angle $\theta$ about the origin of the coordinate system we write:

$$\bar{x} = x\cos(\theta) + y\sin(\theta)$$

$$\bar{y} = -x\sin(\theta) + y\cos(\theta) \quad \ldots\ldots\ldots\ldots(4)$$

Equation (4) are the transformation that rotate a point an angle ($\theta$) about the origin in the ***clockwise*** direction.

**Example 4:** Consider a triangle defind by its three vertices (20,0), (60,10),and (40,100) been rotated 30o counter clockwise. Find the new coordinates of this triangle after Rotation.

## Solution

$\bar{x} = x\cos(\theta) - y\sin(\theta)$

$\bar{y} = x\sin(\theta) + y\cos(\theta)$

Cos(30)=0.866; sin(30)=0.5

p1=(20,0)

x'=20*cos(30)-0*sin(30)

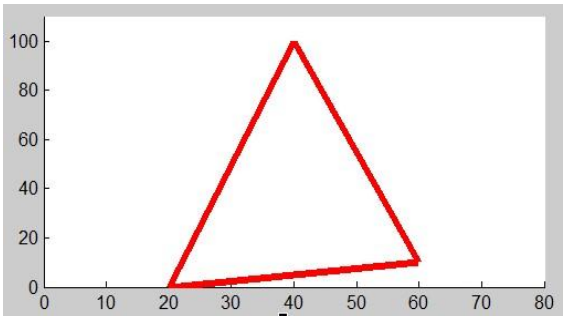
x'=20*0.866 -0= 17.32

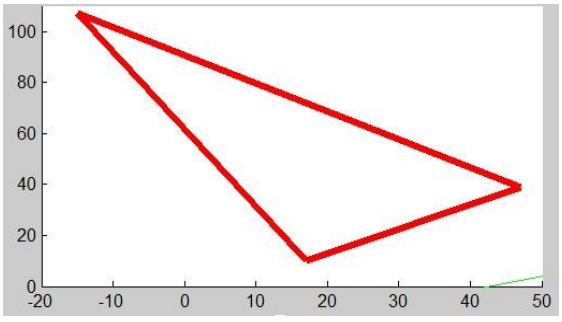
y'= 20*sin(30)+0*cos(30)

y'=20*0.5+0 = 10

**p1(20,0) → p'1(17,10)**

| p(x,y) | P'(x',y') |
|---|---|
| (20,0) | (17,10) |
| (60,10) | (47,39) |
| (40,100) | (-15,107) |

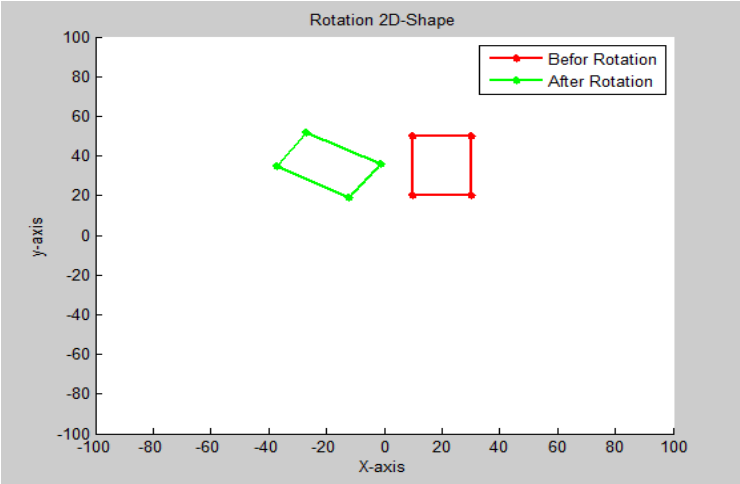


# Program 3: Matlab Program to Rotation 2D -Shape (Anti-Clockwise)

```
% ROTATION TRANSFORMATION (Anti-clockwise) PROGRAM %for 2D-shape
clc;
clear all;
close all
%Enter number of shape or object vertices
g=input ('enter no. of object vertices: ');
%Enter Rotation angle
t=input ('enter the angle: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
   x(i)=input ('enter x-value ');
   y(i)=input ('enter y-value ');
   x1(i)=round ((x(i)) *cos(t)-(y(i)) *sin (t));
   y1(i)=round ((y(i)) *cos(t)+(x(i)) *sin(t));
end
axis ([-100 100 -100 100]);
hold on
for i=1: g-1
   plot([x(i) x(i+1)], [y(i) y(i+1)], ...
      'r*-','linewidth',2,'markersize',5)
   plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...
      'g*-','linewidth',2,'markersize',5)
end
plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...
   'r*-','linewidth',2,'markersize',5)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...
   'g*-','linewidth',2,'markersize',5)

Legend ('Before Rotation', 'After Rotation')
xlabel('X-axis')
ylabel('y-axis')
title ('Rotation 2D-Shape')
```

**The Output of program:**




enter no. of object

vertices: 4

enter the angle: 45

enter x-value 10

enter y-value 20

enter x-value 30

enter y-value 20

enter x-value 30

enter y-value 50

enter x-value 10

enter y-value 50

## 2. Rotate About a Pivot Point

After an object is rotated about a specified pivot point, it is still the same distance away from the pivot point but its orientation has been changed,Figure 3. 5 Show the types a pivot point :

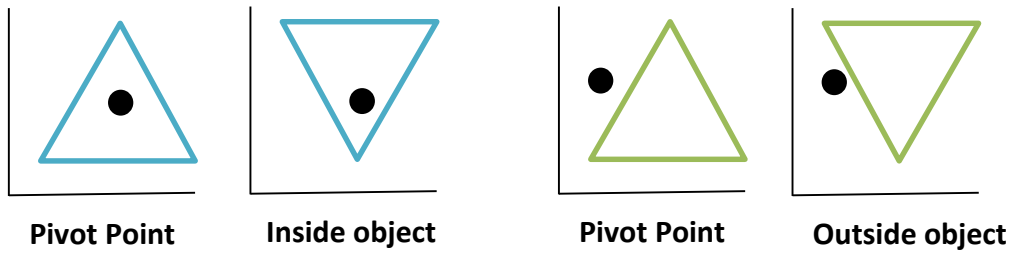



**Figure 3. 5 the types a pivot point**

To rotate an object an angle ($\theta$) about a pivot point **three steps** are required:

**1. Translate** the pivot point ( , $y_p$) to the origin. Every point (x,y) defining the object is translated to a new point ($\bar{x}$, $\bar{y}$) where:

$$\bar{x} = x - x_p$$

$$\bar{y} = y - y_p$$

**2. Rotate** these translated points $(\bar{x}, \bar{y})$ $\theta$ degree about the origin to obtain the new point $(\bar{\bar{x}}, \bar{\bar{y}})$

$$\bar{\bar{x}} = \bar{x} * \cos(\theta) - \bar{y} * \sin(\theta)$$

$$\bar{\bar{y}} = \bar{y} * \cos(\theta) + \bar{x} * \sin(\theta)$$

Substituting for $\bar{x}$ and $\bar{y}$

$$\bar{\bar{x}} = (x - x_p) * \cos(\theta) - (y - y_p) * \sin(\theta)$$

$$\bar{\bar{y}} = (y - y_p) * \cos(\theta) + (x - x_p) * \sin(\theta)$$

**3. Translate** the center of rotation back to the pivot point $(x_p , y_p)$

$$\bar{\bar{\bar{x}}} = \bar{\bar{x}} + x_p$$

$$\bar{\bar{\bar{y}}} = \bar{\bar{y}} + y_p$$

*The final equation of rotate object about a pivot point are:*

$$\bar{\bar{\bar{x}}} = (x - x_p) * \cos(\theta) - (y - y_p) * \sin(\theta) + x_p$$

$$\bar{\bar{\bar{y}}} = (y - y_p) * \cos(\theta) + (x - x_p) * \sin(\theta) + y_p$$

Figure 3.6 show rotate about a pivot point.

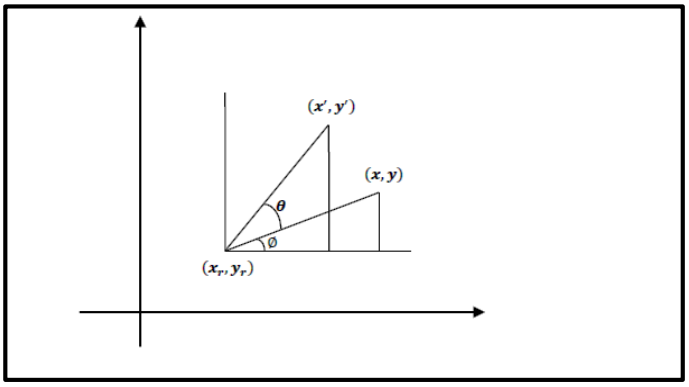



**Figure 3.6 Rotate about a pivot point**

The Figure 3.7 show the three steps of rotate about a pivot point.

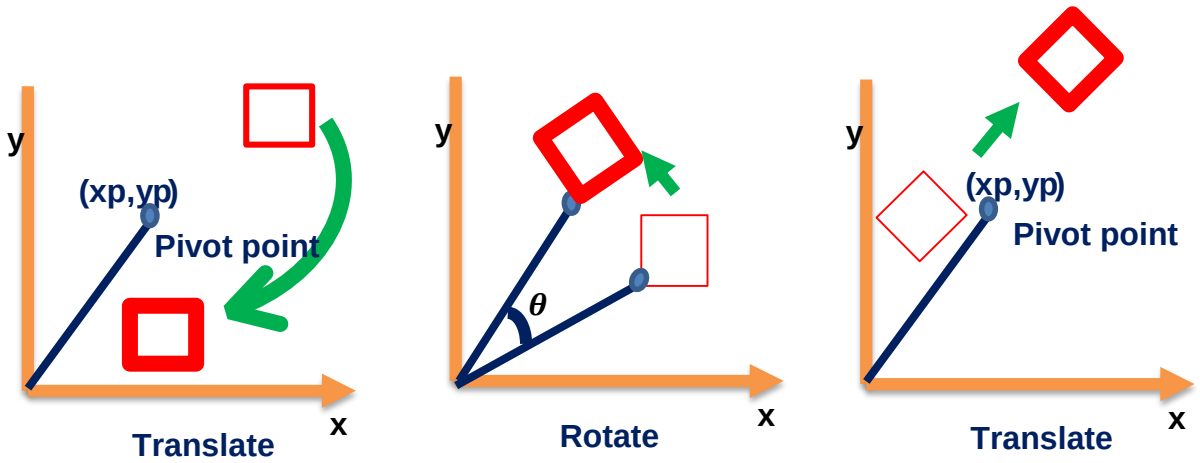



**Figure 3.7 The Three Steps of Rotate about a pivot point.**

**Example 5:** Consider a triangle defind by its three vertices (20,0), (60,10) and (40,100) been rotated 30º counterclockwise about a pivot point (15,20). Find the new coordinates of this triangle after Rotation.

**Solution**

$$\bar{\bar{x}} = (x - x_p) * cos(\theta) - (y - y_p) * sin(\theta) + x_p$$

$$\bar{\bar{y}} = (y - y_p) * cos(\theta) + (x - x_p) * sin(\theta) + y_p$$

cos(30)=0.866 , sin(30)=0.5 ;

xp=15 , yp= 20

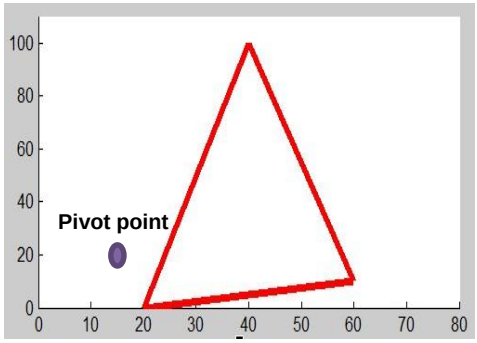
p1=(20,0)

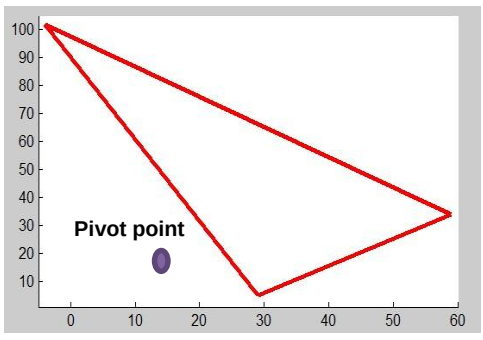
x'''=(20-15)*0.866-(0-20)*0.5+15=29

y'''=(0-20)*0.866+(20-15)*0.5+20=5

**p1(20,0) → p'1(29, 5)**

| p(x,y) | p'(x',y') |
|---|---|
| (20,0) | (29, 5) |
| (60,10) | (59,34) |
| (40,100) | (-4,102) |



## 3.2.4 Reflection

**A reflection** is a Transformation that produces a mirror image of an object**.**

Reflection is a kind of rotation where the angle of rotation is 180 degree,The size of reflected object is same as the size of original object.Consider a point object O has to be reflected in a 2D plane.

Let-

- Initial coordinates of the object O = (Xold, Yold)
- New coordinates of the reflected object O after reflection = (Xnew, Ynew)

**Types of Reflection:**

1. Reflection about the x-axis
2. Reflection about the y-axis
3. Reflection about an axis perpendicular to xy plane and passing through the origin
4. Reflection about line y=x

## 1. Reflection On X-Axis:

This reflection is achieved by using the following reflection equations:

Xnew = Xold

Ynew = -Yold

In this transformation value of x will remain same where as the value of y will become negative. Following figure 3. 8 shows the reflection of the object axis. The object will lie another side of the x-axis.

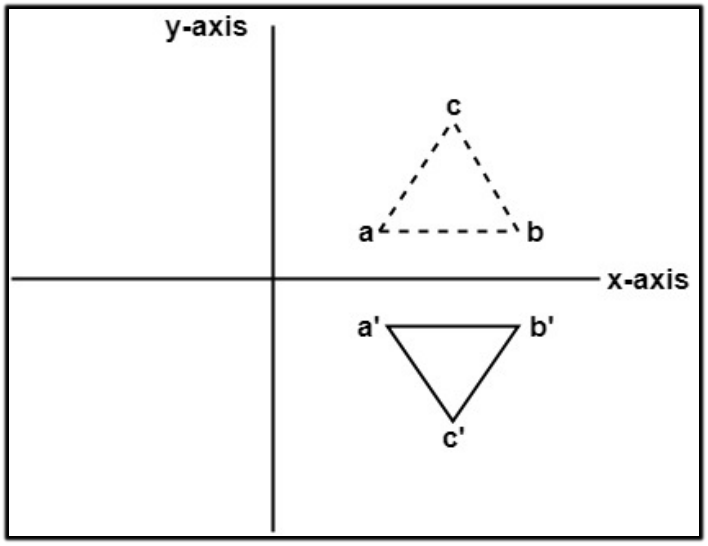



**Figure 3. 8 Reflection of the object on X-axis**

## 2. Reflection On Y-Axis:

This reflection is achieved by using the following reflection equations:

Xnew = -Xold

Ynew = Yold

Here the values of x will be reversed, whereas the value of y will remain the same. The object will lie another side of the y-axis.The following figure 3.9 shows the reflection about the y-axis

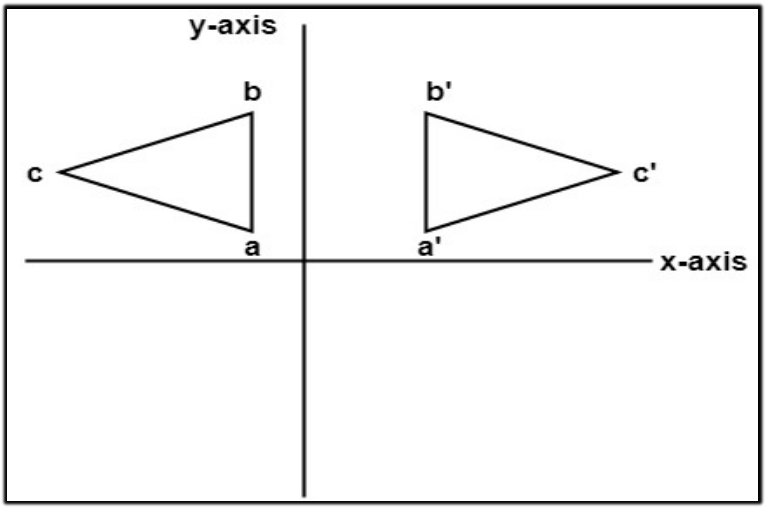



**Figure 3.9 the Reflection about the y-axis**

## 3. Reflection about an axis perpendicular to xy plane and passing through origin:

In this value of x and y both will be reversed. This is also called as half revolution about the origin. The following figure 3.9 shows the reflection about xy plane.

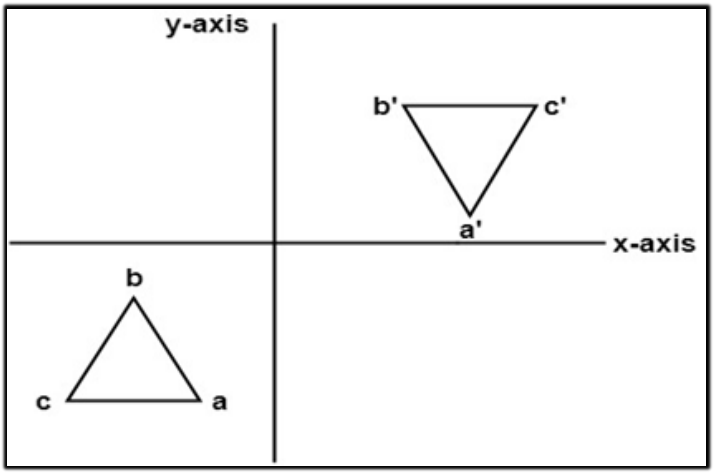



**Figure 3.10 shows the reflection about xy plane.**

## 4. Reflection about line y=x:

The object may be reflected about line y = x with the help of following transformation matrix, Figure 3.10 show the reflection about y=x plane.

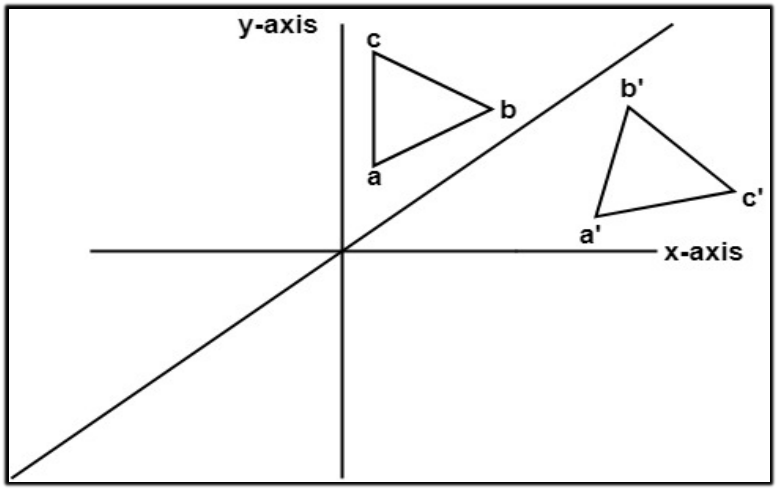



**Figure 3.11 The Reflection about y=x plane.**

First of all, the object is rotated at 45°. The direction of rotation is clockwise. After it reflection is done concerning x-axis. The last step is the rotation of y=x back to its original position that is counterclockwise at 45°.

**Example 6**:Given a triangle with coordinate points A(3, 4), B(6, 4), C(5, 6). Apply the reflection on the X axis and obtain the new coordinates of the object.

**Solution:**

Applying the reflection equations, we have:

**A(3, 4)**

Xnew = Xold = 3

Ynew = -Yold = -4

A’(3,-4)

**B(6, 4)**

Xnew = Xold = 6

Ynew = -Yold = -4

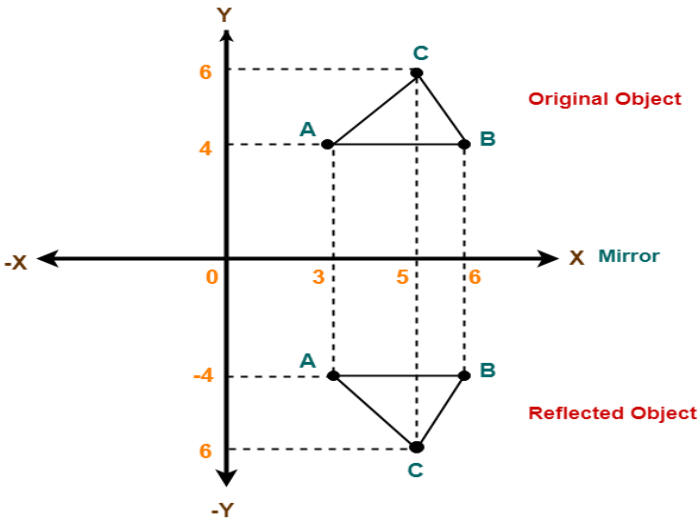
B’(6,-4)

**C(5, 6)**

Xnew = Xold = 5

Ynew = -Yold = -6

C’(5,-6)



# Program 7: Matlab program to Reflection Transformation Program For 2d-Shape

```
% Reflection TRANSFORMATION PROGRAM for 2D-shape
clc; clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of shape vertices: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
   x(i)=input ('enter x-value:');
   y(i)=input ('enter y-value:');
end
%Reflection Types
disp ('Reflection on x, enter 1');
disp ('Reflection on y, enter 2');
disp ('Through the origin, enter 3');
disp ('Over the line y = x, enter 4');
disp ('Over the line y = -x, enter 5');
b=input ('enter type of Reflection:');
switch b
   case 1
      for i=1: g
         x1(i)=x(i);
         y1(i)=-y(i);
      end
```

```
  case 2
    for i=1: g
        x1(i)=-x(i);
        y1(i)=y(i);
    end
  case 3
    for i=1: g
        x1(i)=-x(i);
        y1(i)=-y(i);
    end
  case 4
    for i=1: g
        x1(i)=y(i);
        y1(i)=x(i);
    end
  case 5
    for i=1: g
        x1(i)=-y(i);
        y1(i)=-x(i);
    end
end
axis ([-100 130 -100 130]);
hold on
for i=1: g-1
```

```
   plot([x(i) x(i+1)], [y(i) y(i+1)], ...
      'r*-','linewidth',3,'markersize',10)
   plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...
      'g*-','linewidth',3,'markersize',10)
end
plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...
   'r*-','linewidth',3,'markersize',10)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...
   'g*-','linewidth',3,'markersize',10)
legend ('Before Reflection', 'After Reflection')
xlabel('X-axis')
ylabel('y-axis')
title ('Reflection 2D-shape')
```

## The Output of program:

enter no. of shape vertices: 4

enter x-value:10enter y-value:20

enter x-value:30

enter y-value:20

enter x-value:30

enter y-value:50

enter x-value:10

enter y-value:50

Reflection on x, enter 1

Reflection on y, enter 2

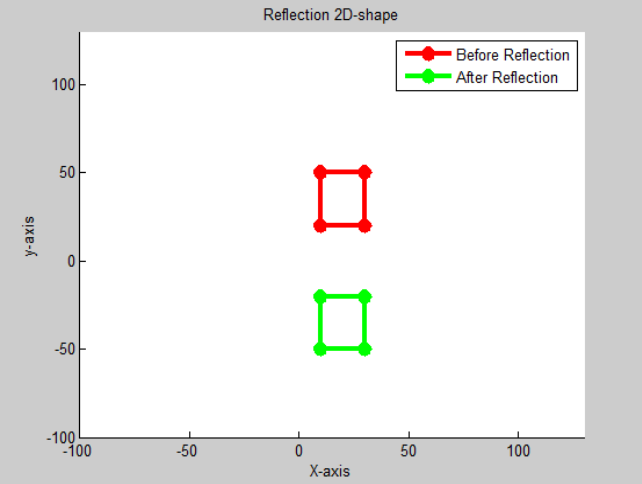
Through the origin, enter 3

Over the line y = x, enter 4

Over the line y = -x, enter 5

enter type of Reflection: 1

Reflection on x



## The Output of program:

enter no. of shape vertices: 4

enter x-value:10

enter y-value:20

enter x-value:30

enter y-value:20

enter x-value:30

enter y-value:50

enter x-value:10

enter y-value:50

Reflection on x, enter 1

Reflection on y, enter 2

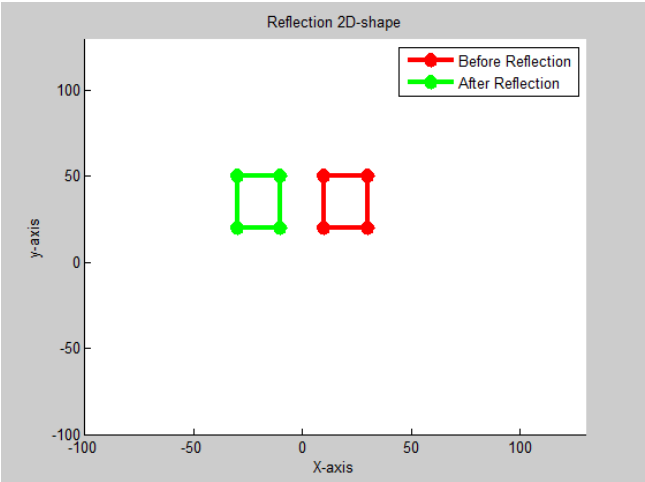
Through the origin, enter 3

Over the line y = x, enter 4

Over the line y = -x, enter 5

enter type of Reflection: 2

Reflection on y



## The Output of program:

enter no.of shape vertices:4  Reflection Through the origin

enter x-value:10

enter y-value:20

enter x-value:30

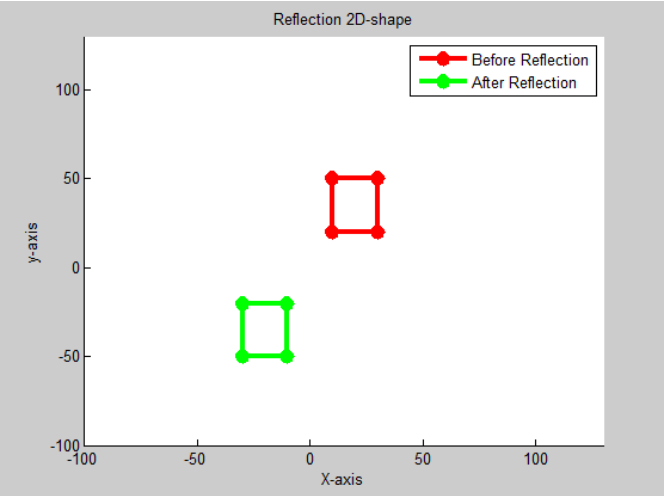
enter y-value:20

enter x-value:30

enter y-value:50

enter x-value:10

enter y-value:50




Reflection on x, enter 1

Reflection on y, enter 2

Through the origin, enter 3

Over the line y = x, enter 4

Over the line y = -x, enter 5

enter type of Reflection: 3

### 3.2.5 Shear

Distorting or changing the shape of an object by differentially moving some of its vertices as if the object internal layers are sided over each other is called ***Shear***.

Shears either shift coordinates x values or y values, Similar to scaling, the shear transformation requires two parameters ($s_x$, $s_y$) not on the main diagonal of the transformation matrix but on the other two positions.

In a two dimensional plane, the object size can be changed along X direction as well as Y direction.

So, there are **two types** of **shearing**:

1. Shearing in X direction
2. Shearing in Y direction

## 1. Shearing in X direction

Shearing in X axis is achieved by using the following shearing equations:

Xnew = Xold + Shx x Yold

Ynew = Yold

Following Figure 3.12 show Shearing in X direction.

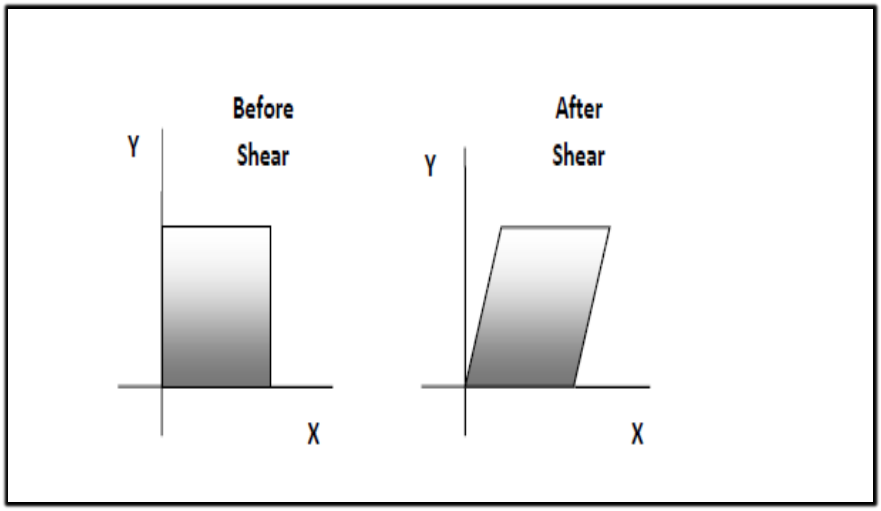



**Figure 3.12 show Shearing in X direction**

## 2. Shearing in Y direction

Shearing in Y axis is achieved by using the following shearing equations-

Xnew = Xold

Ynew = Yold + Shy x Xold

Following Figure 3.13 show **Shearing in Y direction**

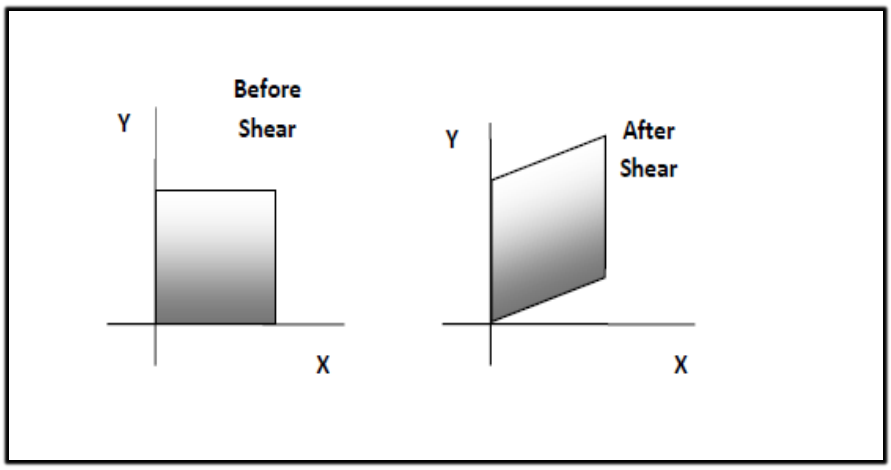



**Figure 3.13 show Shearing in Y direction**

**Example 7**:Given a triangle with points A(1, 1), B(0, 0) and C(1, 0). Apply shear parameter 2 on X axis and 2 on Y axis and find out the new coordinates of the object.

**Solution**

1. **Shearing in X Axis**

   A(1, 1)

   Xnew = Xold + Shx x Yold = 1 + 2 x 1 = 3

   Ynew = Yold = 1

   A'(3,1)

   B(0, 0)

Xnew = Xold + Shx x Yold = 0 + 2 x 0 = 0

Ynew = Yold = 0

**B'(0,0)**

C(1, 0)

Xnew = Xold + Shx x Yold = 1 + 2 x 0 = 1

Ynew = Yold = 0

**C'(1,0)**

Thus, New coordinates of the triangle after shearing in X axis = A'(3, 1), B'(0, 0), C'(1, 0).

## 2. Shearing in Y Axis

A(1, 1)

Applying the shearing equations, we have:

$X_{new} = X_{old} = 1$

$Y_{new} = Y_{old} + Sh_y \times X_{old} = 1 + 2 \times 1 = 3$

**A'(1,3)**

B(0,0)

$X_{new} = X_{old} = 0$

$Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 0 = 0$

**B'(0,0)**

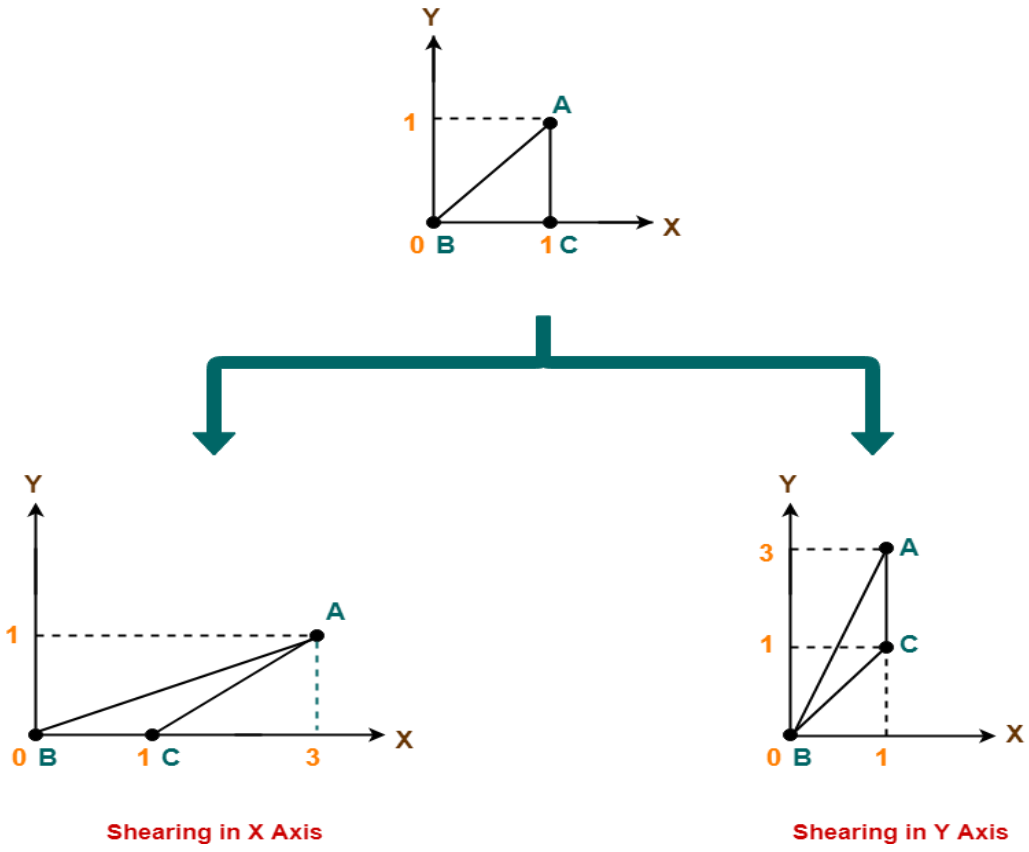
C(1,0)

$X_{new} = X_{old} = 1$

$Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 1 = 2$

**C'(1,2)**

New coordinates of the triangle after shearing in Y axis = A' (1, 3), B'(0, 0), C'(1, 2).




Shearing in X Axis

Shearing in Y Axis

# Program 8: Matlab Program to Shearing Transformation For 2D-Shape

```
% SHEARING TRANSFORMATION PROGRAM for 2D-shape
clc; clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of shape vertices: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
   x(i)=input ('enter x-value:');
   y(i)=input ('enter y-value:');
end
%enter Shearing factor sx & sy
sx=input ('enter Sx: ');
sy=input ('enter Sy: ');

disp ('Shear in the x direction, enter 1');
disp ('Shear in the y direction, enter 2');
disp ('Shear in the both direction, enter 3');
b=input ('enter type of Shearing: ');
switch b
   case 1
      for i=1: g
         x1(i)=x(i)+sx*y(i);
         y1(i)=y(i);
```

```
    end
  case 2
    for i=1: g
        x1(i)=x(i);
        y1(i)=y(i)+sy*x(i);
    end
  case 3
    for i=1: g
        x1(i)=x(i)+sx*y(i);
        y1(i)=y(i)+sy*x(i);
    end
end
```

```
axis ([0 50 0 50]);
hold on
for i=1: g-1
  plot([x(i) x(i+1)], [y(i) y(i+1)], ...
    'r^-','linewidth',3,'markersize',10)
  plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...
    'g^-','linewidth',3,'markersize',10)
end

plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...
  'r^-','linewidth',3,'markersize',10)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...
```

```
  'g^-','linewidth',3,'markersize',10)
legend ('Before Shear', 'After Shear')
xlabel('X-axis')
ylabel('y-axis')
title ('Shearing 2D-shape')
```

## The Output of program:

enter no. of shape vertices: 4

enter x-value:5

enter y-value:5

enter x-value:10

enter y-value:5

enter x-value:10

enter y-value:15

enter x-value:5

enter y-value:15

enter Sx: 2

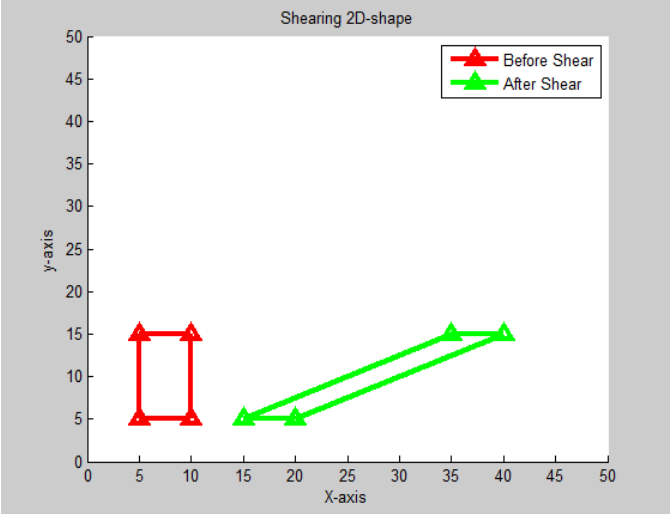
enter Sy: 3

Shear in the x direction, enter 1

Shear in the y direction, enter 2

Shear in the both direction, enter 3

Shear in the x direction



enter type of Shearing: 1

**The Output of program:**

enter no. of shape vertices :4 Shear in the y direction

enter x-value:5

enter y-value:5

enter x-value:10

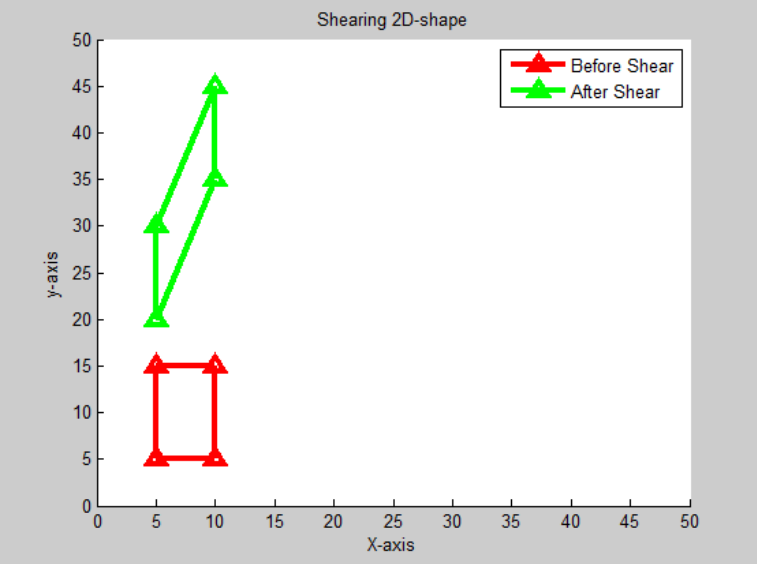
enter y-value:5

enter x-value:10

enter y-value:15

enter x-value:5

enter y-value:15




enter Sx: 2

enter Sy: 3

Shear in the x direction, enter 1

Shear in the y direction, enter 2

Shear in the both direction, enter 3

enter type of Shearing: 1

## The Output of program:

enter no. of shape vertices: 4 Shear in the both direction

enter x-value:5

enter y-value:5

enter x-value:10

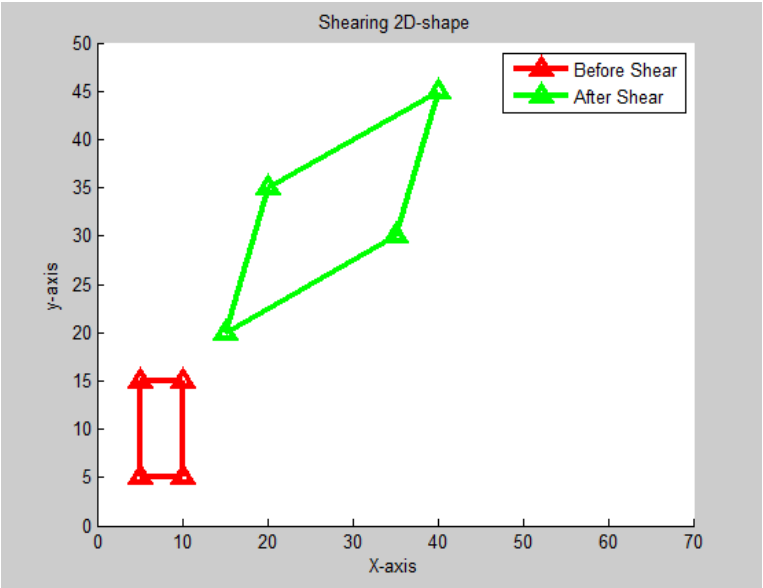
enter y-value:5

enter x-value:10

enter y-value:15

enter x-value:5

enter y-value:15




enter Sx: 2

enter Sy: 3

Shear in the x direction, enter 1

Shear in the y direction, enter 2

Shear in the both direction, enter 3 enter type of Shearing: 1

## 3.3 Matrix Representation of Transformation

Many graphic applications involve sequence of geometric transformations. For example, animation transformation which is require an object to be translated and rotated at each increment of the motion.

- Transformation can be represented as a product of the row vector [x,y] and a 2x2 matrix accept for the translation.
- Transformations can be combined using matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Matrices are convenient to represent a sequence of transformations

### 1- Translation Matrix T(tx , ty)

Wecanrepresent the translation transformation as follows:

P’ = P+T,

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \longrightarrow \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

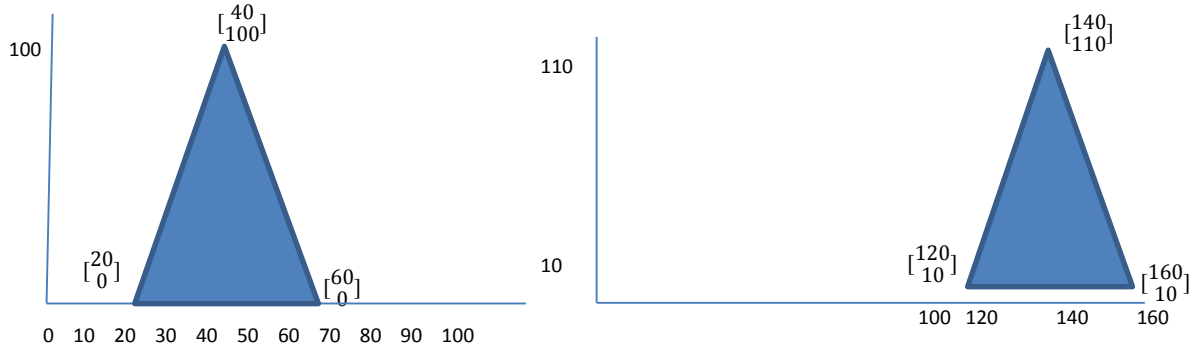
$$P' = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

**Example 8:** Consider a triangle defind by its three vertices (20,0), (60,0), (40,100) been moved 100 units to the right and 10 units up. Find the new coordinates of this triangle after translation. (Using **Matrix**)

So, the new coordinated of the triangle are :

$$T=\begin{bmatrix}100\\10\end{bmatrix}, P=\begin{bmatrix}20 & 60 & 40\\0 & 0 & 100\end{bmatrix}, P'=\begin{bmatrix}20+tx & 60+tx & 40+tx\\0+ty & 0+ty & 100+ty\end{bmatrix}$$

So, the new coordinated of the triangle are : $P'=\begin{bmatrix}120 & 160 & 140\\10 & 10 & 110\end{bmatrix}$



# Program 7 : Matlab program to Translate 2D-shape.(using Matrix)

```
% translation transformation program for 2D-shape
clc;clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of object vertices: ');
%enter Translating factor tx & ty
tx=input ('enter Tx: ');
ty=input ('enter Ty: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
   x(i)=input ('enter x-value:');
   y(i)=input ('enter y-value:');
   x1(i)=x(i)+tx;
   y1(i)=y(i)+ty;
end
axis [0 100 0 100]);
hold on
for i=1: g-1
   plot([x(i) x(i+1)], [y(i) y(i+1)], …'r^- ','linewidth',3,'markersize',10)
   plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], …'g^-','linewidth',3,'markersize',10)
end
plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], …'r^-','linewidth',3,'markersize',10)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], …'g^-
','linewidth',3,'markersize',10)
legend ('Before Translation', 'After Translation')
xlabel('X-axis')
ylabel('y-axis')
title ('Translation 2D-shape')
```

# The Output of program:

enter no. of object vertices: 4

enter Tx: 30

enter Ty: 30

enter x-value 10

enter y-value 20

enter x-value 30

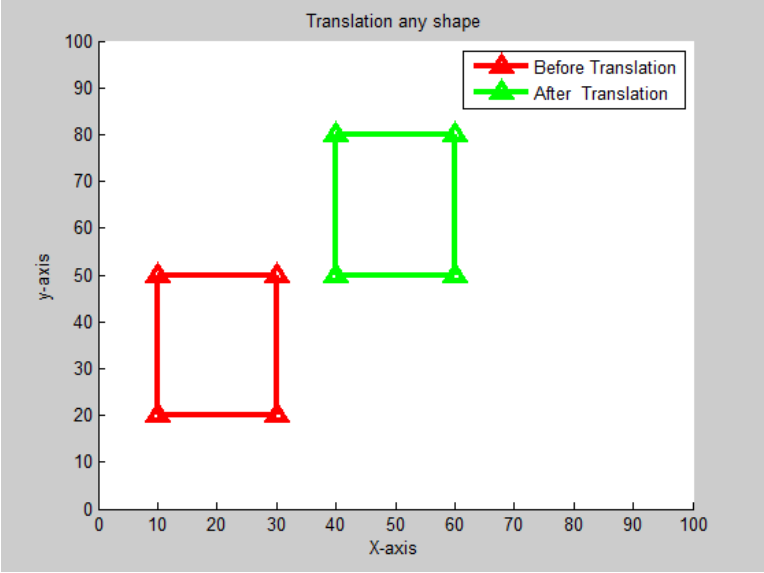
enter y-value 20

enter x-value 30

enter y-value 50

enter x-value 10

enter y-value 50



## 2- Scaling Matrix

If a point P $\begin{bmatrix} x \\ y \end{bmatrix}$ is being a 2x1 vector. If we multiply it by 2x2 matrix

S= $\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$

We will obtain another 2x1 vector which we can interpret as another point: P'= S . P

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

What will happen if we transfer every point by means of multiplication by S and display the result:

**1-** If S is the Identity $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ matrix: S= No change

**2-** If $\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ S= then

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

That mean:

- every new x coordinate would be twice as large as the old value of vertical lines.
- x coordinate would be twice as width and the same tall.

**3-** If $S= \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix}$ shrink all x coordinate (shrink the width with the same tall)

**Example 9:** Stretch the image/object to twice and then compress it to one half of the new width?

$P' = (S_1 S_2).\ P$

$S_1 S_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$

identity matrix then **no change**.

## 3.Rotation Matrix

There are **two types of Rotation:**

### 1. Anti-clockwise direction :

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

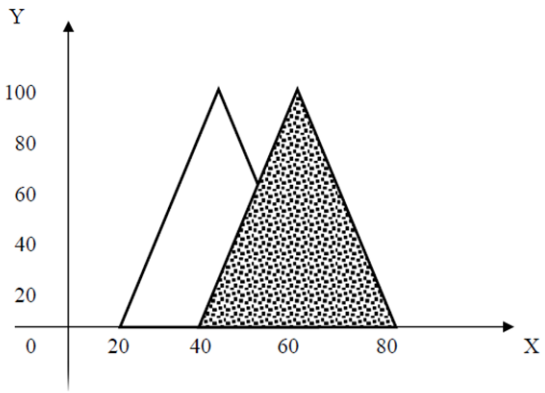
### 2. Clockwise direction :

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

**Example 10 :** Consider a triangle defined by it three vertices ( 40 , 100 ), ( 20 , 0 ), ( 60 , 0 ) be translated 20 units to the right, using matrix representation.

**Solution**

$$\begin{bmatrix} 40 & 100 & 1 \\ 20 & 0 & 1 \\ 60 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 20 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 60 & 100 & 1 \\ 40 & 0 & 1 \\ 80 & 0 & 1 \end{bmatrix}$$



## 4.Reflection Matrix

There are different types of Reflection:

## 1 - Reflection about X – axis:

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

## 2-Reflection about Y – axis:

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

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

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

## 4 - Reflection about the line y = x :

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

**Example 11:** 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

by used matrix representation, and draw the result.

**Solution:**

**1- X – axis:**

$x'=x$

$y'=-y$

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

## 2- Y- axis:

$x' = -x$

$y' = y$

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

## 3- origin (0,0):

$x' = -x$

$y' = -y$

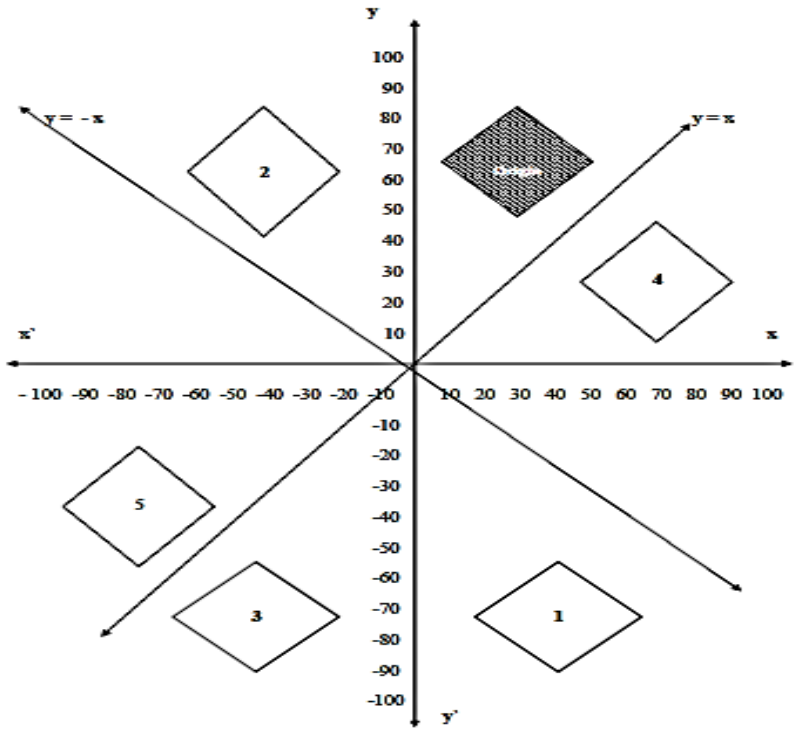
$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -20 & -70 & 1 \\ -40 & -50 & 1 \\ -60 & -70 & 1 \\ -40 & -90 & 1 \end{bmatrix}$$

## 4.y = x

$x' = y$

$y' = x$

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


y
100
90
80
70
60
50
40
30
20
10
y = - x
y = x
2
4
x'
x
- 100 -90 -80 -70 -60 -50 -40 -30 -20 -10 10 20 30 40 50 60 70 80 90 100
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100
5
3
1
y'

## 3.4 2D Viewing

**Viewing** is the process of drawing a view of a model on a 2-dimensional display.

The geometric description of the object or scene provided by the model, is converted into a set of graphical primitives, which are displayed where desired on a 2D display.The same abstract model may be viewed in many different ways:

e.g. faraway, near, looking down, looking up.

### 3.4.1 Real World Coordinates

It is logical to use dimensions which are appropriate to the object for example:

- meters for buildings
- nanometers or microns for molecules, cells, atoms
- light years for astronomy

The **objects** are described with respect to their actual physical size in the **real world**, and then mapped onto screen co-ordinates. It is therefore possible to view an object at various sizes by zooming in and out, without actually having to change the model.

## 3.4.2 How do we convert Real-world coordinates into screen coordinates?

We could have a model of a whole room, full of objects such as chairs, tablets and students.We may want to view the whole room in one go, or zoom in on one single object in the room. We may want to display the object or scene on the full screen, or we may only want to display it on a portion of the screen.Once a model has been constructed, the programmer can specify a view.**2-Dimensional view consists of *two* rectangles**:

1. A ***Window***, given in **real-world** coordinates, which defines the portion of the model that is to be drawn.

2. A ***Viewport*** given in **screen** coordinates,which defines the portion of the screen on which the contents of the window will be displayed.

Figure 3.14 show the window and viewport.

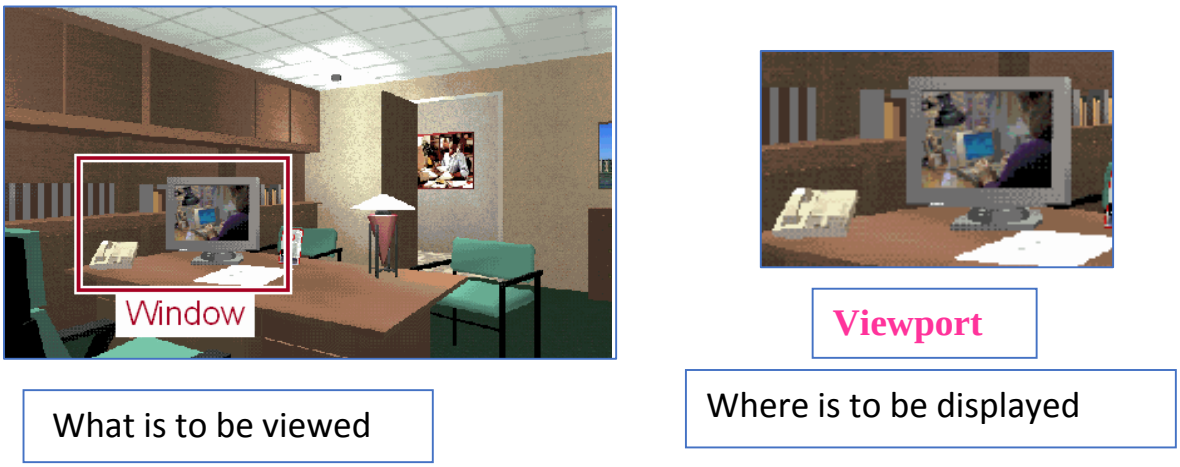



**Figure 3.14 The window and viewport.**

## 3.5 Window to Viewport Transformation ( Mapping)

The window to viewport mapping is a process of transforming or mapping the two dimensional or world coordinate view into device coordinate. The object which is available inside of the clipping window or world is mapped into the viewport and is displayed on the interface window screen, or the clipping window selects the piece of the scene from the display and view port positions it in the output device.The following figure 3.15 show Window to Viewport Mapping.

# Window:

**1-** A world-coordinate area selected for display is called awindow.

**2-** In computer graphics, a window is a graphical control element.

**3-** It consists of a visual area containing some of the graphicaluser interface of the program it belongs to and is framed by awindow decoration.

**4-** A window defines a rectangular area in world coordinates. Youcan define the window to be larger than, the same size as, orsmaller than the actual range of data values, depending onwhether you want to show all of the data or only part of thedata.

**5-** Window defines what is to be viewed .

## Viewport:

**1-** An area on a display device to which a window is mapped iscalled a viewport.

**2-** A viewport is a polygon viewing region in computer graphics.The viewport is an area expressed in rendering-device-specificcoordinates, e.g. pixels for screen coordinates, in which theobjects of interest are going to be rendered.

**3-** A viewport defines in normalized coordinates a rectangulararea on the display device where the image of the data appears.You can have your graph take up the entire display device orshow it in only a portion, say the upper-right part.

**4-** Viewport defines where the window to be displayed.

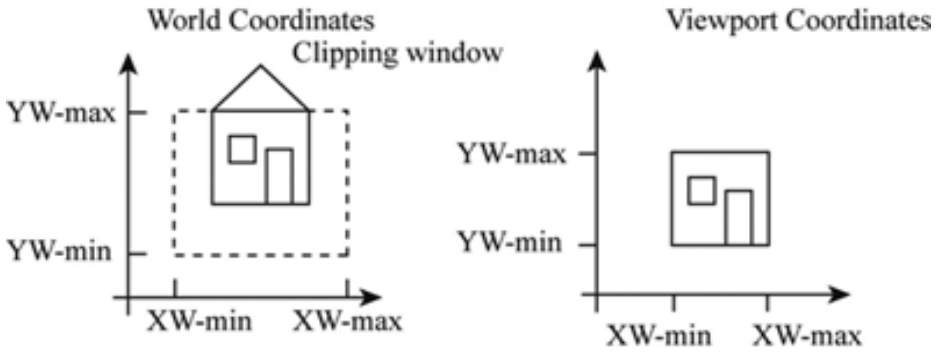
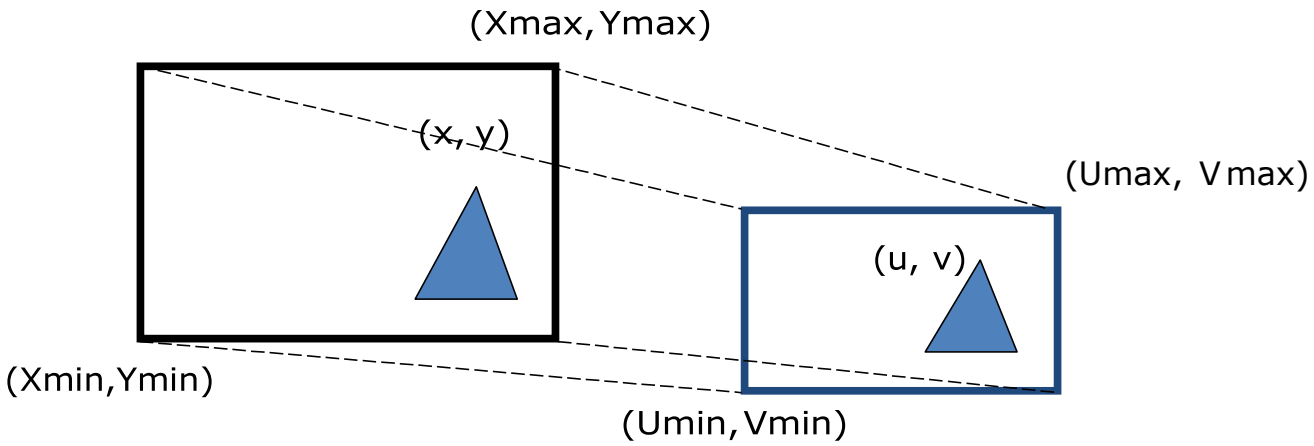



**Figure 3.15 Window to Viewport Mapping**

This transformation involves developing formulas that start with a point in the world window, say (x, y).




The formula for window to viewport mapping is:

(x, y) ⟶ (u, v)

$$\frac{x - x_{min}}{x_{max} - x_{min}} = \frac{u - u_{min}}{u_{max} - u_{min}}$$

$$\frac{y - y_{min}}{y_{max} - y_{min}} = \frac{v - v_{min}}{v_{max} - v_{min}}$$

By rewriting this relationship, we get the following formula:

$u = c_1 x + c_2$ $\qquad c_1 = \frac{u_{max} - u_{min}}{x_{max} - x_{min}}$

$c_2 = u_{min} - c_1 x_{min}$

$v = d_1 y + d_2$ $\qquad d_1 = \frac{v_{max} - v_{min}}{y_{max} - y_{min}}$

$d_2 = v_{min} - d_1 y_{min}$

**Example 12:** A normalized window has left and right boundaries of (-0.05 to +0.05) and lower and upper boundaries of (0.1 to 0.2). the viewport window left and right is (250,550) and lower to upper is (100,400),find the coordinate of any point (u,v) in the viewport window.

**Solution :**

Window( xmin=-0.05 , xmax=+0.05 , ymin=0.1, ymax=0.2)

Viewport ( umin=250, umax=550, vmin=100, vmax=400)

$$u = c_1\, x + c_2$$

$$c_1 = \frac{umax - umin}{xmax - xmin}$$

$$c_1 = \frac{(550 - 250)}{0.05 - (-0.05)} = 300/0.1 = 3000$$

$$c_2 = u_{min} - c_1 x_{min}$$

=250-3000(-0.05) =250+150 =400

**u=3000x+400**

$$v = d_1 y + d_2$$

$$d_1 = \frac{vmax - vmin}{ymax - ymin}$$

$$d_1 = \frac{(400 - 100)}{(0.2 - 0.1)} = 300/0.1 = 3000$$

$$d_2 = v_{min} - d_1 y_{min}$$

=100-3000(0.1)

=-200

**v =3000y-200**

## 3.6 Window to Viewport Transformation N

We can express these two formula for computing (u,v) from (x,y) by term:

**(translate-scale-translate)**

$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ \boldsymbol{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{1} \end{bmatrix} . \boldsymbol{N}$$

$N = T_2 \quad S \quad T_1$

1. T1 is the translation matrix about window origin :

$$T_1 = \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

2. is the scaling transformation matrix:

$$S = \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 0 \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. T2 is the translation matrix position of the viewport :

$$T_2 = \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

**Example 13:** A normalized window has left and right boundaries of (-0.05 to +0.05) and lower and upper boundaries of (0.1 to 0.2). the viewport window left and right is (250,550) and lower to upper is (100,400),find the transformation N.

**Solution**

**N=T2ST1**

$$T_1 = \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 0 \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_1 = \begin{bmatrix} 1 & 0 & -(-0.05) \\ 0 & 1 & -0.1 \\ 0 & 0 & 1 \end{bmatrix} \qquad S = \begin{bmatrix} 3000 & 0 & 0 \\ 0 & 3000 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 1 & 0 & 250 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix}$$

$$N = \begin{bmatrix} 1 & 0 & 250 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3000 & 0 & 0 \\ 0 & 3000 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -(-0.05) \\ 0 & 1 & -0.1 \\ 0 & 0 & 1 \end{bmatrix}$$

## 3.7 Clipping and Windowing

Many graphics application programs give the user the impression of looking through a window at a very large picture. To display an enlarged portion of a picture we must not only apply the appropriate scaling and translation but identify the visible parts of the picture for inclusion in the displayed image. The correct way to select visible information for display is to use **clipping** (a process which divides each element of the picture into its visible and invisible portions, allowing the invisible portion to be discarded ) . Clipping can be applied to a variety of different types of picture elements:

vectors, curves of various kinds, and even polygons. The basis for these clipping operations is a simple pair of inequalities that determine whether a point (x,y) is visible or not.

$$\mathbf{xleft \leq x \leq xright \ , \ ybottom \leq y \leq ytop}$$

Where xleft, xright, ybottom, ytop are the positions of the edges of the screen. These inequalities provide us with a very simple method of clipping pictures on a point by point basis; we

substitute the coordinates of each point for x and y and if the point fails to satisfy either inequality; it is invisible. It would be quite inappropriate to clip pictures by converting all picture elements into points and using these inequalities; the clipping process would take far too long and would leave the picture in a form no longer suitable for a line drawing display. We must attempt to clip larger elements of the picture. This involves developing more powerful clipping algorithms that can be determine the visible and invisible portions of such picture elements.

### 3.7.1 Clipping window

It is refer to a rectangular region whose sides are aligned with the coordinates axes. The x extent is measured from xmin to xmax and the y extent is measured from ymin to ymax.

### 3.7.2 Point Clipping

The basis for these clipping operations is a simple pair of inequalities that determine whether a point (x,y) is visible or not:

$$\mathbf{xmin} \leq \mathbf{x} \leq \mathbf{xmax} \text{ , } \mathbf{ymin} \leq \mathbf{y} \leq \mathbf{ymax}$$

Where **xmin**, **xmax**, y**min**, y**max** are the positions of the edges of the window.

### 3.7.3 Line Clipping

Lines that do not intersect the clipping window are either completely inside the window or completely outside the window. On the other hand a line that intersects the clipping window is divided by the intersection point (s) into segments that are either inside or outside the window. The following algorithm provide efficient way to decide the relationship between an arbitrary line and the clipping window to find intersection point (s).Figure 3.16 show the type of line clipping.

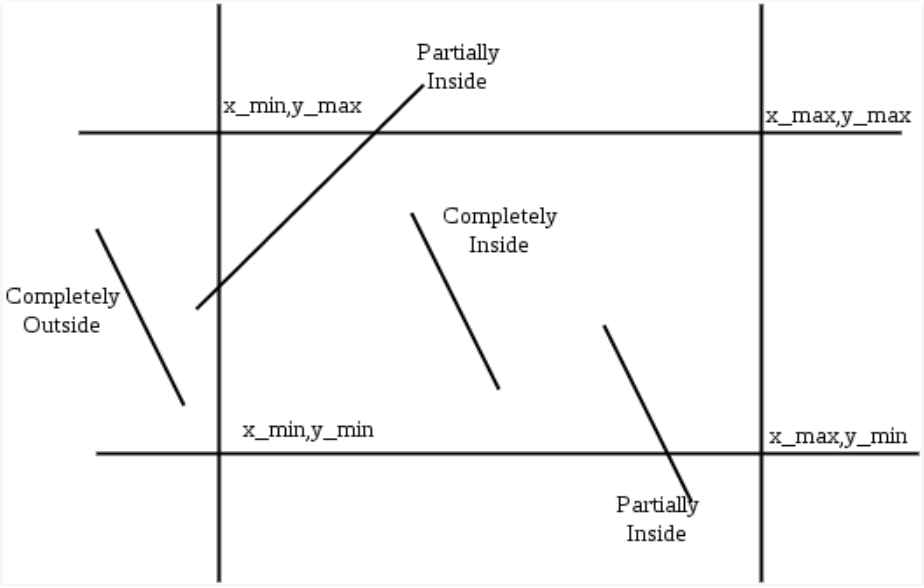
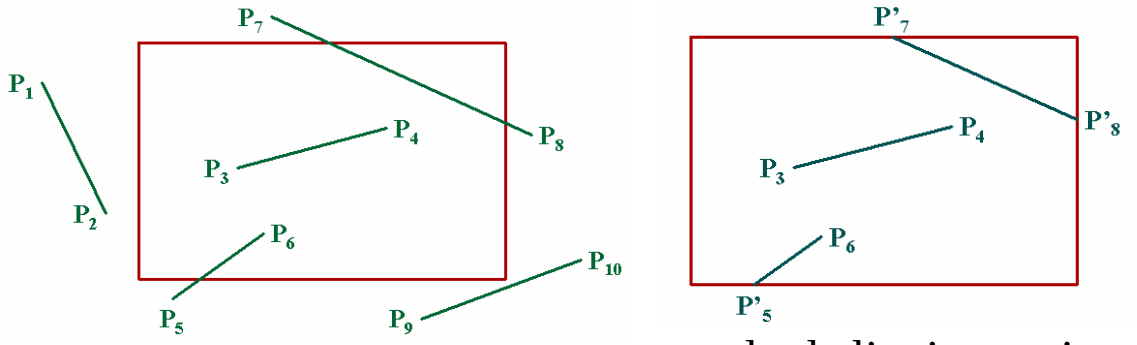



**Figure 3.16 the type of line clipping.**

Figure 3.17 show Possible relationship between line position and a standard clipping region.




**Before Clipping** **After Clipping**

**Figure 3.17 Possible relationship between line position and a standard clipping region.**

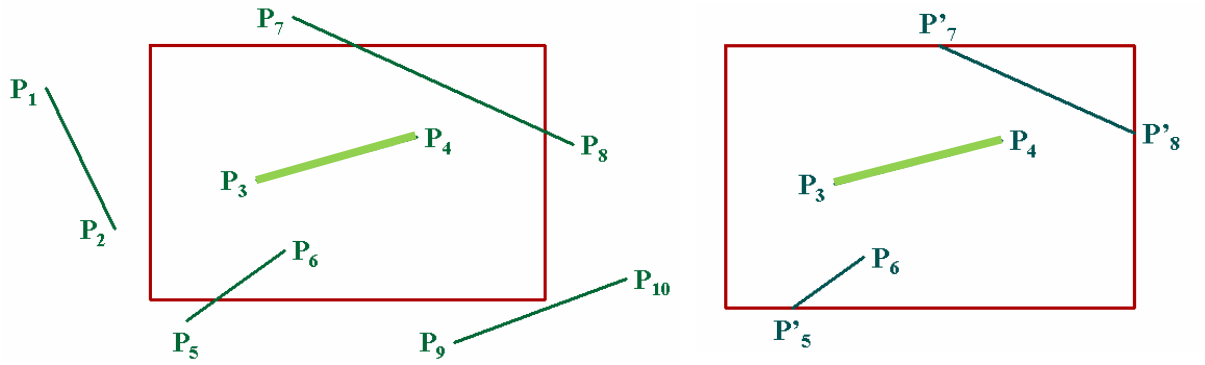
**A line clipping procedure involves several parts:**

**1.** Determine whether line lies completely inside the clipping window.

**2.** Determine whether line lies completely outside the clipping window.

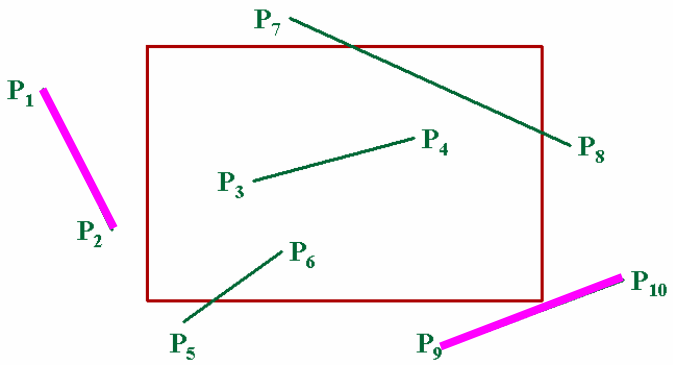
**3.** Perform intersection calculation with one or more clipping boundaries.

A line with both endpoints inside all clipping boundaries is saved ($\overline{P_3P_4}$ )

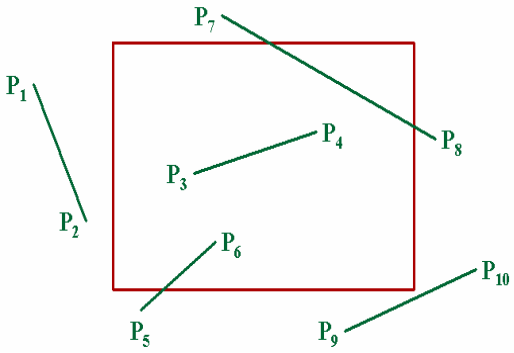



**Before Clipping** **After Clipping**

A line with both endpoints outside all clipping boundaries is **reject** ( $\overline{P_1P_2}$& $\overline{P_9P_{10}}$)



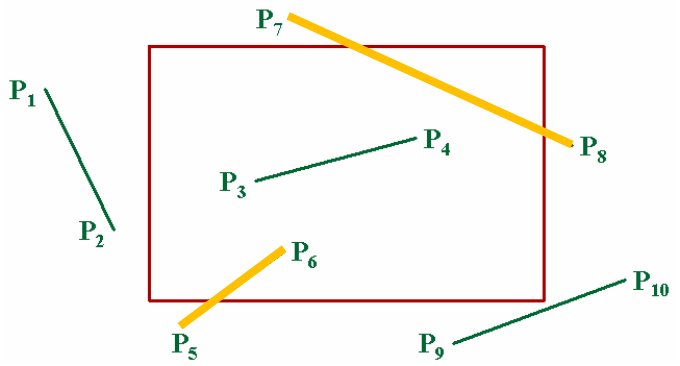

**Before Clipping**



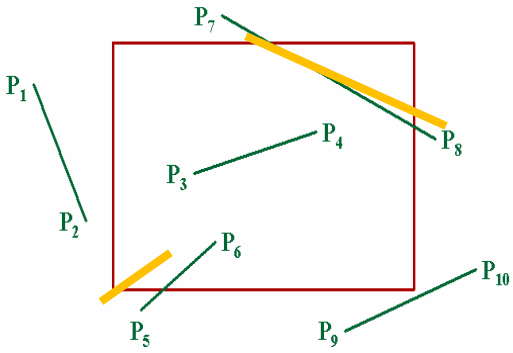

**After Clipping**

If one or both endpoints outside the clipping rectangular, the **parametric representation** could be used to determine values of parameter **u** for intersection with the clipping boundary coordinates.

$$\begin{cases} x = x_1 + u(x_2 - x_1) \\ y = y_1 + u(y_2 - y_1) \end{cases} \qquad 0 \le u \le 1$$




**Before Clipping**




**After Clipping**

**1.** If the value of u is outside the range 0 to 1: The line dose not enter the interior of the window at that boundary.

**2.** If the value of u is within the range 0 to 1, the line segment does cross into the clipping area.

Clipping line segments with these parametric tests requires a good deal of computation, and faster approaches to clipper are possible.

## 3.7.4 The Cohen–Sutherland Algorithm

The **Cohen–Sutherland algorithm** is a computer-graphics algorithm used for line clipping**.**

The Cohen–Sutherland algorithm can be used only on a rectangular clip window.

Given a set of lines and a rectangular area of interest, the task is to remove lines which are outside the area of interest and clip the lines which are partially inside the area.

**Cohen-Sutherland algorithm** divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are inside the given rectangular area.Figure 3.18 show the 9 regions of Cohen-Sutherland algorithm

**The algorithm can be outlines as follows:-**

Nine regions are created, eight "outside" regions and one "inside" region.

For a given line extreme point (x, y), we can quickly find its region's four bit code. Four bit code can be computed by comparing x and y with four values (x_min, x_max, y_min and y_max).

If x is less than x_min then bit number 1 is set.

If x is greater than x_max then bit number 2 is set.

If y is less than y_min then bit number 3 is set.

If y is greater than y_max then bit number 4 is set.

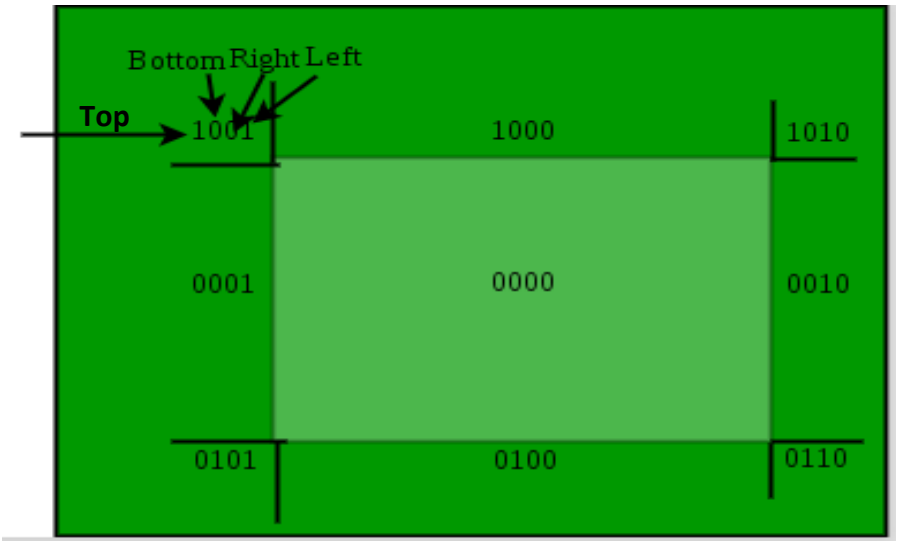



**Figure 3.18 the 9 regions of Cohen-Sutherland algorithm**

The diagram on the above page is associated with the checking order TBRL, corresponding to Top, Bottom, Right, Left. We assign a 1-bit where the region is strictly outside the boundary (in the half-plane not containing the window), and a 0-bit where the region is on the same side as the window. Thus, only the window itself is assigned all zeros. Since the high-order bit is associated

with the top boundary for example, only the three regions above the window (outside the top boundary) have high-order bit equal to 1.

**There are three possible cases for any given line:**

**1. Completely inside the given rectangle :** Bitwise OR of regionof two end points of line is 0 (Both points are inside therectangle)

**2. Completely outside the given rectangle :** Both endpointsshare at least one outside region which implies that the linedoes not cross the visible region. (bitwise AND of endpoints !=0).

**3. Partially inside the window :** Both endpoints are in differentregions. In this case, the algorithm finds one of the two pointsthat is outside the rectangular region. The intersection of theline from outside point and rectangular window becomes newcorner point and the algorithm repeats.

### 3.7.5 Intersection Points

Intersection points with a clipping boundary can be calculated using the slop-intercept form of the line equation. For a line with endpoint coordinates(x1, y1) and (x2, y2), the y coordinate of the intersection point with a **vertical boundary** can be obtained with the calculation **y = y1 + m(x – x1)**

Where the x value is set either to xmin or to xmax, and the slop of the line is calculated as

$$m = (y2 – y1) / (x2 – x1).$$

Similarly, if we are looking for the intersection with a **horizontal boundary**, the x coordinate can be calculated as

$$x = x1 + (y – y1) / m$$

**Note**

**1.**If the boundary line is vertical then:

x=xmin if the line is left

x=xmax if the line is right

**y = y1 + m(x – x1)**

**2.**If the boundary line is horizontal then:

y=ymin if the line is bottom

y=ymax if the line is top

**x = x1 + (y – y1) / m**

**Example 14 :** Apply the Cohen Sutherland line clipping algorithm to clip the line segment with coordinates (30,60) and (60,25) against the window with (*Xmin*,*Ymin*)= (10,10) and (*Xmax*,*Ymax*)= (50,50).

**Solution**

**Clip bit code**

**AB** 1000 AND

0010

---

**0000 (Partially inside) (clipping)**

First ,Find the slop of line AB from the equation:

**m = (y2 – y1) / (x2 – x1)**

m=(25-60)/(60-30)

=-35/30

=-1.16

Then ,We find the coordinate of intersection point from line A A-.

The boundary line A A- is horizontal ,so Ymax=y=50 and calculate xvalue from this :

**x = x1 + (y – y1) / m**

= 30+(50-60)/-1.16

=30+-10/-1.16

= 30+8.6

=38.6

the coordinate of intersection point is **A-(38.6,50)**.

We find the coordinate of intersection point form line BB-.

The boundary line BB- is vertical ,so xmax=x=50 and calculate y valuefrom this:

y = y1 + m(x – x1)

=25+(-1.16)(50-60)

= 25+11.6

=36.6

The coordinate of intersection point is B-(50,36.6).

**Example 15**: Window is defined A(10,20),B(20,20),C(20,10),D(10,10) Find visible portion of line P(15,15),Q(5,5) using Cohen Sutherland line clipping algorithm.

**Solution**

Clip bit code

**PQ** 0000 **AND**

0101

0000 Partially inside (clipping)

Find the slop of line PQ

m = (y2 – y1) / (x2 – x1)

=(5-15)/(5-15)

=-10/-10=1

We fined the coordinate of intersection point from line PP -, The boundary line PP- is horizontal ,so Ymin=y=10 and find x as follow:

x = x1 + (y – y1) / m

=15+(10-15)/1

=15-5

= 10 the coordinate of intersection point is P-(10,10).

**Example 16:** Window is defind

A(20,20),B(90,20),C(90,70),D(20,70) Find visible portion of line1 :P1(10,30),P2(80,90)

Line2: Q1(20,10) , Q2(70,60)

using Cohen Sutherland line clipping algorithm.

**Solution**

Xmin=20 , Xmax=90 , ymin=20 , ymax=70

**Clip bit code**

**P1P2** 0001 **AND**

1000

---

0000 **(Partially inside) (clipping)**

**Q1Q2**

0101 **AND**

0000

---

0000 **(Partially inside) (clipping)**

First find the slop of line P1P2 from the equation:

**m = (y2 – y1) / (x2 – x1)**

=(90-30)/(80-10)

=60/70=0.8

Then find the coordinate of intersection point from line P1P1-.

The boundary line P1P1- is vertical ,so Xmin=x=20 and calculate y value from this :

**y = y1 + m(x – x1)**

=30+0.8(20-10)

=30+8=38

the coordinate of intersection point **P1-(20,38).**

Then find the coordinate of intersection point from line P2P2- .

The boundary line P2P2- is horizontal ,so ymax=y=70 and find x fromthis equation:

**x = x1 + (y – y1) / m**

=80+(70-90)/0.8

=80+(-20)/0.8

=80+(-25)=55

the coordinate of intersection point **P2-(55,70).**

Find the slop of second line Q1Q2

m = (y2 – y1) / (x2 – x1)

=(60-10)/(70-20)=50/50=1

Then find the coordinate of intersection point from line Q1Q1-

The boundary line Q1Q1- is horizontal ,so ymin=y=20 and calculate xvalue from this :

$x = x1 + (y – y1) / m$

$=20+(20-10)/1$

$=20+10=30$

The coordinate of intersection point is Q1- (30,20)

**Example 17**: Rectangular area of interest (defined by below four values which are coordinates of bottom left and top right)

Xmin=4,ymin=4,xmax=10,ymax=8

A set of lines( defined by two corner coordinates)

Line 1: A(5,5), B(7,7)

Line 2: C(7,9), D(11,4)

Line 3: E(1,5), F(3,2)

Apply the Cohen Sutherland line clipping algorithm to clip the line segment.

**Solution:** Clip bit code AB 0000 OR

0000

---

0000 accept (inside)

CD 1000 AND

0110

---

0000 partially inside (clipping)

EF 0001 AND

0101

0001 reject (outside)

Find slop for line CD as follow:

m = (y2 – y1) / (x2 – x1)

=(4-9)/(11-7)

=-5/4=-1.25

We fined the coordinate of intersection point from line CC-,

The boundary line CC- is horizontal ,so Ymax=y=8 and find x as follow:

x = x1 + (y – y1) / m

= 7+(8-9)/-1.25

=7+-1/-1.25

7+0.8=7.8

**The coordinate of intersection point is C-(7.8,8).**

We fined the coordinate of intersection point form line DD-, the boundary line DD- is vertical ,so xmax=x=10 and find y as follow:

y = y1 + m(x – x1)

=4+(-1.25)(10-11)

=4+1.25

=5.25

**The coordinate of intersection point is D -(10,5.25).**

# CHAPTER FOUR

# 3D CONCEPTS & OBJECT REPRESENTATION

# *Chapter Four*

## *3D Concepts & Object Representation*

- **Introducation to 3D Computer Graphics**
- **3D Modeling Transformations**
- **Types of Transformation**
- **3D Translation**
- **Inverse 3D Translation**
- **3D Rotation**
- **3D Scaling**
- **3D Reflection**
- **3D Shearing**

# Chapter Four

# 3D Concepts & Object Representation

## 4.1 Introducation to 3D Computer Graphics

**3D Computer Graphics(CGI)**, is graphics that uses a three-dimensional representation of geometric data that is stored in the computer and produces(known as "rendering") 2D images. A 3D model can be displayed visually as a two-dimensional image through a process called 3D rendering. The world composed of three – dimensional images so the object has height, width and depth. The computer uses a mathematical model to create the image. Figure 4.1 show an example of 3D rendering.

**Figure 4.1 Example of 3D rendering.**

## 4.2 3D Modeling Transformations

The model describes the process of forming the shape of an object.Methods for object modeling transformation in three dimensions are extended from two dimensional methods by including consideration for the z coordinate. Figure 4.2 show the 3d coordinate system.

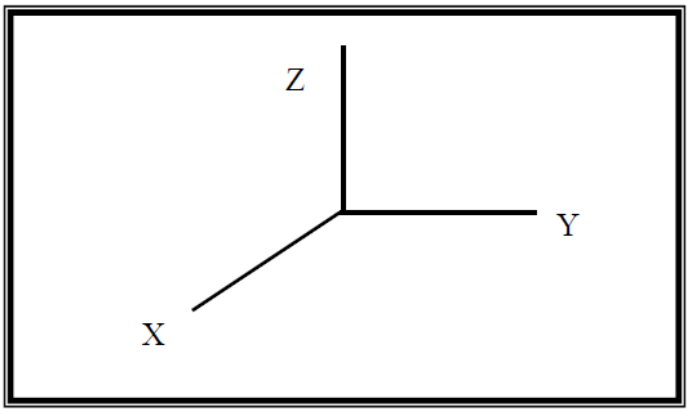
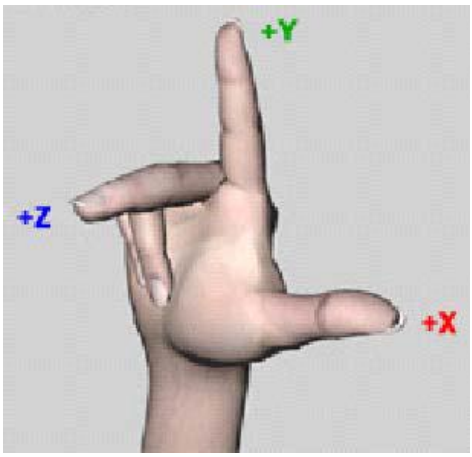


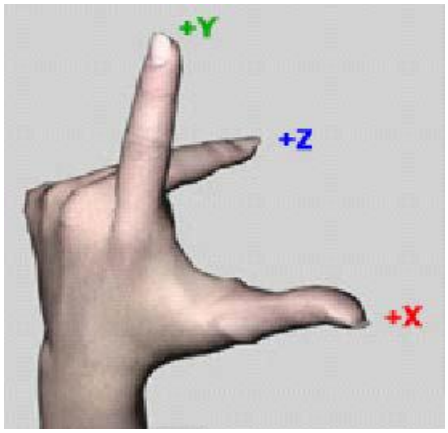

**Figure 4.2 3D Coordinate System**.

## 4.3 3D Coordinate Systems

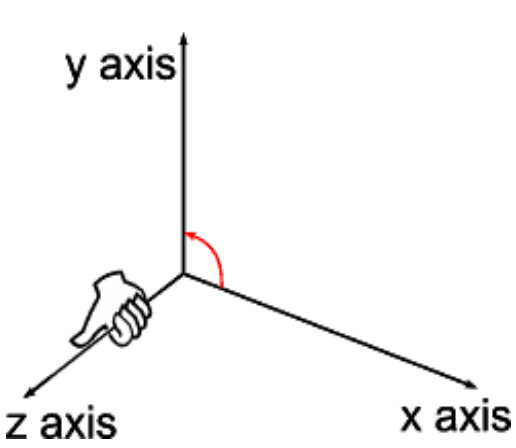
Three dimension system has three axis x, y, z. The orientation of a 3D coordinate system is of two types. **Right-handed system** and **left-handed system**. In the right -handed system thumb of right- hand points to positive z-direction and left- hand system thumb point to negative two directions.

For left-handed coordinates the left thumb points along the z axis in the positive direction and the curled fingers of the left hand represent a motion from the first or x axis to the second or y axis. When viewed from the top or z axis the system is clockwise. Figure 4.3 show 3D Coordinate Systems. A point can be described by triple ( X , Y , Z ) of coordinate values.




| **Right Hand** coordinate system | **Left Hand** coordinate system |
|---|---|



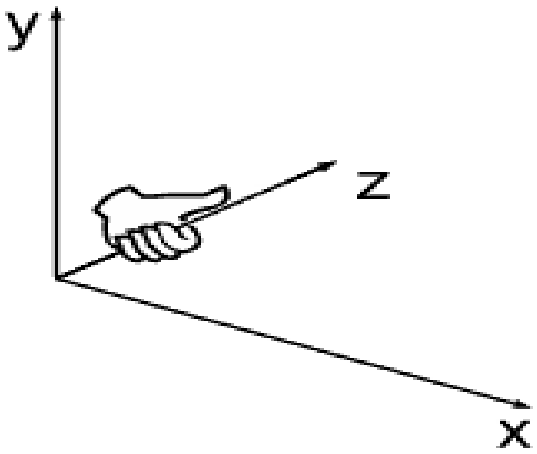
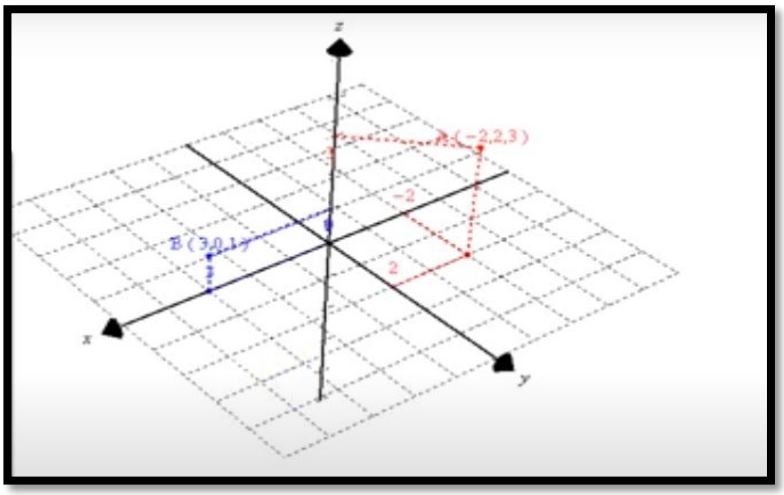
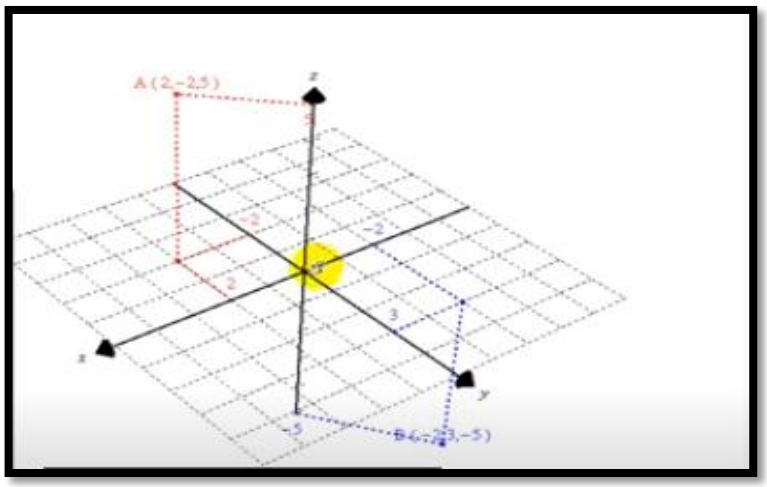




**Figure 4.3 3D Coordinate Systems**

**Exampl 1 : Draw the point A(-2,2,3) and B(3,0,1) using 3D coordinate system?**




**Exampl 2 : Draw the point A(2,-2,5) and B(-2,3,-5) using 3D coordinate system?**



# Program 1: Matlab program to draw any 3D point.

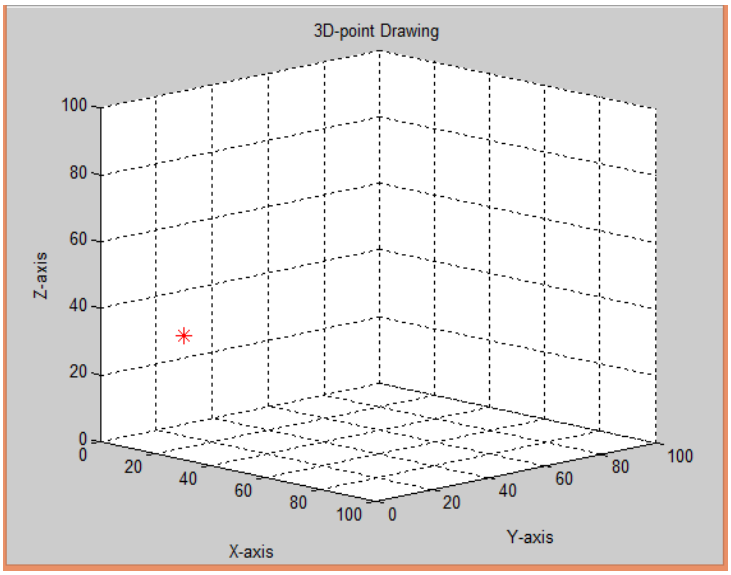
```
% 3D-Point Drawing
clc; clear all; close all
x=input ('enter x-value :');
y=input ('enter y-value :');
z=input ('enter z-value :');
axis ([0 100 0 100 0 100]);
hold on
grid on
plot3(x, y, z, 'r*','markersize',10)
xlabel('X-axis')
ylabel('Y-axis')
zlabel('Z-axis')
title ('3D-point Drawing')
hold off
```

## The Output of program:

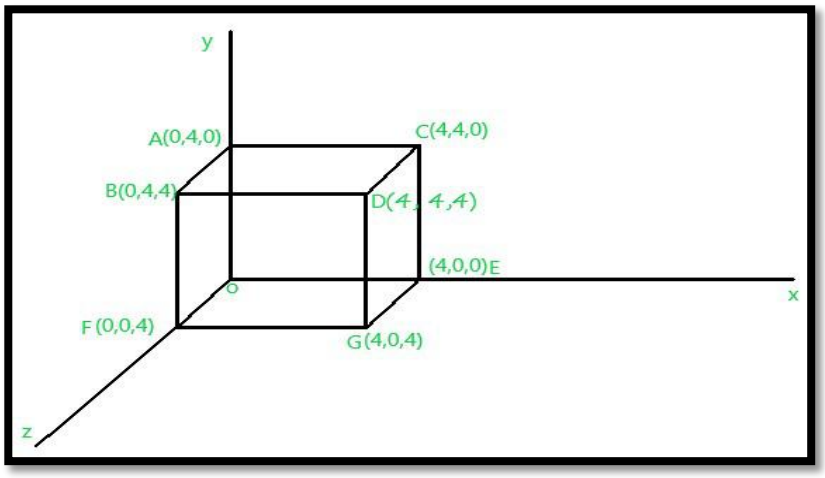
enter x-value : 10

enter y-value : 20

enter z-value :30



**Exampl 3 : Draw Cube by ploting the points A(0,4,0) ,B(0,4,4) C(4,4,0), D(4,4,4), E(4,0,0), F(0,0,4) and G(4,0,4) using 3D coordinate system?**




## 4.4 Types of Transformation

There are two types of transformation in computer graphics.

1) 2D transformation

2) 3D transformation

Types of 2D and 3D transformation:

1) Translation

2) Rotation

3) Scaling

4) Reflection

5) Shearing

## 4.4.1 3D Translation

3D Translation is **a process of moving an object from one position to another in a three dimensional plane**. Consider a point object O has to be moved from one position to another in a 3D plane. Figure 4.4 show 3D translation of point in computer graphics.

A point ( x , y , z ) is translated to a new position ( x1 , y1 , z1 ) by move it dx units in the X – direction and by units in the Y – direction and dz units in Z – direction. **Mathematically** this can be represented as:-

$x' = x + tx$

$y' = y + ty$

$z' = z + tz$

P is translated to P' by:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{P'} = \mathbf{T} \cdot \mathbf{P}$$

Given a Translation vector (Tx, Ty, Tz)-

- *tx* defines the distance the Xold coordinate has to be moved.
- *ty* defines the distance the Yold coordinate has to be moved.
- tz defines the distance the Zold coordinate has to be moved.

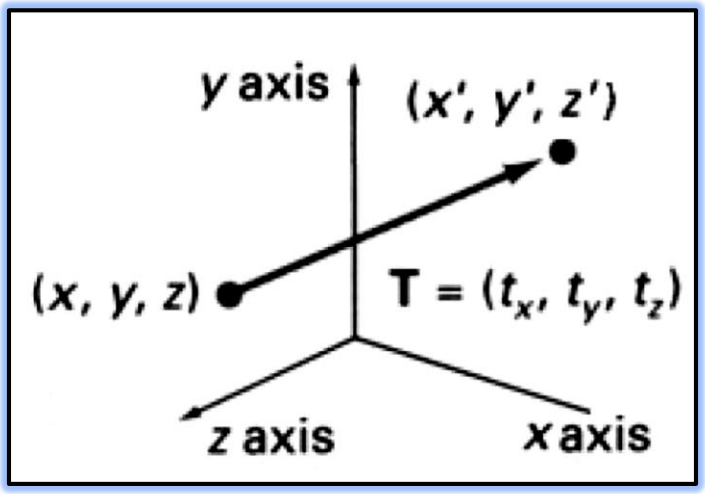
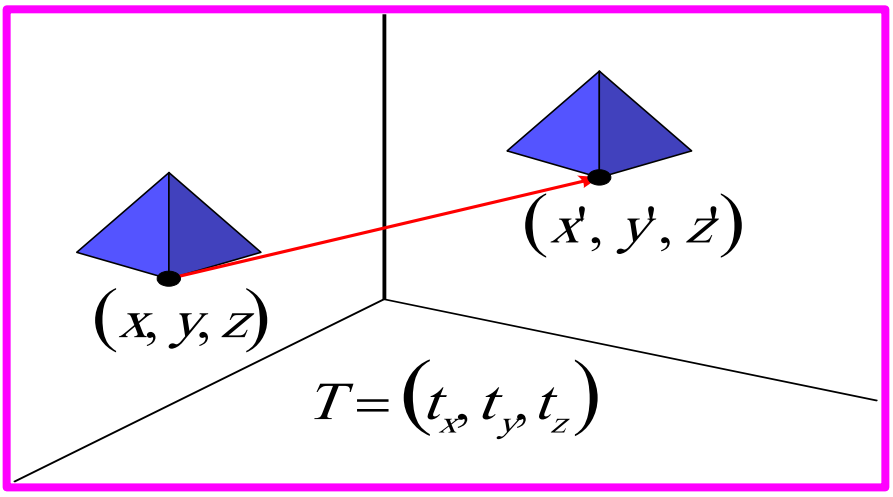



**Figure 4.4 3D Transltion of point**

An object is translated in 3D dimensional by transforming each of the defining points of the objects .

An Object represented as a set of polygon surfaces, is translated by translate each vertex of each surface and redraw the polygon facets in the new position.

Figue 4.5 show the transilation of an object.




**Figue 4.5 Transilation of an object.**

## 4.4.2 Inverse 3D Translation

These are also called as opposite transformations. If T is a translation matrix than inverse translation is representing using T-1. The inverse matrix is achieved using the opposite sign.

$$T^{-1}(t_x \quad t_y \quad t_z) = T(-t_x \quad -t_y \quad -t_z)$$

**Example 4:** Given a 3D object with coordinate points A(0, 3, 1), B(3, 3, 2), C(3, 0, 0), D(0, 0, 0). Apply the translation with the distance 1 towards X axis, 1 towards Y axis and 2 towards Z axis and obtain the new coordinates of the object.

**Solution:**

- Old coordinates of the object = A (0, 3, 1), B(3, 3, 2), C(3, 0, 0), D(0, 0, 0)
- Translation vector = ($T_x$, $T_y$, $T_z$) = (1, 1, 2)

**For Coordinates A(0, 3, 1)**

Let the new coordinates of A = ($X_{new}$, $Y_{new}$, $Z_{new}$).

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 0 + 1 = 1$
- $Y_{new} = Y_{old} + T_y = 3 + 1 = 4$
- $Z_{new} = Z_{old} + T_z = 1 + 2 = 3$

Thus, New coordinates of A = (1, 4, 3).

**For Coordinates B(3, 3, 2)**

Let the new coordinates of B = ($X_{new}$, $Y_{new}$, $Z_{new}$).

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 3 + 1 = 4$
- $Y_{new} = Y_{old} + T_y = 3 + 1 = 4$
- $Z_{new} = Z_{old} + T_z = 2 + 2 = 4$

Thus, New coordinates of B = (4, 4, 4).

**For Coordinates C(3, 0, 0)**

Let the new coordinates of C = ($X_{new}$, $Y_{new}$, $Z_{new}$).

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 3 + 1 = 4$

**For Coordinates D(0, 0, 0)**

Let the new coordinates of D = ($X_{new}$, $Y_{new}$, $Z_{new}$).

Applying the translation equations, we have-

- $X_{new} = X_{old} + T_x = 0 + 1 = 1$

- $Y_{new} = Y_{old} + T_y = 0 + 1 = 1$
- $Z_{new} = Z_{old} + T_z = 0 + 2 = 2$


- $Y_{new} = Y_{old} + T_y = 0 + 1 = 1$
- $Z_{new} = Z_{old} + T_z = 0 + 2 = 2$


Thus, New coordinates of C = (4, 1, 2).

Thus, New coordinates of D = (1, 1, 2).

Thus, New coordinates of the object = A (1, 4, 3), B(4, 4, 4), C(4, 1, 2), D(1, 1, 2).

# Program 2: Matlab program for Translation 3D-Point

```
% Translation 3D-Point
clc;clear all;close all
x=input('enter x-value ');
y=input('enter y-value ');
z=input('enter z-value ');

tx=input('enter Tx : ');
ty=input('enter Ty : ');
tz=input('enter Tz : ');

x1=x+tx;
y1=y+ty;
z1=z+tz;
axis([0 100 0 100 0 100]);
hold on
grid
plot3(x ,y ,z,'r*','linewidth',2,'markersize',10)
plot3(x1,y1,z1,'bo','linewidth',2,'markersize',10)
hold off
legend('before','after')
xlabel('X-axis')
ylabel('Y-axis')
zlabel('Z-axis')
title('Translation 3D-Point')
```

## The Output of program:

enter x-value 10

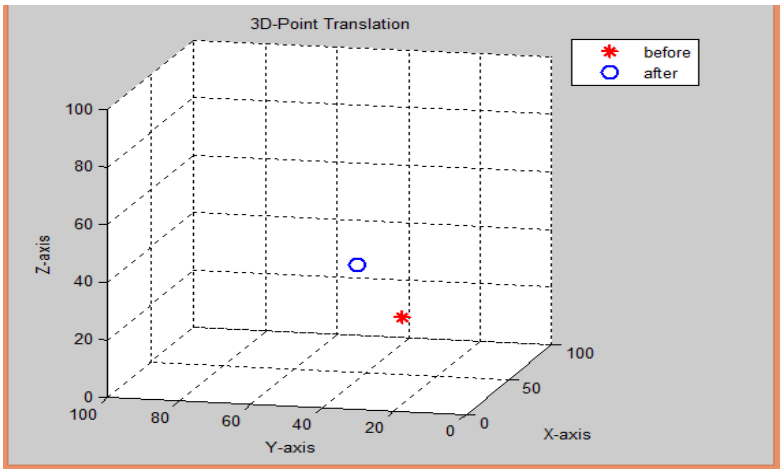
enter y-value 20

enter z-value 30

enter Tx : 30

enter Ty : 20

enter Tz : 10



# Program 3: Matlab program for Translation 3D-shape

```
% Translation TRANSFORMATION PROGRAM for 3D-shape
clc;clear all;close all
%Enter number of shape or object vertices
g=input('enter no. of object vertecis : ');
tx=input('enter Tx : ');
ty=input('enter Ty : ');
tz=input('enter Tz : ');
 for i=1:g
   x(i)=input('enter x-vlaue ');
   y(i)=input('enter y-vlaue ');
   z(i)=input('enter z-vlaue ');
   x1(i)=x(i)+tx;
   y1(i)=y(i)+ty;
   z1(i)=z(i)+tz;
end
axis([0 100 0 100 0 100]);
hold on
for i=1:g-1
  plot3([x(i) x(i+1)],[y(i) y(i+1)],[z(i) z(i+1)],...
      'r^-','linewidth',1,'markersize',2)
plot3([x1(i) x1(i+1)],[y1(i) y1(i+1)],[z1(i) z1(i+1)],...
      'go-','linewidth',1,'markersize',2)
  end
```

```
plot3([x(i+1) x(g-i)],[y(i+1) y(g-i)],[z(i+1) z(g-i)],...
   'r^-','linewidth',3,'markersize',2)
plot3([x1(i+1) x1(g-i)],[y1(i+1) y1(g-i)],[z1(i+1) z1(g-i)],...
   'go-','linewidth',3,'markersize',2)
legend('before','after')
xlabel('X-axis')
ylabel('y-axis')
zlabel('z-axis')
grid on
title('Translation any 3D-shape')
```

## The Output of program:

enter no. of object vertecis : 2

enter Tx : 20

enter Ty : 30

enter Tz : 20

enter x-vlaue 10

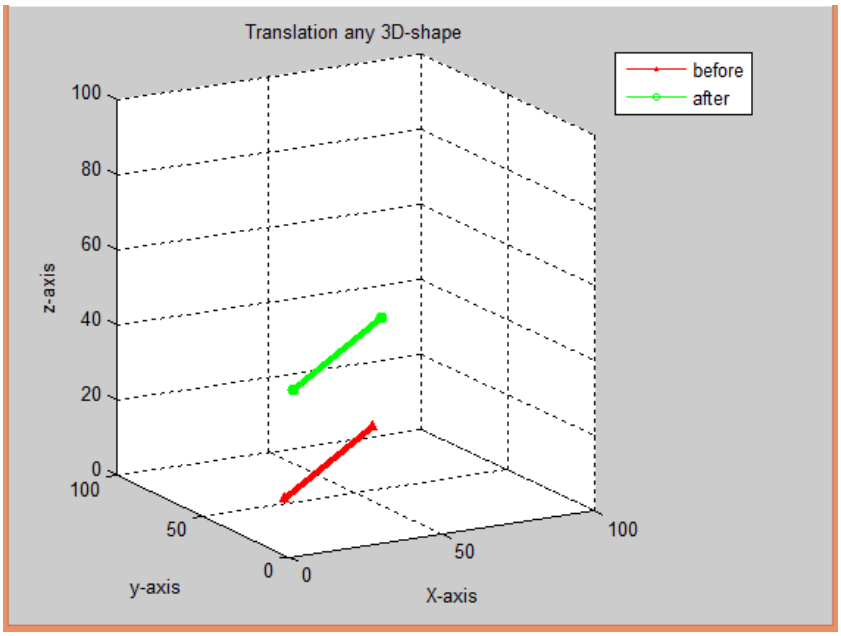
enter y-vlaue 20

enter z-vlaue 10

enter x-vlaue 50

enter y-vlaue 40

enter z-vlaue 20



### 4.4.3 3D Rotation

In general, rotations are specified by a rotation axis and an angle. In two-dimensions there is only one choice of a rotation axis that leaves points in the plane.

The easiest rotation axes are those that parallel to the coordinate axis.

Positive rotation angles produce counterclockwise rotations about a coordinate axix, if we are looking along the positive half of the axis toward the coordinate origin. The Figure 4.6 show Axis Rotation.

In 3 dimensions, there are 3 possible types of rotation:

1- Z- axis Rotation.

2- Y-axis Rotation.

3- X-axis Rotation.

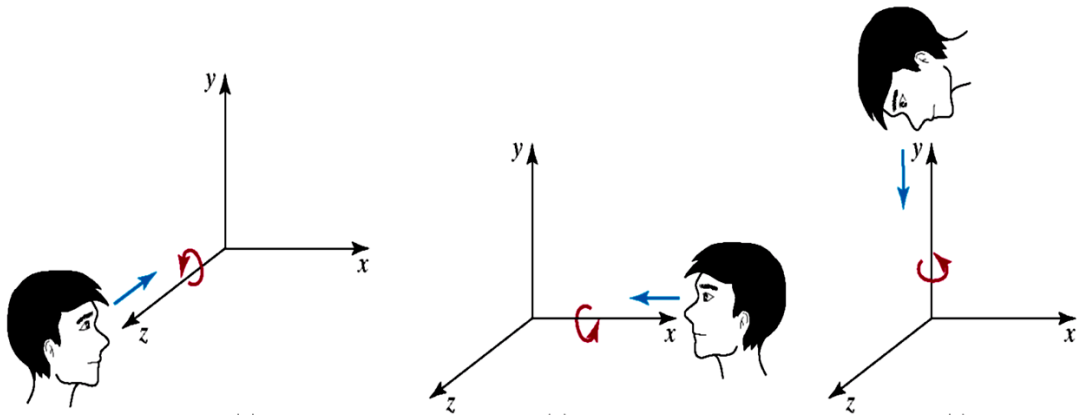



**Figure 4.6 Axis Rotation**

**1. Z-axis Rotation:** For z axis same as 2D rotation, Figure 4.7 show Z-axis Rotation

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

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

$z' = z$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{P'} = \mathbf{R_z}\,(\theta) \cdot \mathbf{P}$$

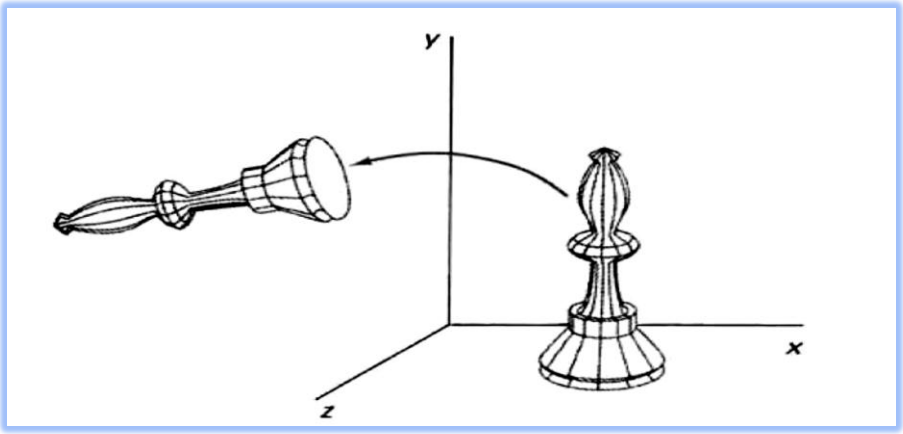



**Figure 4.7 Z-axis Rotation**

## 2. X-axis rotation:

The equation for X-axis rotaion as follow and Figure 4.8 show X-axis Rotation

$x' = x$

$y' = y\cos\theta - z\sin\theta$

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

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{P}' = \mathbf{R}_x(\theta) \cdot \mathbf{P}$$

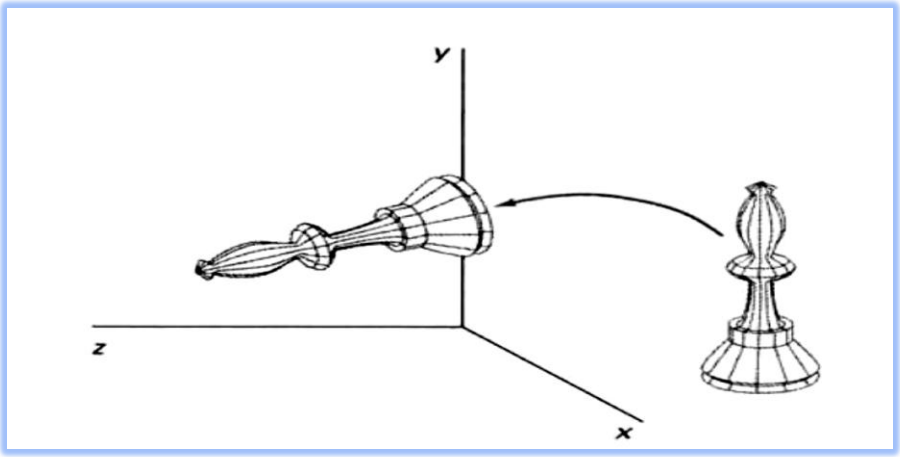



**Figure 4.8 X-axis Rotation**

### 3. Y-axis rotation:

The equation for Y-axis rotaion as follow and Figure 4.9 show Y-axis Rotation

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

$y' = y$

$z' = z \cos\theta - x \sin\theta$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{P}' = \mathbf{R}_y(\theta) \cdot \mathbf{P}$$

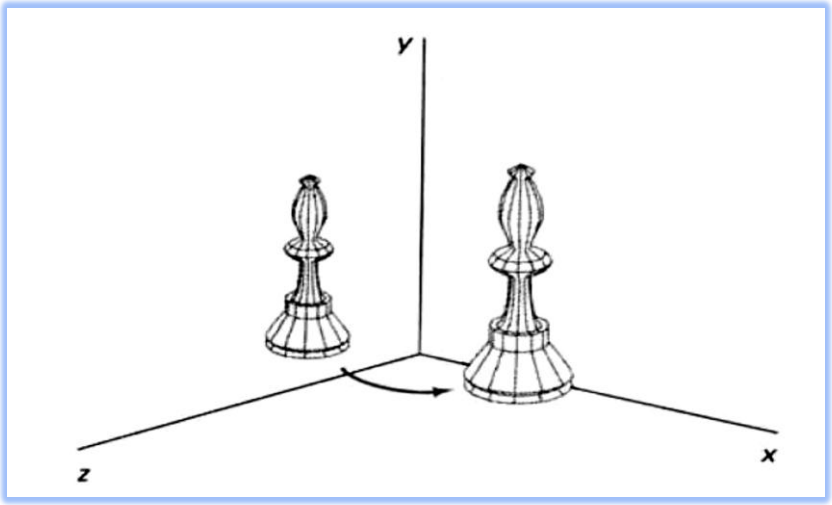



**Figure 4.9 Y-axis Rotation**

**Example 5:**Given a homogeneous point (1, 2, 3). Apply rotation 90 degree towards X, Y and Z axis and find out the new coordinate points.

**Solution**

Old coordinates = (Xold, Yold, Zold) = (1, 2, 3)

Rotation angle = θ = 90º

**For X-Axis Rotation-**

Let the new coordinates after rotation = (Xnew, Ynew, Znew).

Applying the rotation equations, we have

Xnew = Xold = 1

Ynew = Yold x cosθ – Zold x sinθ = 2 x cos90° – 3 x sin90° = 2 x 0 – 3 x 1 = -3

Znew = Yold x sinθ + Zold x cosθ = 2 x sin90° + 3 x cos90° = 2 x 1 + 3 x 0 = 2

Thus, New coordinates after rotation = (1, -3, 2).

**For Y-Axis Rotation-**

Let the new coordinates after rotation = (Xnew, Ynew, Znew).

Applying the rotation equations, we have-

Xnew = Zold x sinθ + Xold x cosθ = 3 x sin90° + 1 x cos90° = 3 x 1 + 1 x 0 = 3

Ynew = Yold = 2

Znew = Yold x cosθ – Xold x sinθ = 2 x cos90° – 1 x sin90° = 2 x 0 – 1 x 1 = -1

Thus, New coordinates after rotation = (3, 2, -1).

**For Z-Axis Rotation-**

Let the new coordinates after rotation = (Xnew, Ynew, Znew).

Applying the rotation equations, we have-

Xnew = Xold x cosθ – Yold x sinθ = 1 x cos90° – 2 x sin90° = 1 x 0 – 2 x 1 = -2

Ynew = Xold x sinθ + Yold x cosθ = 1 x sin90° + 2 x cos90° = 1 x 1 + 2 x 0 = 1

Znew = Zold = 3

Thus, New coordinates after rotation = (-2, 1, 3).

# Program 4: matlab program for rotation 3d-shape

```
% ROTATION TRANSFORMATION PROGRAM for 3D-shape
clc;clear all;close all
%Enter number of shape or object vertices
g=input ('enter no. of object vertecis : ');
%Enter Rotation angle
t=input ('enter the angle : ');
disp('Enter 1 , For X-axis rotatation ');
disp('Enter 2 , For Y-axis rotatation ');
disp('Enter 3 , For Z-axis rotatation ');
m=input ('enter the axis rotation value 1-3: ');
% enter x-value & y-value & z-value for All Shape vertices
 if m ==1 || m==2 || m==3
      for i=1: g
       x(i)=input ('enter x-value ');
      y(i)=input ('enter y-value ');
      z(i)=input ('enter z-value ');
      switch m
         case 1
            x1(i)=x(i);
            y1(i)=round ((y(i))*cos(t)-(z(i))*sin (t));
            z1(i)=round ((y(i))*cos(t)+(z(i))*sin(t));
        case 2
            x1(i)=round ((x(i))*cos(t)+(z(i))*sin (t));
            y1(i)=round(y(i));
            z1(i)=round ((z(i))*cos(t)-(x(i))*sin(t));
         case 3
            x1(i)=round ((x(i))*cos(t)-(y(i))*sin (t));
            y1(i)=round ((y(i))*cos(t)+(x(i))*sin(t));
            z1(i)=z(i);
   end
  end
```

```
axis([0 100 0 100 0 100]);
   hold on
   grid on
   for i=1: g-1
      plot3([x(i) x(i+1)],[y(i) y(i+1)],[z(i) z(i+1)],...
         'r^-','linewidth',2,'markersize',5)
      plot3([x1(i) x1(i+1)],[y1(i) y1(i+1)],[z1(i) z1(i+1)],...
         'go-','linewidth',2,'markersize',5)
   end
   plot3([x(1)  x(g)], [y(1)  y(g)],[z(1)  z(g)],...
      'r^-','linewidth',2,'markersize',5)
   plot3([x1(1) x1(g)],[y1(1) y1(g)],[z1(1) z1(g)],...
      'go-','linewidth',2,'markersize',5)
    legend('Befor Rotation','After Rotation')
   xlabel('X-axis')
   ylabel('Y-axis')
   zlabel('Z-axis')
   title('Rotation 3D-Shape')
   hold off
else
   disp('Invalid choice ');
end
```

## The Output of program:

enter no. of object vertecis : 5

enter the angle : 30

Enter 1 , For X-axis rotatation

Enter 2 , For Y-axis rotatation

Enter 3 , For Z-axis rotatation

enter the axis rotation value 1-3: 3

enter x-value 10

enter y-value 20

enter z-value 10

enter x-value 10

enter y-value 50

enter z-value 20

enter x-value 40

enter y-value 50

enter z-value 25

enter x-value 40

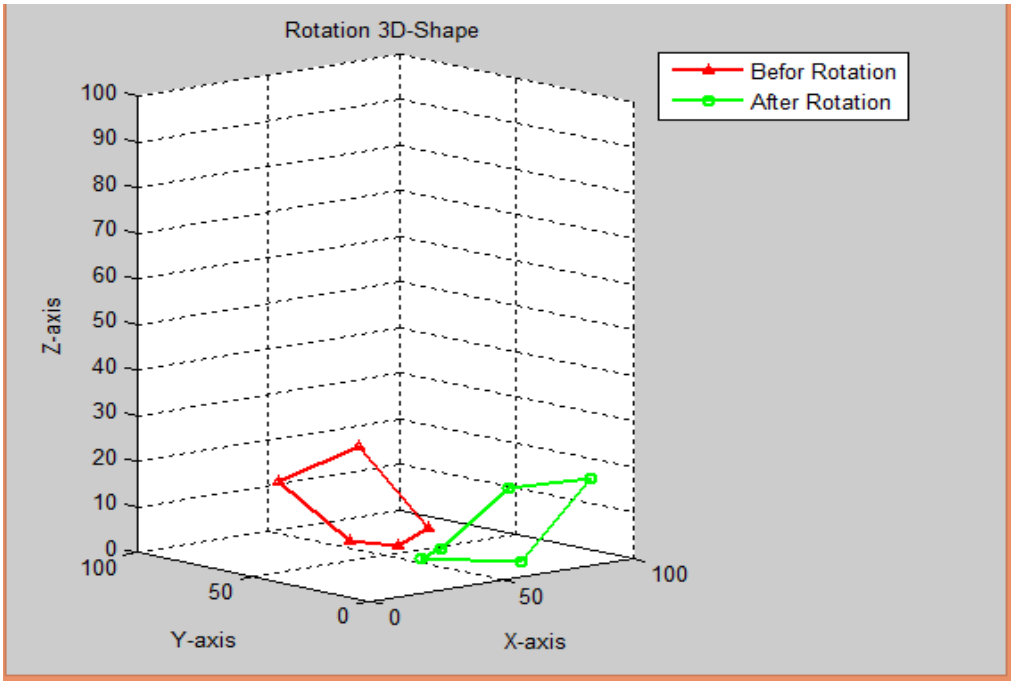
enter y-value 20

enter z-value 10

enter x-value 15

enter y-value 5

enter z-value 10




## 4.4.4 3D Scaling

In computer graphics, scaling is a process of modifying or altering the size of objects.

- Scaling may be used to increase or reduce the size of object.
- Scaling subjects the coordinate points of the original object to change.
- Scaling factor determines whether the object size is to be increased or reduced.
- If scaling factor > 1, then the object size is increased.
- If scaling factor < 1, then the object size is reduced.

## 1. The scale an object from a origin point:

Changes the size of the object and repositions the object relative to the coordinate origin. Figuer 4.10 show the scaling of an object.

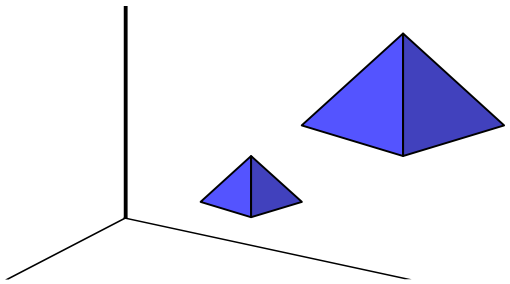
The equations for scaling

$$x' = x \cdot sx$$

$S_{sx,sy,sz}$ $$y' = y \cdot sy$$

$$z' = z \cdot sz$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \mathbf{P' = S \cdot P}$$



**Figuer 4.10 The Scaling of an Object.**

## 2. The Scale About any fixed point:

To scale an object from fixed point ( xp , yp , zp ), we perform the following

**three steps,** Figure 4.11 show scaling about any fixed point.

$$\boldsymbol{T(x_f, y_f, z_f).S(s_x, s_y, s_z).T(-x_f, -y_f, -z_f)}$$

$$= \begin{bmatrix} \boldsymbol{s_x} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{(1-s_x)x_f} \\ \boldsymbol{0} & \boldsymbol{s_y} & \boldsymbol{0} & \boldsymbol{(1-s_y)y_f} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{s_z} & \boldsymbol{(1-s_z)z_f} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{1} \end{bmatrix}$$

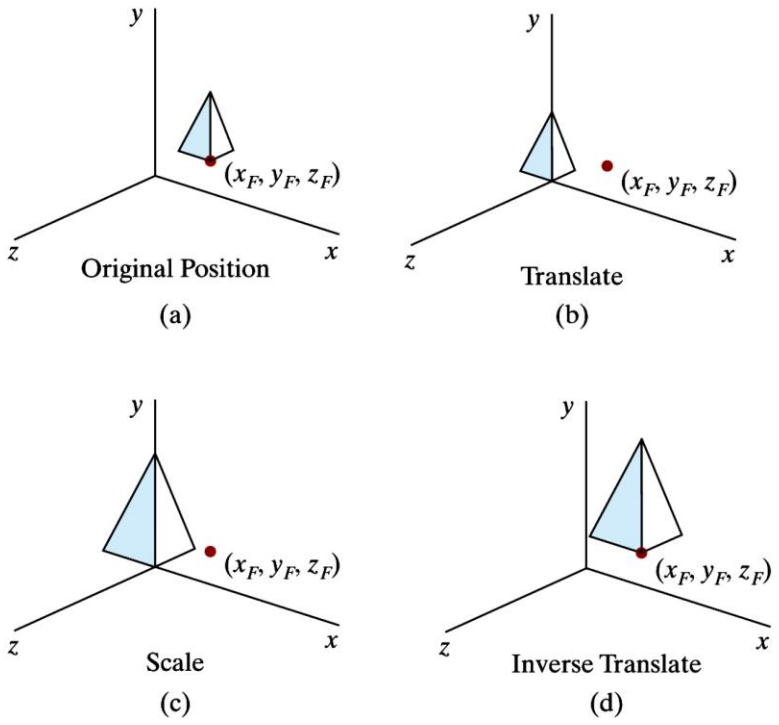



**Figure 4.11 Scaling about any Fixed Point**

**Example 6:** Given a 3D object with coordinate points A(0, 3, 3), B(3, 3, 6), C(3, 0, 1), D(0, 0, 0). Apply the scaling parameter 2 towards X axis, 3 towards Y axis and 3 towards Z axis and obtain the new coordinates of the object.

**Solution:**

Old coordinates of the object = A (0, 3, 3), B(3, 3, 6), C(3, 0, 1), D(0, 0, 0)

Scaling factor along X axis = 2

Scaling factor along Y axis = 3

Scaling factor along Z axis = 3

**For Coordinates A(0, 3, 3)**

Let the new coordinates of A after scaling = (Xnew, Ynew, Znew).

Applying the scaling equations, we have-

Xnew = Xold x Sx = 0 x 2 = 0

Ynew = Yold x Sy = 3 x 3 = 9

Znew = Zold x Sz = 3 x 3 = 9

Thus, New coordinates of corner A after scaling = (0, 9, 9).

**For Coordinates B(3, 3, 6)**

Let the new coordinates of B after scaling = (Xnew, Ynew, Znew).

Applying the scaling equations, we have-

Xnew = Xold x Sx = 3 x 2 = 6

Ynew = Yold x Sy = 3 x 3 = 9

Znew = Zold x Sz = 6 x 3 = 18

Thus, New coordinates of corner B after scaling = (6, 9, 18).

**For Coordinates C(3, 0, 1)**

Let the new coordinates of C after scaling = (Xnew, Ynew, Znew).

Applying the scaling equations, we have-

Xnew = Xold x Sx = 3 x 2 = 6

Ynew = Yold x Sy = 0 x 3 = 0

Znew = Zold x Sz = 1 x 3 = 3

Thus, New coordinates of corner C after scaling = (6, 0, 3).

**For Coordinates D(0, 0, 0)**

Let the new coordinates of D after scaling = (Xnew, Ynew, Znew).

Applying the scaling equations, we have-

Xnew = Xold x Sx = 0 x 2 = 0

Ynew = Yold x Sy = 0 x 3 = 0

Znew = Zold x Sz = 0 x 3 = 0

Thus, New coordinates of corner D after scaling = (0, 0, 0).

# Program 5: Matlab Program for Scaling 3D-Shape

```
% SCALING TRANSFORMATION PROGRAM for 3D-shape
clc;clear all;close all
%Enter number of shape or object vertices
g=input('enter no. of Shape vertecis : ');
%enter SCALING factor sx & sy & sz
sx=input('enter Sx : ');
sy=input('enter Sy : ');
sz=input('enter Sz : ');
% enter x-value & y-value & z-value for All Shape vertecis
 for i=1:g
   x(i)=input('enter x-value ');
   y(i)=input('enter y-value ');
   z(i)=input('enter z-value ');
   x1(i)=x(i)*sx;
   y1(i)=y(i)*sy;
   z1(i)=z(i)*sz;
end

axis([0 100 0 100 0 100]);
hold on
grid on
for i=1:g-1
   plot3([x(i) x(i+1)],[y(i) y(i+1)],[z(i) z(i+1)],...
      'g^-','LineWidth',3,'markersize',5)
   plot3([x1(i) x1(i+1)],[y1(i) y1(i+1)],[z1(i) z1(i+1)],...
      'mo-','LineWidth',3,'markersize',5)
end
   plot3([x(1) x(g)],[y(1) y(g)],[z(1) z(g)],...
      'g^-','LineWidth',3,'markersize',5)
   plot3([x1(1) x1(g)],[y1(1) y1(g)],[z1(1) z1(g)],...
      'mo-','LineWidth',3,'markersize',5)

   legend('Befor Scaling','After Scaling')
   xlabel('X-axis')
   ylabel('y-axis')
   title('Scaling 3D-Shape')
```

## The Output of program:

enter no. of Shape vertecis : 3

enter Sx : 2

enter Sy : 2

enter Sz : 2

enter x-value 10

enter y-value 20

enter z-value 30

enter x-value 35

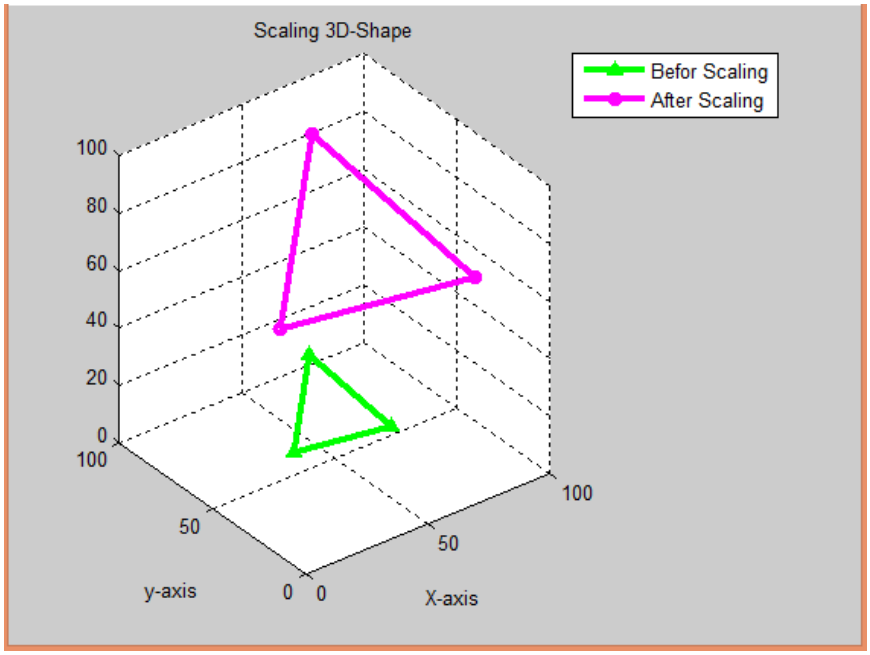
enter y-value 44

enter z-value 44

enter x-value 50

enter y-value 20

enter z-value 25



## 4.4.5 3D Reflections

Reflection in 3D space is quite similar to the reflection in 2D space, but a single difference is there in 3D, here we have to deal with three axes (x, y, z). Reflection is nothing but a mirror image of an object. Three kinds of Reflections are possible in 3D space:

- Reflection relative to XY plane
- Reflection relative to YZ plane
- Reflection relative to XZ plane

# 1. Reflection Relative to XY Plane:

This reflection is achieved by using the following reflection equations-

- Xnew = Xold
- Ynew = Yold
- Znew = -Zold

In Matrix form, the above reflection equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

**3D Reflection Matrix**

**(Reflection Relative to XY plane)**

## 2. Reflection Relative to YZ Plane:

This reflection is achieved by using the following reflection equations-

- Xnew = -Xold
- Ynew = Yold
- Znew = Zold

In Matrix form, the above reflection equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

**3D Reflection Matrix**

**(Reflection Relative to YZ plane)**

## 3. Reflection Relative to XZ Plane:

This reflection is achieved by using the following reflection equations-

- Xnew = Xold
- Ynew = -Yold
- Znew = Zold

In Matrix form, the above reflection equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

**3D Reflection Matrix**

**(Reflection Relative to XZ plane)**

**Example 7**: Given a 3D triangle with coordinate points A(3, 4, 1), B(6, 4, 2), C(5, 6, 3). Apply the reflection on the XY plane and find out the new coordinates of the object.

**Solution**

- Old corner coordinates of the triangle = A (3, 4, 1), B(6, 4, 2), C(5, 6, 3)
- Reflection has to be taken on the XY plane

**For Coordinates A(3, 4, 1)**

Let the new coordinates of corner A after reflection = ($X_{new}$, Ynew, Znew).

Applying the reflection equations, we have-

- $X_{new} = X_{old} = 3$
- $Y_{new} = Y_{old} = 4$
- $Z_{new} = -Z_{old} = -1$

Thus, New coordinates of corner A after reflection = (3, 4, -1).

## For Coordinates B(6, 4, 2)

Let the new coordinates of corner B after reflection = (Xnew, Ynew, Znew).

Applying the reflection equations, we have-

- Xnew = Xold = 6
- Ynew = Yold = 4
- Znew = -Zold = -2

Thus, New coordinates of corner B after reflection = (6, 4, -2).

## For Coordinates C(5, 6, 3)

Let the new coordinates of corner C after reflection = (Xnew, Ynew, Znew).

Applying the reflection equations, we have-

- Xnew = Xold = 5
- Ynew = Yold = 6
- Znew = -Zold = -3

Thus, New coordinates of corner C after reflection = (5, 6, -3).

Thus, New coordinates of the triangle after reflection = A (3, 4, -1), B(6, 4, -2), C(5, 6, -3).

# Program 6: Matlab Program for Reflection 3D-Shape

```
% Reflection TRANSFORMATION PROGRAM for 3D-shape
clc; clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of shape vertices: ');
% enter x-value & y-value & z-value for All Shape vertecis
for i=1: g
   x(i)=input ('enter x-value:');
   y(i)=input ('enter y-value:');
   z(i)=input ('enter z-value: ');
end
%Reflection Types
disp ('Reflection relative to XY plane ,enter 1');
disp ('Reflection relative to YZ plane ,enter 2');
disp ('Reflection relative to XZ plane ,enter 3');
b=input ('Enter type of Reflection : ');
switch b
   case 1
      for i=1: g
         x1(i)=x(i);
         y1(i)=y(i);
         z1(i)=-z(i);
      end
   case 2
     for i=1: g
         x1(i)=-x(i);
         y1(i)=y(i);
         z1(i)=z(i);
     end
   case 3
     for i=1: g
         x1(i)=x(i);
         y1(i)=-y(i);
         z1(i)=z(i);
     end
end
```

```
axis([-100 130 -100 130 -100 130]);
hold on
grid on
for i=1:g-1
   plot3([x(i) x(i+1)], [y(i) y(i+1)], [z(i) z(i+1)],...
      'r*-','linewidth',3,'markersize',5)
   plot3([x1(i) x1(i+1)], [y1(i) y1(i+1)], [z1(i) z1(i+1)],...
      'g*-','linewidth',3,'markersize',5)
end
plot3([x(i+1) x(g-i)], [y(i+1) y(g-i)], [z(i+1) z(g-i)],...
   'r*-','linewidth',3,'markersize',5)
plot3([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], [z1(i+1) z1(g-i)],...
   'g*-','linewidth',3,'markersize',5)
legend ('Before Reflection', 'After Reflection')
xlabel('X-axis')
ylabel('Y-axis')
zlabel('Z-axis')
switch b
   case 1
      title ('Reflection 3D-shape in XY Plane')
   case 2
      title ('Reflection 3D-shape in YZ Plane')
   case 3
      title ('Reflection 3D-shape in XZ Plane')
end
```

## The Output of program:

enter no. of shape vertices: 3

enter x-value:10

enter y-value:20

enter z-value: 30

enter x-value:30

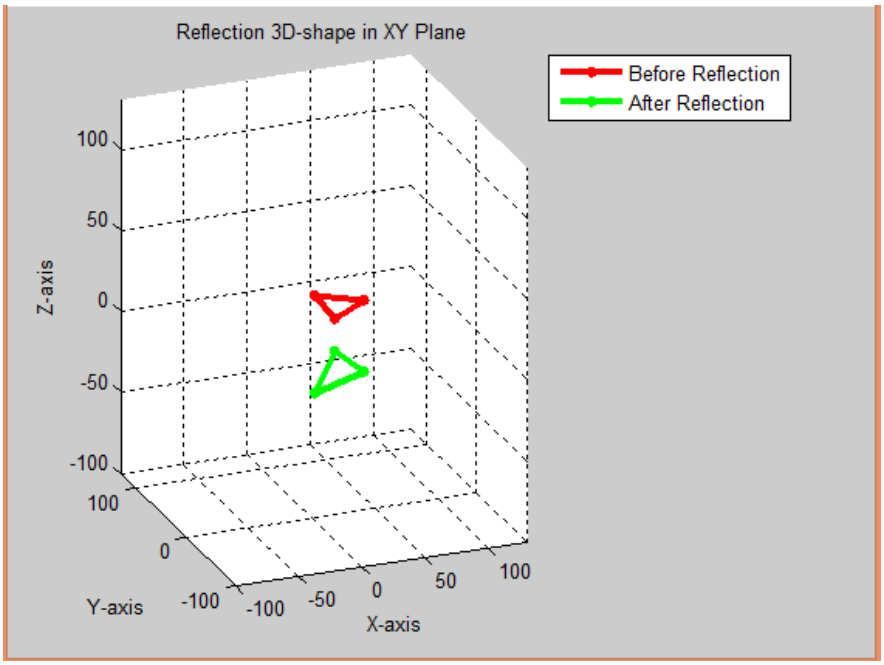
enter y-value:30

enter z-value: 10

enter x-value:50

enter y-value:20

enter z-value: 22




Reflection relative to XY plane ,enter 1

Reflection relative to YZ plane ,enter 2

Reflection relative to XZ plane ,enter 3

Enter type of Reflection : 1

## The Output of program:

enter no. of shape vertices: 3

enter x-value:10

enter y-value:20

enter z-value: 30

enter x-value:30

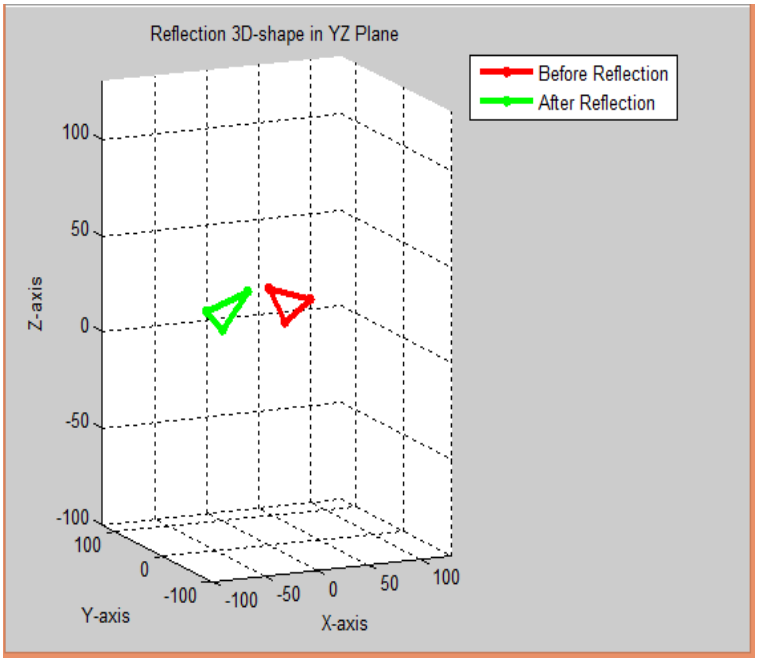
enter y-value:30

enter z-value: 10

enter x-value:50

enter y-value:20

enter z-value: 22




Reflection relative to XY plane ,enter 1

Reflection relative to YZ plane ,enter 2

Reflection relative to XZ plane ,enter 3

Enter type of Reflection : 2

## The Output of program:

enter no. of shape vertices: 3

enter x-value:10

enter y-value:20

enter z-value: 30

enter x-value:30

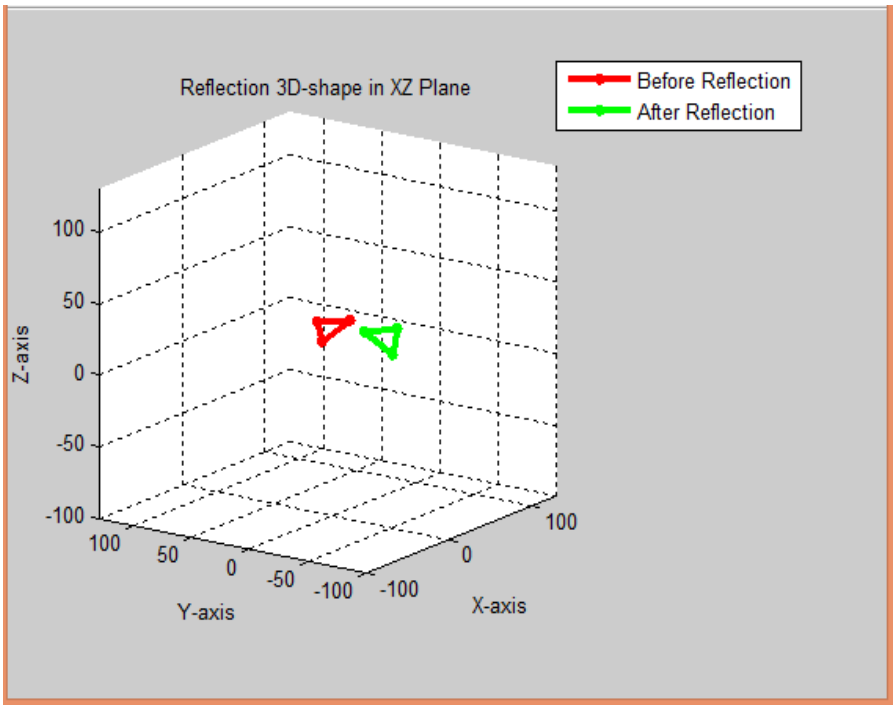
enter y-value:30

enter z-value: 10

enter x-value:50

enter y-value:20

enter z-value: 22




Reflection relative to XY plane ,enter 1

Reflection relative to YZ plane ,enter 2

Reflection relative to XZ plane ,enter 3

Enter type of Reflection : 3

## 4.4.6 3D Shearing

In Computer graphics, 3D Shearing is an ideal technique to change (Modify)

the shape of an existing object in a three dimensional plane. In a three dimensional plane, the object size can be changed along X direction, Y direction as well as Z direction.

**So, there are three versions of shearing-**

**1.** Shearing in X direction

**2.** Shearing in Y direction

**3.** Shearing in Z direction

## 1. Shearing in X Axis

Shearing in X axis is achieved by using the following shearing equations-

- Xnew = Xold
- Ynew = Yold + Shy x Xold
- Znew = Zold + Shz x Xold

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ Sh_y & 1 & 0 & 0 \\ Sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

**3D Shearing Matrix**

**(In X axis)**

## 2. Shearing in Y Axis

Shearing in Y axis is achieved by using the following shearing equations-

- Xnew = Xold + Shx x Yold
- Ynew = Yold
- Znew = Zold + Shz x Yold

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & Sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

**3D Shearing Matrix**

**(In Y axis)**

## 3. Shearing in Z Axis

Shearing in Z axis is achieved by using the following shearing equations-

- Xnew = Xold + Shx x Zold
- Ynew = Yold + Shy x Zold
- Znew = Zold

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & Sh_x & 0 \\ 0 & 1 & Sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

**3D Shearing Matrix**

**(In Z axis)**

**Example 8:** Given a 3D triangle with points (0, 0, 0), (1, 1, 2) and (1, 1, 3). Apply shear parameter 2 on X axis, 2 on Y axis and 3 on Z axis and find out the new coordinates of the object.

**Solution**

Given-

- Old corner coordinates of the triangle = A (0, 0, 0), B(1, 1, 2), C(1, 1, 3)
- Shearing parameter towards X direction ($Sh_X$) = 2
- Shearing parameter towards Y direction ($Sh_Y$) = 2
- Shearing parameter towards Y direction ($Sh_Z$) = 3

**Shearing in X Axis-**

**For Coordinates A(0, 0, 0)**

Let the new coordinates of corner A after shearing = ($X_{new}$, $Y_{new}$, $Z_{new}$).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 0$
- $Y_{new} = Y_{old} + Sh_Y \times X_{old} = 0 + 2 \times 0 = 0$
- $Z_{new} = Z_{old} + Sh_Z \times X_{old} = 0 + 3 \times 0 = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

## For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = (Xnew, Ynew, Znew).

Applying the shearing equations, we have-

- Xnew = Xold = 1
- Ynew = Yold + Shy x Xold = 1 + 2 x 1 = 3
- Znew = Zold + Shz x Xold = 2 + 3 x 1 = 5

Thus, New coordinates of corner B after shearing = (1, 3, 5).

## For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = (Xnew, Ynew, Znew).

Applying the shearing equations, we have-

- Xnew = Xold = 1
- Ynew = Yold + Shy x Xold = 1 + 2 x 1 = 3
- Znew = Zold + Shz x Xold = 3 + 3 x 1 = 6

Thus, New coordinates of corner C after shearing = (1, 3, 6).

Thus, New coordinates of the triangle after shearing in X axis = A (0, 0, 0), B(1, 3, 5), C(1, 3, 6).

## For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing = (Xnew, Ynew, Znew).

Applying the shearing equations, we have-

- $X_{new} = X_{old} + Sh_x \times Z_{old} = 0 + 2 \times 0 = 0$
- $Y_{new} = Y_{old} + Sh_y \times Z_{old} = 0 + 2 \times 0 = 0$
- $Z_{new} = Z_{old} = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

## For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = (Xnew, Ynew, Znew).

Applying the shearing equations, we have-

- Xnew = Xold + Shx x Zold = 1 + 2 x 2 = 5
- Ynew = Yold + Shy x Zold = 1 + 2 x 2 = 5
- Znew = Zold = 2

Thus, New coordinates of corner B after shearing = (5, 5, 2).

## For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = (Xnew, Ynew, Znew).

Applying the shearing equations, we have-

- Xnew = Xold + Shx x Zold = 1 + 2 x 3 = 7
- Ynew = Yold + Shy x Zold = 1 + 2 x 3 = 7
- Znew = Zold = 3

Thus, New coordinates of corner C after shearing = (7, 7, 3).

Thus, New coordinates of the triangle after shearing in Z axis = A (0, 0, 0), B(5, 5, 2), C(7, 7, 3)

# Program 7: Matlab Program for Shearing 3D-Shape

```
% SHEARING TRANSFORMATION PROGRAM for 3D-shape
clc; clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of shape vertices : ');
% enter x-value & y-value & z-value for All Shape vertecis
 for i=1: g
   x(i)=input ('enter x-value: ');
   y(i)=input ('enter y-value: ');
   z(i)=input ('enter z-value: ');
 end
%enter Shearing factor shx & shy & shz
shx=input ('enter Shearing factor Shx : ');
shy=input ('enter Shearing factor Shy : ');
shz=input('enter Shearing factor Shz : ');

disp ('Shearing the x-axis ,enter 1');
disp ('Shearing the y-axis ,enter 2');
disp ('Shearing the z-axis ,enter 3');
b=input('enter version of Shearing 1 or 2 or 3: ');
switch b
   case 1
      for i=1: g
         x1(i)=x(i);
         y1(i) = y(i) + shy * x(i);
         z1(i) = z(i) + shz * x(i);

      end
   case 2
     for i=1: g
         x1(i) = x(i) + shx * y(i);
         y1(i) = y(i);
         z1(i) = z(i) + shz * y(i);
     end
```

```
   case 3
     for i=1: g
         x1(i) = x(i) + shx * z(i);
         y1(i) = y(i) + shy * z(i);
         z1(i) = z(i);
     end
end
axis([0 100 0 100 0 100]);
hold on
grid on
for i=1:g-1
   plot3([x(i) x(i+1)], [y(i) y(i+1)], [z(i) z(i+1)],...
      'r^-','linewidth',3,'markersize',5)
   plot3([x1(i) x1(i+1)], [y1(i) y1(i+1)], [z1(i) z1(i+1)],...
      'g^-','linewidth',3,'markersize',5)
end

plot3([x(i+1) x(g-i)], [y(i+1) y(g-i)], [z(i+1) z(g-i)],...
   'r^-','linewidth',3,'markersize',5)
plot3([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], [z1(i+1) z1(g-i)],...
   'g^-','linewidth',3,'markersize',5)
legend ('Before Shear', 'After Shear')
xlabel('X-axis')
ylabel('Y-axis')
zlabel('Z-axis')
switch b
   case 1
      title ('Shearing 3D-shape in x-axis')
   case 2
      title ('Shearing 3D-shape in y-axis')
   case 3
      title ('Shearing 3D-shape in z-axis')
end
```

**The Output of program:**

enter no. of shape vertices : 3

enter x-value: 10

enter y-value: 20

enter z-value: 30

enter x-value: 30

enter y-value: 30

enter z-value: 10

enter x-value: 50

enter y-value: 20

enter z-value: 22

enter Shearing factor Shx : 2

enter Shearing factor Shy : 2

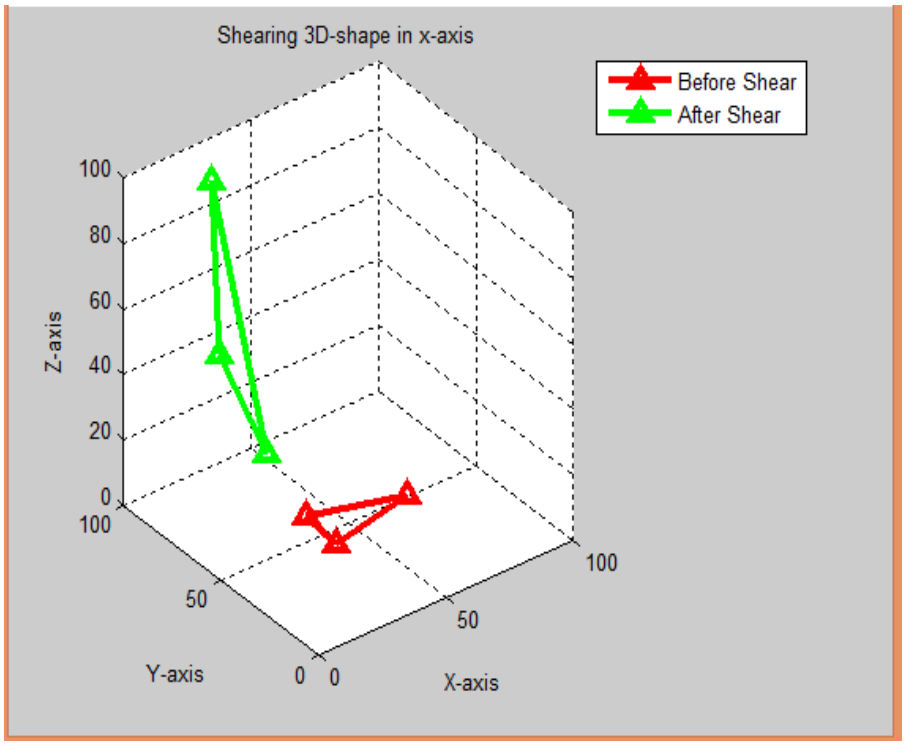
enter Shearing factor Shz : 1

Shearing the x-axis ,enter 1

Shearing the y-axis ,enter 2

Shearing the z-axis ,enter 3

enter version of Shearing 1 or 2 or 3: 1


Shearing 3D-shape in x-axis
Before Shear
After Shear
Z-axis
100
80
60
40
20
0
100
50
0
Y-axis
0
50
100
X-axis

## The Output of program:

enter no. of shape vertices : 3

enter x-value: 10

enter y-value: 20

enter z-value: 30

enter x-value: 30

enter y-value: 30

enter z-value: 10

enter x-value: 50

enter y-value: 20

enter z-value: 22

enter Shearing factor Shx : 2

enter Shearing factor Shy : 2

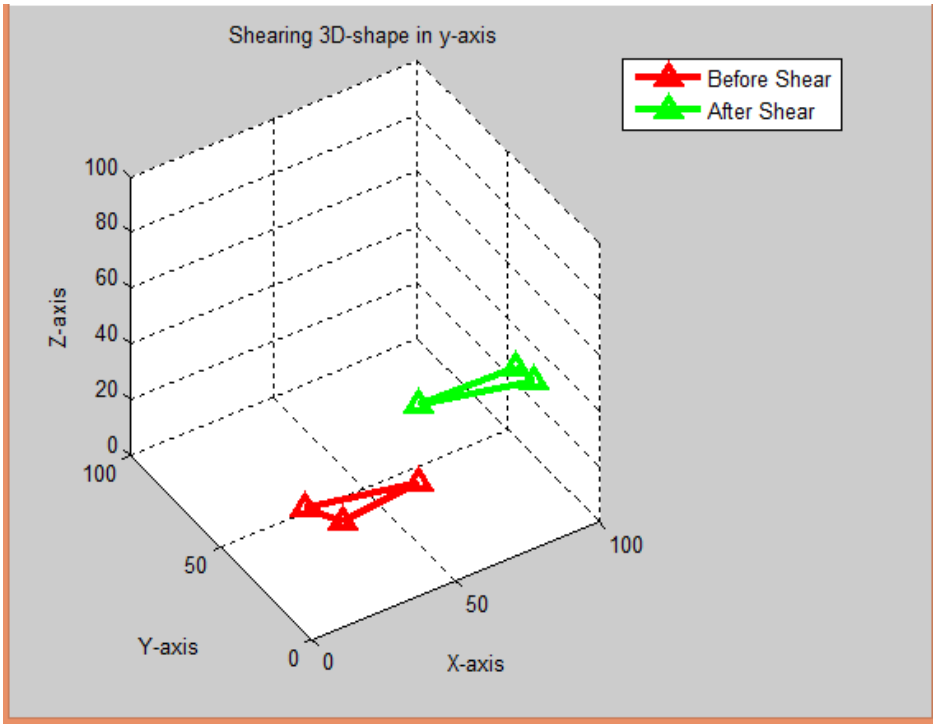
enter Shearing factor Shz : 1

Shearing the x-axis ,enter 1

Shearing the y-axis ,enter 2

Shearing the z-axis ,enter 3

enter version of Shearing 1 or 2 or 3: 2


Shearing 3D-shape in y-axis
Before Shear
After Shear
Z-axis
100
80
60
40
20
0
100
50
0
Y-axis
0
50
100
X-axis

## The Output of program:

enter no. of shape vertices : 3

enter x-value: 10

enter y-value: 20

enter z-value: 30

enter x-value: 30

enter y-value: 30

enter z-value: 10

enter x-value: 50

enter y-value: 20

enter z-value: 22

enter Shearing factor Shx : 2

enter Shearing factor Shy : 2

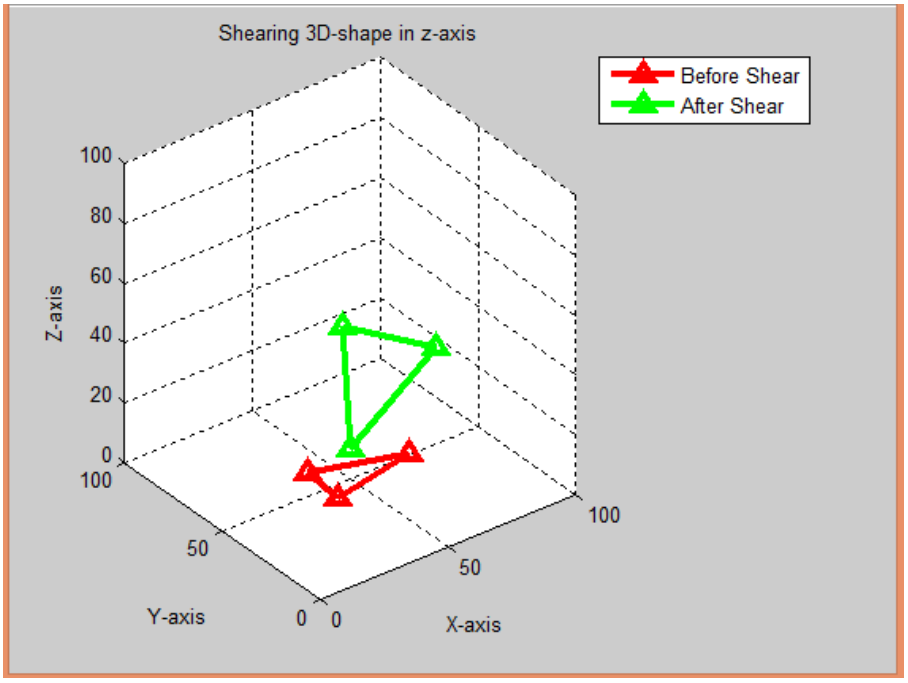
enter Shearing factor Shz : 1

Shearing the x-axis ,enter 1

Shearing the y-axis ,enter 2

Shearing the z-axis ,enter 3

enter version of Shearing 1 or 2 or 3: 3


Shearing 3D-shape in z-axis
Before Shear
After Shear
Z-axis
100
80
60
40
20
0
100
50
0
Y-axis
0
50
100
X-axis

# APPENDIX

# Appendix

## MATLAB Language

## 1. What is MATLAB?

MATLAB is a programming language developed by Math Works. It started out as a matrix programming language where linear algebra programming was simple. It can be run both under interactive sessions and as a batch job.

Matlab stands for **Mat**rix **Lab**oratory. The MATLAB platform is optimized for solving engineering and scientific problems. The matrix-based MATLAB language is the world's most natural way to express computational mathematics. Built-in **graphics** make it easy to visualize and gain insights from data. A vast library of pre-built toolboxes lets you get started right away with algorithms essential to your domain. The desktop environment invites experimentation, exploration, and discovery. These MATLAB tools and capabilities are all rigorously tested and designed to work together.

# 2. MATLAB Language Fundamentals

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. There are two basic ways to create graphs in MATLAB. Either you can use the MATLAB GUI plotting tools to interactively create or you can use the command interface by entering MATLAB graphics commands in the Command Window.

## 2.1 Programming and Scripts

The simplest type of MATLAB program is called a script. A script is a file with a .m extension that contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command line.

To write and run program we used three steps:

**1.** Create a script file-new-script.

**2.** Save the file in the current folder.

**3.** You can also run scripts from the Editor by pressing the **Run** button .

## 2.2 Script Locations

MATLAB looks for scripts and other files in certain places. To run a script, the file must be in the current folder or in a folder on the search path.

By default, the MATLAB folder that the MATLAB Installer creates is on the search path. If you want to store and run programs in another folder, add it to the search path. Select the folder in the Current Folder browser, right-click, and then select Add to Path.

## 2.3 Initializing MATLAB

I like to initialize MATLAB this way…

```
clc;          %erases command window
close all;    %closes all figure windows
clear all;    %clears all variables from memory
```

# 3. MATLAB Advantages

MATLAB provides many techniques for plotting numerical data. Symbolic Math Toolbox expands these graphical capabilities by providing plotting functions for symbolic expressions, equations, and functions. These plots can be in 2-D or 3-D as lines, contours, surfaces, or meshes.

Matlab allows you to

- Implement and test your algorithms easily
- Develop the computational codes easily
- Debug easily

- Use a large database of built in algorithms
- Process still images and create simulation videos easily
- Symbolic computation can be easily done
- Call external libraries
- Perform extensive data analysis and visualization
- Develop application with graphics user interface

## 4. MATLAB Commands

MATLAB provides various commands; the following table provides all such commands :

| Command | Purpose | syntax |
|---|---|---|
| axis | allows you to set the axis scales | axis ( [xmin xmax ymin ymax] ) |
| clc | Clears command window. | clc |
| clear | Removes variables from memory. | clear |
| clear all | Removes all variables from the workspace. | clear all |

| | | |
|---|---|---|
| close | Closes the current plot. | close |
| close all | Closes all plots. | close all |
| grid | Allows you to put the grid lines on the graph. | grid on |
| hold | Retain current graph in figure.<br>hold on :retains the current plot and certain axes properties so that subsequent graphing commands add to the existing graph.<br>hold off resets axes properties to their defaults before drawing new plots | hold on<br>hold off |
| Input | Displays prompts and waits for input. | Variable =input('message') |

| | | |
|---|---|---|
| plot | Generates x,y plot. | plot(X,Y) //creates a 2-D line plot of the data in Y versus the corresponding values in X. plot a blue solid line with no markers(default) plot(X,Y,LineSpec) // creates the plot using the specified line style, marker, and color. |
| Plot3 | Generates x,y,z plot. | plot3(X,Y,Z) plots coordinates in 3-D space. plot3(X,Y,Z,LineSpec) creates the plot using the specified line style, marker, and color. |

| | | |
|---|---|---|
| tittle | Puts text at top of plot. | title('titletext') |
| xlabel | Adds text label to x-axis. | xlabel('text') |
| ylabel | Adds text label to y-axis. | ylabel('text') |

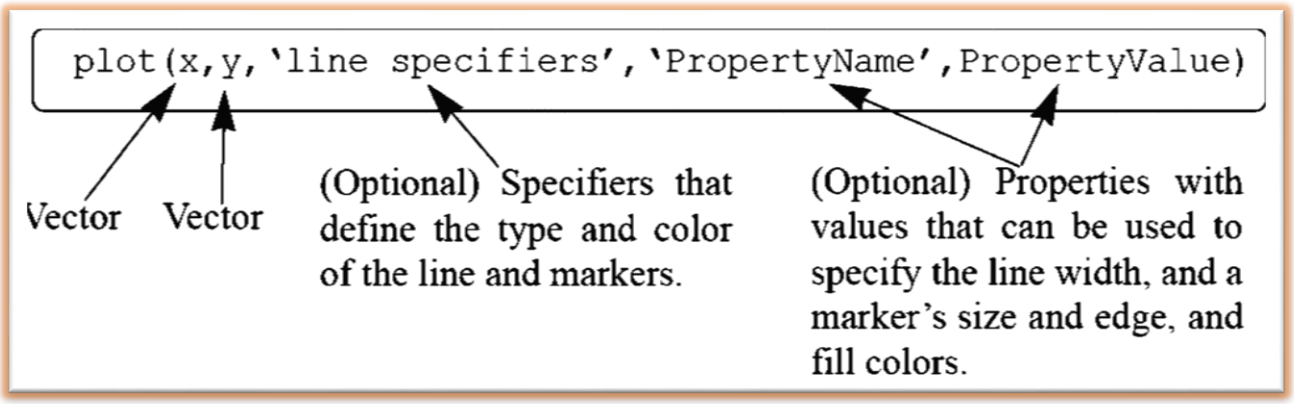
## 4.1 Line Specifies:

Three types of line specifies:

**1**. Line color.

**2**. Line style .

**3.** Marker type.



## 4.2 Line Color

| Line Color | Code |
|---|---|
| White | w |
| Black | k |
| Blue | b |
| Red | r |
| Cyan | c |
| Green | g |
| Magenta | m |
| Yellow | y |

## 4.3 Line Style.

| Line style | sepicifier | Example | Description |
|---|---|---|---|
| Solid line(default) | - | Plot(x,y) | Plot a blue ,solid line |
| dotted line | : | plot(x,y,'y:') | Plots a yellow, dotted line |
| dished line | -- | plot(x,y,'--c') | Plots a cyan ,dished line |
| Point | . | plot(x,y,'w.') | Plots a white, point |
| Dished-dot line | -. | Plot(x,y,'r-.') | Plots a red , Dished-dot line |

## 4.5 Marker Type

| Marker type | Specifier | Example | Desecration |
|---|---|---|---|
| Plus sign | + | plot(x,y,'r+') | Plots a red, Plus sign. |
| Circle | O | plot(x,y,'ro') | Plots a red, circle. |
| Asterisk | * | plot(x,y,'g*') | Plots a green, asterisk |
| Triangle (pointed right) | > | plot(x,y,'k>') | Plots a black, Triangle (pointed right) |
| Triangle (pointed left) | < | plot(x,y,'g<') | Plots a green , Triangle (pointed left) |
| Triangle (pointed up) | ^ | plot(x,y,'y^') | Plots a yellow, Triangle (pointed up) |
| Triangle (pointed down) | v | plot(x,y,'gv') | Plots a green , Triangle (pointed down) |
| Cross | x | plot(x,y,'rx*') | Plots a red ,cross |

| | | | |
|---|---|---|---|
| Square | s | plot(x,y,'gs') | Plots a green, square |
| Diamond | d | plot(x,y,'rd*') | Plots a red ,diamond |
| Five pointed star | P | plot(x,y,'bp') | Plots a blue, Five pointed star |
| six pointed star | h | plot(x,y,'gh') | Plots a green, six pointed star |

# REFRENCES

# References


**1.** César Pérez López,” ***MATLAB Control Systems Engineering*”**, *springer*, 2014**.**

**2.** David McAllister, “***Stereo Computer Graphics and Other True 3D Technologies****”*, Princeton University Press, 1993.

**3.** David Rogers, “***Procedural Elements for Computer Graphics****”*, McGraw-Hill, 1997.

**4.** D.Hearn & M.p.Baker,” ***Computer Graphics*** “, 2nd Ed., Prentice – Hall, 1994.

**5.** Eric Lengyel,”***Mathematics for 3D Game Programming and Computer Graphics”,*** Third Edition, Course Technology, a part of Cengage Learning, 2012.

**6.** F.W. Billmeyer and M. Saltzman,”***Principles of color technology****”,*Wiley-Interscience publication. Wiley, 1981.

**7.** J.D.Foley & A.Dametal,” ***Introduction to Computer Graphic*** “, Addison – wesly, 1993.

**8.** John F. Hughes, Andries Van dam etc. ,"**Computer Graphics Principles and Practice"**, Third Ed., kurt akeley ، 2014.

**9.** M.Berger," ***Computer Graphic with Pascal*** ", B/C Publishing Company, 1984.

**10.** Steve Marschner , Peter Shirley ,"***Fundamentals of Computer Graphics",*** Fourth Edition,by Taylor & Francis Group, LLC, 2016.

**11.** Patrick Marchand**,** *O. THOMAS HOLLAND****, "Graphics and GUIs with MATLAB***", Third Edition, by Chapman & Hall/CRC, 2003.

**12.** Peter Comninos ,"***Mathematical and Computer Programming Techniques for Computer Graphics"***, Springer-Verlag London ,2006.

**13.** Rudra Pratap ,"***Getting Started with MATLAB: A Quick Introduction for Scientists and Engineers",*** New York OXFORD University press 2010.