

# COMPUTER GRAPHICS

**Objective:** To understand the concepts on basic Graphical Techniques, Raster Graphics, Two Dimensional and Three-Dimensional Graphics.

## Unit I

### Introduction to Computer Graphics

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

Computer Graphics can be used in digital photography, film, entertainment, electronic gadgets and all other core technologies which are required. It is a vast subject and area in the field of computer science.

Computer Graphics can be used in UI design, rendering, geometric object, animation and many more. In most areas, computer graphics is an abbreviation of CG. There are several tools used for implementation of Computer Graphics.

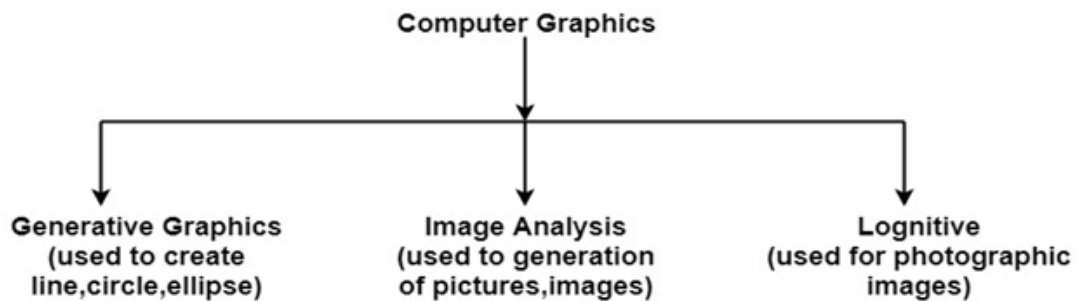
The basic is the <graphics.h> header file in Turbo-C, Unity for advanced and even OpenGL can be used for its implementation. It was invented in 1960 by great researchers Verne Hudson and William Fetter from Boeing.

#### **Computer Graphics refers to several things:**

- The manipulation and the representation of the image or the data in a graphical manner.
- Various technology required for the creation and manipulation.
- Digital synthesis and its manipulation.

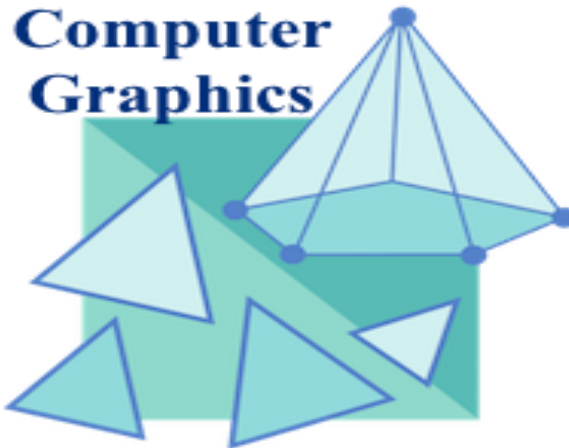
### *Applications*

- **Computer Graphics are used for aided design for engineering and architectural system-** These are used in electrical automobile, electro-mechanical, mechanical, electronic devices. For example: gears and bolts.
- **Computer Art** – MS Paint.
- **Presentation Graphics** – It is used to summarize financial statistical scientific or economic data. For example- Bar chart, Line chart.
- **Entertainment-** It is used in motion picture, music video, television gaming.
- **Education and training-** It is used to understand operations of complex system. It is also used for specialized system such for framing for captains, pilots and so on.
- **Visualization-** To study trends and patterns. For example- Analyzing satellite photo of earth.



### **What drives computer graphics?**

- Movie Industry
- Leaders in quality and artistry
- Not slaves to conceptual purity
- Big budgets and tight schedules
- Reminder that there is more to
- CG than technology



## **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 fine 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, time chart, can be done using the tools present in computer graphics.

- Financial Reports
- Statistical Reports
- Mathematical Reports
- Scientific Reports
- Economic Data for research reports
- Managerial Reports
- Consumer Information Bulletins
- And other types of reports

### **4.Entertainment:**

Computer graphics finds a major part of its utility in the movie industry and game industry. Used for creating motion pictures, music video, television shows, cartoon animation films. In the game industry where focus and interactivity are the key players, computer graphics helps in providing such features in the efficient way.

### **5.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.

### **6.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.

## **7. Visualization:**

Today the need of visualize things have increased drastically, the need of visualization can be seen in many advance technologies, data visualization helps in finding insights of the data, to check and study the behavior of processes around us we need appropriate visualization which can be achieved through proper usage of computer graphics

## **8. 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 of computer graphics.

## **9. Machine Drawing:**

Computer graphics is very frequently used for designing, modifying and creation of various parts of machine and the whole machine itself, the main reason behind using computer graphics for this purpose is the precision and clarity we get from such drawing is ultimate and extremely desired for the safe manufacturing of machine using these drawings.

## **10. Graphical User Interface:**

The use of pictures, images, icons, pop-up menus, graphical objects helps 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.

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.

### **11. Use in Biology:**

Molecular biologist can display a picture of molecules and gain insight into their structure with the help of computer graphics.

### **12. Computer-Generated Maps:**

Town planners and transportation engineers can use computer-generated maps which display data useful to them in their planning work.

### **13. Computer Art:**

Computer Graphics are also used in the field of commercial arts. It is used to generate television

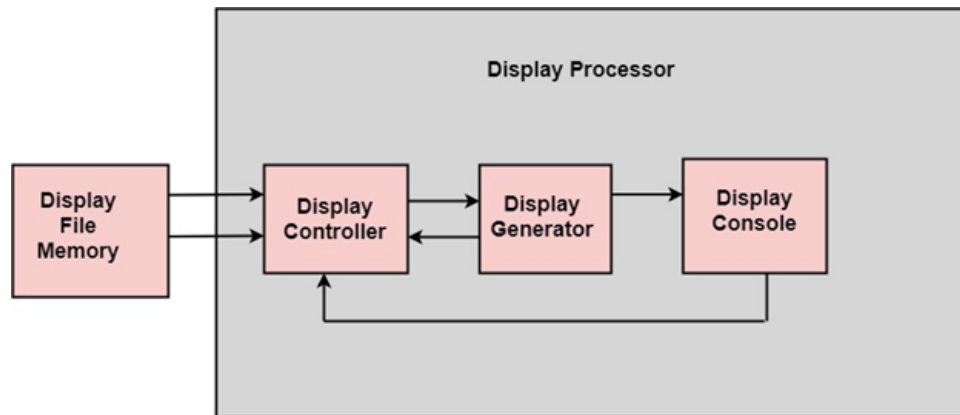
### **15. Printing Technology:**

Computer Graphics is used for printing technology and textile design.

### **Example of Computer Graphics Packages:**

1. LOGO
2. COREL DRAW
3. AUTO CAD
4. 3D STUDIO
5. CORE
6. GKS (Graphics Kernel System)
7. PHIGS
8. CAM (Computer Graphics Metafile)
9. CGI (Computer Graphics Interface)

## Display System



Block diagram of Display System

### Parts of Display Processor

1. Display File Memory
2. Display Processor
3. Display Generator
4. Display Console

**Display Processor:** It is interpreter or piece of hardware that converts display processor code into pictures. It is one of the four main parts of the display processor.

**Display File Memory:** It is used for generation of the picture. It is used for identification of graphic entities.

### Display Controller:

1. It handles interrupt.
2. It maintains timings.
3. It is used for interpretation of instruction.

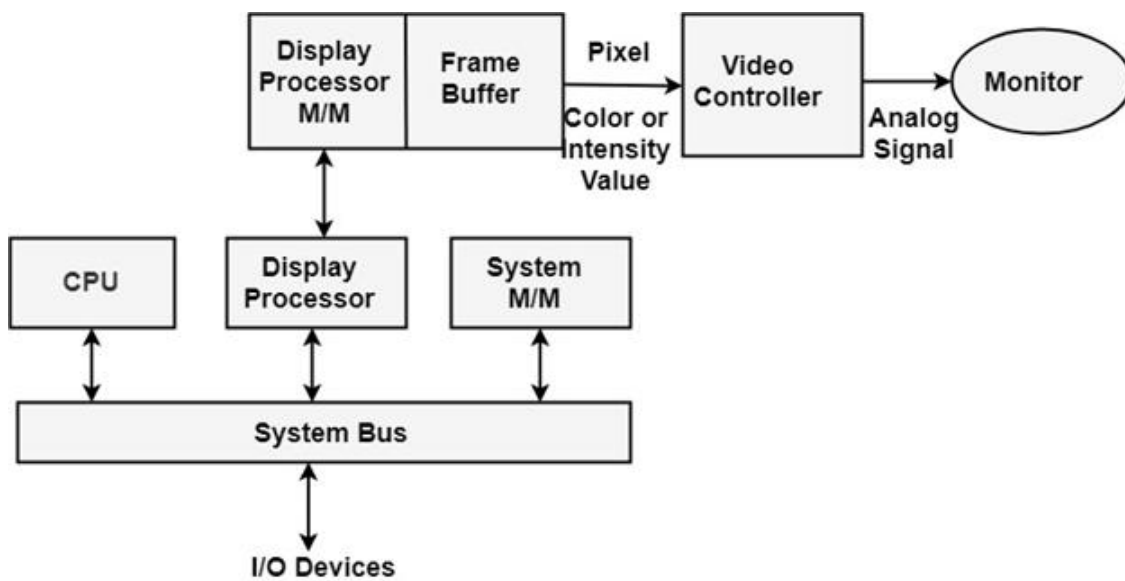
## Display Generator:

1. It is used for the generation of character.
2. It is used for the generation of curves.

**Display Console:** It contains CRT, Light Pen, and Keyboard and deflection system.

The raster scan system is a combination of some processing units. It consists of the control processing unit (CPU) and a particular processor called a display controller. Display Controller controls the operation of the display device. It is also called a video controller.

**Working:** The video controller in the output circuitry generates the horizontal and vertical drive signals so that the monitor can sweep. Its beam across the screen during raster scans.



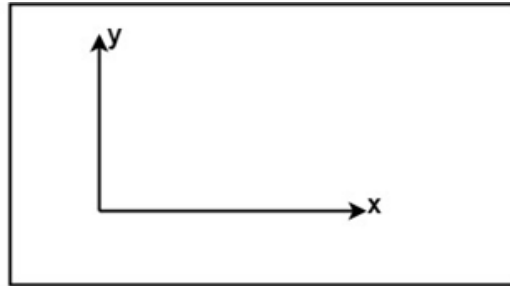
**Fig: Architecture of a Raster Display System with a Display Processor**

As fig showing that 2 registers (X register and Y register) are used to store the coordinate of the screen pixels. Assume that y values of the adjacent scan lines increased by 1 in an upward direction starting from 0 at the bottom of the screen to  $y_{max}$  at the top and along each scan line the screen



pixel positions or x values are incremented by 1 from 0 at the leftmost position to  $x_{max}$  at the rightmost position.

The origin is at the lowest left corner of the screen as in a standard Cartesian coordinate system.



**Fig:**The origin of the coordinate system for identifying screen positions is usually specified in the lower-left corner.

At the start of a **Refresh Cycle**:

X register is set to 0 and y register is set to  $y_{max}$ . This  $(x, y)$  address is translated into a memory address of frame buffer where the color value for this pixel position is stored.

The controller receives this color value (a binary no) from the frame buffer, breaks it up into three parts and sends each element to a separate Digital-to-Analog Converter (DAC).

These voltages, in turn, controls the intensity of 3 e-beam that are focused at the  $(x, y)$  screen position by the horizontal and vertical drive signals.

This process is repeated for each pixel along the top scan line, each time incrementing the X register by Y.

As pixels on the first scan line are generated, the X register is incremented through  $x_{max}$ .

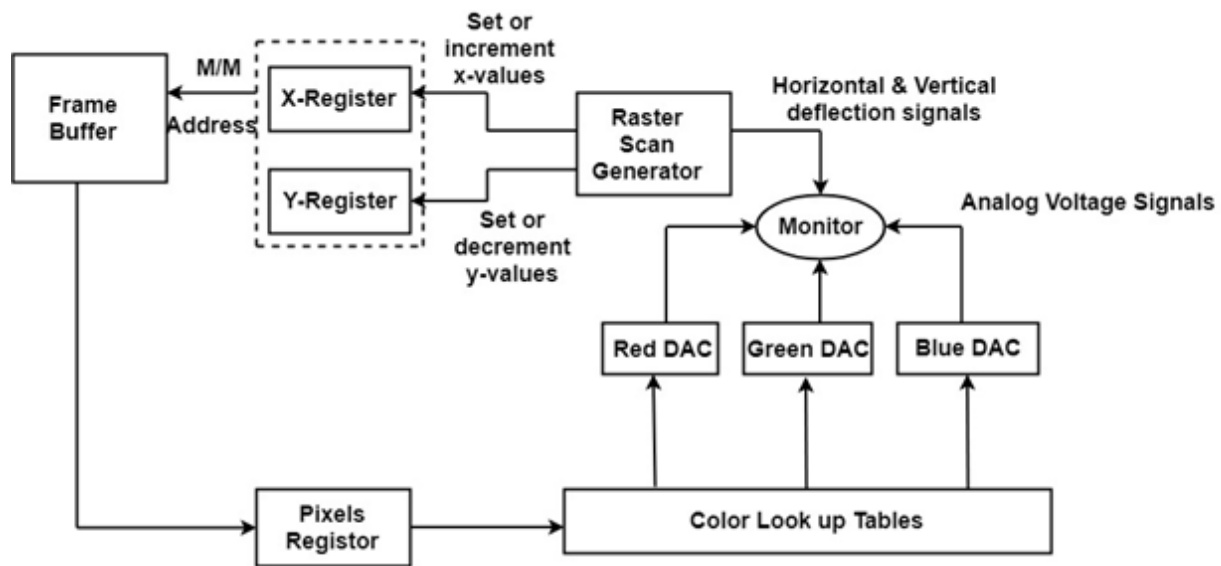
Then x register is reset to 0, and y register is decremented by 1 to access the next scan line.

Pixel along each scan line is then processed, and the procedure is repeated for each successive scan line units pixels on the last scan line ( $y=0$ ) are generated.

For a display system employing a color look-up table frame buffer value is not directly used to control the CRT beam intensity.

It is used as an index to find the three pixel-color value from the look-up table. This lookup operation is done for each pixel on every display cycle.

As the time available to display or refresh a single pixel in the screen is too less, accessing the frame buffer every time for reading each pixel intensity value would consume more time what is allowed:



Multiple adjacent pixel values are fetched to the frame buffer in single access and stored in the register.

After every allowable time gap, the one-pixel value is shifted out from the register to control the warm intensity for that pixel.

The procedure is repeated with the next block of pixels, and so on, thus the whole group of pixels will be processed.

The most commonly used display device is a video monitor. The operation of most video monitors based on CRT (Cathode Ray Tube). The following display devices are used:

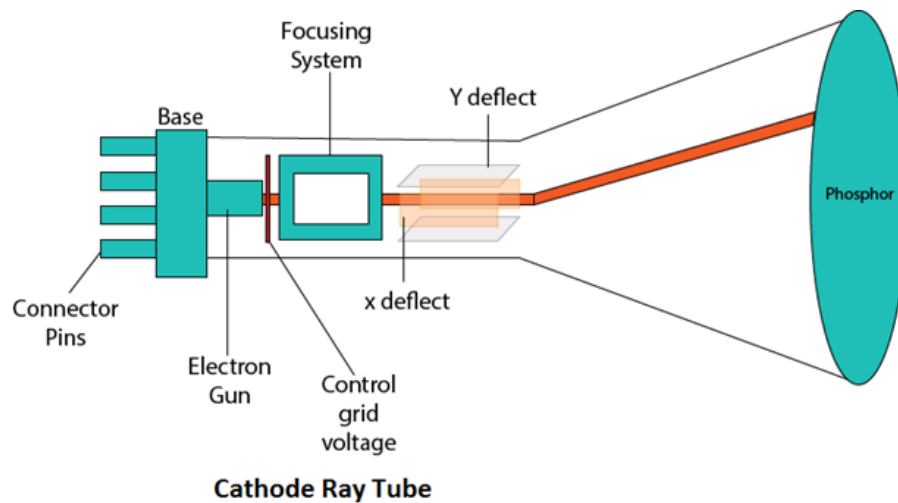
## Video Display Devices:

1. Refresh Cathode Ray Tube
2. Random Scan and Raster Scan
3. Color CRT Monitors
4. Direct View Storage Tubes
5. Flat Panel Display
6. Lookup Table

## Cathode Ray Tube (CRT):

CRT stands for Cathode Ray Tube. CRT is a technology used in traditional computer monitors and televisions. The image on CRT display is created by firing electrons from the back of the tube of phosphorus located towards the front of the screen.

Once the electron heats the phosphorus, they light up, and they are projected on a screen. The color you view on the screen is produced by a blend of red, blue and green light.



## Components of CRT:

Main Components of CRT are:

**1. Electron Gun:** Electron gun consisting of a series of elements, primarily a heating filament (heater) and a cathode. The electron gun creates a source of electrons which are focused into a narrow beam directed at the face of the CRT.

**2. Control Electrode:** It is used to turn the electron beam on and off.

**3. Focusing system:** It is used to create a clear picture by focusing the electrons into a narrow beam.

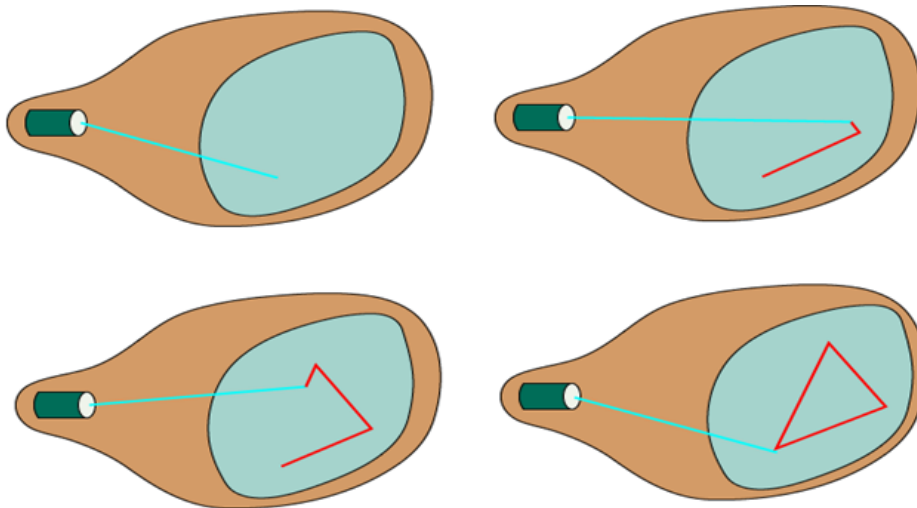
**4. Deflection Yoke:** It is used to control the direction of the electron beam. It creates an electric or magnetic field which will bend the electron beam as it passes through the area. In a conventional CRT, the yoke is linked to a sweep or scan generator. The deflection yoke which is connected to the sweep generator creates a fluctuating electric or magnetic potential.

**5. Phosphorus-coated screen:** The inside front surface of every CRT is coated with phosphors. Phosphors glow when a high-energy electron beam hits them. Phosphorescence is the term used to characterize the light given off by a phosphor after it has been exposed to an electron beam.

## Random Scan Display:

Random Scan System uses an electron beam which operates like a pencil to create a line image on the CRT screen. The picture is constructed out of a sequence of straight-line segments. Each line segment is drawn on the screen by directing the beam to move from one point on the screen to the next, where its x & y coordinates define each point. After drawing the picture. The system cycles

back to the first line and design all the lines of the image 30 to 60 time each second. The process is shown in fig:



Random-scan monitors are also known as vector displays or stroke-writing displays or calligraphic displays.

### Advantages:

1. A CRT has the electron beam directed only to the parts of the screen where an image is to be drawn.
2. Produce smooth line drawings.
3. High Resolution

### Disadvantages:

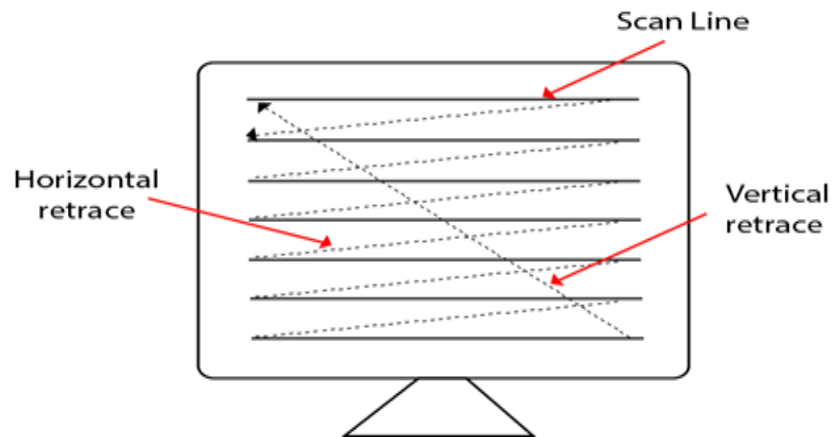
1. Random-Scan monitors cannot display realistic shades scenes.

### **Raster Scan Display:**

A Raster Scan Display is based on intensity control of pixels in the form of a rectangular box called Raster on the screen. Information of on and off pixels is stored in refresh buffer or Frame buffer. Televisions in our house are based on Raster Scan Method. The raster scan system can store

information of each pixel position, so it is suitable for realistic display of objects. Raster Scan provides a refresh rate of 60 to 80 frames per second.

Frame Buffer is also known as Raster or bit map. In Frame Buffer the positions are called picture elements or pixels. Beam refreshing is of two types. First is horizontal retrace and second is vertical retrace. When the beam starts from the top left corner and reaches the bottom right scale, it will again return to the top left side called at vertical retrace. Then it will again more horizontally from top to bottom call as horizontal retrace shown in fig:



### **Types of Scanning or travelling of beam in Raster Scan**

1. Interlaced Scanning
2. Non-Interlaced Scanning

In Interlaced scanning, each horizontal line of the screen is traced from top to bottom. Due to which fading of display of object may occur. This problem can be solved by Non-Interlaced scanning. In this first of all odd numbered lines are traced or visited by an electron beam, then in the next circle, even number of lines are located.

For non-interlaced display refresh rate of 30 frames per second used. But it gives flickers. For interlaced display refresh rate of 60 frames per second is used.

## Advantages:

1. Realistic image
2. Million Different colors to be generated
3. Shadow Scenes are possible.

## Disadvantages:

1. Low Resolution
2. Expensive

## Differentiate between Random and Raster Scan Display:

Random Scan	Raster Scan
1. It has high Resolution	1. Its resolution is low.
2. It is more expensive	2. It is less expensive
3. Any modification if needed is easy	3.Modification is tough
4. Solid pattern is tough to fill	4.Solid pattern is easy to fill
5. Refresh rate depends or resolution	5. Refresh rate does not depend on the picture.
6. Only screen with view on an area is displayed.	6. Whole screen is scanned.
7. Beam Penetration technology come under it.	7. Shadow mark technology came under this.
8. It does not use interlacing method.	8. It uses interlacing

9. It is restricted to line drawing applications

9. It is suitable for realistic display.

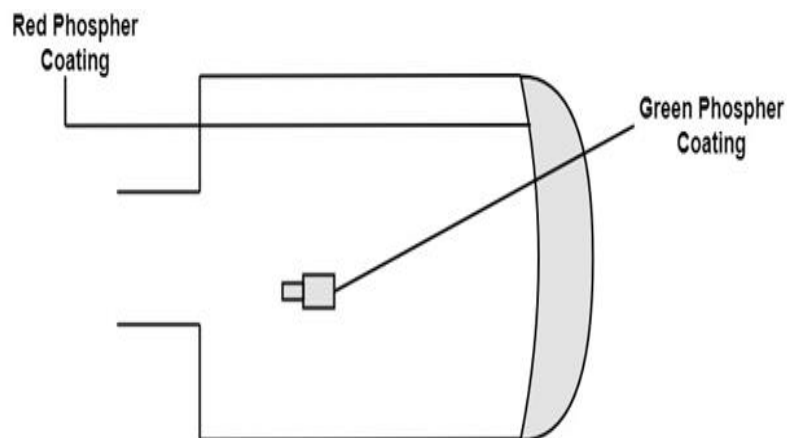
## Color CRT Monitors:

The CRT Monitor display by using a combination of phosphors. The phosphors are different colors. There are two popular approaches for producing color displays with a CRT are:

1. Beam Penetration Method
2. Shadow-Mask Method

### 1. Beam Penetration Method:

The Beam-Penetration method has been used with random-scan monitors. In this method, the CRT screen is coated with two layers of phosphor, red and green and the displayed color depends on how far the electron beam penetrates the phosphor layers. This method produces four colors only, red, green, orange and yellow. A beam of slow electrons excites the outer red layer only; hence screen shows red color only. A beam of high-speed electrons excites the inner green layer. Thus screen shows a green color.





## Advantages:

1. Inexpensive

## Disadvantages:

1. Only four colors are possible
2. Quality of pictures is not as good as with another method.

## 2. Shadow-Mask Method:

- Shadow Mask Method is commonly used in Raster-Scan System because they produce a much wider range of colors than the beam-penetration method.
- It is used in the majority of color TV sets and monitors.

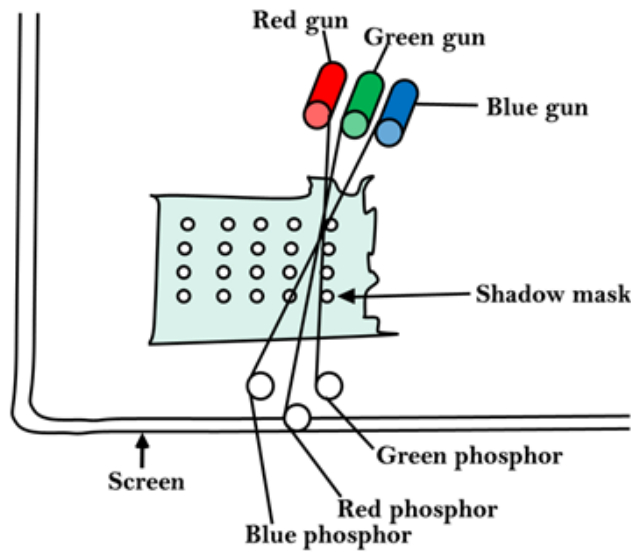
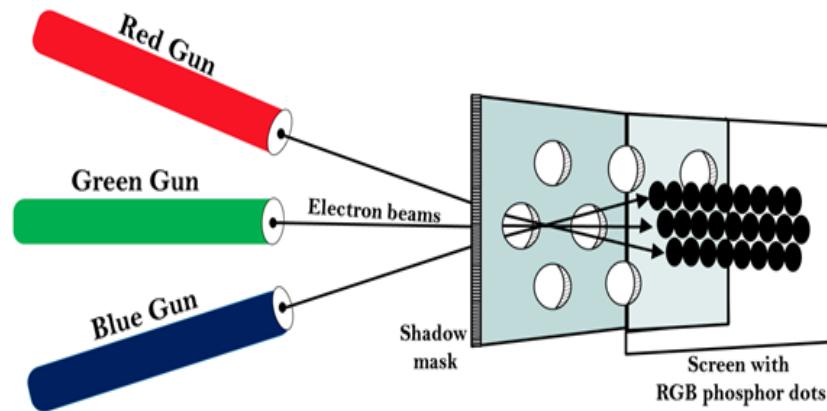
Construction: A shadow mask CRT has 3 phosphor color dots at each pixel position.

- One phosphor dot emits: red light
- Another emits: green light
- Third emits: blue light

This type of CRT has 3 electron guns, one for each color dot and a shadow mask grid just behind the phosphor coated screen.

Shadow mask grid is pierced with small round holes in a triangular pattern.

Figure shows the delta-delta shadow mask method commonly used in color CRT system.



### The Shadow mask CRT

**Working:** Triad arrangement of red, green, and blue guns.

The deflection system of the CRT operates on all 3 electron beams simultaneously; the 3 electron beams are deflected and focused as a group onto the shadow mask, which contains a sequence of holes aligned with the phosphor-dot patterns.

When the three beams pass through a hole in the shadow mask, they activate a dotted triangle, which occurs as a small color spot on the screen.

The phosphor dots in the triangles are organized so that each electron beam can activate only its corresponding color dot when it passes through the shadow mask.

**Inline arrangement:** Another configuration for the 3 electron guns is an Inline arrangement in which the 3

electron guns and the corresponding red-green-blue color dots on the screen, are aligned along one scan line rather of in a triangular pattern.

This inline arrangement of electron guns is easier to keep in alignment and is commonly used in high-resolution color CRT's.

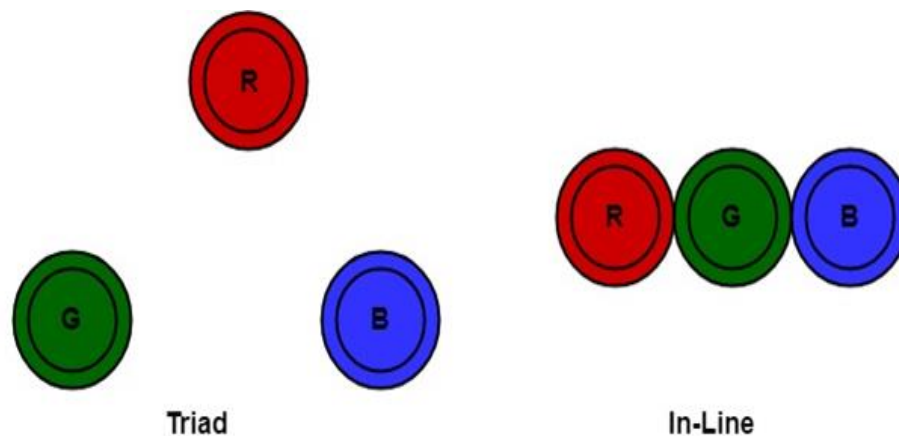


Fig: Triad-and -in-line arrangements of red, green and blue electron guns of CRT for color monitors.

### **Advantage:**

1. Realistic image
2. Million different colors to be generated
3. Shadow scenes are possible

## Disadvantage:

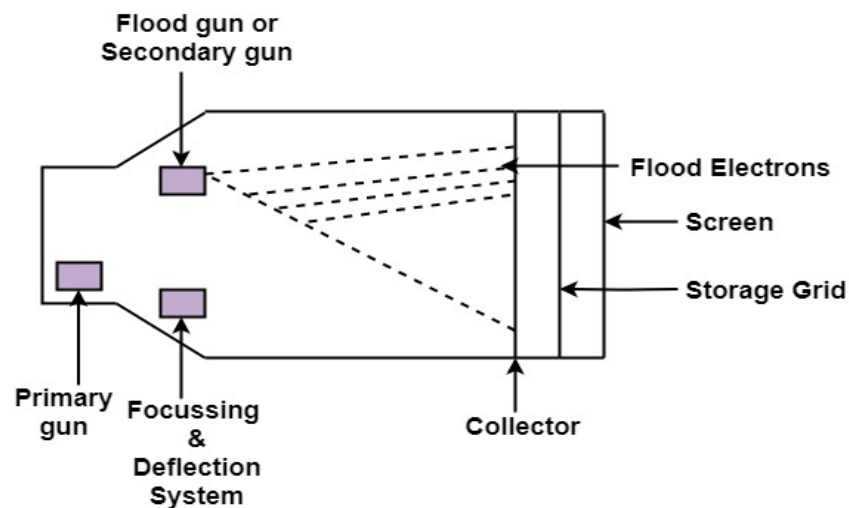
1. Relatively expensive compared with the monochrome CRT.
2. Relatively poor resolution
3. Convergence Problem

## Direct View Storage Tubes:

DVST terminals also use the random scan approach to generate the image on the CRT screen. The term "storage tube" refers to the ability of the screen to retain the image which has been projected against it, thus avoiding the need to rewrite the image constantly.

**Function of guns:** Two guns are used in DVST

1. **Primary guns:** It is used to store the picture pattern.
2. **Flood gun or Secondary gun:** It is used to maintain picture display.



Direct View Storage Tube

## Advantage:

1. No refreshing is needed.
2. High Resolution
3. Cost is very less

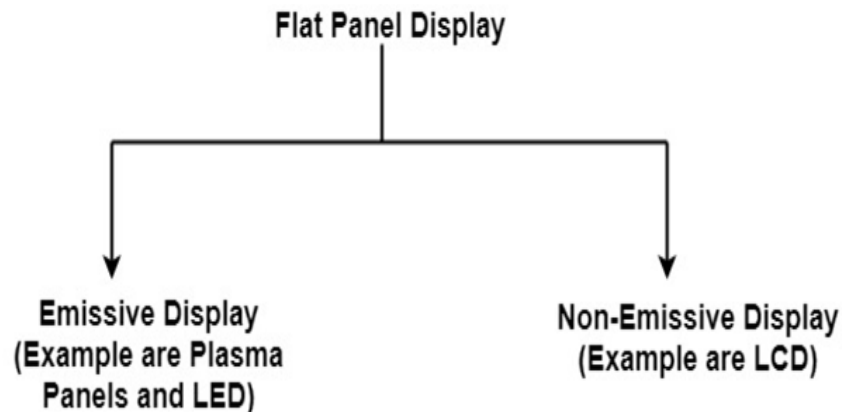
## Disadvantage:

1. It is not possible to erase the selected part of a picture.
2. It is not suitable for dynamic graphics applications.
3. If a part of picture is to modify, then time is consumed.

## Flat Panel Display:

The Flat-Panel display refers to a class of video devices that have reduced volume, weight and power requirement compare to CRT.

**Example:** Small T.V. monitor, calculator, pocket video games, laptop computers, an advertisement board in elevator.



**1. Emissive Display:** The emissive displays are devices that convert electrical energy into light. Examples are Plasma Panel, thin film electroluminescent display and LED (Light Emitting Diodes).

**2. Non-Emissive Display:** The Non-Emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns. Examples are LCD (Liquid Crystal Device).

## Plasma Panel Display:

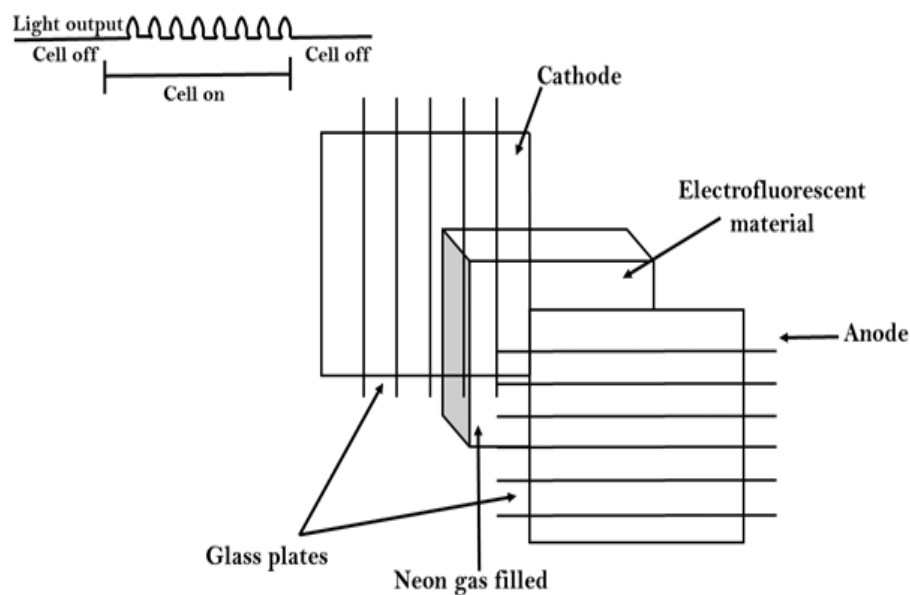
Plasma-Panels are also called as Gas-Discharge Display. It consists of an array of small lights. Lights are fluorescent in nature. The essential components of the plasma-panel display are:

1. **Cathode:** It consists of fine wires. It delivers negative voltage to gas cells. The voltage is released along with the negative axis.
2. **Anode:** It also consists of line wires. It delivers positive voltage. The voltage is supplied along positive axis.
3. **Fluorescent cells:** It consists of small pockets of gas liquids when the voltage is applied to this liquid (neon gas) it emits light.
4. **Glass Plates:** These plates act as capacitors. The voltage will be applied, the cell will glow continuously.

The gas will glow when there is a significant voltage difference between horizontal and vertical wires. The voltage level is kept between 90 volts to 120 volts. Plasma level does not require refreshing. Erasing is done by reducing the voltage to 90 volts.

Each cell of plasma has two states, so cell is said to be stable. Displayable point in plasma panel is made by the crossing of the horizontal and vertical grid. The resolution of the plasma panel can be up to 512 \* 512 pixels.

**Figure shows the state of cell in plasma panel display:**



## Advantage:

1. High Resolution
2. Large screen size is also possible.
3. Less Volume
4. Less weight
5. Flicker Free Display

## Disadvantage:

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

## **LED (Light Emitting Diode):**

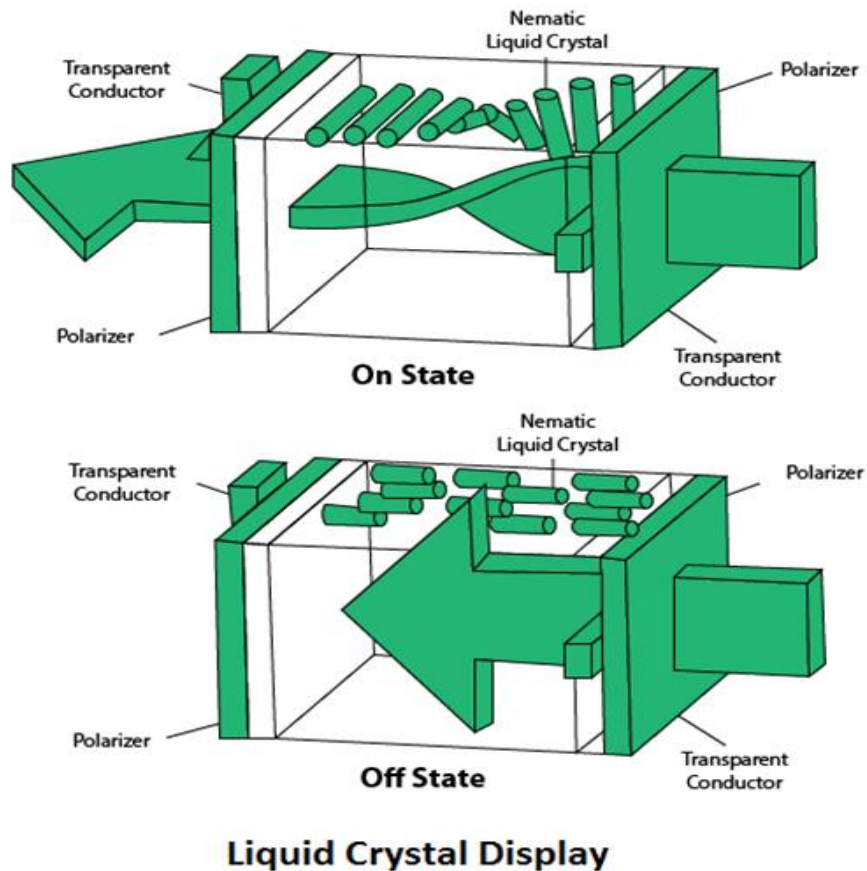
In an LED, a matrix of diodes is organized to form the pixel positions in the display and picture definition is stored in a refresh buffer. Data is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light pattern in the display.

## **LCD (Liquid Crystal Display):**

Liquid Crystal Displays are the devices that produce a picture by passing polarized light from the surroundings or from an internal light source through a liquid-crystal material that transmits the light.

LCD uses the liquid-crystal material between two glass plates; each plate is the right angle to each other between plates liquid is filled. One glass plate consists of rows of conductors arranged in vertical direction. Another glass plate is consisting of a row of conductors arranged in horizontal direction. The pixel position is determined by the intersection of the vertical & horizontal conductor. This position is an active part of the screen.

Liquid crystal display is temperature dependent. It is between zero to seventy degree Celsius. It is flat and requires very little power to operate.



### Advantage:

1. Low power consumption.
2. Small Size
3. Low Cost

### Disadvantage:

1. LCDs are temperature-dependent (0-70°C)
2. LCDs 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.



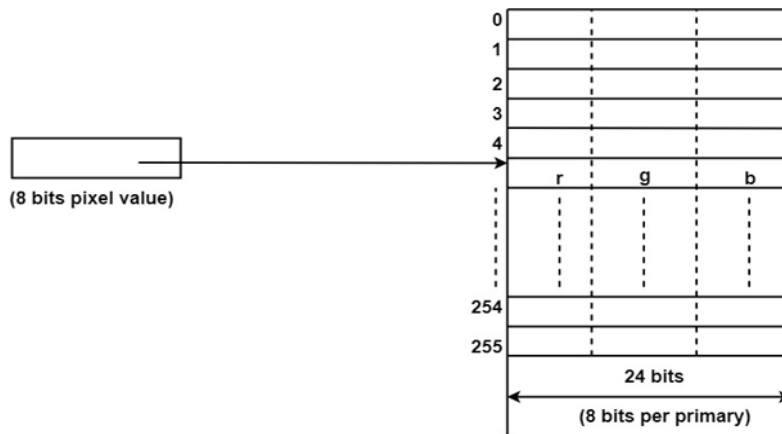
## Look-Up Table:

Image representation is essentially the description of pixel colors. There are three primary colors: R (red), G (green) and B (blue). Each primary color can take on intensity levels produces a variety of colors. Using direct coding, we may allocate 3 bits for each pixel, with one bit for each primary color. The 3-bit representation allows each primary to vary independently between two intensity levels: 0 (off) or 1 (on). Hence each pixel can take on one of the eight colors.

Bit 1:r	Bit 2:g	Bit 3:b	Color name
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

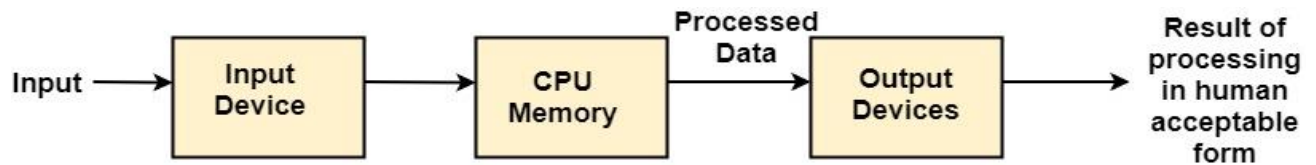
A widely accepted industry standard uses 3 bytes, or 24 bytes, per pixel, with one byte for each primary color. The way, we allow each primary color to have 256 different intensity levels. Thus a pixel can take on a color from 256 x 256 x 256 or 16.7 million possible choices. The 24-bit format is commonly referred to as the actual color representation.

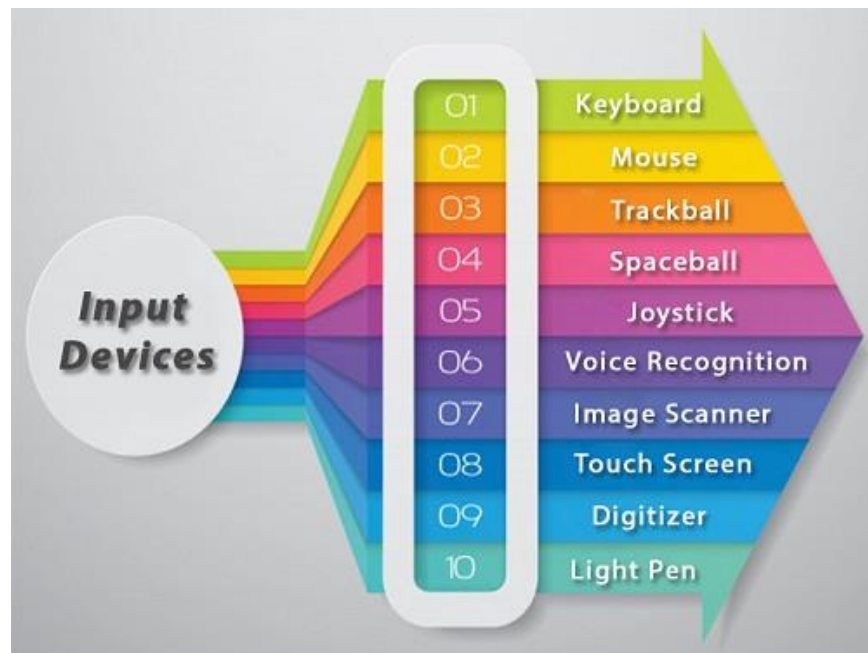
Lookup Table approach reduces the storage requirement. In this approach pixel values do not code colors directly. Alternatively, they are addresses or indices into a table of color values. The color of a particular pixel is determined by the color value in the table entry that the value of the pixel references. Figure shows a look-up table with 256 entries. The entries have addresses 0 through 255. Each entry contains a 24-bit RGB color value. Pixel values are now 1-byte. The color of a pixel whose value is  $i$ , where  $0 < i < 255$ , is persistence by the color value in the table entry whose address is  $i$ . It reduces the storage requirement of a  $1000 \times 1000$  image to one million bytes plus 768 bytes for the color values in the look-up table.



## Input Devices

The Input Devices are the hardware that is used to transfer transfers input to the computer. The data can be in the form of text, graphics, sound, and text. Output device display data from the memory of the computer. Output can be text, numeric data, line, polygon, and other objects.





## Keyboard:

The most commonly used input device is a keyboard. The data is entered by pressing the set of keys. All keys are labeled. A keyboard with 101 keys is called a QWERTY keyboard.

The keyboard has alphabetic as well as numeric keys. Some special keys are also available.

1. **Numeric Keys:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
2. **Alphabetic keys:** a to z (lower case), A to Z (upper case)
3. **Special Control keys:** Ctrl, Shift, Alt
4. **Special Symbol Keys:** ; , " ? @ ~ ? :
5. **Cursor Control Keys:** ↑ → ← ↓
6. **Function Keys:** F1 F2 F3....F9.
7. **Numeric Keyboard:** It is on the right-hand side of the keyboard and used for fast entry of numeric data.

## Functions of Keyboard:

1. Alphanumeric Keyboards are used in CAD. (Computer Aided Drafting)
2. Keyboards are available with special features line screen co-ordinates entry, Menu selection or graphics functions, etc.
3. Special purpose keyboards are available having buttons, dials, and switches. Dials are used to enter scalar values. Dials also enter real numbers. Buttons and switches are used to enter predefined function values.

## Advantage:

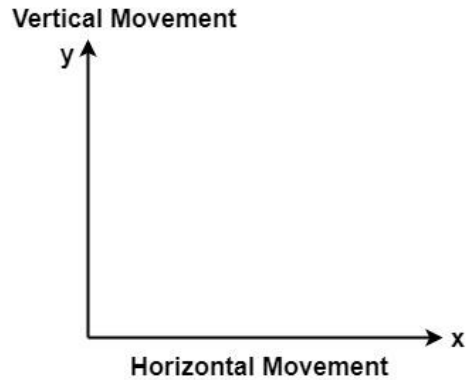
1. Suitable for entering numeric data.
2. Function keys are a fast and effective method of using commands, with fewer errors.

## Disadvantage:

1. Keyboard is not suitable for graphics input.

## Mouse:

A Mouse is a pointing device and used to position the pointer on the screen. It is a small palm size box. There are two or three depression switches on the top. The movement of the mouse along the x-axis helps in the horizontal movement of the cursor and the movement along the y-axis helps in the vertical movement of the cursor on the screen. The mouse cannot be used to enter text. Therefore, they are used in conjunction with a keyboard.

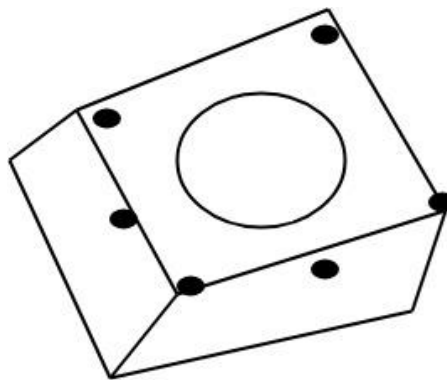


### Advantage:

1. Easy to use
2. Not very expensive

## Trackball

It is a pointing device. It is similar to a mouse. This is mainly used in notebook or laptop computer, instead of a mouse. This is a ball which is half inserted, and by changing fingers on the ball, the pointer can be moved.



**TrackBall**

### Advantage:

1. Trackball is stationary, so it does not require much space to use it.
2. Compact Size

## **Spaceball:**

It is similar to trackball, but it can move in six directions where trackball can move in two directions only. The movement is recorded by the strain gauge. Strain gauge is applied with pressure. It can be pushed and pulled in various directions. The ball has a diameter around 7.5 cm. The ball is mounted in the base using rollers. One-third of the ball is an inside box, the rest is outside.

## **Applications:**

1. It is used for three-dimensional positioning of the object.
2. It is used to select various functions in the field of virtual reality.
3. It is applicable in CAD applications.
4. Animation is also done using spaceball.
5. It is used in the area of simulation and modeling.

## **Joystick:**

A Joystick is also a pointing device which is used to change cursor position on a monitor screen. Joystick is a stick having a spherical ball as its both lower and upper ends as shown in fig. The lower spherical ball moves in a socket. The joystick can be changed in all four directions. The function of a joystick is similar to that of the mouse. It is mainly used in Computer Aided Designing (CAD) and playing computer games.

Joystick



## **Light Pen**

Light Pen (similar to the pen) is a pointing device which is used to select a displayed menu item or draw pictures on the monitor screen. It consists of a photocell and an optical system placed in a small tube. When its tip is moved over the monitor screen, and pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signals to the CPU.



**Light Pen**

### **Uses:**

1. Light Pens can be used as input coordinate positions by providing necessary arrangements.
2. If background color or intensity, a light pen can be used as a locator.
3. It is used as a standard pick device with many graphics system.
4. It can be used as stroke input devices.
5. It can be used as valuator

### **Digitizers:**

The digitizer is an operator input device, which contains a large, smooth board (the appearance is similar to the mechanical drawing board) & an electronic tracking device, which can be changed over the surface to follow existing lines. The electronic tracking device contains a switch for the user to record the desire x & y coordinate positions. The coordinates can be entered into the computer memory or stored on an off-line storage medium such as magnetic tape.



Digitizer

#### Advantages:

1. Drawing can easily be changed.
2. It provides the capability of interactive graphics.

#### Disadvantages:

1. Costly
2. Suitable only for applications which required high-resolution graphics.

### **Touch Panels:**

Touch Panels is a type of display screen that has a touch-sensitive transparent panel covering the screen. A touch screen registers input when a finger or other object comes in contact with the screen.

When the wave signals are interrupted by some contact with the screen, that located is recorded. Touch screens have long been used in military applications.

### **Voice Systems (Voice Recognition):**

Voice Recognition is one of the newest, most complex input techniques used to interact with the computer. The user inputs data by speaking into a microphone. The simplest form of voice recognition is a one-word command spoken by one person. Each command is isolated with pauses between the words.



Voice Recognition is used in some graphics workstations as input devices to accept voice commands. The voice-system input can be used to initiate graphics operations or to enter data. These systems operate by matching an input against a predefined dictionary of words and phrases.

#### Advantage:

1. More efficient device.
2. Easy to use
3. Unauthorized speakers can be identified

#### Disadvantages:

1. Very limited vocabulary
2. Voice of different operators can't be distinguished.

## Image Scanner

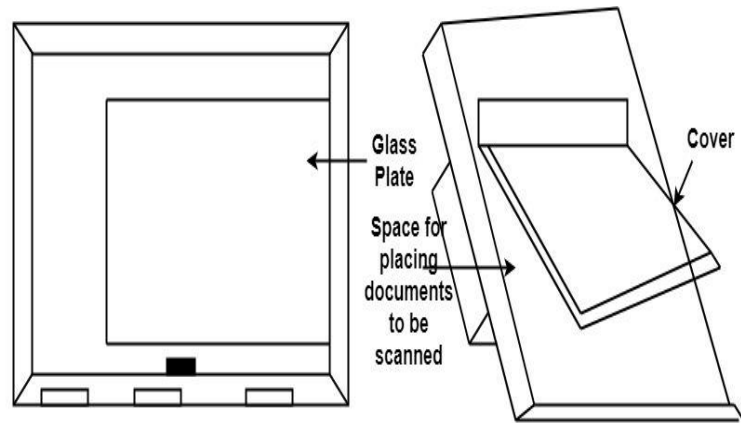
It is an input device. The data or text is written on paper. The paper is feeded to scanner. The paper written information is converted into electronic format; this format is stored in the computer. The input documents can contain text, handwritten material, picture extra.

By storing the document in a computer document became safe for longer period of time. The document will be permanently stored for the future. We can change the document when we need. The document can be printed when needed.

Scanning can be of the black and white or colored picture. On stored picture 2D or 3D rotations, scaling and other operations can be applied.

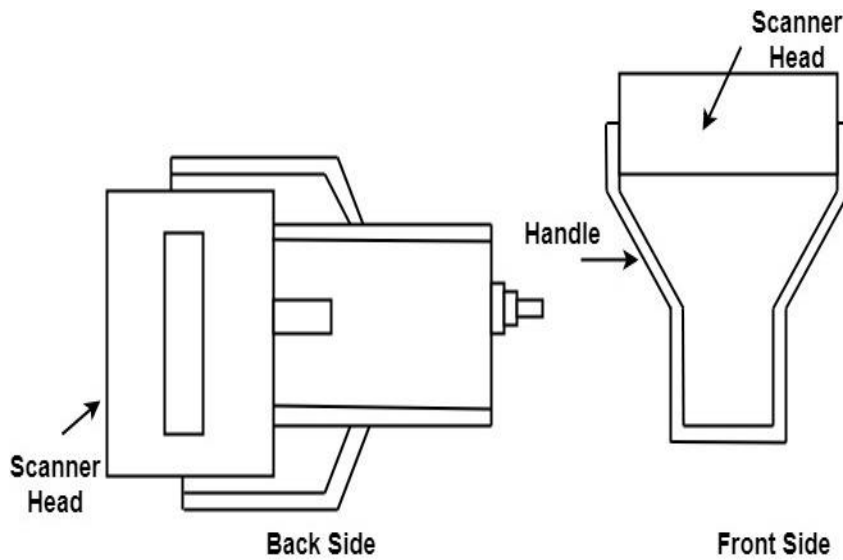
### Types of image Scanner:

**1. Flat Bed Scanner:** It resembles a photocopy machine. It has a glass top on its top. Glass top in further covered using a lid. The document to be scanned is kept on glass plate. The light is passed underneath side of glass plate. The light is moved left to right. The scanning is done the line by line. The process is repeated until the complete line is scanned. Within 20-25 seconds a document of 4" \* 6" can be scanned.



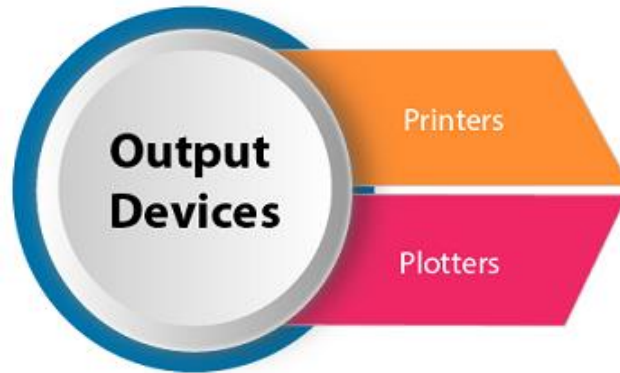
Flat Bed Scanner

**2. Hand Held Scanner:** It has a number of LED's (Light Emitting Diodes) the LED's are arranged in the small case. It is called a Hand held Scanner because it can be kept in hand which performs scanning. For scanning the scanner is moved over document from the top towards the bottom. Its light is on, while we move it on document. It is dragged very slowly over document. If dragging of the scanner over the document is not proper, the conversion will not correct.



Hand Held Scanner

## Output Devices



It is an electromechanical device, which accepts data from a computer and translates them into form understand by users.

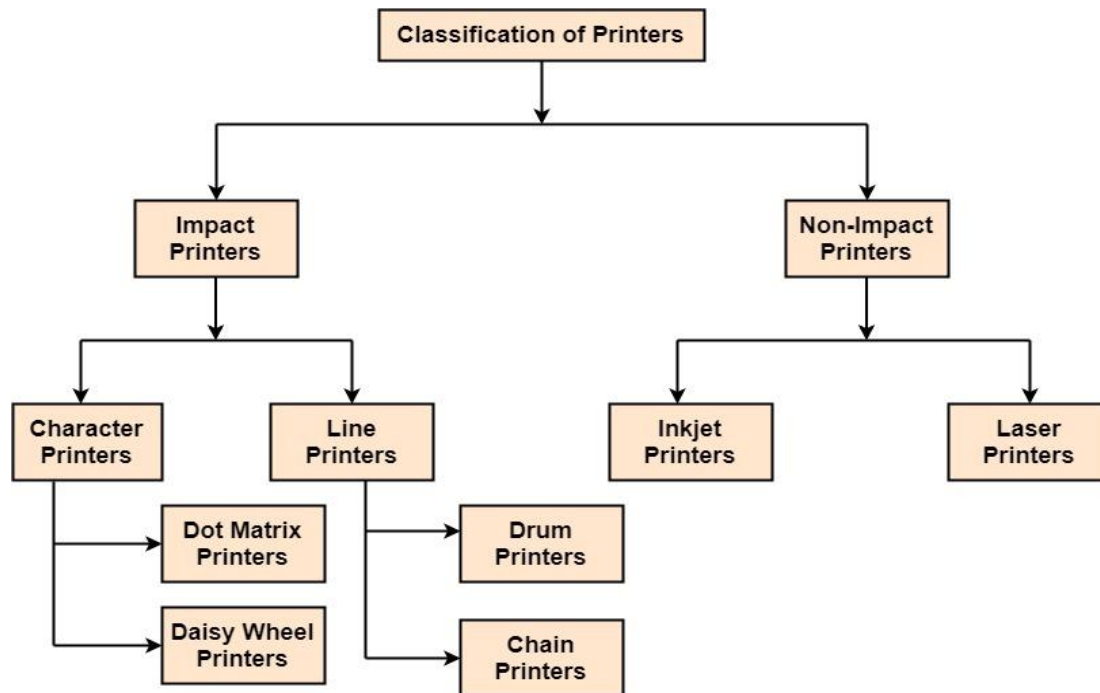
Following are Output Devices:

1. Printers
2. Plotters

### **Printers:**

Printer is the most important output device, which is used to print data on paper.

**Types of Printers:** There are many types of printers which are classified on various criteria as shown in fig:



**1. Impact Printers:** The printers that print the characters by striking against the ribbon and onto the papers are known as Impact Printers.

These Printers are of two types:

1. Character Printers
2. Line Printers

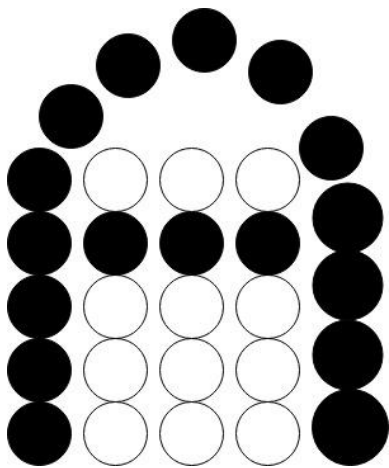
**2. Non-Impact Printers:** The printers that print the characters without striking against the ribbon and onto the papers are called Non-Impact Printers. These printers print a complete page at a time, therefore, also known as Page Printers.

Page Printers are of two types:

1. Laser Printers
2. Inkjet Printers

## Dot Matrix Printers:

Dot matrix has printed in the form of dots. A printer has a head which contains nine pins. The nine pins are arranged one below other. Each pin can be activated independently. All or only the same needles are activated at a time. When needless is not activated, and then the tip of needle stay in the head. When pin work, it comes out of the print head. In nine pin printer, pins are arranged in 5 \* 7 matrixes.



Dot Matrix Printer

### Advantage:

1. Dot Matrix Printers prints output as dots, so it can print any shape of the character. This allows the printer to print special character, charts, graphs, etc.
2. Dot Matrix Printers come under the category of impact printers. The printing is done when the hammer pin strikes the inked ribbon. The impressions are printed on paper. By placing multiple copies of carbon, multiple copies of output can be produced.
3. It is suitable for printing of invoices of companies.

## Daisy Wheel Printers:

Head is lying on a wheel and Pins corresponding to characters are like petals of Daisy, that's why called Daisy wheel printer.



Daisy Wheel Printer

**Advantage:**

1. More reliable than DMPs
2. Better Quality

**Disadvantage:**

1. Slower than DMPs

**Drum Printers:**

These are line printers, which prints one line at a time. It consists of a drum. The shape of the drum is cylindrical. The drum is solid and has characters embossed on it in the form of vertical bands. The characters are in circular form. Each band consists of some characters. Each line on drum consists of 132 characters. Because there are 96 lines so total characters are  $(132 * 96) = 12,672$ .

Drum contains a number of hammers also.

**Chain Printers:**

These are called as line printers. These are used to print one line at a line. Basically, chain consists of links. Each link contains one character. Printers can follow any character set style, i.e., 48, 64 or 96 characters. Printer consists of a number of hammers also.

### Advantages:

1. Chain or Band if damaged can be changed easily.
2. It allows printing of different form.
3. Different Scripts can be printed using this printer.

### Disadvantages:

1. It cannot print charts and graphs.
2. It cannot print characters of any shape.
3. Chain Printers is impact printer, hammer strikes so it is noisy.

## **Non-Impact Printers:**

### **Inkjet Printers:**

These printers use a special ink called electrostatic ink. The printer head has a special nozzle. Nozzle drops ink on paper. Head contains up to 64 nozzles. The ink dropped is deflected by the electrostatic plate. The plate is fixed outside the nozzle. The deflected ink settles on paper.



**Inkjet Printer**

### Advantages:

1. These produce high quality of output as compared to the dot matrix.
2. A high-quality output can be produced using 64 nozzles printed.
3. Inkjet can print characters in a variety of shapes.

4. Inkjet can print special characters.
5. The printer can print graphs and charts.

#### Disadvantages:

1. Inkjet Printers are slower than dot matrix printers.
2. The cost of inkjet is more than a dot matrix printer.

### **Laser Printers:**

These are non-impact page printers. They use laser lights to produce the dots needed to form the characters to be printed on a page & hence the name laser printers.

#### **The output is generated in the following steps:**

**Step1:** The bits of data sent by processing unit act as triggers to turn the laser beam on & off.

**Step2:** The output device has a drum which is cleared & is given a positive electric charge. To print a page the modulated laser beam passing from the laser scans back & forth the surface of the drum. The positive electric charge on the drum is stored on just those parts of the drum surface which are exposed to the laser beam create the difference in electric which charges on the exposed drum surface.



Laser Printer



**Step3:** The laser exposed parts of the drum attract an ink powder known as toner.

**Step4:** The attracted ink powder is transferred to paper.

**Step5:** The ink particles are permanently fixed to the paper by using either heat or pressure technique.

**Step6:** The drum rotates back to the cleaner where a rubber blade cleans off the excess ink & prepares the drum to print the next page.

## Plotters

Plotters are a special type of output device. It is suitable for applications:

1. Architectural plan of the building.
2. CAD applications like the design of mechanical components of aircraft.
3. Many engineering applications.

### Plotter



**Advantage:**

1. It can produce high-quality output on large sheets.

2. It is used to provide the high precision drawing.
3. It can produce graphics of various sizes.
4. The speed of producing output is high.

### **Drum Plotter:**

It consists of a drum. Paper on which design is made is kept on the drum. The drum can rotate in both directions. Plotters comprised of one or more pen and penholders. The holders are mounted perpendicular to drum surface. The pens are kept in the holder, which can move left to the right as well as right to the left. The graph plotting program controls the movement of pen and drum.



Drum Plotter

### **Flatbed Plotter:**

It is used to draw complex design and graphs, charts. The Flatbed plotter can be kept over the table. The plotter consists of pen and holder. The pen can draw characters of various sizes. There can be one or more pens and pen holding mechanism. Each pen has ink of different color. Different colors help to produce multicolor design of document. The area of plotting is also variable. It can vary A4 to 21'\*52'.



## Flatbed Plotter

It is used to draw

1. Cars
2. Ships
3. Airplanes
4. Shoe and dress designing
5. Road and highway design

## Graphics Software:

There are two types of Graphics Software.

**1. General Purpose Packages:** Basic Functions in a general package include those for generating picture components (straight lines, polygons, circles and other figures), setting color and intensity values, selecting views, and applying transformations.

Example of general purpose package is the GL (Graphics Library), GKS, PHIGS, PHIGS+ etc.

**2. Special Purpose Packages:** These packages are designed for non programmers, so that these users can use the graphics packages, without knowing the inner details.

Example of special purpose package is

1. Painting programs
2. Package used for business purpose
3. Package used for medical systems.
4. CAD packages

Graphics Software There are two general categories of graphics software

♣ General programming packages:

♣ Provides extensive set of graphics functions for high level languages (FORTRAN, C etc).

♣ Basic functions include those for generating picture components (straight lines, polygons, circles, and other figures), setting color and intensity values, selecting views, and applying transformations.

♣ Example: GL(Graphics Library)

♣ Special-purpose application packages:

♣ Designed for nonprogrammers, so that users can generate displays without worrying about how graphics operations work.

♣ The interface to the graphics routines in such packages allows users to communicate with the programs in their own terms.

♣ Example: artist's painting programs and various business, medical, and CAD systems. Software standards Primary goal of standardized graphics software is portability. When packages are designed with standard graphics functions, software can be moved easily from one hardware system to another and used in different implementations and applications. International and national standards planning

organizations in many countries have cooperated in an effort to develop a generally accepted standard for computer graphics. After considerable effort, this work led to following standards:

♣ GKS (Graphical Kernel System): This system was adopted as the first graphics software standard by the International Standards Organization (ISO) and American National Standards Institute (ANSI). Although GKS was originally designed as a two-dimensional graphics package, a three-dimensional GKS extension was subsequently developed.

♣ PHIGS (Programmer's Hierarchical Interactive Graphics Standard): Extension to GKS, Increased Capabilities for object modeling, color specifications, surface rendering and picture manipulations are provided. Subsequently, an extension of PHIGS, called PHIGS+, was developed to provide three-dimensional surface-shading capabilities not available in PHIGS. Although PHIGS presents a specification for basic graphics functions, it does not provide a standard methodology for a graphics interface to output devices (i.e. still machine dependent). Nor does it specify methods for storing and transmitting pictures. Separate standards have been developed for these areas:

♣ CGI (Computer Graphics interface): Standardization for device interface

♣ CGM (Computer Graphics Metafile): Standards for archiving and transporting pictures

## UNIT II

### Output Primitives:

Output primitives are the geometric structures such as straight-line segments (pixel array) and polygon color areas, used to describe the shapes and colors of the objects. Points and straight-line segments are the simplest geometric components of pictures. Additional output primitive includes: circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon color areas and character strings. Here, we discuss picture generation algorithm by examining device-level algorithms for displaying two-dimensional output primitives, with emphasis on scan-conversion methods for raster graphics system.

### Points and Lines

- ♣ Point plotting is done in CRT monitor by turning on the electron beam to illuminate at the screen phosphor at the selected location.

- o Random-scan systems: stores point plotting instructions in the display list and coordinate values in these instructions are converted into deflection voltages that position the electron beam at selected location.

- o B/W raster system: Within frame buffer, bit value is set to 1 for specified screen position. Electron beam then sweeps across each horizontal scan line, it emits a burst of electrons (plots a point) whenever value of 1 is encountered in the frame buffer.

- o RGB raster system: Frame buffer is loaded with the color codes for the intensities that are to be displayed at the screen pixel positions.

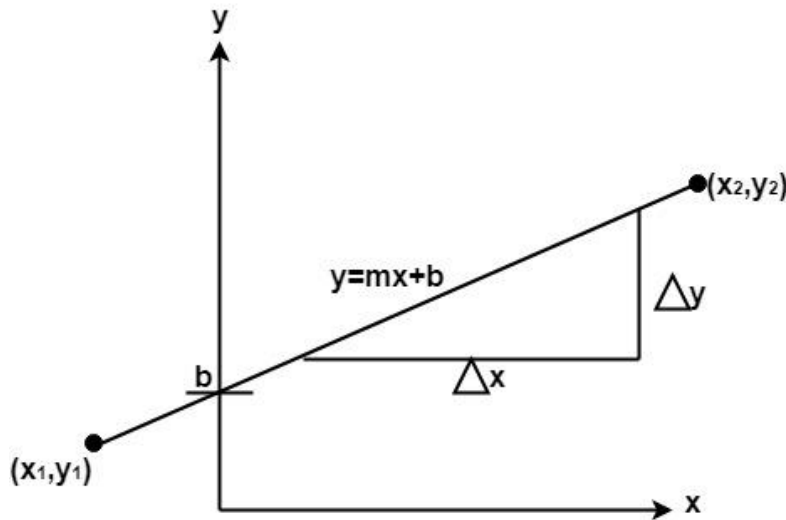
- ♣ Line drawing is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. An output device is then directed to fill in these positions between the endpoints.

o For analog devices (vector-pen plotter and random-scan display), a straight line can be drawn smoothly between two points. [Reason: linearly varying horizontal and vertical deflection voltages are generated that are proportional to the required changes in the x and y directions]

o Digital devices display a straight-line segment by plotting discrete points between two endpoints. Discrete integer coordinates are calculated from the equation of the line. Since rounding of coordinate values occur [viz. (4.48, 48.51) would be converted to (4, 49)], line is displayed with stairstep appearance.

## Scan Converting a Straight Line

A straight line may be defined by two endpoints & an equation. In fig the two endpoints are described by  $(x_1, y_1)$  and  $(x_2, y_2)$ . The equation of the line is used to determine the x, y coordinates of all the points that lie between these two endpoints.



Using the equation of a straight line,  $y = mx + b$  where  $m = \frac{\Delta y}{\Delta x}$  &  $b$  = the y intercept, we can find values of y by incrementing x from  $x = x_1$ , to  $x = x_2$ . By scan-converting these calculated x, y values, we represent the line as a sequence of pixels.

## Properties of Good Line Drawing Algorithm:

**1. Line should appear Straight:** We must appropriate the line by choosing addressable points close to it. If we choose well, the line will appear straight, if not, we shall produce crossed lines.

The lines must be generated parallel or at  $45^\circ$  to the x and y-axes. Other lines cause a problem: a line segment through it starts and finishes at addressable points, may happen to pass through no another addressable points in between.

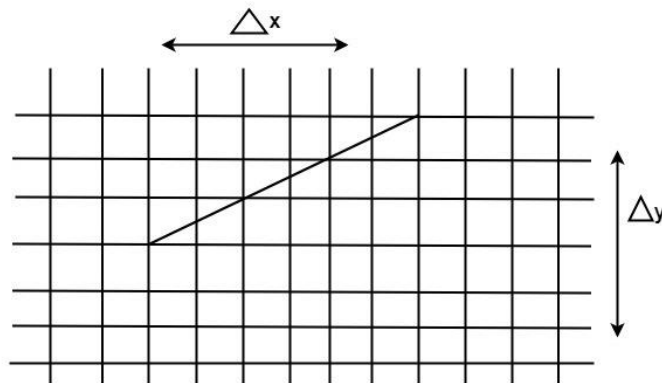


Fig: A straight line segment connecting 2 grid intersection may fail to pass through any other grid intersections.

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

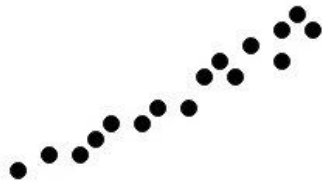


Fig: Uneven line density caused by bunching of dots.

**3. Lines should have constant density:** Line density is proportional to the no. of dots displayed divided by the length of the line.



To maintain constant density, dots should be equally spaced.

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

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

## **Bresenham's Line Algorithm**

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

In this method, next pixel selected is that one who has the least distance from true line.

The method works as follows:

Assume a pixel  $P_1'(x_1', y_1')$ , then select subsequent pixels one pixel position at a time in the horizontal direction toward  $P_2'(x_2', y_2')$ .

Once a pixel is chosen at any step

The next pixel is

1. Either the one to its right (lower-bound for the line)
2. One top its right and up (upper-bound for the line)

The line is best approximated by those pixels that fall the least distance from the path between  $P_1', P_2'$ .

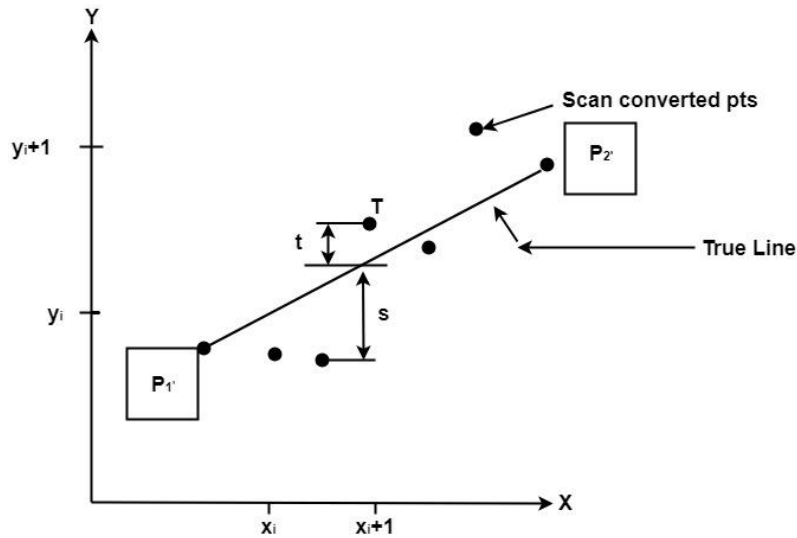


Fig: Scan Converting a line.

To choose the next one between the bottom pixel S and top pixel T.

If S is chosen

We have  $x_{i+1} = x_i + 1$  and  $y_{i+1} = y_i$

If T is chosen

We have  $x_{i+1} = x_i + 1$  and  $y_{i+1} = y_i + 1$

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

$$y = mx_{i+1} + b$$

$$y = m(x_i + 1) + b$$

The distance from S to the actual line in y direction

$$s = y - y_i$$

The distance from T to the actual line in y direction

$$t = (y_i + 1) - y$$

Now consider the difference between these 2 distance values

$$s - t$$

When  $(s-t) < 0 \Rightarrow s < t$

The closest pixel is S

When  $(s-t) \geq 0 \Rightarrow s > t$

The closest pixel is T

This difference is

$$\begin{aligned} s-t &= (y-y_i) - [(y_{i+1}) - y] \\ &= 2y - 2y_i - 1 \end{aligned}$$

$$s - t = 2m(x_i + 1) + 2b - 2y_i - 1$$

[Putting the value of (1)]

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

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

$$\begin{aligned} d_i &= \Delta x \left( 2 \frac{\Delta y}{\Delta x} (x_i + 1) + 2b - 2y_i - 1 \right) \\ &= 2\Delta x y_i - 2\Delta y - 1\Delta x + 2b - 2y_i\Delta x - \Delta x \end{aligned}$$

$$d_i = 2\Delta y \cdot x_i - 2\Delta x \cdot y_i + c$$

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

We can write the decision variable  $d_{i+1}$  for the next slip on

$$\begin{aligned} d_{i+1} &= 2\Delta y \cdot x_{i+1} - 2\Delta x \cdot y_{i+1} + c \\ d_{i+1} - d_i &= 2\Delta y \cdot (x_{i+1} - x_i) - 2\Delta x (y_{i+1} - y_i) \end{aligned}$$

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

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

## Special Cases

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

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

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

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

Finally, we calculate  $d_1$

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

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

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

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

## Advantage:

1. It involves only integer arithmetic, so it is simple.
2. It avoids the generation of duplicate points.
3. It can be implemented using hardware because it does not use multiplication and division.
4. It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

## Disadvantage:

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

## **Bresenham's Line Algorithm:**

**Step1:** Start Algorithm

**Step2:** Declare variable  $x_1, x_2, y_1, y_2, d, i_1, i_2, dx, dy$

**Step3:** Enter value of  $x_1, y_1, x_2, y_2$

Where  $x_1, y_1$  are coordinates of starting point

And  $x_2, y_2$  are coordinates of Ending point

**Step4:** Calculate  $dx = x_2 - x_1$

Calculate  $dy = y_2 - y_1$

Calculate  $i_1 = 2 * dy$

Calculate  $i_2 = 2 * (dy - dx)$

Calculate  $d = i_1 - dx$

**Step5:** Consider  $(x, y)$  as starting point and  $x_{end}$  as maximum possible value of  $x$ .

If  $dx < 0$

Then  $x = x_2$

$y = y_2$

$x_{end} = x_1$

If  $dx > 0$

Then  $x = x_1$

$y = y_1$

$x_{end} = x_2$

**Step6:** Generate point at  $(x, y)$  coordinates.

**Step7:** Check if whole line is generated.

If  $x \geq x_{end}$

Stop.

**Step8:** Calculate co-ordinates of the next pixel

If  $d < 0$

Then  $d = d + i_1$

If  $d \geq 0$

Then  $d = d + i_2$

Increment  $y = y + 1$

**Step9:** Increment  $x = x + 1$

**Step10:** Draw a point of latest  $(x, y)$  coordinates

**Step11:** Go to step 7

**Step12:** End of Algorithm

**Example:** Starting and Ending position of the line are  $(1, 1)$  and  $(8, 5)$ . Find intermediate points.

X	y	$d=d+I_1$ or $I_2$
1	1	$d+I_2=1+(-6)=-5$
2	2	$d+I_1=-5+8=3$
3	2	$d+I_2=3+(-6)=-3$
4	3	$d+I_1=-3+8=5$

5	3	$d+I_2=5+(-6)=-1$
6	4	$d+I_1=-1+8=7$
7	4	$d+I_2=7+(-6)=1$
8	5	

**Solution:**  $x_1=1$

$x_2=8$

$dx=x_2-$

$dy=y_2-$

$I_1=2* \Delta y=2*4=8$

$I_2=2*(\Delta y-\Delta x)=2*(4-7)=-6$

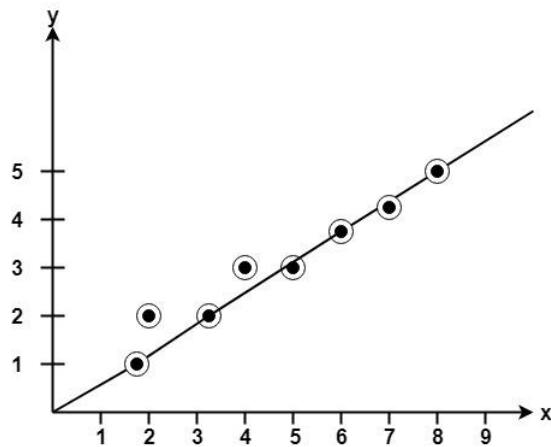
$d = I_1-\Delta x=8-7=1$

$y_1=1$

$y_2=5$

$x_1=8-1=7$

$y_1=5-1=4$



### Program to implement Bresenham's Line Drawing Algorithm:

1. `#include<stdio.h>`
2. `#include<graphics.h>`
3. `void drawline(int x0, int y0, int x1, int y1)`

```

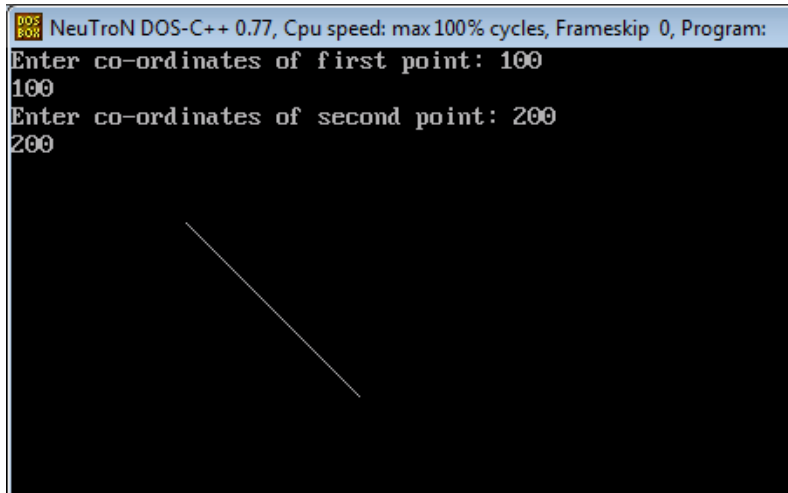
4. {
5.     int dx, dy, p, x, y;
6.     dx=x1-x0;
7.     dy=y1-y0;
8.     x=x0;
9.     y=y0;
10.    p=2*dy-dx;
11.    while(x<x1)
12.    {
13.        if(p>=0)
14.        {
15.            putpixel(x,y,7);
16.            y=y+1;
17.            p=p+2*dy-2*dx;
18.        }
19.        else
20.        {
21.            putpixel(x,y,7);
22.            p=p+2*dy;}
23.            x=x+1;
24.        }
25.    }
26. int main()
27. {
28.     int gdriver=DETECT, gmode, error, x0, y0, x1, y1;
29.     initgraph(&gdriver, &gmode, "c:\\turbo3\\bgi");
30.     printf("Enter co-ordinates of first point: ");
31.     scanf("%d%d", &x0, &y0);
32.     printf("Enter co-ordinates of second point: ");
33.     scanf("%d%d", &x1, &y1);
34.     drawline(x0, y0, x1, y1);

```



```
35. return 0;
36. }
```

### Output:



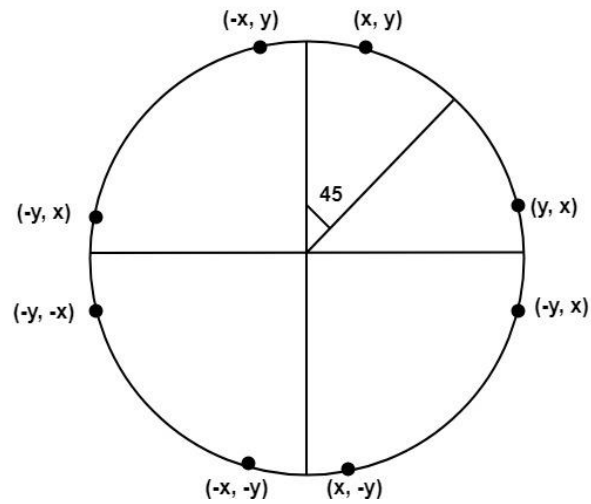
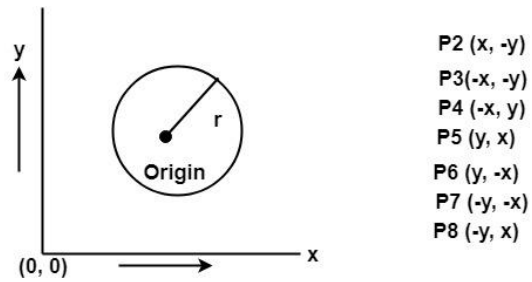
```
DOS
NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program:
Enter co-ordinates of first point: 100
100
Enter co-ordinates of second point: 200
200
```

The screenshot shows a DOS window with a black background and white text. The text displays the program's input and output. The first input is '100', followed by the second input '200'. Below the text, a white line is drawn on the black background, starting from the left and sloping downwards to the right, representing a line segment between the two input points.

### Defining a Circle:

Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are two octants. If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.

For drawing, circle considers it at the origin. If a point is  $P_1(x, y)$ , then the other seven points will be



So we will calculate only 45°arc. From which the whole circle can be determined easily.

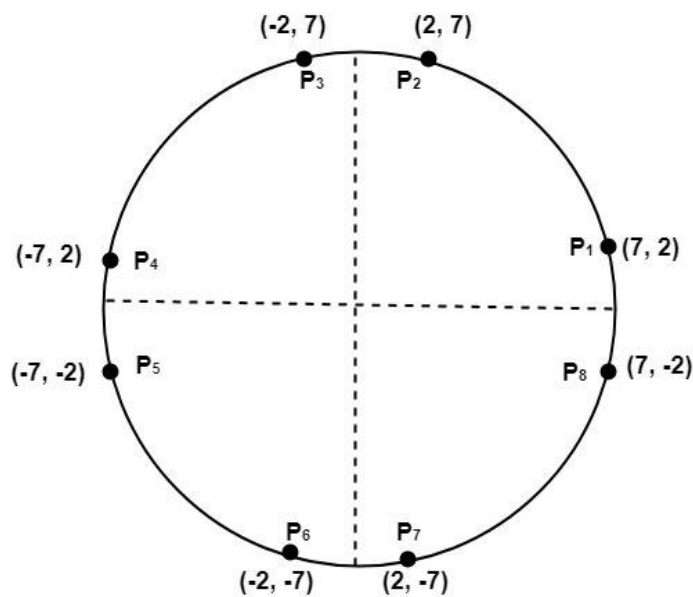
If we want to display circle on screen then the putpixel function is used for eight points as shown below:

```
putpixel (x, y, color)
putpixel (x, -y, color)
putpixel (-x, y, color)
putpixel (-x, -y, color)
putpixel (y, x, color)
putpixel (y, -x, color)
putpixel (-y, x, color)
putpixel (-y, -x, color)
```

**Example:** Let we determine a point  $(2, 7)$  of the circle then other points will be  $(2, -7)$ ,  $(-2, -7)$ ,  $(-2, 7)$ ,  $(7, 2)$ ,  $(-7, 2)$ ,  $(-7, -2)$ ,  $(7, -2)$

These seven points are calculated by using the property of reflection. The reflection is accomplished in the following way:

The reflection is accomplished by reversing  $x, y$  co-ordinates.



**Eight way symmetry of a Circle**

here are two standards methods of mathematically defining a circle centered at the origin.

1. Defining a circle using Polynomial Method
2. Defining a circle using Polar Co-ordinates

### **Defining a circle using Polynomial Method:**

The first method defines a circle with the second-order polynomial equation as shown in fig:

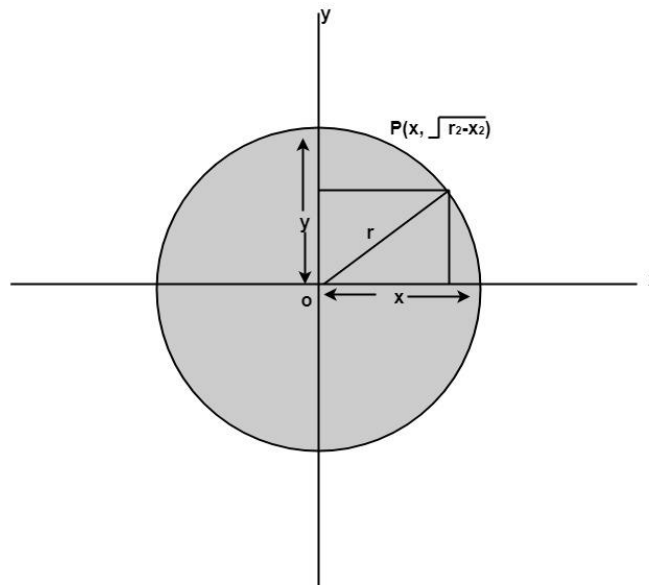
$$y^2=r^2-x^2$$

Where  $x$  = the  $x$  coordinate

$y$  = the  $y$  coordinate

$r$  = the circle radius

With the method, each  $x$  coordinate in the sector, from  $90^\circ$  to  $45^\circ$ , is found by stepping  $x$  from 0 to  $\frac{r}{\sqrt{2}}$  & each  $y$  coordinate is found by evaluating  $\sqrt{r^2 - x^2}$  for each step of  $x$ .



## Algorithm:

**Step1:** Set the initial variables

$r$  = circle radius

$(h, k)$  = coordinates of circle center

$x=0$

$I$  = step size

$x_{end} = \frac{r}{\sqrt{2}}$

**Step2:** Test to determine whether the entire circle has been scan-converted.

If  $x > x_{end}$  then stop.

**Step3:** Compute  $y = \sqrt{r^2 - x^2}$

**Step4:** Plot the eight points found by symmetry concerning the center (h, k) at the current (x, y) coordinates.

Plot (x + h, y +k)	Plot (-x + h, -y + k)
Plot (y + h, x + k)	Plot (-y + h, -x + k)
Plot (-y + h, x + k)	Plot (y + h, -x + k)
Plot (-x + h, y + k)	Plot (x + h, -y + k)

**Step5:** Increment  $x = x + i$

**Step6:** Go to step (ii).

### Program to draw a circle using Polynomial Method:

```
1. #include<graphics.h>
2. #include<conio.h>
3. #include<math.h>
4. voidsetPixel(int x, int y, int h, int k)
5. {
6.     putpixel(x+h, y+k, RED);
7.     putpixel(x+h, -y+k, RED);
8.     putpixel(-x+h, -y+k, RED);
9.     putpixel(-x+h, y+k, RED);
10.    putpixel(y+h, x+k, RED);
11.    putpixel(y+h, -x+k, RED);
12.    putpixel(-y+h, -x+k, RED);
13.    putpixel(-y+h, x+k, RED);
14. }
15. main()
16. {
17.     intgd=0, gm,h,k,r;
18.     double x,y,x2;
19.     h=200, k=200, r=100;
20.     initgraph(&gd, &gm, "C:\\TC\\BGI ");
21.     setbkcolor(WHITE);
22.     x=0,y=r;
23.     x2 = r/sqrt(2);
```

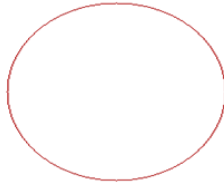
```

24. while(x<=x2)
25. {
26.     y = sqrt(r*r - x*x);
27.     setPixel(floor(x), floor(y), h,k);
28.     x += 1;
29. }
30. getch();
31. closegraph();
32. return 0;
33. }

```

**Output:**

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip: 0, Program: TC



## Defining a circle using Polar Co-ordinates:

The second method of defining a circle makes use of polar coordinates as shown in fig:

$$x=r \cos \theta \quad y = r \sin \theta$$

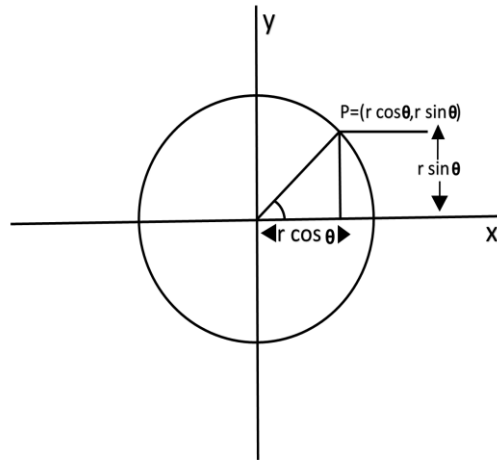
Where  $\theta$ =current angle

$r$  = circle radius

$x$  = x coordinate

$y$  = y coordinate

By this method,  $\theta$  is stepped from 0 to  $\frac{\pi}{4}$  & each value of  $x$  &  $y$  is calculated.



## Algorithm:

**Step1:** Set the initial variables:

$r$  = circle radius  
 $(h, k)$  = coordinates of the circle center  
 $i$  = step size

$$\theta_{end} = \left(\frac{2\pi}{7}\right) / 4$$

$$\theta = 0$$

**Step2:** If  $\theta > \theta_{end}$  then stop.

**Step3:** Compute

$$x = r * \cos \theta \quad y = r * \sin \theta$$

**Step4:** Plot the eight points, found by symmetry i.e., the center  $(h, k)$ , at the current  $(x, y)$  coordinates.

Plot $(x + h, y + k)$	Plot $(-x + h, -y + k)$
Plot $(y + h, x + k)$	Plot $(-y + h, -x + k)$

Plot  $(-y + h, x + k)$       Plot  $(y + h, -x + k)$   
Plot  $(-x + h, y + k)$       Plot  $(x + h, -y + k)$

**Step5:** Increment  $\theta = \theta + i$

**Step6:** Go to step (ii).

## Program to draw a circle using Polar Coordinates:

```
1. #include <graphics.h>
2. #include <stdlib.h>
3. #define color 10
4. void eightWaySymmetricPlot(int xc,int yc,int x,int y)
5. {
6.     putpixel(x+xc,y+yc,color);
7.     putpixel(x+xc,-y+yc,color);
8.     putpixel(-x+xc,-y+yc,color);
9.     putpixel(-x+xc,y+yc,color);
10.    putpixel(y+xc,x+yc,color);
11.    putpixel(y+xc,-x+yc,color);
12.    putpixel(-y+xc,-x+yc,color);
13.    putpixel(-y+xc,x+yc,color);
14. }
15. void PolarCircle(int xc,int yc,int r)
16. {
17.     int x,y,d;
18.     x=0;
19.     y=r;
20.     d=3-2*r;
21.     eightWaySymmetricPlot(xc,yc,x,y);
22.     while(x<=y)
23.     {
24.         if(d<=0)
25.         {
26.             d=d+4*x+6;
27.         }
28.         else
29.         {
30.             d=d+4*x-4*y+10;
31.             y=y-1;
32.         }
33.         x=x+1;
```

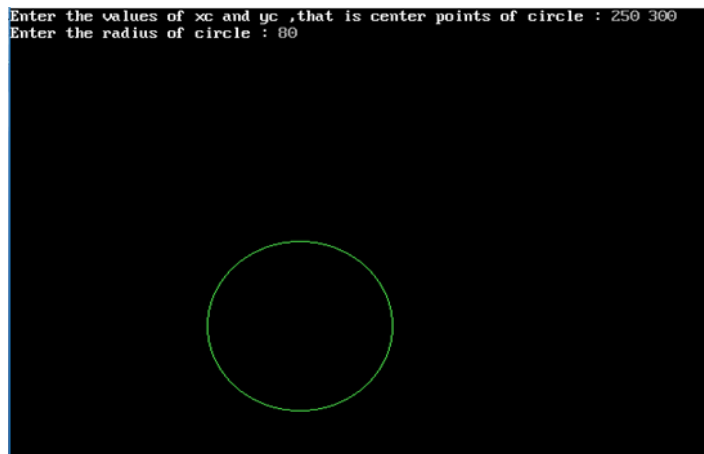


```

34.     eightWaySymmetricPlot(xc,yc,x,y);
35. }
36. }
37. int main(void)
38. {
39.     int gdriver = DETECT, gmode, errorcode;
40.     int xc,yc,r;
41.     initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
42. errorcode = graphresult();
43. if (errorcode != grOk)
44.     {
45.         printf("Graphics error: %s\n", grapherrormsg(errorcode));
46.         printf("Press any key to halt:");
47.         getch();
48.         exit(1);
49.     }
50. printf("Enter the values of xc and yc ,that is center points of circle : ");
51.     scanf("%d%d",&xc,&yc);
52.     printf("Enter the radius of circle : ");
53.     scanf("%d",&r);
54.     PolarCircle(xc,yc,r);
55.     getch();
56.     closegraph();
57.     return 0;
58. }

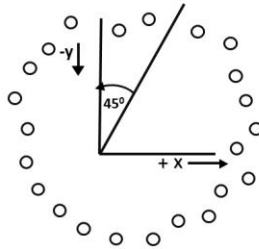
```

**Output:**

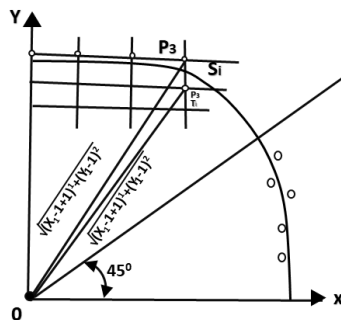


## Bresenham's Circle Algorithm:

Scan-Converting a circle using Bresenham's algorithm works as follows: Points are generated from  $90^\circ$  to  $45^\circ$ , moves will be made only in the  $+x$  &  $-y$  directions as shown in fig:



The best approximation of the true circle will be described by those pixels in the raster that falls the least distance from the true circle. We want to generate the points from.



$90^\circ$  to  $45^\circ$ . Assume that the last scan-converted pixel is  $P_1$  as shown in fig. Each new point closest to the true circle can be found by taking either of two actions.

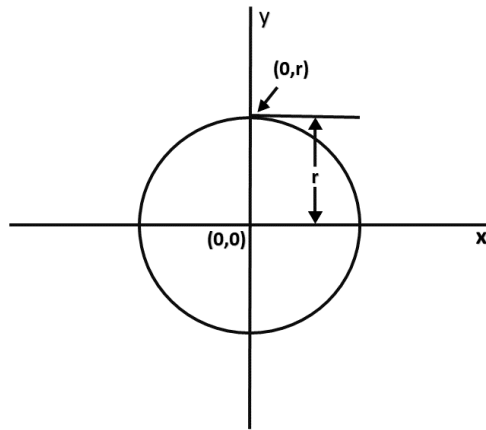
1. Move in the x-direction one unit or
2. Move in the x- direction one unit & move in the negative y-direction one unit.

Let  $D(S_i)$  is the distance from the origin to the true circle squared minus the distance to point  $P_3$  squared.  $D(T_i)$  is the distance from the origin to the true circle squared minus the distance to point  $P_2$  squared. Therefore, the following expressions arise.

$$D(S_i) = (x_{i-1} + 1)^2 + y_{i-1}^2 - r^2$$

$$D(T_i) = (x_{i-1} + 1)^2 + (y_{i-1} - 1)^2 - r^2$$

Since  $D(S_i)$  will always be +ve &  $D(T_i)$  will always be -ve, a decision variable  $d$  may be defined as follows:



$$d_i = D(S_i) + D(T_i)$$

Therefore,

$$d_i = (x_{i-1} + 1)^2 + y_{i-1}^2 - r^2 + (x_{i-1} + 1)^2 + (y_{i-1} - 1)^2 - r^2$$

From this equation, we can drive initial values of  $d_i$  as

If it is assumed that the circle is centered at the origin, then at the first step  $x = 0$  &  $y = r$ .

Therefore,

$$\begin{aligned}d_i &= (0+1)^2 + r^2 - r^2 + (0+1)^2 + (r-1)^2 - r^2 \\ &= 1+1+r^2-2r+1-r^2 \\ &= 3 - 2r\end{aligned}$$

Thereafter, if  $d_i < 0$ , then only x is incremented.

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

& if  $d_i \geq 0$ , then x & y are incremented

$$x_{i+1} = x_i + 1$$

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

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

## **Bresenham's Circle Algorithm:**

**Step1:** Start Algorithm

**Step2:** Declare p, q, x, y, r, d variables

p, q are coordinates of the center of the circle  
r is the radius of the circle

**Step3:** Enter the value of r

**Step4:** Calculate  $d = 3 - 2r$

**Step5:** Initialize  $x=0$   
&  $y=r$

**Step6:** Check if the whole circle is scan converted

If  $x \geq y$   
Stop

**Step7:** Plot eight points by using concepts of eight-way symmetry. The center is at (p, q).

Current active pixel is (x, y).

putpixel (x+p, y+q)  
putpixel (y+p, x+q)  
putpixel (-y+p, x+q)  
putpixel (-x+p, y+q)

```

putpixel (-x+p, -y+q)
putpixel (-y+p, -x+q)
putpixel (y+p, -x+q)
putpixel (x+p, -y-q)

```

**Step8:** Find location of next pixels to be scanned

```

If d < 0
then d = d + 4x + 6
increment x = x + 1
If d ≥ 0
then d = d + 4 (x - y) + 10
increment x = x + 1
decrement y = y - 1

```

**Step9:** Go to step 6

**Step10:** Stop Algorithm

**Example:** Plot 6 points of circle using Bresenham Algorithm. When radius of circle is 10 units. The circle has centre (50, 50).

**Solution:** Let  $r = 10$  (Given)

**Step1:** Take initial point (0, 10)

```

d = 3 - 2r
d = 3 - 2 * 10 = -17
d < 0 ∴ d = d + 4x + 6
    = -17 + 4 (0) + 6
    = -11

```

**Step2:** Plot (1, 10)

```

d = d + 4x + 6 (∴ d < 0)
  = -11 + 4 (1) + 6
  = -1

```

**Step3:** Plot (2, 10)

```

d = d + 4x + 6 (∴ d < 0)
  = -1 + 4 x 2 + 6
  = 13

```

**Step4:** Plot (3, 9)  $d$  is  $> 0$  so  $x = x + 1, y = y - 1$

```

d = d + 4 (x-y) + 10 (∴ d > 0)
  = 13 + 4 (3-9) + 10
  = 13 + 4 (-6) + 10
  = 23-24=-1

```

**Step5:** Plot (4, 9)

$$\begin{aligned}d &= -1 + 4x + 6 \\ &= -1 + 4(4) + 6 \\ &= 21\end{aligned}$$

**Step6:** Plot (5, 8)

$$\begin{aligned}d &= d + 4(x-y) + 10 (\because d > 0) \\ &= 21 + 4(5-8) + 10 \\ &= 21-12 + 10 = 19\end{aligned}$$

So  $P_1(0,0) \Rightarrow (50,50)$

$P_2(1,10) \Rightarrow (51,60)$

$P_3(2,10) \Rightarrow (52,60)$

$P_4(3,9) \Rightarrow (53,59)$

$P_5(4,9) \Rightarrow (54,59)$

$P_6(5,8) \Rightarrow (55,58)$

## Program to draw a circle using Bresenham's circle drawing algorithm:

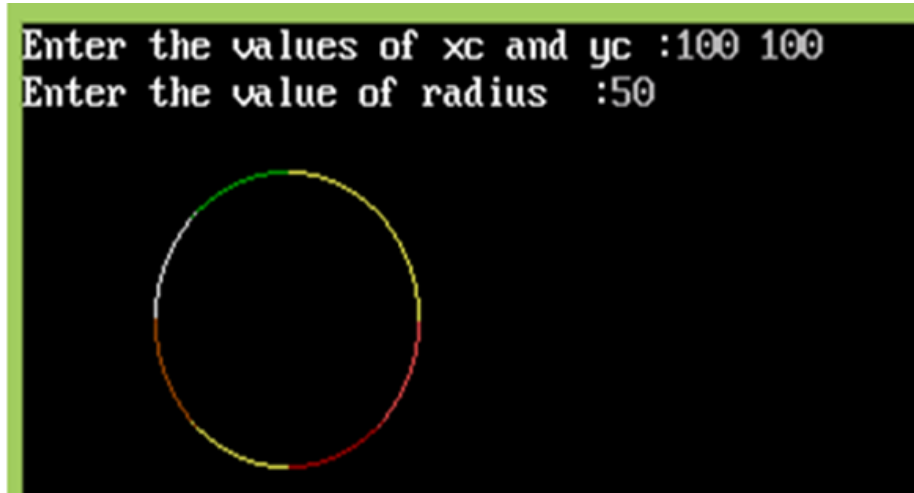
```
1. #include <graphics.h>
2. #include <stdlib.h>
3. #include <stdio.h>
4. #include <conio.h>
5. #include <math.h>
6. void EightWaySymmetricPlot(int xc,int yc,int x,int y)
7. {
8.     putpixel(x+xc,y+yc,RED);
9.     putpixel(x+xc,-y+yc,YELLOW);
10.    putpixel(-x+xc,-y+yc,GREEN);
11.    putpixel(-x+xc,y+yc,YELLOW);
12.    putpixel(y+xc,x+yc,12);
13.    putpixel(y+xc,-x+yc,14);
14.    putpixel(-y+xc,-x+yc,15);
15.    putpixel(-y+xc,x+yc,6);
16. }
17. void BresenhamCircle(int xc,int yc,int r)
18. {
19.     int x=0,y=r,d=3-(2*r);
20.     EightWaySymmetricPlot(xc,yc,x,y);
21.     while(x<=y)
22.     {
```

```

23.  if(d<=0)
24.      {
25.          d=d+(4*x)+6;
26.      }
27.  else
28.      {
29.          d=d+(4*x)-(4*y)+10;
30.          y=y-1;
31.      }
32.      x=x+1;
33.      EightWaySymmetricPlot(xc,yc,x,y);
34.  }
35. }
36. int main(void)
37. {
38.     /* request auto detection */
39.     int xc,yc,r,gdriver = DETECT, gmode, errorcode;
40.     /* initialize graphics and local variables */
41.     initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");
42.     /* read result of initialization */
43.     errorcode = graphresult();
44.     if (errorcode != grOk) /* an error occurred */
45.     {
46.         printf("Graphics error: %s\n", grapherrormsg(errorcode));
47.         printf("Press any key to halt:");
48.         getch();
49.         exit(1); /* terminate with an error code */
50.     }
51.     printf("Enter the values of xc and yc :");
52.     scanf("%d%d",&xc,&yc);
53.     printf("Enter the value of radius :");
54.     scanf("%d",&r);
55.     BresenhamCircle(xc,yc,r);
56.     getch();
57.     closegraph();
58.     return 0;
59. }

```

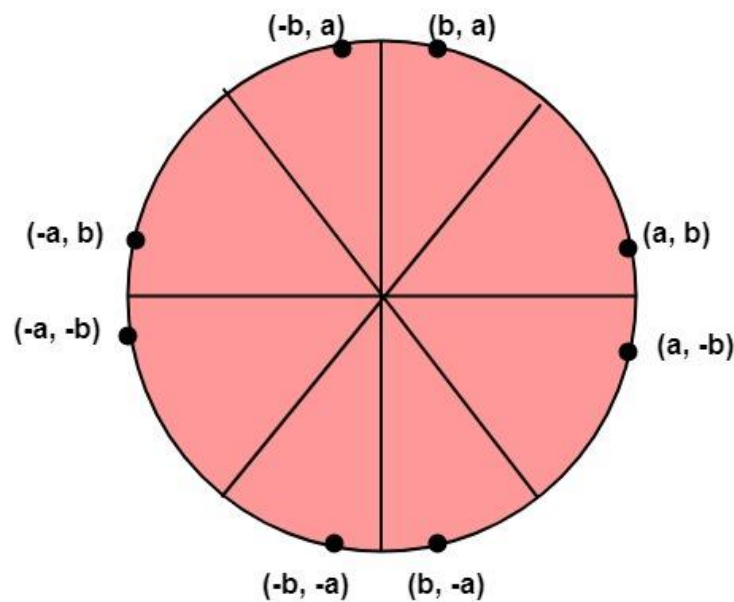
Output:



## Midpoint Circle Algorithm

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

$$f(x, y) = x^2 + y^2 - r^2 \quad \left[ \begin{array}{l} < 0 \text{ for } (x, y) \text{ inside the circle} \\ = 0 \text{ for } (x, y) \text{ on the circle} \\ > 0 \text{ for } (x, y) \text{ outside the circle} \end{array} \right] \dots \text{equation 1}$$





Now, consider the coordinates of the point halfway between pixel T and pixel S

This is called midpoint  $(x_{i+1}, y_i - \frac{1}{2})$  and we use it to define a decision parameter:

$$P_i = f(x_{i+1}, y_i - \frac{1}{2}) = (x_{i+1})^2 + (y_i - \frac{1}{2})^2 - r^2 \dots\dots\dots \text{equation 2}$$

If  $P_i$  is -ve  $\implies$  midpoint is inside the circle and we choose pixel T

If  $P_i$  is +ve  $\implies$  midpoint is outside the circle (or on the circle) and we choose pixel S.

The decision parameter for the next step is:

$$P_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - \frac{1}{2})^2 - r^2 \dots\dots\dots \text{equation 3}$$

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

$$\begin{aligned} P_{i+1} - P_i &= ((x_i + 1) + 1)^2 - (x_i + 1)^2 + (y_{i+1} - \frac{1}{2})^2 - (y_i - \frac{1}{2})^2 \\ &= x_i^2 + 4 + 4x_i - x_i^2 + 1 - 2x_i + y_{i+1}^2 + \frac{1}{4} - y_{i+1} - y_i^2 - \frac{1}{4} - y_i \\ &= 2(x_i + 1) + 1 + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i) \\ P_{i+1} &= P_i + 2(x_i + 1) + 1 + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i) \dots\dots\dots \text{equation 4} \end{aligned}$$

If pixel T is chosen  $\implies P_i < 0$

We have  $y_{i+1} = y_i$

If pixel S is chosen  $\implies P_i \geq 0$

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

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

We can continue to simplify this in n terms of  $(x_i, y_i)$  and get

$$P_{i+1} = \begin{cases} P_i + 2x_i + 3, & \text{if } P_i < 0 \\ P_i + 2(x_i - y_i) + 5, & \text{if } P_i \geq 0 \end{cases} \dots\dots\dots \text{equation 6}$$

Now, initial value of  $P_1(0, r)$  from equation 2

$$P_1 = (0 + 1)^2 + \left(r - \frac{1}{2}\right)^2 - r^2$$
$$= 1 + \frac{1}{4} - r^2 = \frac{5}{4} - r$$

We can put  $\frac{5}{4} \cong 1$

$\therefore r$  is an integer

So,  $P_1 = 1 - r$

### Algorithm:

**Step1:** Put  $x = 0$ ,  $y = r$  in equation 2

We have  $p = 1 - r$

**Step2:** Repeat steps while  $x \leq y$

Plot  $(x, y)$

If  $(p < 0)$

Then set  $p = p + 2x + 3$

Else

$p = p + 2(x - y) + 5$

$y = y - 1$  (end if)

$x = x + 1$  (end loop)

**Step3:** End

### Program to draw a circle using Midpoint Algorithm:

1. `#include <graphics.h>`
2. `#include <stdlib.h>`
3. `#include <math.h>`
4. `#include <stdio.h>`
5. `#include <conio.h>`
6. `#include <iostream.h>`
7. `class bresen`

```

8. {
9.   float x, y,a, b, r, p;
10.  public:
11.  void get ();
12.  void cal ();
13. };
14.  void main ()
15.  {
16.  bresen b;
17.  b.get ();
18.  b.cal ();
19.  getch ();
20.  }
21.  Void bresen :: get ()
22.  {
23.  cout<<"ENTER CENTER AND RADIUS";
24.  cout<< "ENTER (a, b)";
25.  cin>>a>>b;
26.  cout<<"ENTER r";
27.  cin>>r;
28.  }
29.  void bresen ::cal ()
30.  {
31.  /* request auto detection */
32.  int gdriver = DETECT,gmode, errorcode;
33.  int midx, midy, i;
34.  /* initialize graphics and local variables */
35.  initgraph (&gdriver, &gmode, " ");
36.  /* read result of initialization */
37.  errorcode = graphresult ();
38.  if (errorcode != grOK) /*an error occurred */
39.  {
40.  printf("Graphics error: %s \n", grapherrormsg (errorcode));
41.  printf ("Press any key to halt:");
42.  getch ();
43.  exit (1); /* terminate with an error code */
44.  }
45.  x=0;
46.  y=r;
47.  putpixel (a, b+r, RED);

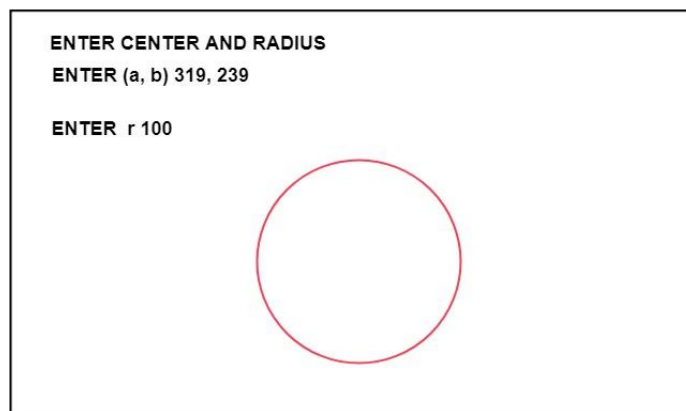
```

```

48. putpixel (a, b-r, RED);
49. putpixel (a-r, b, RED);
50. putpixel (a+r, b, RED);
51. p=5/4-r;
52. while (x<=y)
53. {
54.     If (p<0)
55.     p+= (4*x)+6;
56.     else
57.     {
58.         p+=(2*(x-y))+5;
59.         y--;
60.     }
61.     x++;
62.     putpixel (a+x, b+y, RED);
63.     putpixel (a-x, b+y, RED);
64.     putpixel (a+x, b-y, RED);
65.     putpixel (a-x, b-y, RED);
66.     putpixel (a+x, b+y, RED);
67.     putpixel (a+x, b-y, RED);
68.     putpixel (a-x, b+y, RED);
69.     putpixel (a-x, b-y, RED);
70. }
71. }

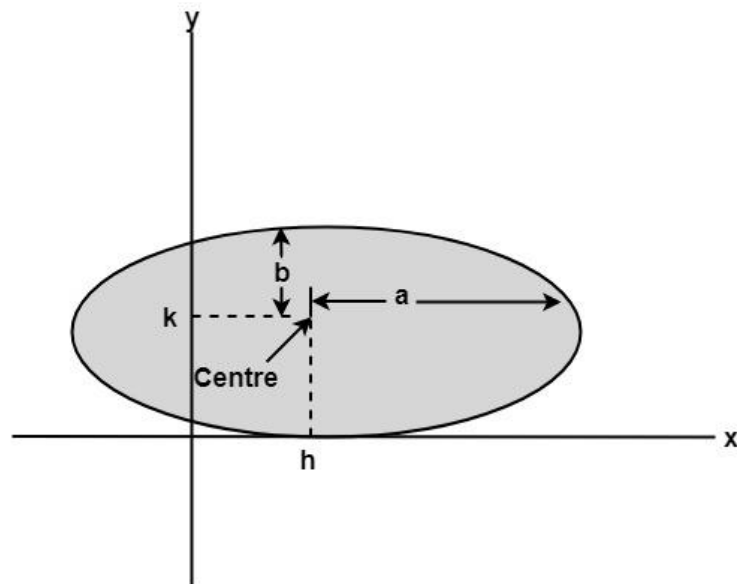
```

**Output:**



## Scan Converting an Ellipse:

The ellipse is also a symmetric figure like a circle but is four-way symmetry rather than eight-way.



### Program to Implement Ellipse Drawing Algorithm:

```
1. #include<stdio.h>
2. #include<conio.h>
3. #include<graphics.h>
4. #include<math.h>
5. void disp();
6. float x,y;
7. int xc,yc;
8. void main()
9. {
10.     intgd=DETECT,gm,a,b;
11.     float p1,p2;
12.     clrscr();
13.     initgraph(&gd,&gm,"c:\\turbo3\\bgi");
14.     printf("*** Ellipse Generating Algorithm ***\n");
15.     printf("Enter the value of Xc\t");
16.     scanf("%d",&xc);
17.     printf("Enter the value of yc\t");
18.     scanf("%d",&yc);
19.     printf("Enter X axis length\t");
20.     scanf("%d",&a);
21.     printf("Enter Y axis length\t");
```

```

22.     scanf("%d",&b);
23.     x=0;y=b;
24.     disp();
25.     p1=(b*b)-(a*a*b)+(a*a)/4;
26.     while((2.0*b*b*x)<=(2.0*a*a*y))
27.     {
28.         x++;
29.         if(p1<=0)
30.             p1=p1+(2.0*b*b*x)+(b*b);
31.         else
32.     {
33.             y--;
34.             p1=p1+(2.0*b*b*x)+(b*b)-(2.0*a*a*y);
35.     }
36.         disp();
37.         x=-x;
38.         disp();
39.         x=-x;
40.         delay(50);
41.     }
42.     x=a;
43.     y=0;
44.     disp();
45.     p2=(a*a)+2.0*(b*b*a)+(b*b)/4;
46.     while((2.0*b*b*x)>(2.0*a*a*y))
47.     {
48.         y++;
49.         if(p2>0)
50.             p2=p2+(a*a)-(2.0*a*a*y);
51.         else
52.     {
53.             x--;
54.             p2=p2+(2.0*b*b*x)-(2.0*a*a*y)+(a*a);
55.     }
56.         disp();
57.         y=-y;
58.         disp();
59.         y=-y;
60.         delay(50);
61.     }

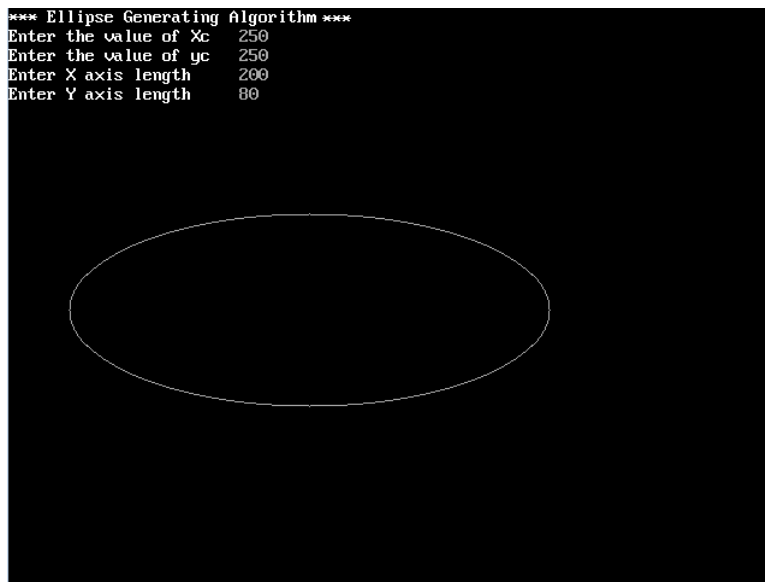
```

```

62.         getch();
63.         closegraph();
64.     }
65.     void disp()
66.     {
67.         putpixel(xc+x,yc+y,7);
68.         putpixel(xc-x,yc+y,7);
69.         putpixel(xc+x,yc-y,7);
70.         putpixel(xc-x,yc-y,7);
71.     }

```

### Output:



## ATTRIBUTES

The features or characteristics of an output primitive are known as Attribute. In other words, any parameter that affects the way a primitive is to be displayed is known as Attribute. Some attributes, such as colour and size, are basic characteristics of primitive. Some attributes control the basic display properties of primitives. For example, lines can be dotted or dashed, thin or thick. Areas can be filled with one colour or with multiple colours pattern. Text can appear from left to right, slanted or vertical.

## LINE ATTRIBUTES

Basic attributes of a straight-line segment are its type, its width, and its color. In some graphics packages, lines can also be displayed **using** selected pen or brush options.

## Line Type

Solid lines, Dashed lines, and Dotted lines.

We modify a line drawing algorithm to generate such lines by setting the length and spacing of displayed solid sections along the line path.

A dashed line could be displayed by generating an inter dash spacing that is equal to the length of the solid sections. Both the length of the dashes and the inter dash spacing are often specified as user options. A dotted line can be displayed by generating very short dashes with the spacing equal to or greater than the dash size. Similar methods are used to produce other line-type variations.



i. Solid Line



ii. Dashed Line



iii. Dotted Line

To set line type attributes in a **PHIGS** application program, a user invokes the function.

### **setLinetype (It)**

Where parameter **I t** is assigned a positive integer value of 1,2,3, or 4 to generate lines that are, respectively, solid, dashed, dotted, or dash-dotted.



## Line Width

We set the line-width attribute with the command: Line-width parameter `lw.` is assigned a positive number to indicate the relative width of the line to be displayed. A value of  $1$  specifies a standard-width line. On.

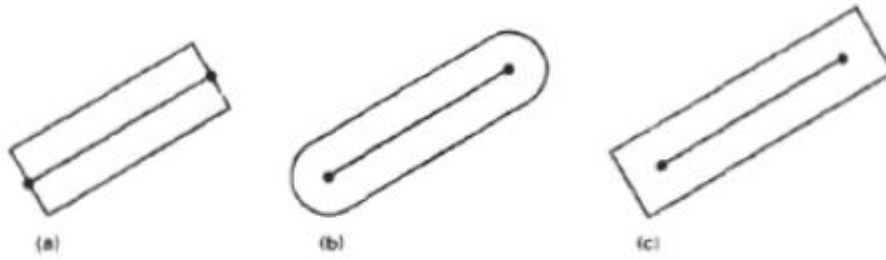
For lines with slope magnitude greater than  $1$ , we can plot thick lines with horizontal spans, alternately picking up pixels to the right and left of the line path.

Problem with implementing width options using horizontal or vertical pixel spans is that the method produces lines whose ends are horizontal or vertical regardless of the slope of the line. This effect is more noticeable with very thick lines. We can adjust the shape of the line ends to give them a better appearance by adding line caps

One kind of line cap is the butt cap obtained by adjusting the end positions of the component parallel lines so that the thick line is displayed with square ends that are perpendicular to the line path. If the specified line has slope  $m$ , the square end of the thick line has slope  $-1/m$ .

Another line cap is the round cap obtained by adding a filled semicircle to each butt cap. The circular arcs are centered on the line endpoints and have a diameter equal to the line thickness.

A third type of line cap is the projecting square cap. Here, we simply extend the line and add butt caps that are positioned one-half of the line width beyond the specified endpoints.



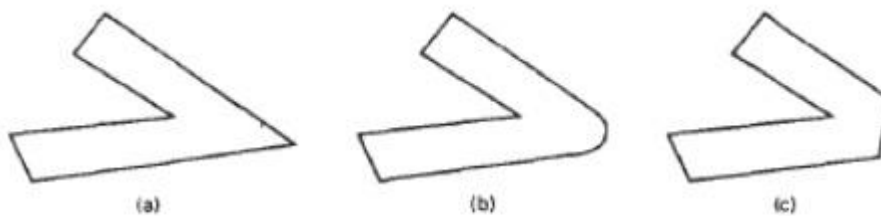
*Figure 4-5*  
Thick lines drawn with (a) butt caps, (b) round caps, and (c) projecting square caps.

We can generate thick polylines that are smoothly joined at the cost of additional processing at the segment endpoints.

A miter join is accomplished by extending the outer boundaries of each of the two lines until they meet.

A round join is produced by capping the connection between the two segments with a circular boundary whose diameter is equal to the linewidth.

And a bevel join is generated by displaying the line segments with butt caps and filling in the triangular gap where the segments meet.



*Figure 4-6*  
Thick line segments connected with (a) miter join, (b) round join, and (c) bevel join.

## Pen and Brush Options

Lines can be displayed with pen or brush selections. Options in this category include shape, size, and pattern.

These shapes can be stored in a pixel mask that identifies the array of pixel positions that are to be set along the line path. Lines generated with pen (or brush) shapes can be displayed in various widths by changing the size of the mask.

## **Line Color**

When a system provides color (or intensity) options, a parameter giving the current.

color index is included in the list of system-attribute values. A polyline routine displays a line in the current color by setting this color value in the framebuffer at pixel locations along the line path using the **setpixel** procedure.

The number of color choices depends on the number of bits available per pixel in the frame buffer. We set the line color value in **PHIGS** with the function

Set PolylineColourIndex (le)

## **CURVE ATTRIBUTES**

Parameters for curve attributes are the **same** as those for line segments. We can display curves with varying colors, widths, dotdash patterns, and available pen or brush options.

## **AREA FILL ATTRIBUTES**

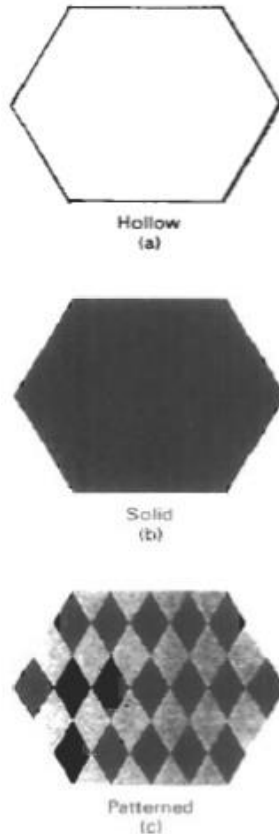


Figure 4-18

## AREA-FILL ATTRIBUTES

Options for filling a defined region include a choice between a solid color or a patterned fill and choices for the particular colors and patterns. These fill options can be applied to polygon regions or to areas defined with curved boundaries, depending on the capabilities of the available package. In addition, areas can be painted using various brush styles, colors, and transparency parameters.

### Fill Styles

Areas are displayed with three basic fill styles: hollow with a color border, filled with a solid color, or filled with a specified pattern or design. A basic fill style is selected in a PHIGS program with the function

```
setInteriorStyle (fs)
```

Values for the fill-style parameter *fs* include *hollow*, *solid*, and *pattern* (Fig. 4-18). Another value for fill style is *hatch*, which is used to fill an area with selected hatching patterns—parallel lines or crossed lines—as in Fig. 4-19. As with line attributes, a selected fill-style value is recorded in the list of system attributes and applied to fill the interiors of subsequently specified areas. Fill selections for parameter *fs* are normally applied to polygon areas, but they can also be implemented to fill regions with curved boundaries.

Hollow areas are displayed using only the boundary outline, with the interior color the same as the background color. A solid fill is displayed in a single color up to and including the borders of the region. The color for a solid interior or for a hollow area outline is chosen with

```
setInteriorColourIndex (fc)
```

where fill-color parameter *fc* is set to the desired color code. A polygon hollow

fill

is generated with a line drawing routine as a closed polyline

## Pattern Fill

We select fill patterns with

```
setInteriorStyleIndex (pi)
```

where pattern index parameter *pi* specifies a table position. For example, the following set of statements would fill the area defined in the *fillArea* command with the second pattern type stored in the pattern table:

```
setInteriorStyle (pattern);  
setInteriorStyleIndex (2);  
fillArea (n, points);
```

Separate tables are set up for hatch patterns. If we had selected *hatch* fill for the interior style in this program segment, then the value assigned to parameter *pi* is an index to the stored patterns in the hatch table.

For fill style *pattern*, table entries can be created on individual output devices with

```
setPatternRepresentation (ws, p., nx, ny, cp)
```

**TABLE 4-3**  
A WORKSTATION  
PATTERN TABLE WITH  
TWO ENTRIES, USING  
THE COLOR CODES OF  
TABLE 4-1

Index ( <i>pi</i> )	Pattern ( <i>cp</i> )
1	$\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$
2	$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$

## CHARACTER ATTRIBUTES

The appearance of displayed characters is controlled by attributes such as font, size, color, and orientation. Attributes can be set for entire character strings (text) and for individual characters defined as marker symbols.

## TEXT ATTRIBUTES

There are a great many text options that can be made available to graphics programmers.

First of all, there is the choice of font (or typeface), which is a set of characters with a particular design style such as New York, Courier, Helvetica, London, "Times Roman, and various special symbol groups.

The width only of text can be set with the function

```
setCharacterExpansionFactor (cw)
```

where the character-width parameter *cw* is set to a positive real value that scales the body width of characters. Text height is unaffected by this attribute setting. Examples of text displayed with different character expansions is given in Fig. 4-27.

Spacing between characters is controlled separately with

```
setCharacterSpacing (cs)
```

where the character-spacing parameter *cs* can be assigned any real value. The value assigned to *cs* determines the spacing between character bodies along print lines. Negative values for *cs* overlap character bodies; positive values insert space to spread out the displayed characters. Assigning the value 0 to *cs* causes text to be displayed with no space between character bodies. The amount of spacing to be applied is determined by multiplying the value of *cs* by the character height (distance between baseline and capline). In Fig. 4-28, a character string is displayed with three different settings for the character-spacing parameter.

width 0.5  
width 1.0  
width 2.0

Figure 4-27  
The effect of different character-width settings on displayed text.

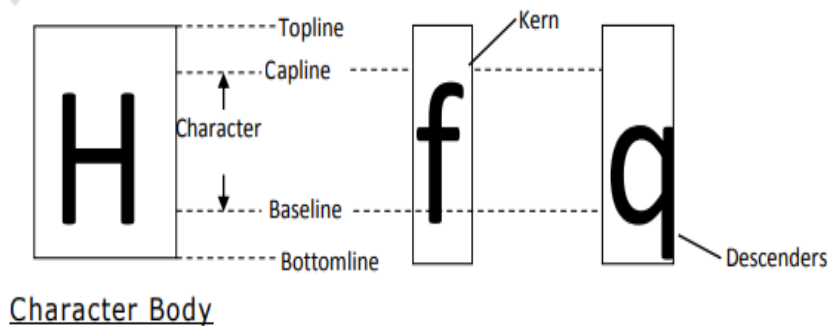
## Character Attributes:

The appearance of displayed characters is controlled by attributes such as font, size, colour and orientation. Attributes can be set both for entire character strings and for individual characters, known as Marker symbols.

1. Text Attributes: There are many text options available, such as font, colour, size, spacing, and orientation.
2. Text Style: The characters in a selected font can also be displayed in various underlining styles (solid, dotted, dashed, double), in bold, in italics, shadow style, etc. Font options can be made available as pre defined sets of grid patterns or as character sets designed with lines and curves.
3. Text Colour: Colour settings for displayed text are stored in the system attribute list and transferred to the frame buffer by character loading functions. When a character string is displayed, the current colour is used to set pixel values in the frame buffer corresponding to the character shapes and position.

4. Text Size: We can adjust text size by changing the overall dimensions, i.e., width and height, of characters or by changing only the width.
5. Character size is specified in Points, where 1 point is 0.013837 inch or approximately 1/72 inch. Point measurements specify the size of the Character Body. Different fonts with the same point specifications can have different character sizes depending upon the design of the font. Character Body Topline Capline Bottomline Baseline Character H Height f q Descenders

dimensions, i.e., width and height, of characters or by changing only the width. Character size is specified in **Points**, where 1 point is 0.013837 inch or approximately 1/72 inch. Point measurements specify the size of the **Character Body**. Different fonts with the same point specifications can have different character sizes depending upon the design of the font.



The distance between Topline and Bottomline is same for all characters in a particular size and font, but the width may be different. A smaller body width is assigned to narrow characters such as i, j, l, etc.

compared to broad characters such as W or M. The Character Height is the distance between the Baseline and Capline of characters. Kerned characters, such as f and j, extend beyond the character-width limits. And letters with descenders, such as g, j, p, q, extend below the baseline.

The size can be changed in such a way so that the width and spacing of characters is adjusted to maintain the same text proportions.

For example, doubling the height also doubles the character width and the spacing between characters. Also, only the width of the character s can be changes without affecting its height. Similarly, spacing between characters can be increased without changing height or width of individual characters.

The effect of different character-height setting, character-width setting, and character spacing on a text is shown below.

**HEIGHT 1**

**WIDTH1**

S P A C I N G 1

**HEIGHT2**

**WIDTH2**

S P A C I N G 2

**HEIGHT3**

**WIDTH3**

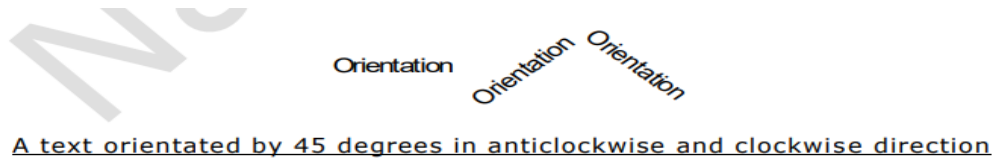
S P A C I N G 3

Effect of changing Height, Width and Spacing

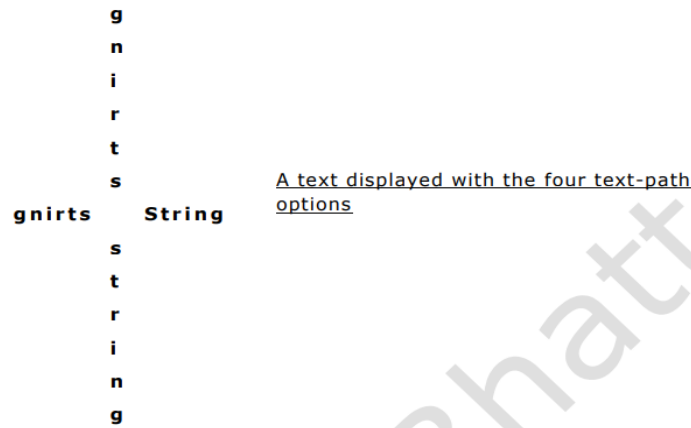
↑ **Text Orientation:** The text can be displayed at various angles, known as orientation. A procedure for orienting text rotates characters so that the sides of character bodies, from baseline to topline at aligned at some angle. Character strings can be arranged vertically or horizontally. A text orientated by 45 degrees in anticlockwise and clockwise direction

↑ **Text Path:** In some applications the character strings are arranged vertically or horizontally. This is known as Text Path. Text path can be right, left, up or down.

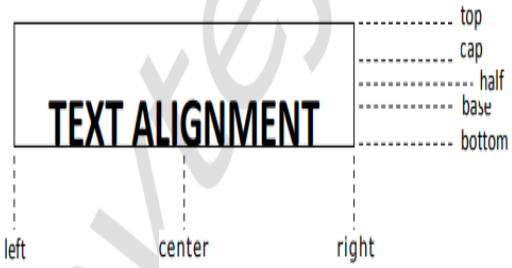




**Text Path:** In some applications the character strings are arranged vertically or horizontally. This is known as Text Path. Text path can be right, left, up or down.



**Text Alignment:** Another attribute for character strings is alignment. This attribute specifies how text is to be positioned with respect to the start coordinates. Vertical alignment can be top, cap, half, base and bottom. Similarly, horizontal alignment can be left, center and right.



Alignment values for a string

## UNIT III

### 2D Geometric Transformations:

#### Introduction of Transformations

Computer Graphics provide the facility of viewing object from different angles. The architect can study building from different angles i.e.

#### Front Evaluation

#### Side elevation

#### Top plan

A Cartographer can change the size of charts and topographical maps. So if graphics images are coded as numbers, the numbers can be stored in memory. These numbers are modified by mathematical operations called as Transformation.

The purpose of using computers for drawing is to provide facility to user to view the object from different angles, enlarging or reducing the scale or shape of object called as Transformation.

Two essential aspects of transformation are given below:

Each transformation is a single entity. It can be denoted by a unique name or symbol.

It is possible to combine two transformations, after connecting a single transformation is obtained, e.g., A is a transformation for translation. The B transformation performs scaling. The combination of two is  $C=AB$ . So C is obtained by concatenation property.

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

**Geometric Transformation:** The object itself is transformed relative to the coordinate system or background. The mathematical statement of this viewpoint is defined by geometric transformations applied to each point of the object.

**Coordinate Transformation:** The object is held stationary while the coordinate system is transformed relative to the object. This effect is attained through the application of coordinate transformations.

An example that helps to distinguish these two viewpoints:

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

Moving the automobile while keeping the background fixed-(Geometric Transformation)

We can keep the car fixed while moving the background scenery- (Coordinate Transformation)

## **Types of Transformations:**

- **Translation**
- **Scaling**
- **Rotating**
- **Reflection**
- **Shearing**

### **Translation**

It is the straight-line movement of an object from one position to another is called Translation. Here the object is positioned from one coordinate location to another.

Translation of point:

To translate a point from coordinate position (x, y) to another (x1 y1), we add algebraically the translation distances Tx and Ty to original coordinate.

$$x1=x+Tx$$

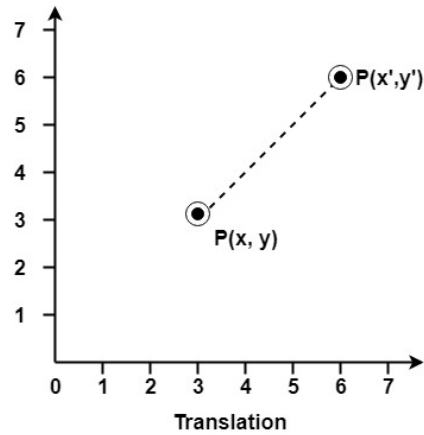
$$y1=y+Ty$$

The translation pair (Tx,Ty) is called as shift vector.

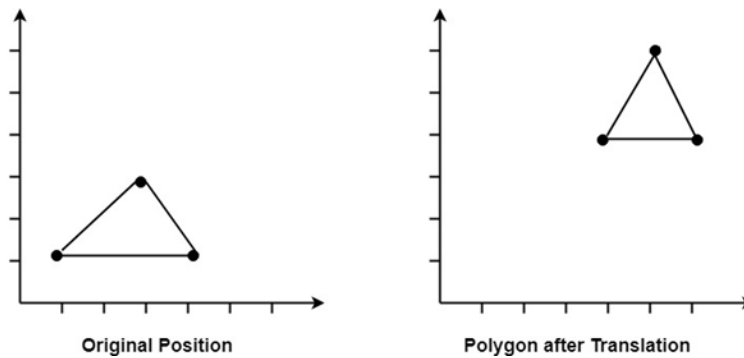
Translation is a movement of objects without deformation. Every position or point is translated by the same amount. When the straight line is translated, then it will be drawn using endpoints.

For translating polygon, each vertex of the polygon is converted to a new position. Similarly, curved objects are translated. To change the position of the circle or ellipse its center coordinates are transformed, then the object is drawn using new coordinates.

Let P is a point with coordinates (x, y). It will be translated as (x1 y1).



Translation of Polygon



*Matrix for Translation:*

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix} \text{ Or } \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

## Scaling:

It is used to alter or change the size of objects. The change is done using scaling factors. There are two scaling factors, i.e.  $S_x$  in x direction  $S_y$  in y-direction. If the original position is  $x$  and  $y$ . Scaling factors are  $S_x$  and  $S_y$  then the value of coordinates after scaling will be  $x^1$  and  $y_1$ .

If the picture to be enlarged to twice its original size then  $S_x = S_y = 2$ . If  $S_x$  and  $S_y$  are not equal then scaling will occur but it will elongate or distort the picture.

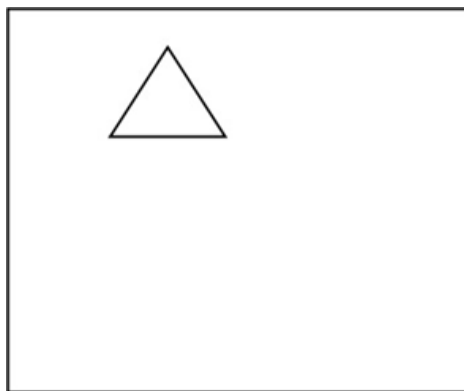
If scaling factors are less than one, then the size of the object will be reduced. If scaling factors are higher than one, then the size of the object will be enlarged.

If  $S_x$  and  $S_y$  are equal it is also called as Uniform Scaling. If not equal then called as Differential Scaling. If scaling factors with values less than one will move the object closer to coordinate origin, while a value higher than one will move coordinate position farther from origin.

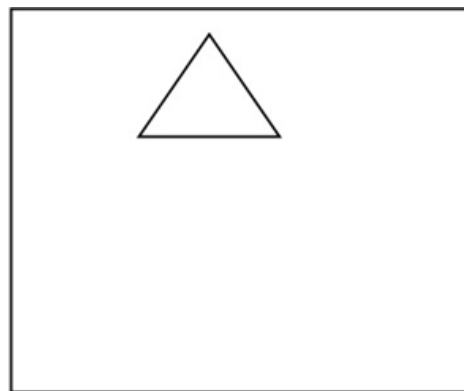
**Enlargement:** If  $T_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ , If  $(x_1 \ y_1)$  is original position and  $T_1$  is translation vector then  $(x_2 \ y_2)$  are coordinated after scaling

$$[x_2 \ y_2] = [x_1 \ y_1] \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = [2x_1 \ 2y_1]$$

**The image will be enlarged two times**



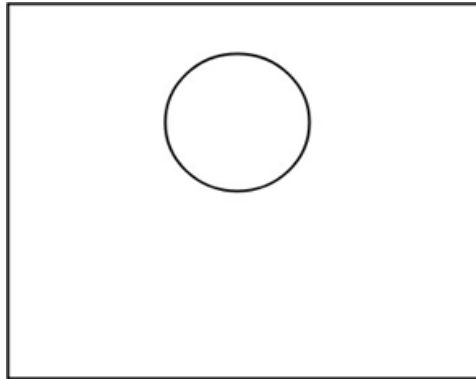
Original Image



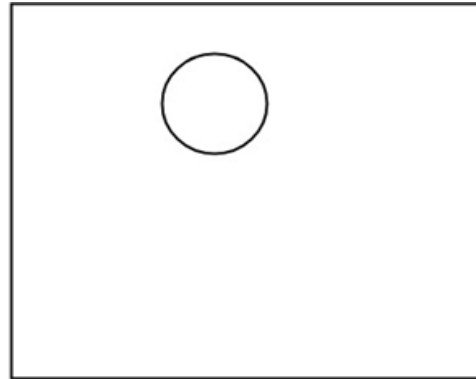
Enlarged Image

**Reduction:** If  $T_1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ . If  $(x_1 \ y_1)$  is original position and  $T_1$  is translation vector, then  $(x_2 \ y_2)$  are coordinates after scaling

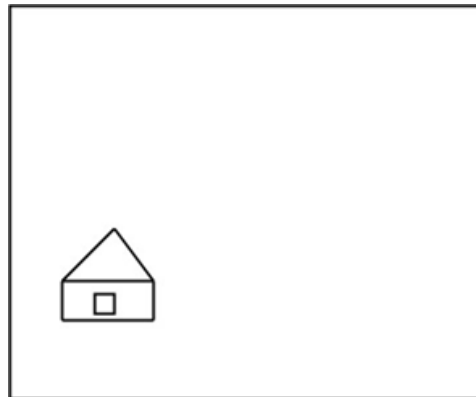
$$[x_2 \ y_2] = [x_1 \ y_1] \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} = [0.5x_1 \ 0.5y_1]$$



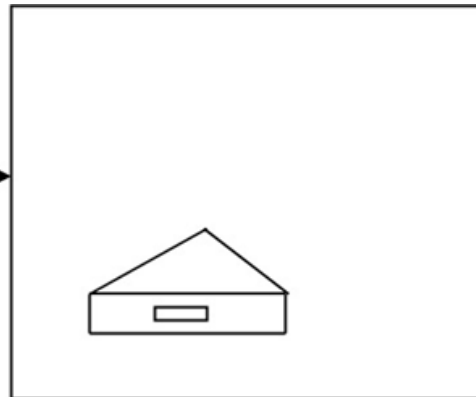
Original Image



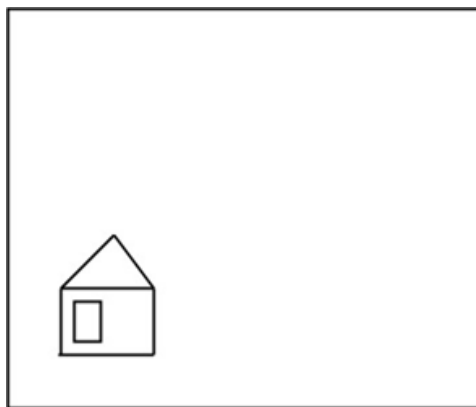
Reduction Image



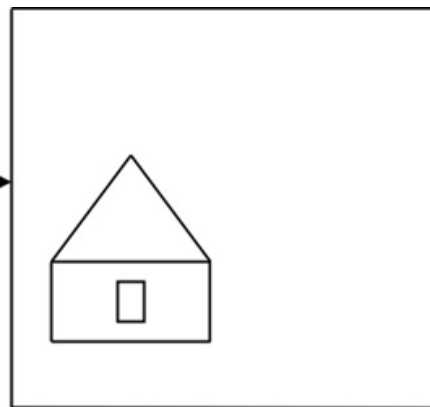
Original Object



Object after scaling in X direction



Original Object



Object after scaling in Y direction

Matrix for Scaling:

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**Example: Prove that 2D Scaling transformations are commutative i.e,  $S_1 S_2 = S_2 S_1$ .**

**Solution:  $S_1$  and  $S_2$  are scaling matrices**

$$S_1 = \begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_1 * S_2 = \begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_{x_1}S_{x_2} & 0 & 0 \\ 0 & S_{y_1}S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \dots \dots \dots \text{equation 1}$$

$$S_2 * S_1 = \begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_{x_2}S_{x_1} & 0 & 0 \\ 0 & S_{y_2}S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \dots \dots \dots \text{equation 2}$$

From equation 1 and 2

$$S_1 S_2 = S_2 S_1. \text{ Hence Proved}$$

**Rotation:**

It is a process of changing the angle of the object. Rotation can be clockwise or anticlockwise. For rotation, we have to specify the angle of rotation and rotation point. Rotation point is also called a pivot point. It is print about which object is rotated.

**Types of Rotation:**

1. Anticlockwise



## 2. Counterclockwise

The positive value of the pivot point (rotation angle) rotates an object in a counter-clockwise (anti-clockwise) direction.

The negative value of the pivot point (rotation angle) rotates an object in a clockwise direction.

When the object is rotated, then every point of the object is rotated by the same angle.

**Straight Line:** Straight Line is rotated by the endpoints with the same angle and redrawing the line between new endpoints.

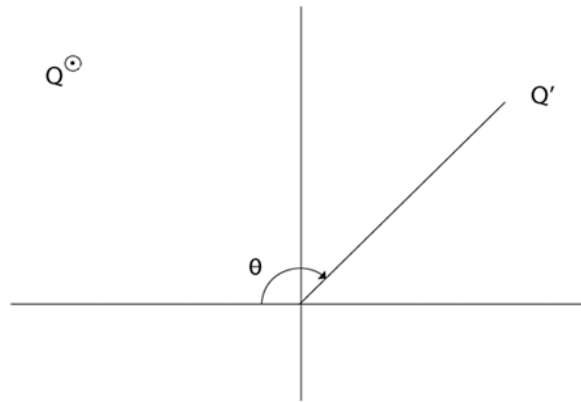
**Polygon:** Polygon is rotated by shifting every vertex using the same rotational angle.

**Curved Lines:** Curved Lines are rotated by repositioning of all points and drawing of the curve at new positions.

**Circle:** It can be obtained by center position by the specified angle.

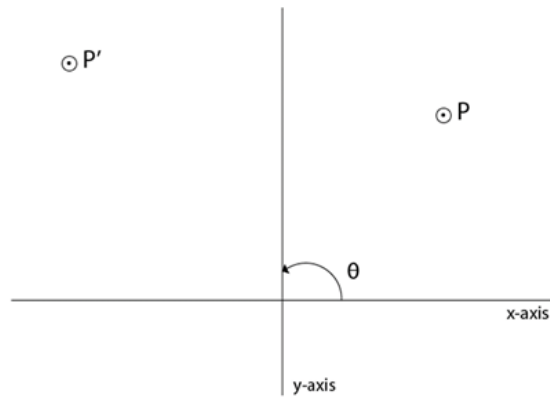
**Ellipse:** Its rotation can be obtained by rotating major and minor axis of an ellipse by the desired angle.

Rotation in anticlockwise direction



**Q** is original position  
**Q'** is final rotated position

Rotation of P in clockwise direction



**P** is original Position  
**P'** is final position or position after rotation  
where  $\theta$  is angle of rotation

Matrix for rotation is a clockwise direction.

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

Matrix for rotation is an anticlockwise direction.

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

Matrix for homogeneous co-ordinate rotation (clockwise)

$$R = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Matrix for homogeneous co-ordinate rotation (anticlockwise)

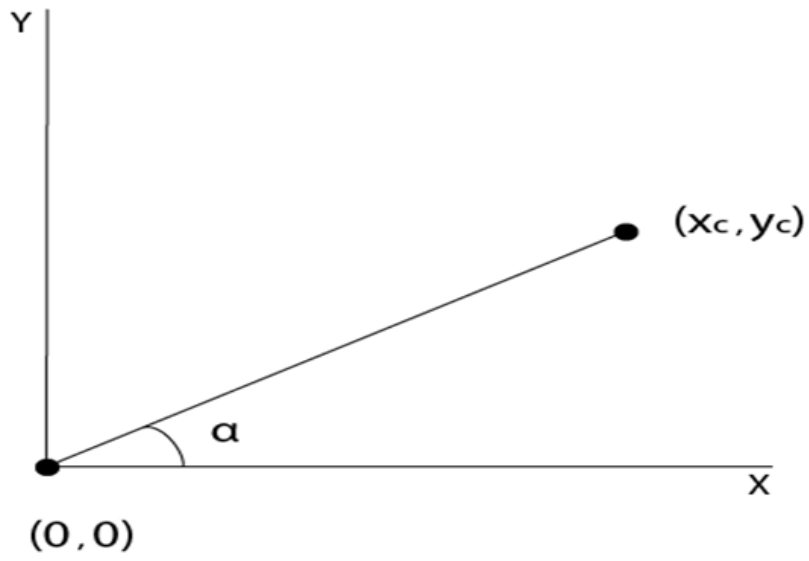
$$R = \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

**Rotation about an arbitrary point:** If we want to rotate an object or point about an arbitrary point, first of all, we translate the point about which we want to rotate to the origin. Then rotate point or object about the origin, and at the end, we again translate it to the original place. We get rotation about an arbitrary point.

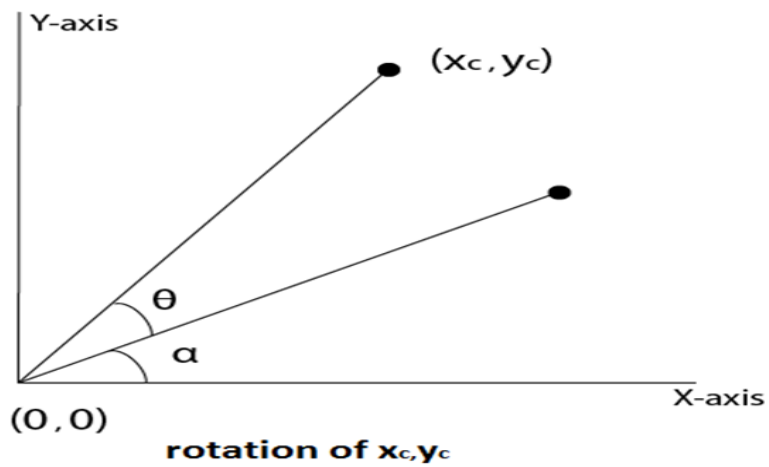
**Example:** The point  $(x, y)$  is to be rotated

The  $(x_c, y_c)$  is a point about which counterclockwise rotation is done

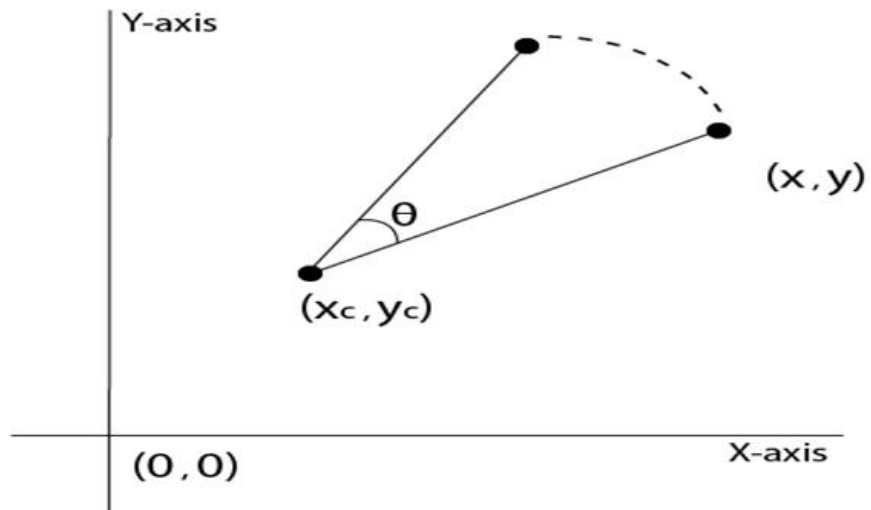
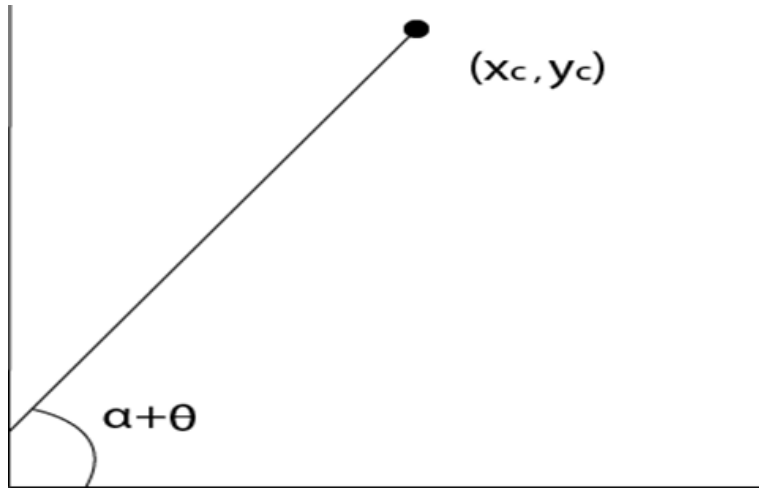
**Step1:** Translate point  $(x_c, y_c)$  to origin



**Step2:** Rotation of  $(x, y)$  about the origin



**Step3:** Translation of center of rotation back to its original position



**Example1:** Prove that 2D rotations about the origin are commutative i.e.  $R_1 R_2 = R_2 R_1$ .

**Solution:**  $R_1$  and  $R_2$  are rotation matrices

$$R_1 = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_1 * R_2 = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_1 \cos\theta_2 & -\sin\theta_1 \sin\theta_2 & \cos\theta_1 \sin\theta_2 + \sin\theta_1 \cos\theta_2 & 0 \\ -\sin\theta_1 \cos\theta_2 & -\cos\theta_1 \sin\theta_2 & -\sin\theta_1 \sin\theta_2 + \cos\theta_1 \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots \text{eq 1}$$

$$R_2 * R_1 = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_2 \cos\theta_1 & -\sin\theta_2 \sin\theta_1 & \cos\theta_2 \sin\theta_1 + \sin\theta_2 \cos\theta_1 & 0 \\ -\sin\theta_2 \cos\theta_1 & -\cos\theta_2 \sin\theta_1 & -\sin\theta_2 \sin\theta_2 + \cos\theta_2 \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \dots \text{eq 2}$$

From eq 1 & eq 2

$R_1 R_2 = R_2 R_1$ . Hence Proved.

**Example2:** Rotate a line CD whose endpoints are (3, 4) and (12, 15) about origin through a  $45^\circ$  anticlockwise direction.

**Solution:** The point C (3, 4)

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

$$\theta = 45^\circ$$

Let

$$R = \begin{bmatrix} \cos 45^\circ & \sin 45^\circ \\ -\sin 45^\circ & \cos 45^\circ \end{bmatrix}$$

$$R = \begin{bmatrix} 0.707 & 0.707 \\ -0.707 & -0.707 \end{bmatrix}$$

The point A (3, 4) after rotation will be

$$\begin{aligned} [x, y] &= [3, 4] \begin{bmatrix} 0.707 & 0.707 \\ -0.707 & -0.707 \end{bmatrix} \\ &= [3 * 0.707 - 4 * 0.707 \quad 3 * 0.707 + 4 * 0.707] \\ &= [2.121 - 2.828 \quad 2.121 + 2.828] \\ &= [-.707 \quad 4.949] \end{aligned}$$

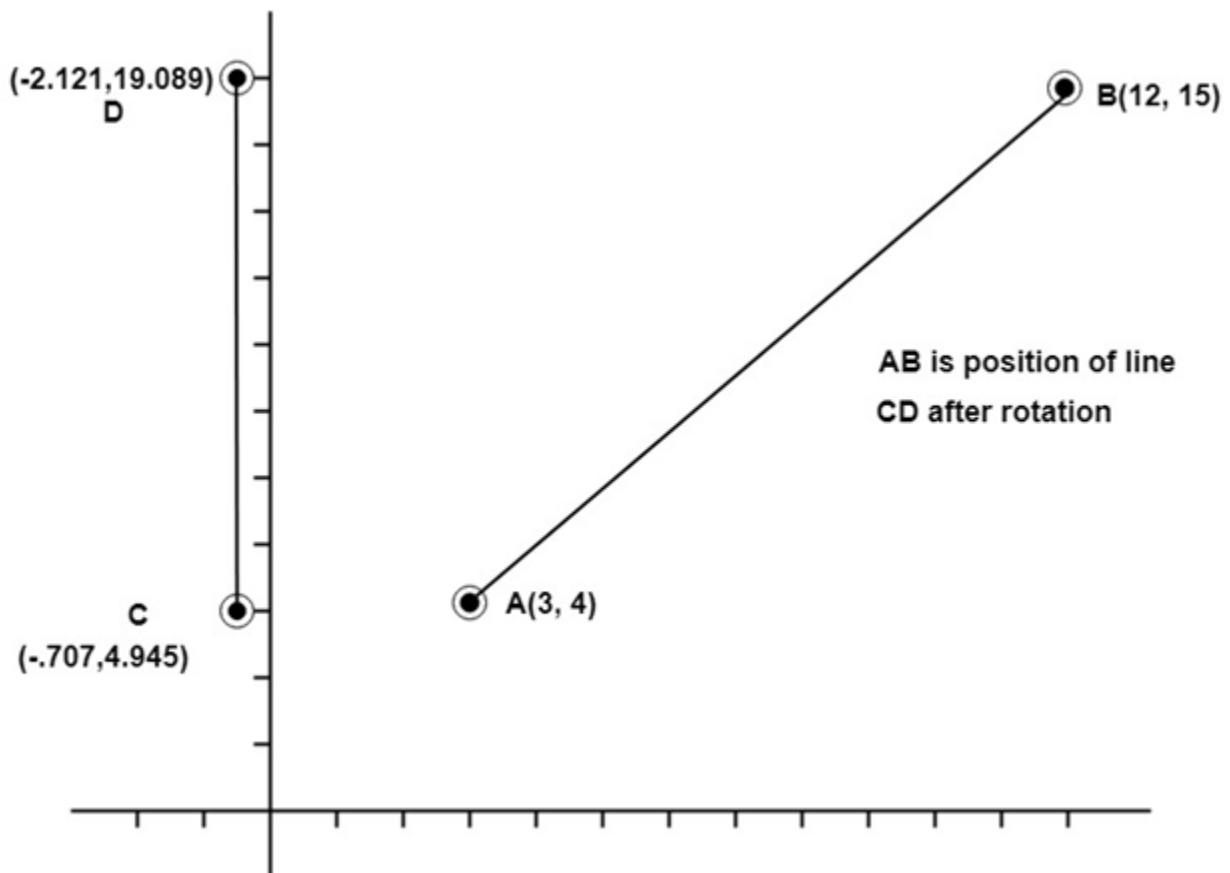
The rotation of point B (12, 15)

$$\begin{aligned} R_1 &= \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} 0.707 & 0.707 \\ -0.707 & -0.707 \end{bmatrix} \end{aligned}$$

The point B (12, 15) will be

$$\begin{aligned} [x, y] &= [12, 15] \begin{bmatrix} 0.707 & 0.707 \\ -0.707 & -0.707 \end{bmatrix} \\ &= [12 * 0.707 + 15 * 0.707 \quad 12 * 0.707 + 15 * 0.707] \\ &= [(8.484 - 10.605) \quad (8.484 + 10.605)] \\ &= [-2.121 \quad 19.089] \end{aligned}$$

So line AB after rotation at 45° become  $[-.707, 4.945]$  and  $[-2.121, 19.089]$



**Example3:** Rotate line AB whose endpoints are A (2, 5) and B (6, 12) about origin through a 30° clockwise direction.



**Solution:** For rotation in the clockwise direction. The matrix is

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

**Step1:** Rotation of point A (2, 5). Take angle  $30^\circ$

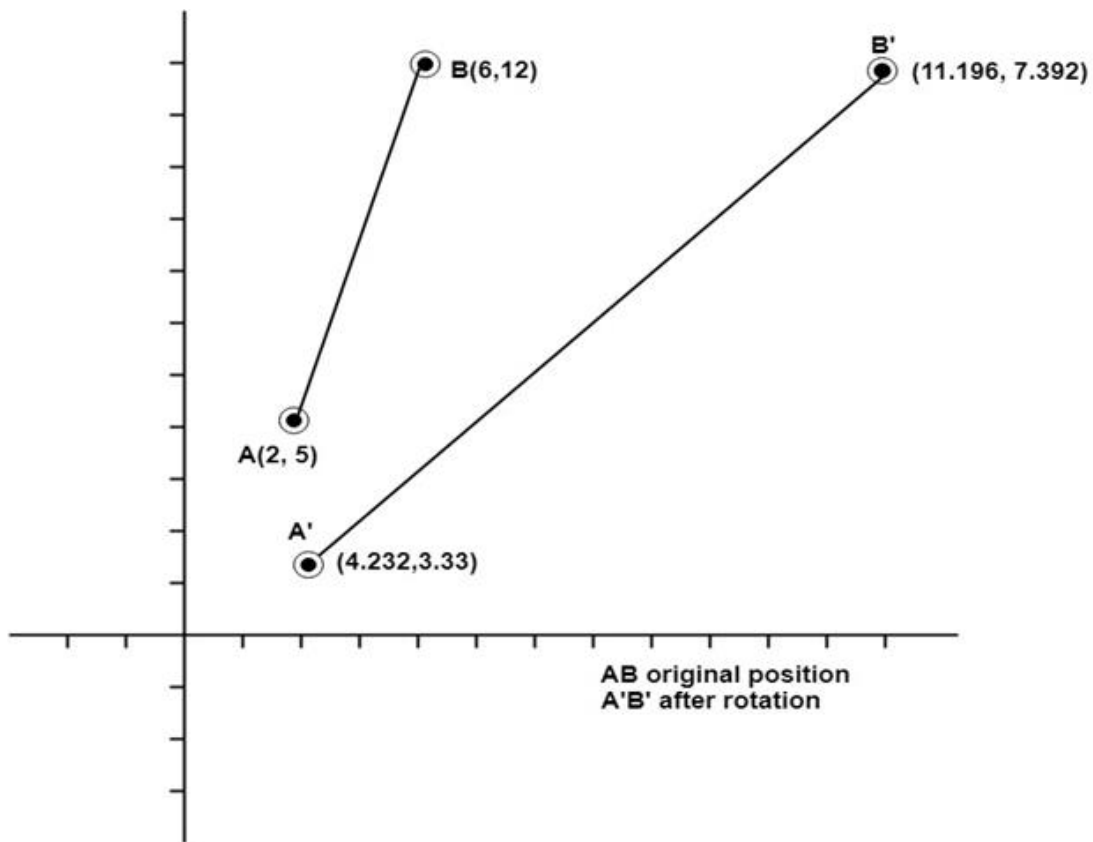
$$\begin{aligned} R &= \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ \\ \sin 30^\circ & \cos 30^\circ \end{bmatrix} \\ &= [2, 5] \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ \\ -\sin 30^\circ & \cos 30^\circ \end{bmatrix} \\ &= [2, 5] \begin{bmatrix} .866 & -0.5 \\ .5 & .866 \end{bmatrix} \\ &= [2 * .866 + 5 * .5 & 2 * (-.5) + 5 (.866)] \\ &= [(1.732 + 2.5 & -1 + 4.33)] \\ &= [4.232 & 3.33] \end{aligned}$$

A point (2, 5) become (4.232, 3.33)

**Step2:** Rotation of point B (6, 12)

$$\begin{aligned} &= [6 \ 12] \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ \\ \sin 30^\circ & \cos 30^\circ \end{bmatrix} \\ &= [6 \ 12] \begin{bmatrix} .866 & -0.5 \\ .5 & .866 \end{bmatrix} \\ &= [6 * .866 + 12 * .5 & 6 * (-.5) + 12 * (.866)] \\ &= [5.196 + 6 & -3 + 10.392] \\ &= [11.196 & 7.392] \end{aligned}$$

B point (6, 12) becomes (11.46, 7.312) after rotation  $30^\circ$  in clockwise direction.



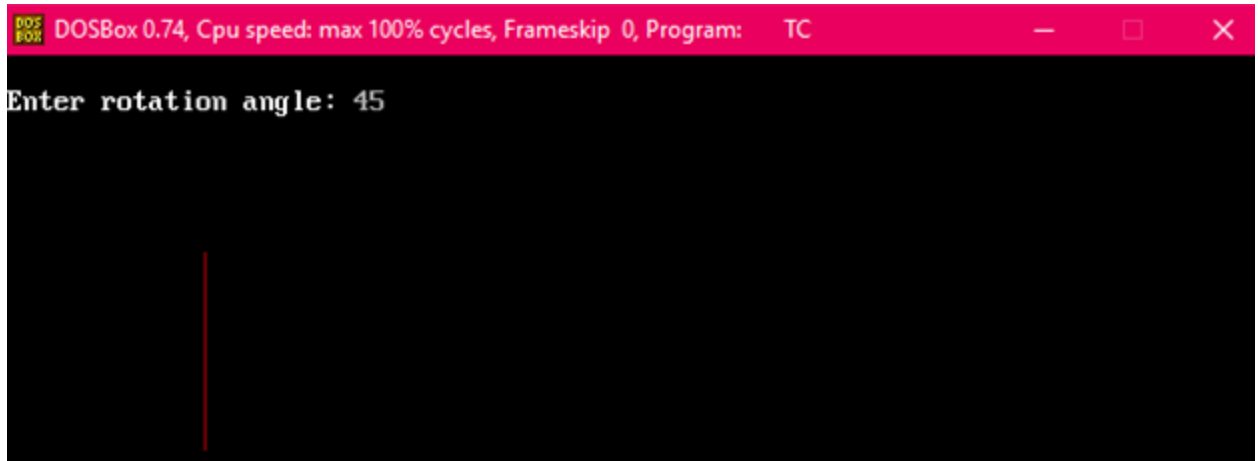
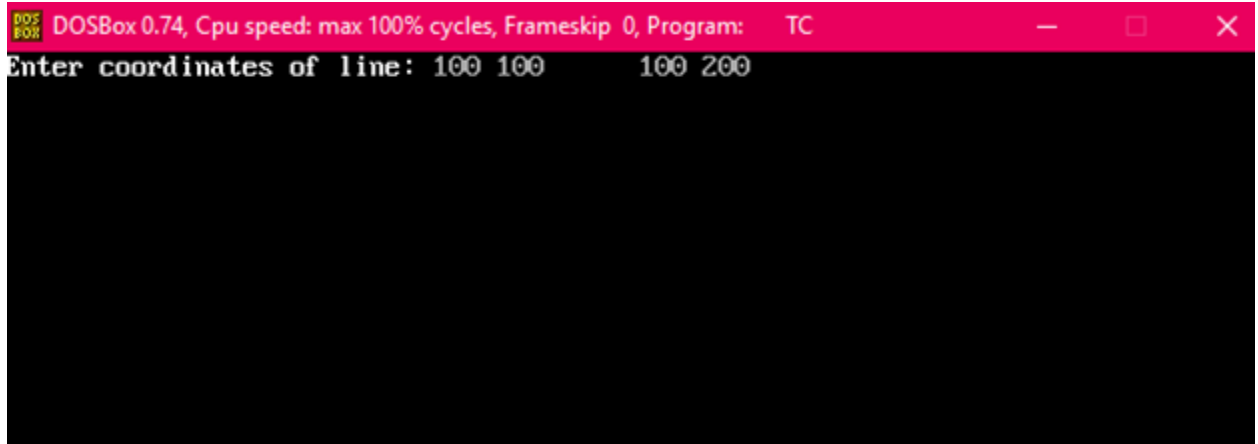
Program to rotate a line:

1. `#include<stdio.h>`
2. `#include<graphics.h>`
3. `#include<math.h>`
4. **int** main()
5. {
6.   `intgd=0,gm,x1,y1,x2,y2;`
7.   **double** s,c, angle;
8.   `initgraph(&gd, &gm, "C:\\TC\\BGI");`
9.   `setcolor(RED);`
10.   `printf("Enter coordinates of line: ");`
11.   `scanf("%d%d%d%d",&x1,&y1,&x2,&y2);`
12.   `cleardevice();`
13.   `setbkcolor(WHITE);`

```
14. line(x1,y1,x2,y2);
15. getch();
16. setbkcolor(BLACK);
17. printf("Enter rotation angle: ");
18. scanf("%lf", &angle);
19. setbkcolor(WHITE);
20. c = cos(angle *3.14/180);
21. s = sin(angle *3.14/180);
22. x1 = floor(x1 * c + y1 * s);
23. y1 = floor(-x1 * s + y1 * c);
24. x2 = floor(x2 * c + y2 * s);
25. y2 = floor(-x2 * s + y2 * c);
26. cleardevice();
27. line(x1, y1 ,x2, y2);
28. getch();
29. closegraph();
30. return 0;
31. }
```

### **Output:**

#### **Before rotation**



**After rotation**



Program to rotate a Triangle:

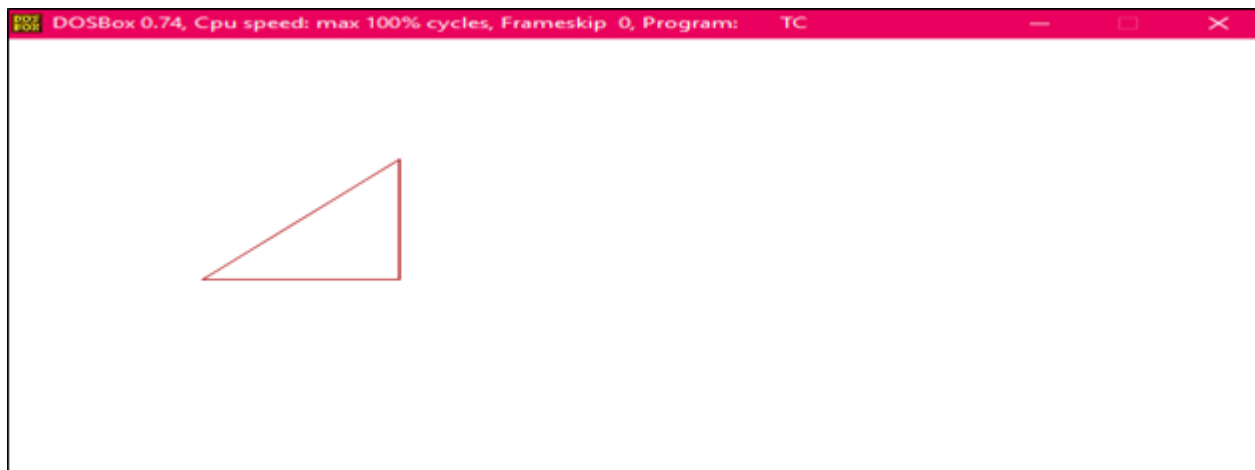
1. `#include<stdio.h>`
2. `#include<graphics.h>`
3. `#include<math.h>`
4. `main()`
5. `{`
6. `intgd=0,gm,x1,y1,x2,y2,x3,y3;`
7. `double s,c, angle;`
8. `initgraph(&gd, &gm, "C:\\\\TURBOC3\\\\BGI");`
9. `setcolor(RED);`
10. `printf("Enter coordinates of triangle: ");`
11. `scanf("%d%d%d%d%d%d",&x1,&y1,&x2,&y2, &x3, &y3);`
12. `setbkcolor(WHITE);`
13. `cleardevice();`
14. `line(x1,y1,x2,y2);`
15. `line(x2,y2, x3,y3);`
16. `line(x3, y3, x1, y1);`
17. `getch();`
18. `setbkcolor(BLACK);`
19. `printf("Enter rotation angle: ");`

```
20. scanf("%lf", &angle);
21. setbkcolor(WHITE);
22. c = cos(angle *M_PI/180);
23. s = sin(angle *M_PI/180);
24. x1 = floor(x1 * c + y1 * s);
25. y1 = floor(-x1 * s + y1 * c);
26. x2 = floor(x2 * c + y2 * s);
27. y2 = floor(-x2 * s + y2 * c);
28. x3 = floor(x3 * c + y3 * s);
29. y3 = floor(-x3 * s + y3 * c);
30. cleardevice();
31. line(x1, y1 ,x2, y2);
32. line(x2,y2, x3,y3);
33. line(x3, y3, x1, y1);
34. getch();
35. closegraph();
36. return 0;
37. }
```

**Output:**

**Before rotation**

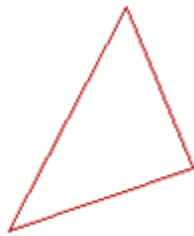
```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter coordinates of triangle: 200 200 200 100 100 200
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter rotation angle: 20
```

A window titled "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC" showing a black background with a red-outlined right-angled triangle. The triangle is rotated 20 degrees counter-clockwise from its original orientation. The right angle is now at the top-right corner, and the hypotenuse is the leftmost side.

**After rotation**



**Reflection:**

It is a transformation which produces a mirror image of an object. The mirror image can be either about x-axis or y-axis. The object is rotated by  $180^\circ$ .

**Types of Reflection:**

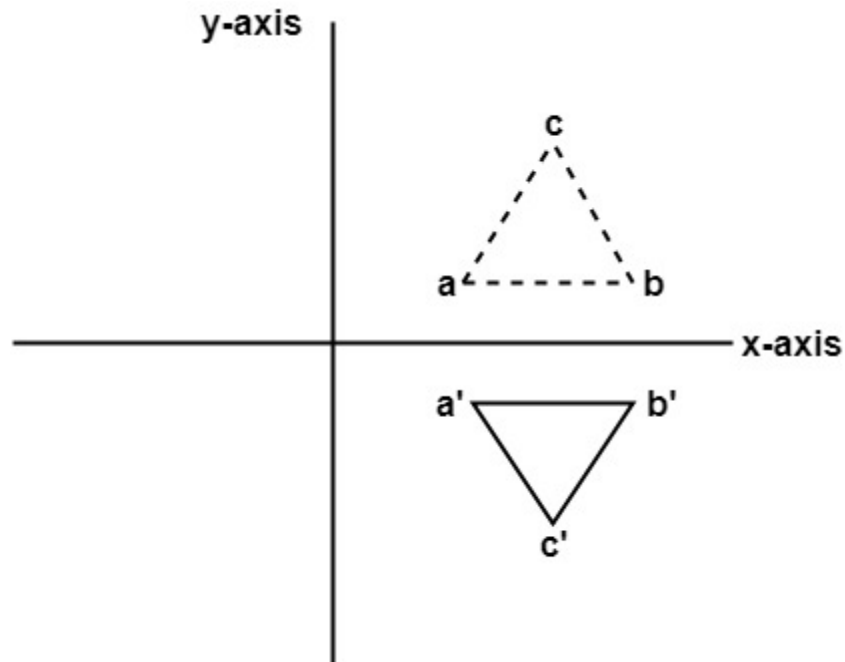
1. Reflection about the x-axis
2. Reflection about the y-axis
3. Reflection about an axis perpendicular to xy plane and passing through the origin
4. Reflection about line  $y=x$

**1. Reflection about x-axis:** The object can be reflected about x-axis with the help of the following matrix

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



In this transformation value of x will remain same whereas the value of y will become negative. Following figures shows the reflection of the object axis. The object will lie another side of the x-axis.

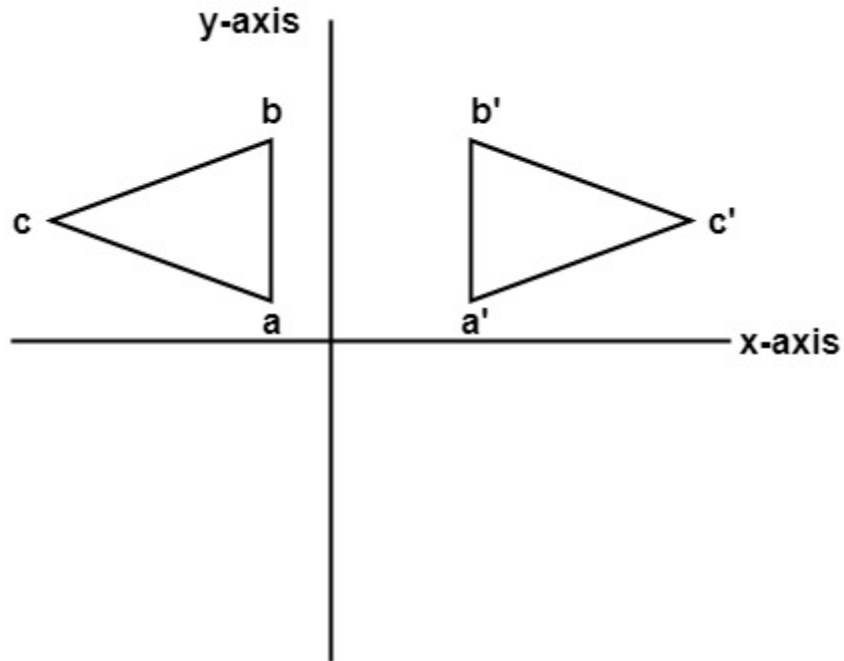


**2. Reflection about y-axis:** The object can be reflected about y-axis with the help of following transformation matrix

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

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.

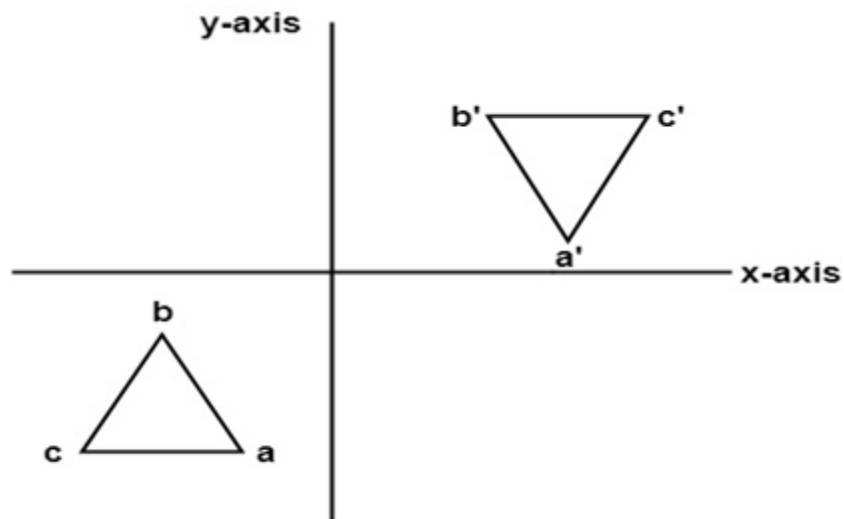
The following figure shows the reflection about the y-axis



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

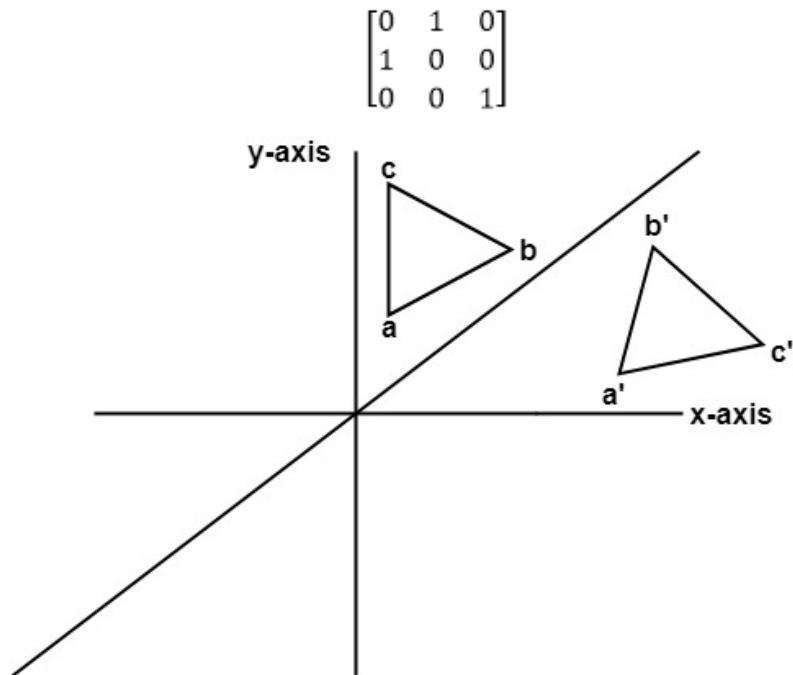
In the matrix of this transformation is given below

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



In this value of x and y both will be reversed. This is also called as half revolution about the origin.

**4. Reflection about line  $y=x$ :** The object may be reflected about line  $y = x$  with the help of following transformation matrix



First of all, the object is rotated at  $45^\circ$ . 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  $45^\circ$ .

**Example:** A triangle ABC is given. The coordinates of A, B, C are given as

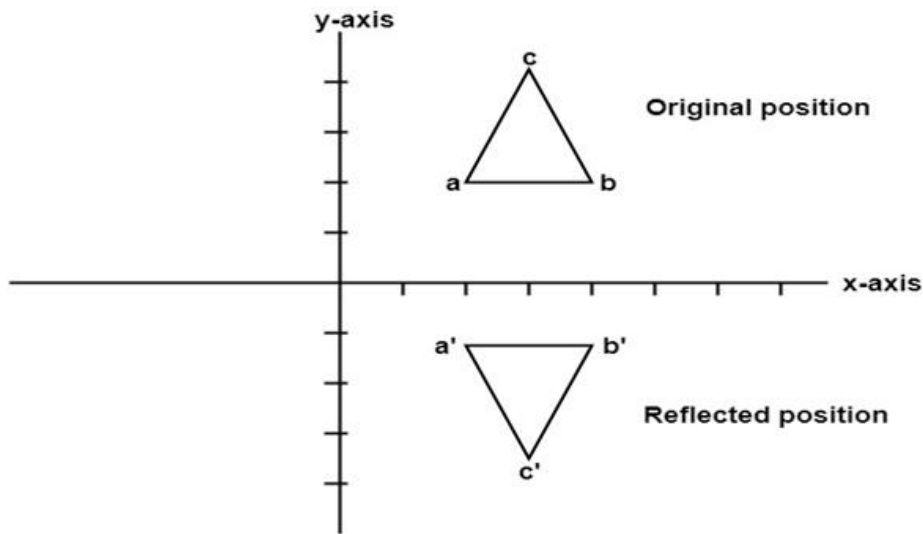
A (3 4)

B (6 4)

C (4 8)

Find reflected position of triangle i.e., to the x-axis.

**Solution:**



The matrix for reflection about x axis  $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

The a point coordinates after reflection

$$(x, y) = (3, 4) \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$(x, y) = [3, -4]$$

The b point coordinates after reflection

$$(x, y) = (6, 4) \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$(x, y) = [6, -4]$$

The coordinate of point c after reflection

$$(x, y) = (4, 8) \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$(x, y) = [4, -8]$$

- a (3, 4) becomes  $a^1$  (3, -4)
- b (6, 4) becomes  $b^1$  (6, -4)
- c (4, 8) becomes  $c^1$  (4, -8)

Program to perform Mirror Reflection about a line:

1. `#include <iostream.h>`
2. `#include <conio.h>`
3. `#include <graphics.h>`
4. `#include <math.h>`
5. `#include <stdlib.h>`
6. `#define pi 3.14`
7. `class arc`
8. `{`
9. `float x[10],y[10],theta,ref[10][10],ang;`
10. `float p[10][10],p1[10][10],x1[10],y1[10],xm,ym;`
11. `int i,k,j,n;`
12. `public:`
13. `void get();`
14. `void cal ();`
15. `void map ();`
16. `void graph ();`
17. `void plot ();`
18. `void plot1();`
19. `};`
20. `void arc::get ()`
21. `{`

```

22. cout<<"\n ENTER ANGLE OF LINE INCLINATION AND Y INTERCEPT";
23. cin>> ang >> b;
24. cout <<"\n ENTER NO OF VERTICES";
25. cin >> n;
26. cout <<"\n ENTER";
27. for (i=0; i<n; i++)
28. {
29.     cout<<"\n x["<<i<<"] and y["<<i<<"];
30. }
31. theta =(ang * pi)/ 180;
32. ref [0] [0] = cos (2 * theta);
33. ref [0] [1] = sin (2 * theta);
34. ref [0] [2] = -b *sin (2 * theta);
35. ref [1] [0] = sin (2 * theta);
36. ref [1] [1] = -cos (2 * theta);
37. ref [1] [2] = b * (cos (2 * theta)+1);
38. ref [2] [0]=0;
39. ref [2] [1]=0;
40. ref [2] [2] = 1;
41. }
42. void arc :: cal ()
43. {
44.     for (i=0; i < n; i++)
45.     {
46.         p[0] [i] = x [i];
47.         p [1] [i] = y [i];
48.         p [2] [i] = 1;
49.     }
50.     for (i=0; i<3;i++)
51.     {
52.         for (j=0; j<n; j++)

```

```

53.  {
54.    p1 [i] [j]=0;
55.    for (k=0;k<3; k++)
56.  }
57.  p1 [i] [j] += ref [i] [k] * p [k] [j];
58.  }
59. for (i=0; i<n; i++)
60. {
61.  x1 [i]=p1[0] [i];
62.  y1 [i] = p1 [1] [i];
63. }
64. }
65. void arc :: map ()
66. {
67.  int gd = DETECT, gm;
68.  initgraph (&gd, &gm, " ");
69.    int errorcode = graphresult ();
70.  /* an error occurred */
71.  if (errorcode != grOK)
72.  {
73.    printf ("Graphics error: %s \n", grapherrormsg (errorcode));
74.    printf ("Press any key to halt:");
75.    getch ();
76.    exit (1); /* terminate with an error code */
77.  }
78. }
79. void arc :: graph ()
80. {
81.  xm=getmaxx ()/2;
82.  ym=getmaxy ()/2;
83.  line (xm, 0, xmm 2*ym);

```

```

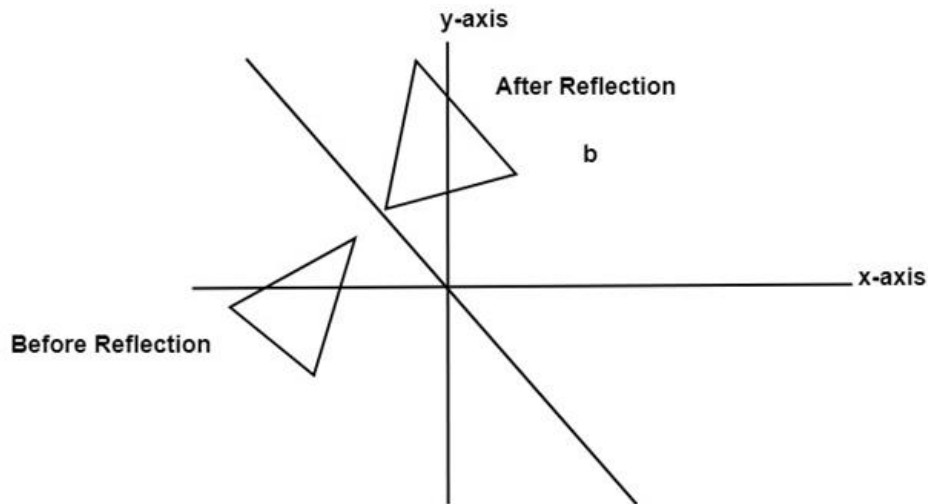
84. }
85. void arc :: plot 1 ()
86. {
87.     for (i=0; i <n-1; i++)
88.     {
89.         circle (x1[i]+xm, (-y1[i]+ym), 2);
90.         line (x1[i]+xm, (-y1[i]+ym), x1[i+1]+xm, (-y1[i+1]+ym));
91.     }
92.     line (x1[n-1]+xm, (-y1[n-1]+ym), x1[0]+xm, (-y1[0]+ym));
93.     getch();
94. }
95. void arc :: plot ()
96. {
97.     for (i=0; i <n-1; i++)
98.     {
99.         circle (x1[i]+xm, (-y1[i]+ym), 2);
100.         line (x1[i]+xm, (-y1[i]+ym), x[i+1]+xm, (-y1[i+1]+ym));
101.     }
102.     line (x[n-1]+xm, (-y1[n-1]+ym), x[0]+xm, (-y[0]+ym));
103.     getch();
104. }
105. void main ()
106. {
107.     class arc a;
108.     clrscr();
109.     a.map();
110.     a.graph();
111.     a.get();
112.     a.cal();
113.     a.plot();
114.     a.plot1();

```



```
115.     getch();
116.     }
```

**Output:**



**Shearing:**

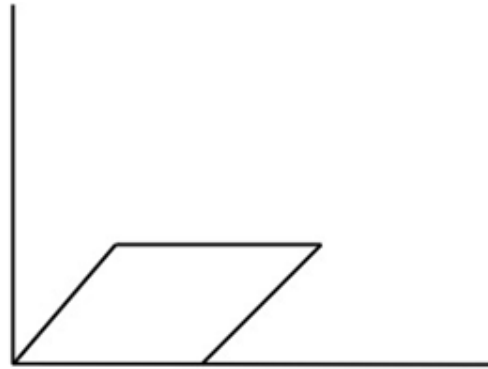
It is transformation which changes the shape of object. The sliding of layers of object occur. The shear can be in one direction or in two directions.

**Shearing in the X-direction:** In this horizontal shearing sliding of layers occur. The homogeneous matrix for shearing in the x-direction is shown below:

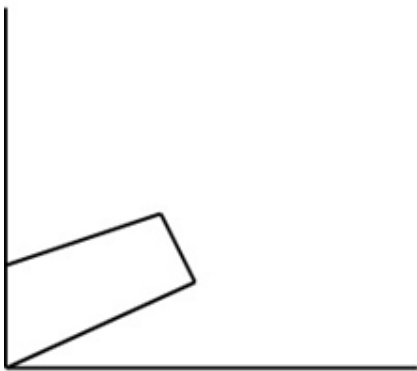
$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



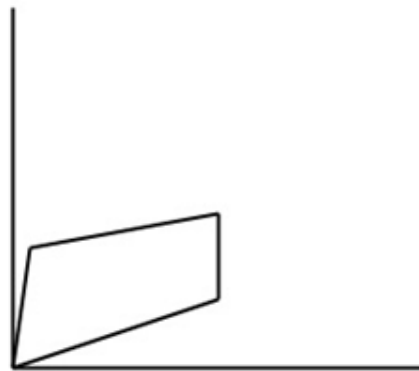
Original Object



Shear in X direction



Shear in Y direction



Shear in both directions

**Shearing in the Y-direction:** Here shearing is done by sliding along vertical or y-axis.

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

**Shearing in X-Y directions:** Here layers will be slid in both x as well as y direction. The sliding will be in horizontal as well as vertical direction. The shape of the object will be distorted. The matrix of shear in both directions is given by:

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

## Matrix Representation of 2D Transformation

1. Scaling	$\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$
2. Rotation (clockwise)	$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
3. Rotation (anti-clock)	$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$
4. Translation	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ t_x & t_y \end{bmatrix}$
5. Reflection (about x axis)	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
6. Reflection (about y axis)	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$
7. Reflection (about origin)	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$
8. Reflection about Y=X	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
9. Reflection about Y= -X	$\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$
10. Shearing in X direction	$\begin{bmatrix} 1 & 0 \\ Sh_x & 1 \end{bmatrix}$
11. Shearing in Y direction	$\begin{bmatrix} 1 & Sh_y \\ 0 & 1 \end{bmatrix}$
12. Shearing in both x and y direction	$\begin{bmatrix} 1 & Sh_y \\ Sh_x & 1 \end{bmatrix}$