1

INTRODUCTION

CHAPTER OUTLINE

- Advantage of Computer Graphics and Areas of Applications
- Hardware of Software for Computer Graphics. (Hard Copy, Display Technologies)
- Random Scan Display System, Video Controller, Random Scan Display Processor
- Raster Graphics
- Scan Conversion Algorithms (Line, Circle, Ellipse)
- Area Filling

INTRODUCTION TO COMPUTER GRAPHICS

Computer graphics is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. In other words, we can say that computer graphics is a rendering tool for the generation and manipulation of images. Computer Graphics is a field related to the generation of graphics using computers. It includes the creation, storage, and manipulation of images of objects. These objects come from diverse fields such as physical, mathematical, engineering, architectural, abstract structures and natural phenomenon. Computer graphics today is largely interactive, i.e., the user controls the contents, structure, and appearance of images of the objects by using input devices, such as keyboard, mouse, or touch-sensitive panel on the screen. In short, Computer graphics refer different things in different contexts:

- **Pictures**, scenes that are generated by a computer.
- **Tools** used to make such pictures, software and hardware, input/output devices.
- The **whole field of study** that involves these tools and the pictures they produce.

TYPES OF COMPUTER GRAPHICS

Basically, there are two types of computer graphics namely.

- Interactive Computer Graphics and
- Non-interactive Computer Graphics

Interactive Computer Graphics

Interactive Computer Graphics involves a two-way communication between computer and user. Here the observer is given some control over the image by providing him with an input device for example the video game controller of the ping pong game. This helps him to signal his request to the computer. The computer on receiving signals from the input device can modify the displayed picture appropriately. To the user it appears that the picture is changing instantaneously in response to his commands. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

Interactive computer graphics affects our lives in a number of indirect ways. For example, it helps to train the pilots of our airplanes. We can create a flight simulator which may help the pilots to get trained not in a real aircraft but on the grounds at the control of the flight simulator. The flight simulator is a mock up of an aircraft flight deck, containing all the usual controls and surrounded by screens on which we have the projected computer-generated views of the terrain visible on takeoff and landing. Flight simulators have many advantages over the real aircrafts for training purposes, including fuel savings, safety, and the ability to familiarize the trainee with a large number of the world's airports.

Non-Interactive Computer Graphics

The non-interactive computer graphics is also known as passive computer graphics. It is the computer graphics in which user does not have any kind of control over the image. Image is merely the product of static stored program and will work according to the instructions given in the program linearly. The image is totally under the control of program instructions not under the user. Example: screen savers.

EARLY HISTORY OF COMPUTER GRAPHICS

We need to take a brief look at the historical development of computer graphics to place today's system in context. Crude plotting of hardcopy devices such as teletypes and line printers' dates from the early days of computing. The whirlwind computer developed in 1950 at the Massachusetts Institute of Technology (MIT) had computer driven CRT displays for output. The SAGE air-defense system developed in the middle 1950s was the first to use command and control CRT display consoles on which operators identified targets with light pens (hand held pointing devices that sense light emitted by objects on the screen) later on Sketchpad system by Ivan Sutherland came in light. That was the beginning of modern interactive graphics. In this system, keyboard and light pen were used for pointing, making choices and drawing.

At the same time, it was becoming clear to computer, automobile, and aerospace manufacturers that (CAD) and computer aided manufacturing (CAM) activities had enormous potential for automating drafting and other drawing intensive activities. The General Motors DAC system for automobile design and the Itek-Digitek system for lens design were pioneering efforts that showed the utility of graphical interaction in the iterative design cycles common in engineering. By the mid 60s, a number of commercial products using these systems had appeared at that time only the most technology intensive organizations could use the interactive computer graphics where as others used punch cards, a non-interactive system.

To understand the many issues in today's modern computer graphics, we need to know how developed computer graphics from its beginnings to this day. In brief the history of computer graphics can be categorized as below:

- **1950's** - the first graphic displays, military applications
- **1962** - the first graphics station (sketchpad) consisting of a monitor, light pen and software for interactive operation constructed by Ivan Sutherland
- **1964** - research team working on the algorithms in computer graphics employed at the University of Utah (including Ivan Sutherland, James Blinn, Edwin Catmull)
- **1965** - the first commercial graphics station: IBM 2250 Display Unit and the IBM PC 1130
- **1969** - beginning of a group SIGGRAPH (Special Interest Group on Graphics) in the organization of ACM (Association for Computing Machinery) gathering of IT professionals
- **1974** - creation of graphics laboratory at the New York Institute of Technology

- **1980** - Turner Whitted published article about creating realistic images, beginning of method of ray tracing
- **1982** - *TRON*, the first film that uses computer graphics. The first completely computer-generated scene in the movie Star Trek II: The Wrath of Khan
- **1983** - Development of fractal techniques and their use in computer graphics. Fractals are used for example in the movie Star Trek II: The Wrath of Khan
- **1984** - work of C. Goral, K. Torrance, D. Greenberg and B. Battaile and proposing a new approach for visualization – the method of radiosity
- **1988** – the first film sequence with morphing in Willow
- **1989** - the first character created using 3D graphics in the studio Industrial Light & Magic (ILM)
- **1993** - dinosaurs in Jurassic Park – the first complete and detailed living organisms generated digital technology
- **1995** - Toy Story implemented complete using computer graphics, the first photo-realistic hair and fur computer generated
- **1999** - The World Wide Web Consortium (W3C) begins development of SVG (Scalable Vector Graphics), a way of using text-based (XML) files to provide higher-quality images on the Web. SVG images can include elements of both conventional vector and raster graphics.
- **2007-** Apple launches its iPhone and iPod Touch products with touch screen graphical user interfaces.
- **2017-** Microsoft announces it will not kill off its basic but very popular Paint program, loved by computer artists for over 30 years.
- The beginnings of computer graphics were related to the military industry, due to the very high cost of equipment. The development of new graphics techniques has forced the film industry requires realistic special effects. Currently, computer graphics is used in many areas of life.

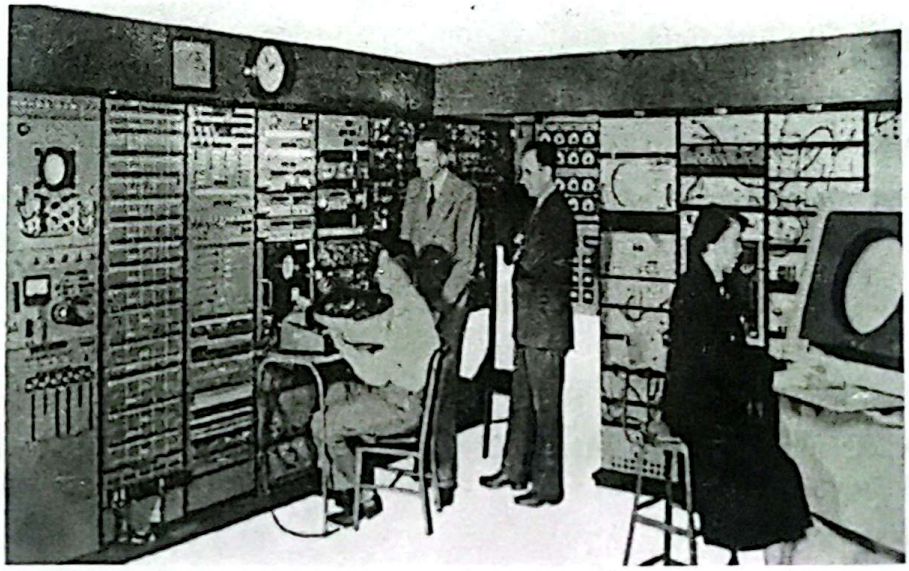


Figure 1.1: Whirlwind: early graphics using Vector Scope (1951)

Figure 1.2: Images on a plotter exploring cockpit design using a 3D model of Human body created by William Fetter in 1960.

Figure 1.3: DAC-1, first commercial CAD system, developed in 1959 by IBM

Figure 1.4: Sketch Pad: First interactive graphics (1961)

Advantages of Computer Graphics

Following are the advantages of computer graphics:

- Today, with the advanced and high-quality computer graphics, it is possible to interact effectively.
- It provides tools for producing procures not only of real-world objects but also of abstract, synthetic objects, such as 3D mathematical surfaces and also of the data that have no inherent geometry, etc.
- It has the ability to show moving pictures and thus, it is possible to produce animations with computer graphics.
- Graphics enables the user to control the animation by adjusting the speed, the portion of the total scene in the view, the geometry of the objects in the scene, the amount of detailing, the light and color adjustments, etc.

Uses of Computer Graphics

There are several uses of CG which are very useful in the current scenario. Some of its uses include:

- Computer program Development
- Making movies
- Video Games (Ex: Plat formers, Role-playing games, side scrollers, first person shooters)
- Catalogs designing
- Creating Commercial Arts
- Scientific Modeling (Ex: Weather Forecasts, Meteorological data)

APPLICATION OF COMPUTER GRAPHICS

Computer graphics is used today in many different areas of science, engineering, industry, business, education, entertainment, medicine, art and training. All of these are included in the following categories.

1. User interfaces

Most applications have user interfaces that rely on desktop windows systems to manage multiple simultaneous activities, and on point-and click facilities to allow users to select menu items, icons and objects on the screen. These activities fall under computer graphics. Typing is necessary only to input text to be stored and manipulated. For example, Word processing, spreadsheet, and desktop-publishing programs are the typical examples where user-interface techniques are implemented.

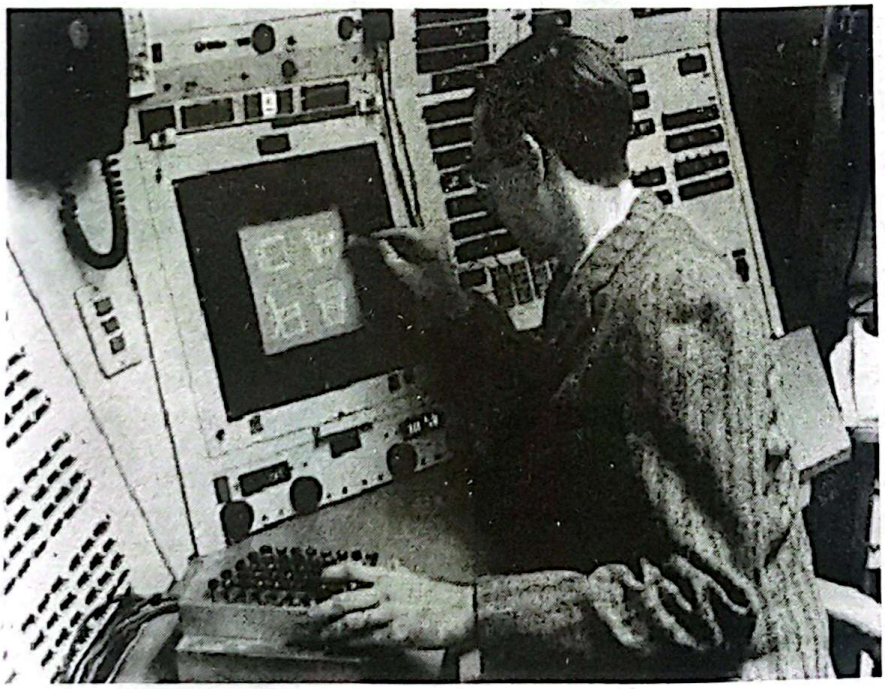


Figure 1.5: Ivan Sutherland's sketchpad (1963)

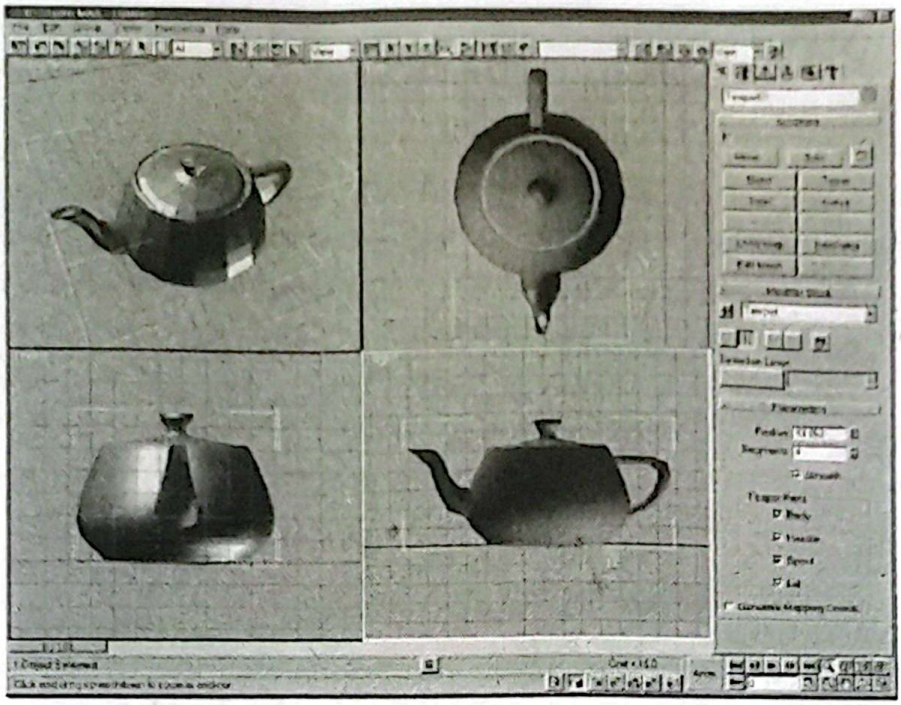


Figure 1.6: 3D Studio Max

2. Plotting

Plotting 2D and 3D graphs of mathematical, physical, and economic functions use computer graphics extensively. The histograms, bar, and pie charts; the task-scheduling charts are the most commonly used plotting. These all are used to present meaningfully and concisely the trends and patterns of complex data.

Figure 1.7: Cannon iPF607-A1 CAD Plotter

3. Office Automation and Electronic Publishing

Computer graphics has facilitated the office automation and electronic publishing which is also popularly known as desktop publishing, giving more power to the organizations to print the meaningful materials Pictures, Drawings, etc. This application is majorly used by the architects, interior designers, engineers, structural designers etc. which makes possible for them to design various components and systems, such as automobile bodies, structures of building, airplanes, ships, optical systems and computer networks.

4. Computer Aided Drafting and Design

One of the major uses of computer graphics is to design components and systems of mechanical, electrical, electrochemical, and electronic devices, including structures such as buildings, automobile bodies, airplane and ship hulls, very large scale integrated (VLSI) chips, optical systems and telephone and computer networks. More frequently however the emphasis is on interacting with a computer-based model of the component or system being designed in order to test, for example, its structural, electrical, or thermal properties. Often, the model is interpreted by a simulator that feeds back the behavior of the system to the user for further interactive design and test cycles. After objects have been designed, utility programs can postprocess the design database to make parts lists, to process 'bills of materials', to define numerical control taps for cutting or drilling parts and so on.

5. Scientific and business Visualization

Generating computer graphics for scientific, engineering, and medical data sets is termed as scientific visualization whereas business visualization is related with the non-scientific data sets such as those obtained in economics. Visualization makes easier to understand the trends and patterns inherent in the huge amount of data sets. It would, otherwise, be almost impossible to analyze those data numerically.

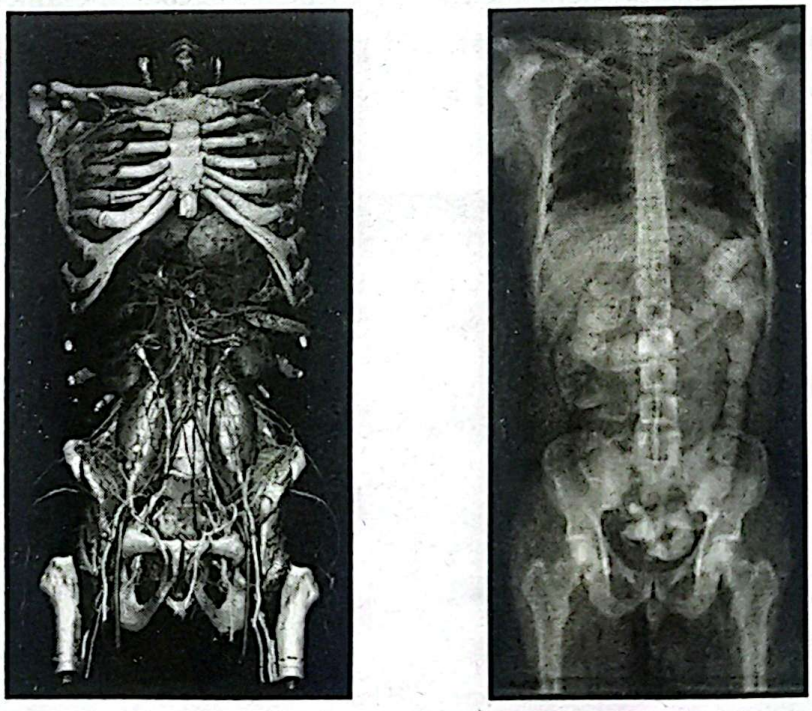


Figure 1.8: Scientific Visualization

6. Simulation and Modeling

Simulation is the imitation of the conditions like those, which is encountered in real life. Simulation thus helps to learn or to feel the conditions one might have to face in near future without being in danger at the beginning of the course. For example, astronauts can exercise the feeling of weightlessness in a simulator; similarly, a pilot training can be conducted in flight simulator. The military tank simulator, the naval simulator, driving simulator, air traffic control

simulator, heavy-duty vehicle simulator, and so on are some of the mostly used simulator in practice. Simulators are also used to optimize the system, for example the vehicle, observing the reactions of the driver during the operation of the simulator.

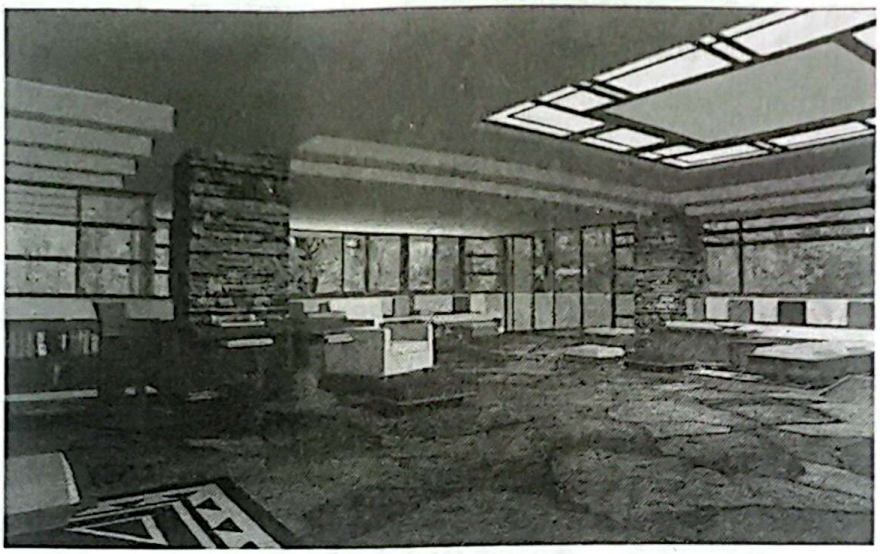


Figure 1.9: Simulation of interior design

7. Entertainment

Computer graphics methods are now commonly used in making motions pictures, music videos and television shows. Sometimes the graphics scenes are displayed by themselves and sometimes graphics objects are combined with the actors and live scenes. A number of hit movies and shows are made using computer graphics technology. Disney movies such as Lion Kings and The Beauty of Beast, and other scientific movies like Jurassic Park, the lost world etc are the best example of the application of computer graphics in the field of entertainment. Instead of drawing all necessary frames with slightly changing scenes for the production of cartoon-film, only the key frames are sufficient for such cartoon-film where the in between frames are interpolated by the graphics system dramatically decreasing the cost of production while maintaining the quality. Computer and video games such FIFA, Doom, Pools are few to name where graphics is used extensively.

Figure 1.10: The flocking behaviour of the wild beast in Ling King

8. Art and Commerce

Here computer graphics is used to produce pictures that express a message and attract attention such as a new model of a car moving along the ring of the Saturn. These pictures are frequently seen at transportation terminals supermarkets, hotels etc. The slide production for commercial, scientific, or educational presentations is another cost-effective use of computer graphics. One of such graphics packages is a PowerPoint.

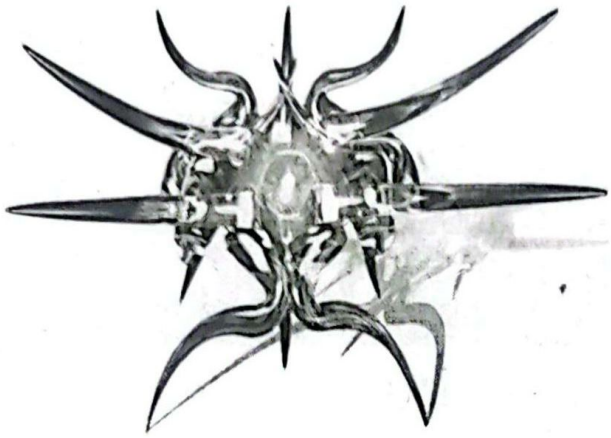


Figure 1.11: Art

9. Medicine and Virtual Surgery

Computer graphics has extensive use in tomography and simulations of operations. Tomography is the technique that allows cross-sectional views of physiological systems in X-ray's photography. Moreover, recent advancement is to make model and study physical functions to design artificial limbs and even plan and practice surgery.

Figure 1.12: Virtual Surgery

10. Process control

By the use of computer, it is possible to control various processed in the industry from remote control room. In such cases, process system and processing parameters are shown on the computer with graphic symbol and identification which makes it easy for operator to monitor and control various processing parameters at a time.

11. History and Cultural Heritage

Another important application of computer graphics is in the field of history and cultural heritage. A lot of work is done in this area to preserve history and cultural heritage. The features so for provide are:

- Innovative graphics presentations developed for cultural heritage applications.
- Interactive computer techniques for education in art history and archeology
- New analytical tools designed for art historians
- Computer simulations of different classes of artistic media.

12. Cartography

Cartography is a subject, which deals with the making of maps and charts. Computer graphics is used to produce both accurate and schematic representations of geographical and other natural phenomena from measurement data. Examples include geographic maps, oceanographic charts, weather maps, contour maps, exploration maps for drilling and mining, relief maps, and population-density maps. Surfer is one of such graphics packages, which is extensively used for cartography.

INTRODUCTION TO IMAGE PROCESSING

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools
- Analyzing and manipulating the image
- Output in which result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, **analogue** and **digital** image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement, and display, information extraction.

Differences between computer graphics and Image processing

The process of Synthesize pictures from mathematical or geometrical models is called computer graphics. Simply in computer graphics we take pixels as mathematical input parameters and draw different images from these pixels. Whereas the process of analyze pictures to derive descriptions (often in mathematical or geometrical forms) of objects appeared in the pictures is called image processing. Image processing is the reverse process of computer graphics. Here we take any one of the pictures as input parameter and return their mathematical form.

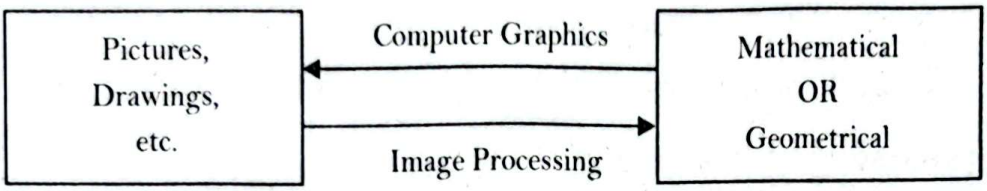


Figure 1.13: Image processing vs. computer graphics

HARDWARE AND SOFTWARE FOR COMPUTER GRAPHICS

A computer system consists of two major elements: hardware and software. Computer hardware is the collection of all the parts you can physically touch. Hardware refers to the physical elements of a computer. This is also sometime called the machinery or the equipment of the computer. Examples of hardware in a computer are the keyboard, the monitor, the mouse and the central processing unit. However, most of a computer's hardware cannot be seen; in other words, it is not an external element of the computer, but rather an internal one, surrounded by the computer's casing (tower). A computer's hardware is comprised of many different parts, but perhaps the most important of these is the motherboard. The motherboard is made up of even more parts that power and control the computer.

In contrast to software, hardware is a physical entity. Hardware and software are interconnected, without software; the hardware of a computer would have no function. However, without the creation of hardware to perform tasks directed by software via the central processing unit, software would be useless.

Hardware is limited to specifically designed tasks that are, taken independently, very simple. Software implements algorithms (problem solutions) that allow the computer to complete much more complex tasks.

Computer software, on the other hand, is not something you can touch. Software is a set of instructions for a computer to perform specific operations. Software, commonly known as programs or apps, consists of all the instructions that tell the hardware how to perform a task. These instructions come from a software developer in the form that will be accepted by the platform (operating system + CPU) that they are based on. For example, a program that is designed for the Windows operating system will only work for that specific operating system. Compatibility of software will vary as the design of the software and the operating system differ. Software that is designed for Windows XP may experience a compatibility issue when running under Windows 2000 or NT.

Software is capable of performing many tasks, as opposed to hardware which can only perform mechanical tasks that they are designed for. Software provides the means for accomplishing many different tasks with the same basic hardware. Practical computer systems divide software systems into two major classes:

- **System software:** Helps run the computer hardware and computer system itself. System software includes operating systems, device drivers, diagnostic tools and more. System software is almost always pre-installed on your computer.
- **Application software:** Allows users to accomplish one or more tasks. It includes word processing, web browsing and almost any other task for which you might install software. (Some application software is pre-installed on most computer systems.)

INPUT DEVICES

The devices that are used to take data from user are called input devices. An **input device** is a hardware or peripheral **device** used to send data to a computer. An **input device** allows users to communicate and feed instructions and data to computers for processing, display, storage and transmission. Following are some of the important input devices which are used in a computer:

- Keyboard
- Mouse
- Joy Stick
- Light pen
- Track Ball
- Scanner
- Graphic Tablet
- Microphone
- Magnetic Ink Card Reader (MICR)
- Optical Character Reader (OCR)
- Bar Code Reader
- Optical Mark Reader (OMR)

Mouse

A mouse is a small hand-held device used to position the cursor on the screen. Mice are relative devices, that is, they can be picked up, moved in space, and then put down gain without any change in the reported position. For this, the computer maintains the current mouse position, which is incremented or decremented by the mouse movements. Following are the mice, which are mostly used in computer graphics.

Figure 1.14: Mouse

Mechanical mouse

A mechanical mouse consists of a heavy rubber ball whose movement makes the cursor move on the screen. Commonly known as the 'rolling rubber ball' mouse, it is considerably heavy, thanks to the rubber ball, a few wheels, and a number of other mechanical parts present inside it. When a roller in the base of this mechanical mouse is moved, a pair of orthogonally arranged toothed wheels, each placed in between a LED and a photo detector, interrupts the light path. The numbers of interrupts so generated are used to report the mouse movements to the computer. Mechanical mice were quite popular in the past decade, but due to their clunkers design and relatively lessened durability, they were quickly replaced by optical mice.

Optical mouse

An optical mouse is technologically much more advanced than a mechanical mouse. Unlike the latter, an optical mouse is completely electronic and therefore has no moving parts. It consists of an LED (that generates the signature red light), a light-detector chip, a switch mechanism and a few other simple components. Some mice have another LED that lights up a plastic strip installed at the back of the mouse as an indication of the mouse's operation.

The optical mouse is used on a special pad having a grid of alternating light and dark lines. A LED on the bottom of the mouse directs a beam of light down onto the pad, from which it is reflected and sensed by the detectors on the bottom of the mouse. As the mouse is moved, the reflected light beam is broken each time a dark line is crossed. The number of pulses so generated, which is equal to the number of lines crossed, and it is used to report mouse movements to the computer.

Trackball

A trackball is a pointing device consisting of a ball housed in a socket containing sensors to detect rotation of the ball about two axes—like an upside-down mouse with an exposed protruding ball. The user rolls the ball with his thumb, fingers, or the palm of his hand to move a cursor. A potentiometer captures the track ball orientation which is calibrated with the translation of the cursor on screen. Tracker balls are common on CAD workstations for ease of use and, before the advent of the touchpad, on portable computers, where there may be no desk space on which to use a mouse.

Figure 1.15: Trackball

Touch Screen

A touch screen is a computer display screen that accepts input directly through the monitor. The screens are sensitive to pressure a user interacts with the computer by touching pictures or words on the screen. They are useful where environmental conditions prohibit the use of a keyboard or mouse. Touch screens are used with automated teller machine (ATM), computer-based training devices etc. There are three types of touch screen technology:

Figure 1.16: Touch Screen

- Surface wave
- Capacitive
- Resistive

Tablet

Figure 1.17: Tablet

A tablet is digitizer. In general, a digitizer is a device which is used to scan over an object, and to input a set of discrete coordinate positions. These positions can then be joined with straight-line segments to approximate the shape of the original object. A tablet digitizes an object detecting the position of a movable stylus (pencil-shaped device) or puck (link mouse with cross hairs for sighting positions) held in the user's hand. A tablet is flat surface, and its size of the tablet varies from about 6 by 6 inches up to 48 by 72 inches or more. The accuracy of the tablets usually falls below 0.2 mm. There are mainly three types of tablets.

- Electrical tablet
- Sonic tablet and
- Resistive tablet

Electrical tablet

A grid of wires on $\frac{1}{4}$ to $\frac{1}{2}$ inch centers is embedded in the tablet surface and electromagnetic signals generated by electrical pulses applied in sequence to the wires in the grid induce an electrical signal in a wire coil in the stylus (or puck). The strength of the signal induced by each pulse is used to determine the position of the stylus. The signal strength is also used to determine roughly how far the stylus is from the tablet. When the stylus is within ½ inch from the tablet, it is taken as near otherwise it is either far or touching. When the stylus is near or touching, a cursor is usually shown on the display to provide visual feedback to the user. A signal is sent to the computer when the tip of the stylus is pressed against the tablet, or when any button on the puck is pressed. The information provided by the tablet repeats 30 to 60 time per second.

Sonic tablet

The sonic tablet uses sound waves to couple the stylus to microphones positioned on the periphery of the digitizing area. An electrical spark at the tip of the stylus creates sound bursts. The position of the stylus or the coordinate values is calculated using the delay between when the spark occurs and when its sound arrives at each microphone. The main advantage of sonic tablet is that it does not require a dedicated working area for the microphones can be placed on any surface to form the "tablet" work area. This facilitates digitizing drawing on thick books. Because in an electrical tablet this is not convenient for the stylus cannot get closer to the tablet surface.

Resistive tablet

The tablet is just a piece of glass coated with a thin layer of conducting material. When a buttery-powered stylus is activated at certain position, it emits high-frequency radio signals, which induces the radio signals on the conducting layer. The strength of the signal received at

the edges of the tablet is used to calculate the position of the stylus. Several types of tablets are transparent, and thus can be backlit for digitizing x- ray films and photographic negatives. The resistive tablet can be used to digitize the objects on CRT because it can be curved to the shape of the CRT. The mechanism used in the electrical or sonic tablets can also be used to digitize the 3Dobjects.

Light pen

Light pen is an input device that utilizes a light-sensitive detector to select objects on a display screen. A light pen is similar to a mouse, except that with a light pen you can move the pointer and select objects on the display screen by directly pointing to the objects with the pen.

Figure 1.18: Light Pen

It is a pencil-shaped device to determine the coordinates of a point on the screen where it is activated such as pressing the button. Because of the following drawbacks the light pens are not popular now days. Light pen obscures the screen image as it is pointed to the required spot Prolong use of it can cause arm fatigue. It cannot report the coordinates of a point that is completely black. As a remedy one can display a dark blue field in place of the regular image for a single frame time. It gives sometimes false reading due to background lighting in a room

Data Glove

A data glove is an interactive device, resembling a glove worn on the hand, which facilitates tactile sensing and fine-motion control in robotics and virtual reality . Data gloves are one of several types of electromechanical devices used in haptics applications.

Figure 1.19: Data Glove

Tactile sensing involves simulation of the sense of human touch and includes the ability to perceive pressure, linear force, torque, temperature, and surface texture. Fine-motion control involves the use of sensors to detect the movements of the user's hand and fingers, and the translation of these motions into signals that can be used by a virtual hand (for example, in gaming) or a robotic hand (for example, in remote-control surgery).

Touch panel

The touch panel allows the users to point at the screen directly with a finger to move the cursor around the screen, or to select the icons. Following are the mostly used touch panels.

- Optical touch panel
- Sonic panel and
- Electrical touch panel

Optical touch panel

It uses a series of infra-red light emitting diodes (LED) along one vertical edge and along one horizontal edge of the panel. The opposite vertical and horizontal edges contain photo-detectors to form a grid of invisible infrared light beams over the display area. Touching the screen breaks one or two vertical and horizontal light beams, thereby indicating the finger's position. The cursor is then moved to this position, or the icon at this position is selected. It two parallel beams are broken, the finger is presumed to be centered between them; if one is broken, the finger is presumed to be on the beam. There is a low-resolution panel, which offers 10 to 50 positions in each direction.

Sonic panel

Bursts of high-frequency sound waves traveling alternately horizontally and vertically are generated at the edge of the panel. Touching the screen causes part of each wave to be reflected back to its source. The screen position at the point of contact is then calculated using the time elapsed between when the wave is emitted and when it arrives back at the source. This is a high-resolution touch panel having about 500 positions in each direction.

Electrical touch panel

It consists of slightly separated two transparent plates one coated with a thin layer of conducting material and the other with resistive material. When the panel is touched with a finger, the two plates are forced to touch at the point of contact thereby creating the touched position. The resolution of this touch panel is similar to that of sonic touch panel.

Keyboard

A keyboard creates a code such as ASCII uniquely corresponding to a pressed key. It usually consists of alphanumeric keys, function keys, cursor-control keys, and separate numeric pad. It is used to move the cursor, to select the menu item, pre-defined functions. In computer graphics keyboard is mainly used for entering screen coordinates and text, to invoke certain functions. Now-a-days ergonomically designed keyboard (Ergonomic keyboard) with removable palm rests is available. The slope of each half of the keyboard can be adjusted separately.

Figure 1.20: Keyboard

Bar Code Reader

A barcode reader is a hand-held or stationary input device used to capture and read information contained in a bar code. A barcode reader consists of a scanner, a decoder and a cable used to connect the reader with a computer. Because a barcode reader merely captures and translates the barcode into numbers and/or letters, the data must be sent to a computer so that a software application can make sense of the data. Barcode scanners can be

Figure 1.21: Bar Code Reader

connected to a computer through a serial port, keyboard port, or an interface device called a wedge. A barcode reader works by directing a beam of light across the bar code and measuring the amount of light that is reflected back. (The dark bars on a barcode reflect less light than the white spaces between them.) The scanner converts the light energy into electrical energy, which is then converted into data by the decoder and forwarded to a computer.

OUTPUT DEVICES

Output devices allow computers to communicate with users and with other devices. This can include peripherals, which may be used for input/output (I/O) purposes, like network interface cards (NICs), modems, IR ports, RFID systems and wireless networking devices, as well as mechanical output devices, like solenoids, motors and other electromechanical devices. Some of the most common output devices that people are familiar with include monitors, which produce video output; speakers, which produce audio output; and printers, which produce text or graphical output.

The most common graphics output device is the video monitor which is based on the standard cathode ray tube (CRT) design, but several other technologies exist such as LCDs, LEDs, the direct view storage tube (DVST) etc. and solid-state monitors may eventually predominate. The display devices are categorized as follows:

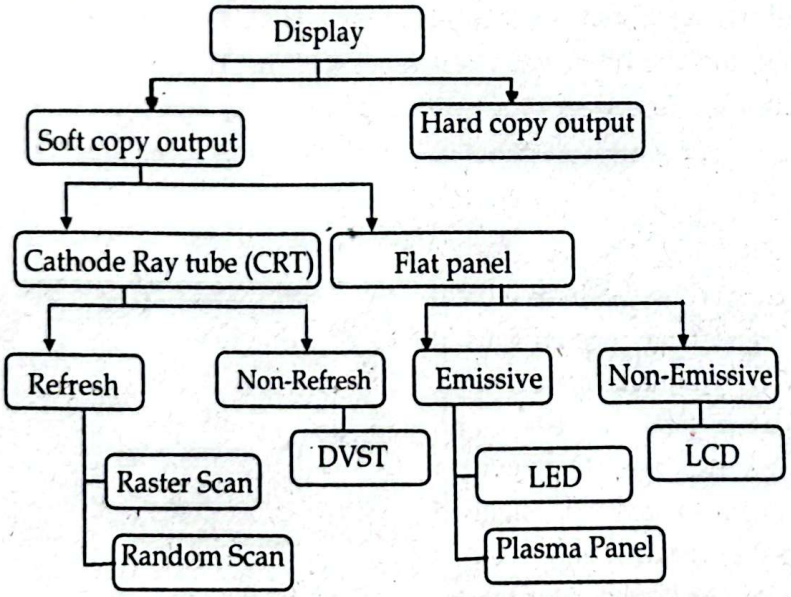


Figure 1.22: Display devices in Hierarchical order

Cathode Ray Tube (CRT)

CRT is the most common display devices on computer today. It can generate only a single color at a time.

A CRT is an evacuated glass tube, with a heating element on one end and a phosphor-coated screen on the other end. When a current flow through this heating element (filament) the conductivity of metal is reduced due to high temperature. These cause electrons to pile up on the filament. These electrons are attracted to a strong positive charge from the outer surface of the

Figure 1.23: CRT Monitor

focusing anode cylinder. Due to the weaker negative charge inside the cylinder, the electrons head towards the anode forced into a beam and accelerated by the inner cylinder walls in just the way that water is speeds up when its flow though a small diameter pipe. The forwarding fast electron beam is called Cathode Ray. A cathode ray tube is shown in figure below.

When electrons strike the phosphor coating on the tube, light is emitted. The direction of the beam is controlled by two pairs of deflection plates. The output of the computer is converted, by digital-to-analog converters, to voltages across the x and y deflection plates. Light appears on the surface of the CRT when a sufficiently intense beam of electrons is directed at the phosphor.

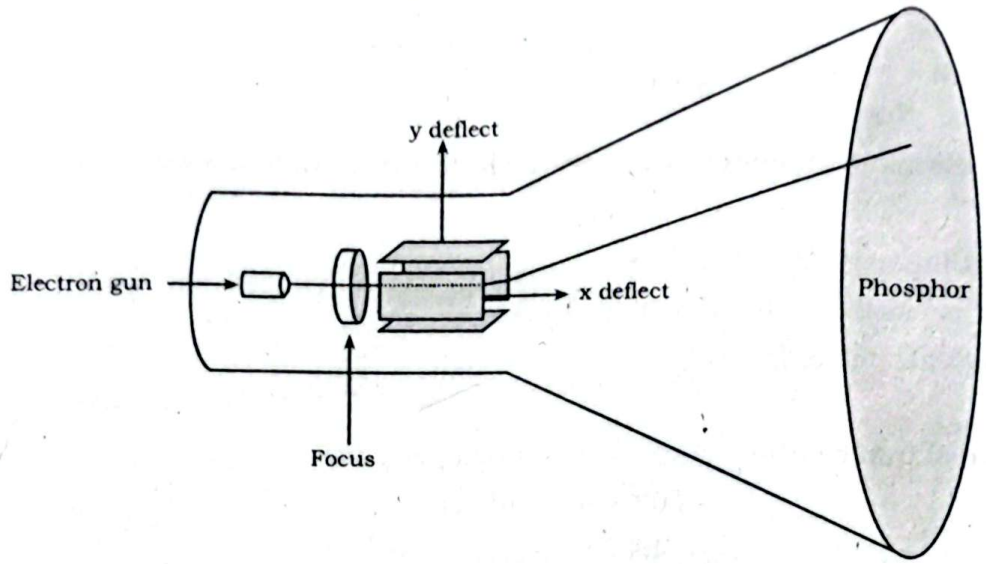


Figure 1.24: Cathode Ray Tube

The amount of light emitted by the phosphor coating depends on the number of electrons striking the screen. The brightness of the display is controlled by varying the voltage on the control grid.

Properties of CRT

Pixel or PEL

It is also called PEL. It comes from the words Picture Element (PEL). Pixel is the smallest addressable screen element. Each pixel has its own intensity, name or address by which we can control. It is a measure of screen resolution. Graphics monitors display pictures by dividing the display screen into thousands (or millions) of pixels, arranged in rows and columns.

The pixels are so close together that they appear connected. The number of bits used to represent each pixel determines how many colors or shades of gray can be displayed. For example, in 8-bit color mode, the color monitor uses 8 bits for each pixel, making it possible to display $2^8 = 256$ different colors or shades of gray.

On color monitors, each pixel is actually composed of three dots - a red, a blue, and a green one. Ideally, the three dots should all converge at the same point, but all monitors have some convergence error that can make color pixels appear fuzzy.

Frame buffer

It is the large contiguous piece of memory into which the intensity values for all pixels are placed. It contains the internal representation of the image. A frame buffer (frame buffer, or sometimes frame store) is a portion of RAM containing a bitmap that drives a video display.

The frame buffer is the video memory that is used to hold or map the image displayed on the screen. The amount of memory required to hold the image depend primarily on the resolution of the screen image and the color depth. The formula to calculate how much video memory is required at a given resolution and bit depth is given below:

$$\text{Memory MB} = \frac{\text{x-resolution} \times \text{y-resolution} \times \text{Bit per pixel}}{8 \times 1024 \times 1024}$$

Example 1.1: How many kilobytes does a frame buffer need in 600 × 400 pixels?

Solution:

Resolution = 600 × 400

Let, n bits are required to store 1 pixel

Size of frame buffer = ?

Now,

$$\text{Size of frame buffer} = \text{resolution} \times \text{bits per pixel}$$
$$= 600 \times 400 \times n \text{ bits}$$
$$= 240000 \times n \text{ bits}$$
$$= \frac{(240000 \times n)}{(8 \times 1024)} \text{ Kilobytes}$$
$$= (29.30 \times n) \text{ Kilobytes}$$

Example 1.2: A system with 24 bits per pixel and resolution of 1024 by 1024, Calculate the size of frame buffer (in Megabytes)

Solution:

Frame size in bits = 24 × 1024 × 1024 bits

$$\text{Frame size in bytes} = \frac{24 \times 1024 \times 1024}{8} \text{ bytes}$$

$$\text{Frame size in Kilobytes} = \frac{24 \times 1024 \times 1024}{8 \times 1024} \text{ KB}$$

$$\text{Frame size in Megabytes} = \frac{24 \times 1024 \times 1024}{8 \times 1024 \times 1024} \text{ MB}$$
$$= 3 \text{ MB}$$

Example 1.3: How long would it take to load a (640 × 480) frame buffer with 12 bits per pixel, if 10^5 bits can be transferred per second? How long would it take to load a 24 bits per pixel frame buffer with a resolution of 1280 by 1024 using the same transfer rate?

Solution:

$$\text{Size of frame buffer} = 640 \times 480 \text{ pixels}$$
$$= 640 \times 480 \times 12 \text{ bits [since, 1 pixel = 12 bits]}$$

Now,

Time taken to transfer $= \frac{640 \times 480 \times 12}{10^5}$ seconds [$\because 10^5$ bits takes 1 sec. to transfer]

$= 36.864$ seconds

Example 1.4: Consider a RGB raster system is to be designed using 8 inches by 10 inches screen with a resolution of 100 pixels per inch in each direction. If we want to store 8 bits per pixel in the frame buffer. How much storage (in bytes) do we need for the frame buffer?

Solution:

Size of screen = 8 inches × 10 inches

Bit per pixel storage = 8

Pixel per inch (Resolution) = 100

Then,

Total no of pixels = 8×100 × 10 × 100 pixels

Now,

Total storage required in frame buffer $= (800 \times 1000 \times 8)$ bits

$= \frac{(800 \times 1000 \times 8)}{8 \text{ Bytes}}$,

$= 8,00,000$ Bytes

Pixel density

Number of pixels per unit area is called pixel density. Pixel density is a metric telling us how many pixels there are in a fixed area of a display. It's a very important metric because it lets us know how closely packed the pixels on a display area. This is something that determines the quality, clarity, and readability of the image displayed. It is usually measured in a unit called pixels per inch (ppi). If a careen of size 4 inch by 3 inch and of resolution 800 × 600 then pixel density is,

$$\text{Pixel Density (PD)} = \frac{800}{4} \times \frac{600}{3} = 200 \times 200$$

If we have two displays with the same resolution, the smaller one will have the higher pixel density. And if we have displays with the same size, the one with a higher resolution will have the higher pixel density. So, it's clear that the two parameters that we need in order to calculate the pixel density are display size and resolution. We can calculate the pixel density knowing the resolution and size (diagonal) or a display as below,

$$\text{Pixel Density (PD)} = \frac{\sqrt{\text{Width}^2 + \text{Length}^2}}{\text{Screen Size (diagonal)}}$$

Example 1.5: **Let's say we have a computer display that is 21.5 inches with a 1920×1080 resolution.**

Display size = 21.5 inch
Width = 1920, Length = 1080

Solution:

Display size = 21.5 inches, Width = 1920, Length = 1080

$$\text{Diagonal resolution} = \sqrt{1920^2 + 1080^2}$$

$$\text{Diagonal resolution} = \sqrt{3686400 + 1166400}$$

$$\text{Diagonal resolution} = \sqrt{4852800}$$

$$\text{Diagonal resolution} = 22030 \text{ (approx.)}$$

$$\text{Pixel density} = \frac{\text{diagonal resolution}}{\text{display size}}$$

$$\text{Pixel density} = \frac{2203}{21.5}$$

$$\text{Pixel density} = 103 \text{ ppi (approx.)}$$

Persistence

Persistence is the one of the major property of phosphor used in CRT's. It means how long phosphors continue to emit light after the electron beam is removed. Persistence is defined as the time it takes the emitted light from the screen to decay to one-tenth of its original intensity. Lower persistence phosphors require higher refresh rates to maintain a picture on the screen. A phosphor with lower persistence is useful for animation and a higher-persistence phosphor is useful for displaying highly complex static picture. Graphics monitor are usually constructed with the persistence 10 to 60 microseconds.

Resolution

The maximum number of points (pixel) that can be displayed without overlap on a CRT is referred to as their solution. It is also defined as maximum number of points displayed horizontally and vertically without overlap on a display screen. More precise definition of resolution is no of dots per inch (dpi/pixel per inch) that can be plotted horizontally and vertically. Resolution of 1280 × 720 means that there are 1280 horizontal lines and 720 vertical lines or 1280 pixels horizontally and 720 pixels vertically. We can say that the higher is the pixel resolution, the higher is the quality of the image. There are two types of resolutions:

- Image resolution and
- Screen resolution

Image resolution is defined as the distance from one pixel to the next pixel i.e., pixel spacing.

Screen resolution is defined as the number of pixels in the horizontal and vertical directions on the screen.

Example 1.6: What is the resolution of a 2 × 2 inches image with 512 × 512 pixels?

Solution:

Image size = 2 × 2 inches

Pixels = 512 × 512 pixels

Resolution = ?

Now,

$$\text{Resolution} = \frac{512}{2} = 256 \text{ pixels/inch}$$

Megapixels

We can calculate mega pixels of a display device using pixel resolution.

Megapixels = Column pixels (width) × row pixels (height) / 1 Million.

The size of an image can be defined by its pixel resolution.

Size = pixel resolution × bpp (bits per pixel)

Example 1.7: Let's say we have an image of dimension: 2500 × 3192.

Its pixel resolution = 2500 × 3192 = 7982350 bytes.

Dividing it by 1 million = 7.9 = 8 mega pixels (approximately).

Refresh Rate

Light emitted by phosphor fades very rapidly, so to keep the drawn picture glowing constantly; it is required to redraw the picture repeatedly and quickly directing the electron beam back over some point. This process is called refresh operation.

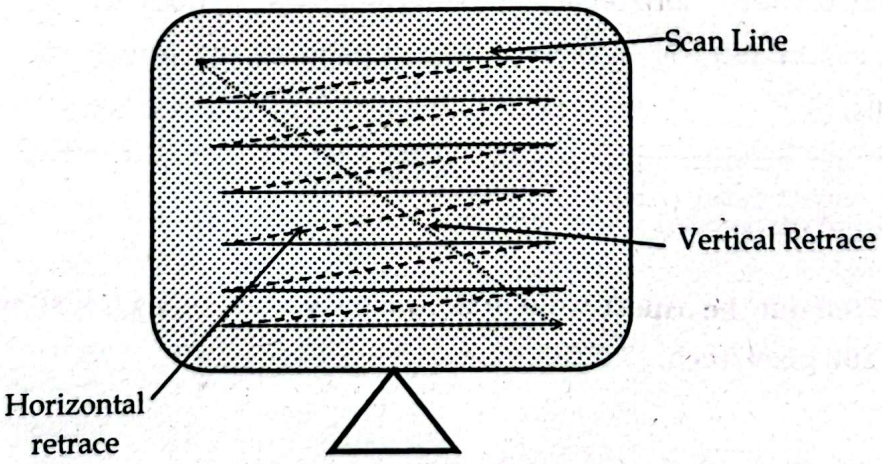


Figure 1.25: A Schematic diagram of refreshing

The no of times/sec the image is redrawn to give a feeling of non-flickering pictures is called refresh-rate. Refresh rates are described in units of cycles per second, or Hertz (Hz), where a cycle corresponds to one frame. It is also called a frame rate, horizontal scan rate, and vertical scan rate. A refresh rate of 70 Hz means that the image is redrawn 70 times second. When

choosing a monitor, one of the factors that the customer usually considers is the refresh rate. A high refresh rate is important in providing a clear picture and avoiding eye fatigue. An image appears on screen when electron beams strike the surface of the screen in a **zig-zag** pattern. A refresh rate is the number of times a screen is redrawn in one second and is measured in Hertz (Hz). Therefore, a monitor with a refresh rate of 85 Hz is redrawn 85 times per second. A monitor should be "flicker-free meaning that the image is redrawn quickly enough so that the user cannot detect flicker, a source of eye strain. Today, a refresh rate of 75 Hz or above is considered to be flicker-free.

Bit depth (color depth)

Color depth or **color depth** also known as **bit depth**, is either the number of **bits** used to indicate the **color** of a single pixel, in a bitmapped image or video frame buffer, or the number of **bits** used for each **color** component of a single pixel. It is defined as the number of bits assigned to each pixel in the image. It specifies the number of colors that a monitor can display.

Aspect ratio

It is defined as the number of the horizontal points to the vertical points required to produce equal length lines in both directions on the screen. Aspect ratio is the ratio between width of an image and the height of an image. It is commonly explained as two numbers separated by a colon (8:9). This ratio differs in different images, and in different screens. The common aspect ratios are:

1.33:1, 1.37:1, 1.43:1, 1.50:1, 1.56:1, 1.66:1, 1.75:1, 1.78:1, 1.85:1, 2.00:1, etc.

Aspect ratio maintains a balance between the appearances of an image on the screen, means it maintains a ratio between horizontal and vertical pixels. It does not let the image to get distorted when aspect ratio is increased. Aspect ratio of the screen can be calculated by following formula,

$$\text{Aspect ratio (AR)} = \frac{\text{Width of the image}}{\text{Height of the image}}$$

Example 1.8: Find out the aspect ratio of the raster system using 8 × 10 inches screen and 100 pixel/inch.

Solution:

Width of image = 8 ×100 = 800 inches

Height of image = 10 × 100 = 1000 inches

Aspect ratio = ?

Now,

$$\text{Aspect ratio} = \frac{800}{1000} = \frac{4}{5} = 4:5$$

Example 1.9: An image has a height of 3 inches and an aspect ratio of 2.5. What is its width?

Solution:

Height of image = 3 inches
Aspect ratio = 2.5
Width of image = ?

Now,

Width of image = aspect ratio × height of image = 2.5 × 3 = 7.5 inches

Bit map and pixel map

An image of two colors (generally black and white) is called as a bitmap. An image of more than two colors is called as a pixel map.

Scan line

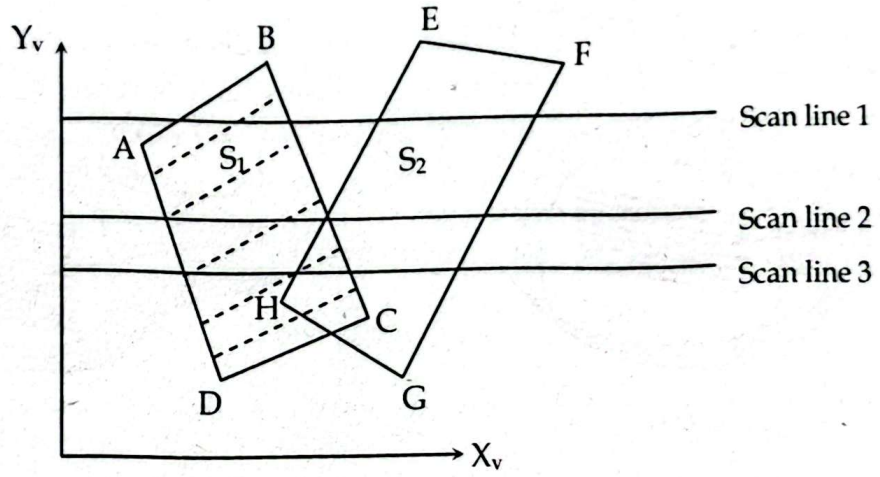


Figure 1.26: Scan Line

Time taken to access horizontal pixels is called scan line. A scan line is one line, or row, in a raster scanning pattern, such as a line of video on a cathode ray tube (CRT) display of television set or computer monitor.

TYPES OF REFRESH CATHODE RAY TUBES (CRT'S)

A **cathode ray tube** (CRT) is a specialized vacuum tube in which images are produced when an electron beam strikes phosphorescent surface. Most desktop computer displays make use of CRTs. There are two types of Refresh CRT's:

- Raster-Scan Displays
- Random-Scan Displays

Raster-Scan Display

In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots. Picture definition is stored in memory area called

the **Refresh Buffer** or **Frame Buffer**. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and painted on the screen one row (scan line) at a time as illustrated in figure 1.25. Each screen point is referred to as a **pixel (picture element)** or **pel**. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line. In comparisons of random scan display method, it is more accurate method.

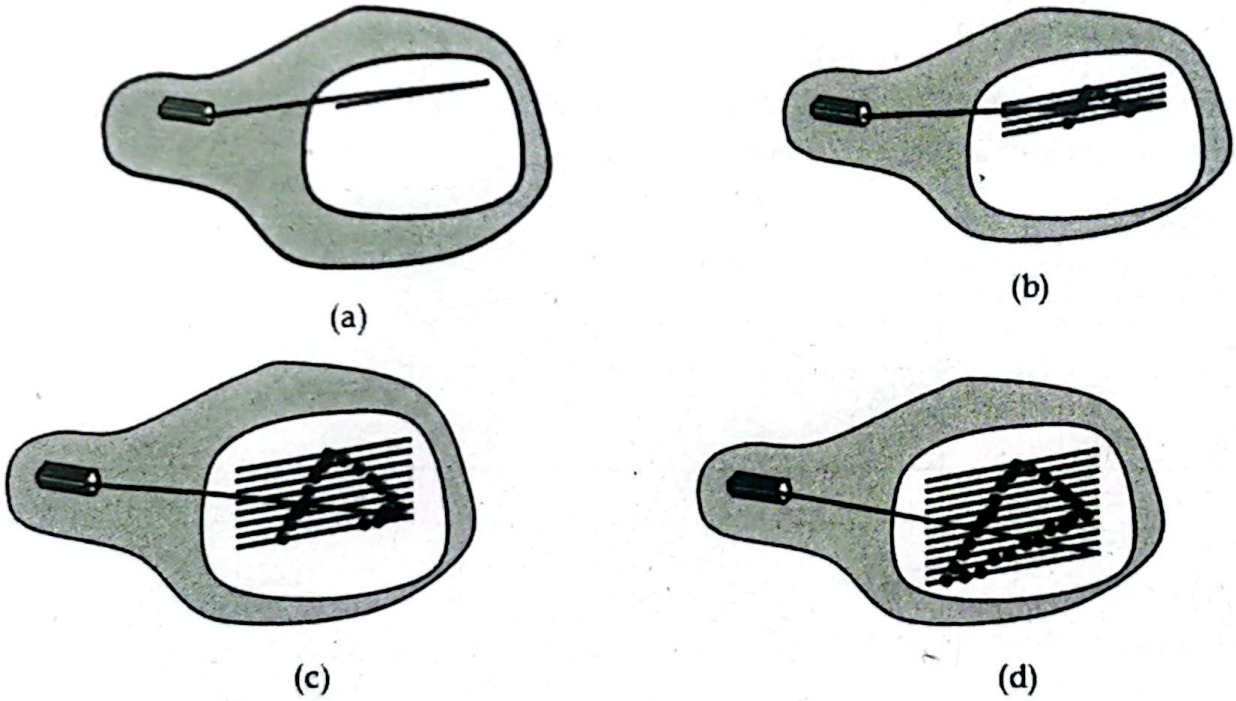


Figure 1.27: A raster-scan system displays an object as a set of points across each screen scan line

The stored intensity value is retrieved from frame buffer and painted on the scan line at a time. The stored intensity value is retrieved from frame buffer and painted on the screen. Refreshing on Raster-Scan display is carried out at the rate of 60 or higher frames per second. 60 frames per second is also termed as 60 cycle per second usually used unit Hertz (HZ). Most of display devices are based on this technology. For example, CRT, color CRT, LCD, and LED etc.

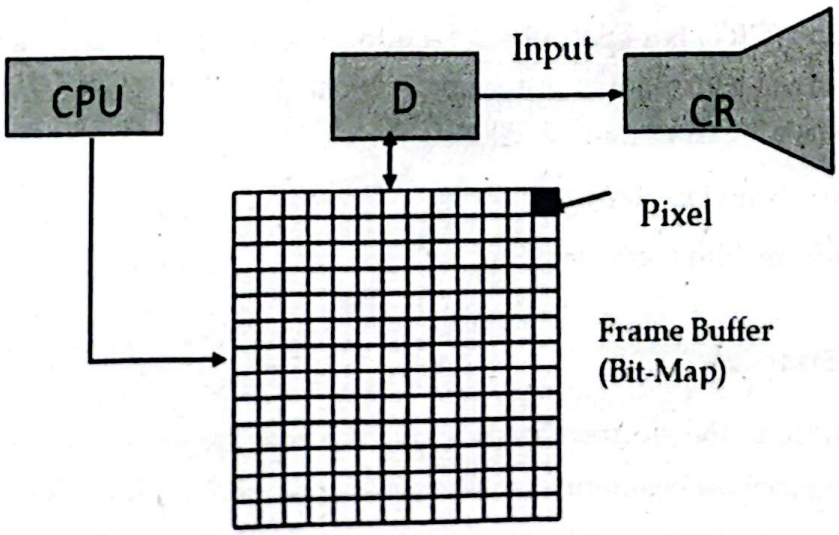


Figure 1.28: Raster Scan display system

Horizontal retrace/ vertical retrace

Returning of electron beam from right end to left end after refreshing each scan line is horizontal retrace. At the end of each frame, the electron beam returns to the top left corner to begin next frame called vertical retrace.

Architecture of Raster-Scan System

The raster graphics systems typically consist of several processing units. CPU is the main processing unit of computer systems. Besides CPU, graphics system consists of a special buffer processor called video controller or display controller. It is used to control the operation of the display device. In addition, to the video controller, raster scan systems can have other processors as co-processors which are called graphic controller or display processors. A fixed area of system memory is reserved for the frame buffer. The video controller has the direct access to the frame buffer for refreshing the screen. The video controller cycles through the frame buffer, one scan line at a time, typically at 60 times per second or higher. The contents of frame buffer are used to control the CRT beam's intensity or color. The organization of simple raster system is shown in the figure below.

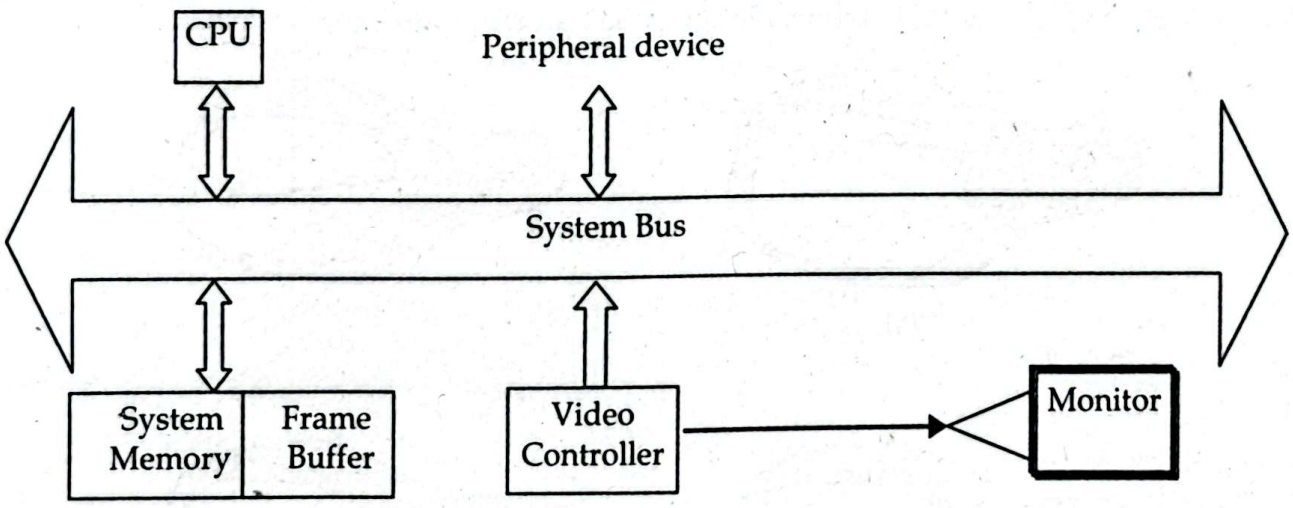


Figure 1.29: Architecture of simple Raster-scan system

Advantages of Raster scan system

- Show Realistic pictures
- Million Unique hues can be produced
- Shadow scenes are conceivable

Disadvantages of Raster scan system

- Low Resolution
- Electron beam coordinated to whole screen not exclusively to those parts of the screen where picture is to be drawn so tedious when the drawn picture estimate is especially not as much as the whole screen.
- Expensive

Random-Scan (Vector display) system

In this technique, the electron beam is directed only to the part of the screen where the picture is to be drawn rather than scanning from left to right and top to bottom as in raster scan. It is also called vector display, stroke-writing display, or calligraphic display. Picture definition is stored as a set of line-drawing commands in an area of memory referred to as the refresh display file. To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn. After all the line-drawing commands are processed, the system cycles back to the first line command in the list. Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.

The refresh rate of vector display depends upon the no of lines to be displayed for any image. Picture definition is stored as a set of line drawing instructions in an area of memory called the refresh display file (Display list or display file). It is designed for drawing all component lines 30 to 60 times per second. Such systems are designed for line-drawing applications and cannot display realistic shaded scenes. Since CRT beam directly follows the line path, the vector display system produces smooth line. Plotter is the best example of this system.

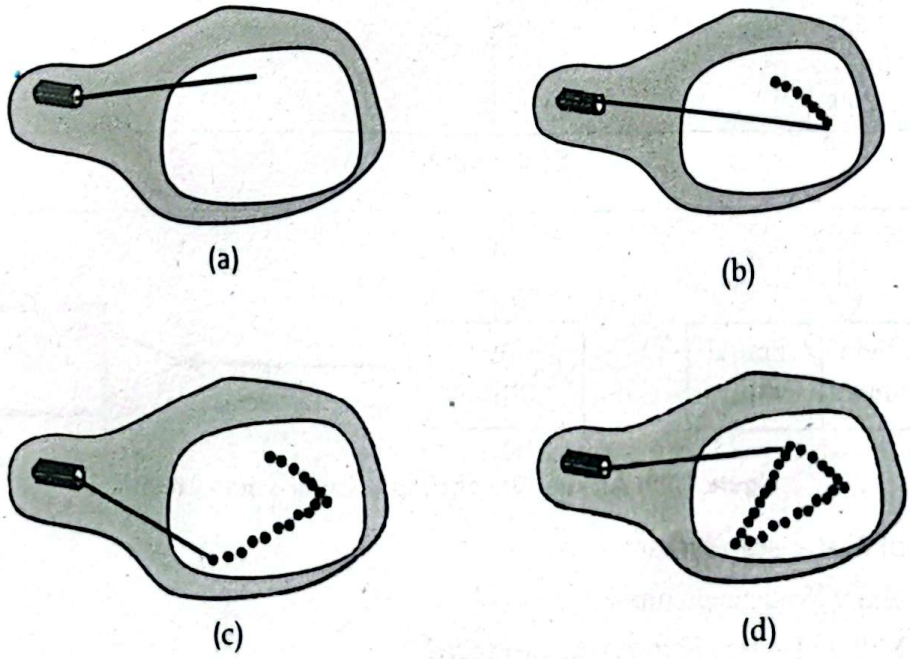


Figure 1.30: Random Scan Display

Architecture of Random Scan Systems

Vector display system consists of additional processing unit along with CPU which is called the display processor or graphics controller. Picture definition is stored as a set of line drawing commands in an area of memory called display list. A display list (or display file) contains a series of graphics commands that define an output image. This display list is then accessed by the display processor to create an image.

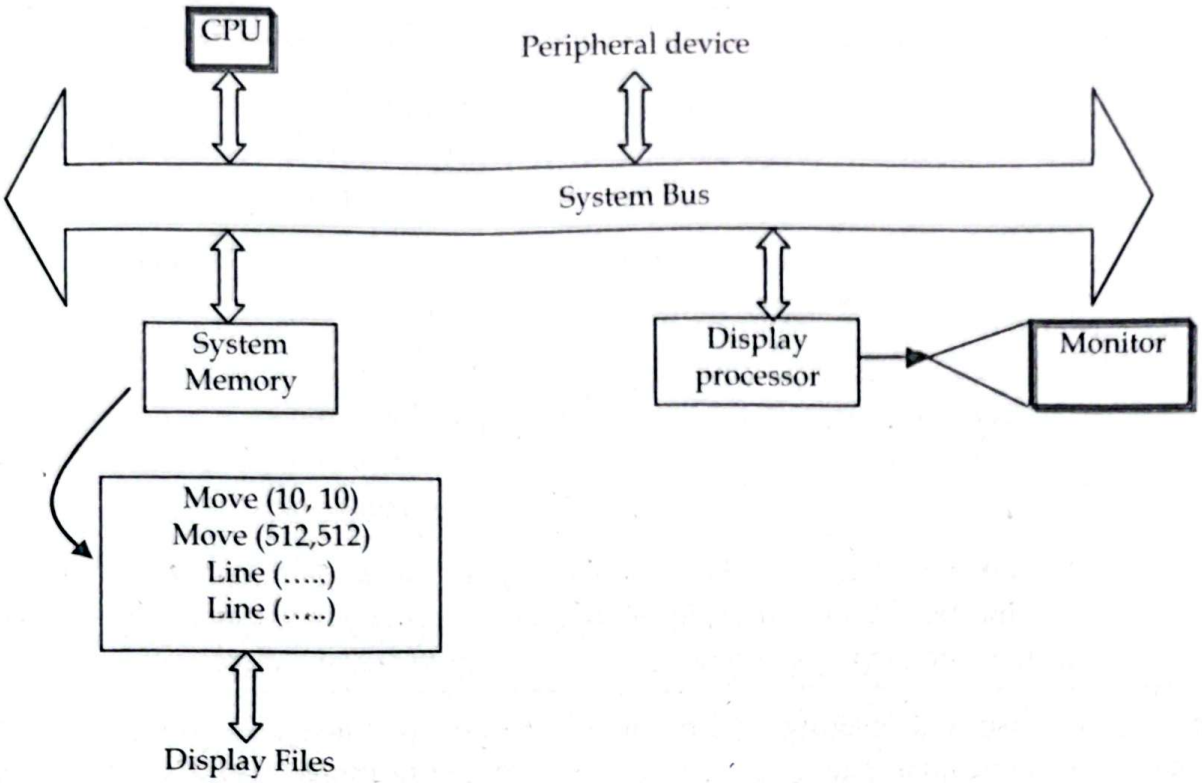


Figure 1.31: Architecture of Vector Display System

Advantages of Random scan display

- high resolution
- Produce smooth line drawings.

Disadvantages of random scan display

- costlier
- Just does wire frame.
- Complex scene causes visible flicker.

Comparison between Raster and Random (Vector) scan display systems

	Raster Scan System	Random Scan System
Resolution	It has poor or less resolution because picture definition is stored as an intensity value.	It has High Resolution because it stores picture definition as a set of line commands.
Electron-Beam	Electron beam is directed from top to bottom and one row at a time on screen, but electron beam is directed to whole screen.	Electron beam is directed to only that part of screen where picture is required to be drawn, one line at a time so also called vector display.
Cost	It is less expensive than Random Scan System.	It is Costlier than Raster Scan System.

	Raster Scan System	Random Scan System
Refresh Rate	The refresh rate is independent of picture complexity. Refresh rate is 60 to 80 frames per second.	Refresh Rate depends on the number of lines to be displayed i.e., 30 to 60 times per second.
Picture Definition	It Stores picture definition in Refresh Buffer also called Frame Buffer.	It Stores picture definition as a set of line commands called Refresh Display File.
Line Drawing	Zig-Zag line is produced because plotted values are discrete.	Smooth line is produced because directly the line path is followed by electron beam.
Realism in display	It contains shadow, advance shading and hidden surface technique so gives the realistic display of scenes.	It does not contain shadow and hidden surface technique so it cannot give realistic display of scenes.
Image Drawing	It uses Pixels along scan lines for drawing an image.	It is designed for line drawing applications and uses various mathematical functions to draw.

Color CRT

Normal CRT generates only a single color at a time but we can also generate multi- colors using multilayer phosphor. In color CRT, the phosphor on the face of CRT screen is laid in to different fashion. The basic principle behind colored displays is that combining the 3 basic colors –Red, Blue and Green can produce every color. By choosing different ratios of these three colors, we can produce different colors – millions of them in-fact. We also have basic phosphors, which can produce these basic colors. So, one should have a technology to combine them in different combinations. There are two popular techniques for producing color displays with a CRT are:

- Beam penetration method
- Shadow mask method

Beam Penetration method

This CRT is similar to the simple CRT, but it makes use of multi colored phosphorus of number of layers. Each phosphorus layer is responsible for one color. All other arrangements are similar to simple CRT. It can produce a maximum of 4 to 5 colors. The organization is something like this - The red, green and blue phosphorus are coated in layers - one behind the other. If a low-speed beam strikes the CRT, only the red colored phosphorus is activated, a slightly accelerated beam would activate both red and green (because it can penetrate deeper) and a much more activated one would add the blue component also. But the basic problem is a reliable

technology to accelerate the electronic beam to precise levels to get the exact colors - it is easier said than done. However, a limited range of colors can be conveniently produced using the concept.

This method is commonly used for random scan display or vector display. In random scan display CRT, the two layers of phosphor usually red and green are coated on CRT screen. Display color depends upon how far electrons beam penetrate the phosphor layers. Slow electron excites only red layer so that we can see red color displayed on the screen pixel where the beam strikes. Fast electron beam excites green layer penetrating the red layer and we can see the green color displayed at the corresponding position. Intermediate is combination of red and green so two additional colors are possible - orange and yellow so only four colors are possible. In this type of display method, no good quality picture occurs.

Advantages of beam penetration CRT

- Half cost as compared to that of shadow mask CRT
- It is an inexpensive way to produce color in random scan monitor
- Its resolution is better

Disadvantages of beam penetration CRT

- A problem arises at the time of switching of colors as it is time consuming.
- It consumes significant amount of accelerating potential in order to switch color.

Shadow Mask Method

In this display, there are three electron guns, one for each primary color i.e., red, green and blue. The electron guns are frequently arranged in a triangular pattern called delta corresponding to a similar triangular pattern of red, green and blue phosphor dots on the face of the CRT. To ensure that individual electron guns excite the correct phosphor dots, a perforated metal grid (Shadow mask) is placed between the electron guns and the face of the CRT. Shadow mask method is used for raster scan system so they can produce wide range of colors. In shadow mask color CRT, the phosphor on the face of the screen is laid out in a precise geometric pattern. There are two primary variations.

- The stripe pattern of inline tube
- The delta pattern of delta tube

The color variations in a shadow mask CRT can be obtained by varying the intensity levels of the three beams.

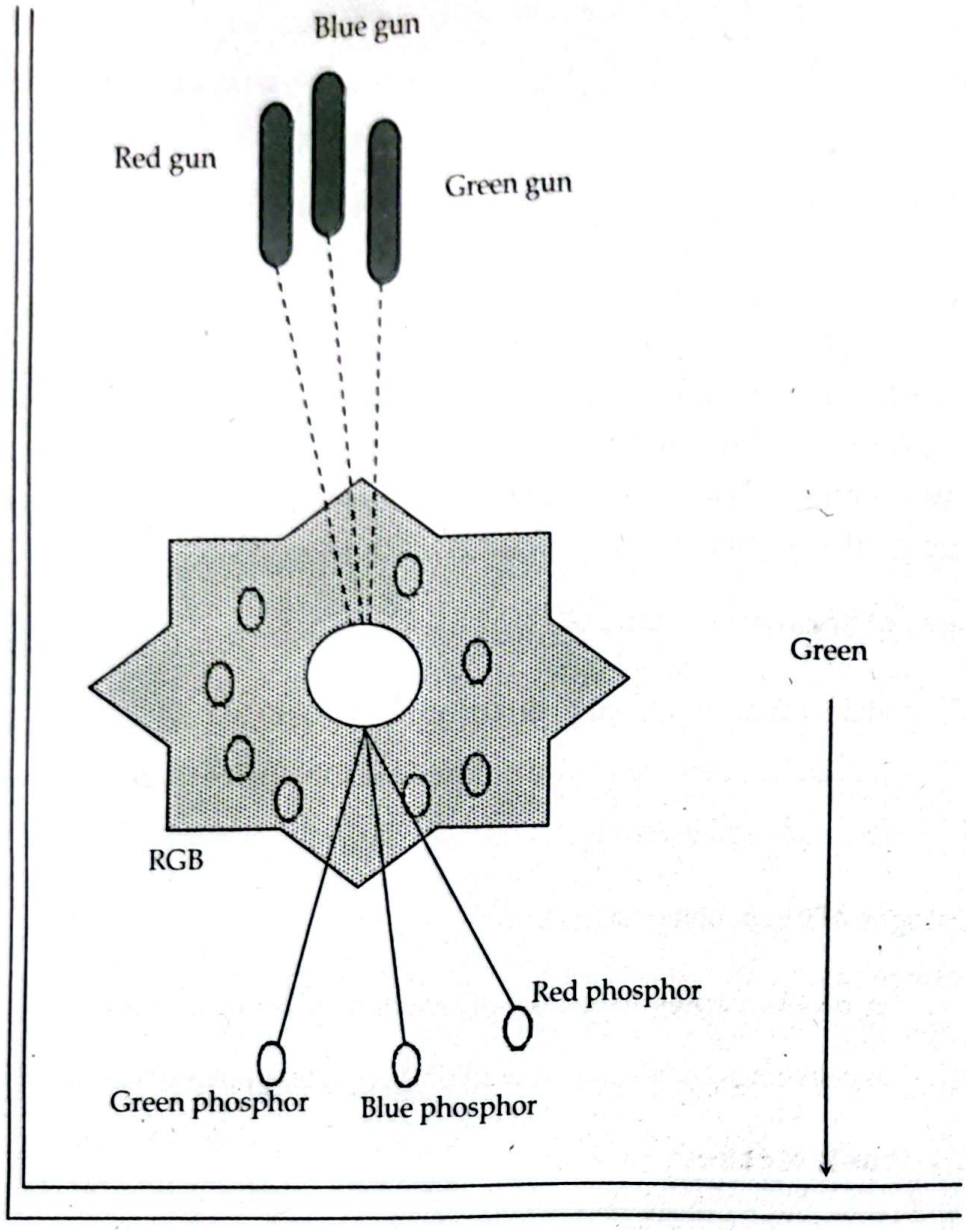


Figure 1.32: Shadow Mask Method

Advantages of shadow mask method

- Produces a much wider range of colors than the beam penetration method.
- An in-line arrangement of RGB color dots on the screen along one scan line is used in higher resolution color CRTs.
- It produces realistic images

Disadvantages of shadow mask method

- It is very difficult to coverage all three beams on same hole in shadow mask. This is because the screen has a limited area.
- They have poor resolution

Differences between Beam Penetration method and Shadow Mask method

	Beam Penetration method	Shadow Mask method
Where Used	It is used with **Random Scan System** to display color.	It is used with **Raster Scan System** to display color.
Colors	It can display only four colors i.e., Red, Green, Orange and Yellow.	It can display Millions of colors.
Color Dependency	Fewer colors are available because the colors in Beam penetration depend on the speed of the electron beam.	Millions of colors are available because the color in Shadow Mask depends on the type of the ray.
Cost	It is **Less Expensive** as compared to Shadow Mask.	It is **More Expensive** than other methods.
Picture Quality	Quality of picture is not so good i.e., Poor with Beam Penetration Method.	Shadow Mask gives realism in picture with shadow effect and millions of colors.
Resolution	It gives High Resolution.	It gives Low Resolution.
Criteria	In Beam Penetration method, Color display depends on how far electron excites outer Red layer and then Green layer.	In Shadow Mask Method, there are no such criteria for producing colors. It is used in computers, in color TV etc.

Flat Panel Display

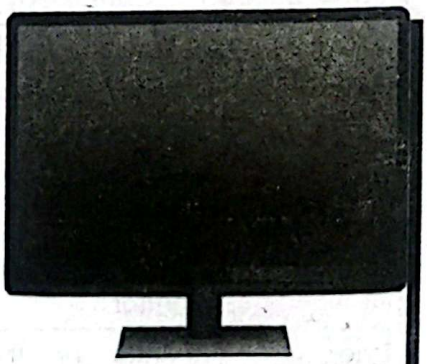


Figure 1.33: Flat Panel Display

A flat-panel display is a thin screen display found on all portable computers and is the new standard for desktop computers. Unlike CRT monitors, flat-panel displays use liquid-crystal display (LCD) or light-emitting diode (LED) technology to make them much lighter and thinner compared to a traditional CRT monitor. Flat panel display refers to a class of video device that have reduced volume, weight and power requirement compared to a CRT. These emerging display technologies tend to replace CRT monitors. Current uses of flat-panel displays include TV monitors, calculators, pocket video games, laptops, displays in airlines and ads etc.

There are two categories of flat-panel displays:

Emissive displays

The emissive displays (or emitters) are devices that displays and light - emitting diodes are examples of emissive displays. Non-emissive displays (or non-emitters) use optical effects to convert sunlight or light from some other source into graphics patterns. Plasma panels, electroluminescent displays and light-emitting diodes (LED) are examples of emission display devices.

Non-emissive displays

Use optical effects to convert sunlight or light from other sources into graphics patterns. Example: liquid-crystal displays (LCD).

Advantages

- Can produce output with high resolutions.
- Better for animation than raster system since only end point information is needed.

Disadvantages

- Cannot fill area with pattern and manipulate bits.
- Refreshing image depends upon its complexity.

Differences between CRT and LCD display

Topic	CRT	LCD
Size	Because of the CRT in a CRT monitor the physical size of these displays is much larger than an LCD and usually awkward on small desks.	LCD monitors are much thinner than CRT monitors, being only a few inches in thickness (some can be near 1 inch thick). They can fit into smaller, tighter spaces, whereas a CRT monitor can't in most cases.
Dead pixels	Although a CRT can have display issues there is no such thing as a dead pixel on a CRT monitor. Many issues can also be fixed by degaussing the monitor.	LCD monitors can encounter dead pixels, which causes small black or other colored dots in the display.
Weight	A CRT monitor can weigh 40 pounds or more depending on the size of the monitor.	LCD monitors can be pretty light, weighing as little as 8 to 10 pounds.
Price	Because of the popularity of LCD monitors the price of most CRT monitors is very cheap. You can also usually pick up a used CRT for next to nothing.	LCD monitors are a newer technology and have more demand so will be more expensive than a CRT.
Viewable area	The frame around the glass screen of the monitor causes the viewable area of the screen to be smaller than an LCD.	LCD monitors have a slightly bigger viewable area than a CRT monitor. A 19" LCD monitor has a diagonal screen size of 19" and a 19" CRT monitor has a diagonal screens size of about 18".
Picture	Because of the older technology most CRT monitors will not have as good as quality as picture as most LCD displays.	Depending on the quality of the LCD monitor, the picture quality can be quite superb and amazing, almost like looking out a window.
Viewing angle	Almost every CRT have a better viewing angle than many LCD displays.	Not all LCD monitors can be viewed at every angle, which makes it difficult for anyone who is not in front of the monitor to see the screen.

Topic	CRT	LCD
Glare	Most monitors have a glass screen, which can cause much more glare than an LCD.	An LCD monitor doesn't have a glass screen, virtually eliminating any glare.
Burn	If the same image is left on a CRT for days the image can burn into the display causing a permanent ghost image to appear on the screen.	Unlike a CRT an LCD monitor does is not affected by a burn in or ghosting problem.
Flicker	Some people can see the flicker in a CRT monitor, which is due in part to the refresh rate of the screen refreshing the image.	On an LCD monitor, screen flicker is very minimal if noticeable at all because of the higher refresh rate.
Response	The response on early LCD monitors was not as good as most monitors at the time, making games and movies not as enjoyable to play.	Any LCD made within the last 5 years has no refresh rate issues. Making the display just as enjoyable as CRT monitors.
Power	A 17" CRT monitor will use as much as 80 watts, depending on the age.	LCD monitors are very energy efficient. A 19" LCD monitor only uses about 17 to 31 watts on average.

GRAPHICS SOFTWARE

Interactive graphics allow users to make change over the displayed objects. Several graphics software packages are now available. There are two general classifications for graphics software:

- General programming packages
- Special-purpose application packages

General programming packages

It contains graphics functions that can be used with high level programming languages such as C, FORTRAN, JAVA etc. **Example**: OpenGL (Graphics library). A general-purpose graphics package provides users with a variety of functions for creating and manipulating pictures. These graphic functions include tools for generating picture components, setting color, selecting views, and applying transformations.

Special-purpose application packages

It is specifically designed for particular applications. Maya, CINEMA 4D are particularly used for animations, different types of CAD applications are designed for medical and business purposes. These are primarily oriented to non-programmers.

SOFTWARE STANDARDS

Many problems have been encountered due to the plate-form dependency of the graph software. So, the primary goal of standardized graphics software is portability. When graphi packages are designed with standard graphics functions (set of specifications that independent of any programming languages), software can be easily moved from one H/ system to another. And it can be used in different implementations and applications.

a. **Graphical Kernel System (GKS)**

GKS was the first graphics software standard adopted by the internation standards organization (ISO). It was originally designed as a 2-dimension graphics package. GKS supports the grouping of logically related primitives suc as; lines, polygons, character strings.

b. **Programmer's Hierarchical Interactive Graphics System (PHIGS)**

It is an extension of GKS. Increased capabilities in object modeling, colo specifications, surface rendering, and picture manipulations are provided i PHIGS. PHIGS include all primitives supported by GKS; in addition, it als includes geometric transformations (Scaling, Translation, and Rotation).

c. **PHIGS+**

It is extension of earlier PHIGS. 3D surface shading capabilities are added to th PHIGS.

SCAN CONVERSION ALGORITHM

This chapter introduces basic concepts that are required for the understanding of two dimensional graphics. Almost all output devices for graphics like computer monitors o printers are pixel-oriented. Therefore, it is crucial to distinguish between the representation c images on these devices and the model of the image itself which is usually not pixel-oriente but defined as scalable vector graphics, i.e., floating point values are used for coordinates.

The basic building blocks for pictures are referred to as output primitives. They includ character strings, and geometric entities such as points, straight lines, curved lines, polygon circles etc. Points and straight-line segments are the most basic components of a picture. picture can be displayed in several ways. Assuming we have raster display, a picture completely specified by the set of intensities for the pixel positions in the display. The process which the object is represented as the collection of discrete pixels is called scan conversion. the other extreme, we can describe a picture as a set of geometrical objects, such as lin triangles, polygons etc. Scan lines are numbered consecutively from 0, starting at the bottom the screen; and pixel columns are numbered from 0, from left to right across each scan line.

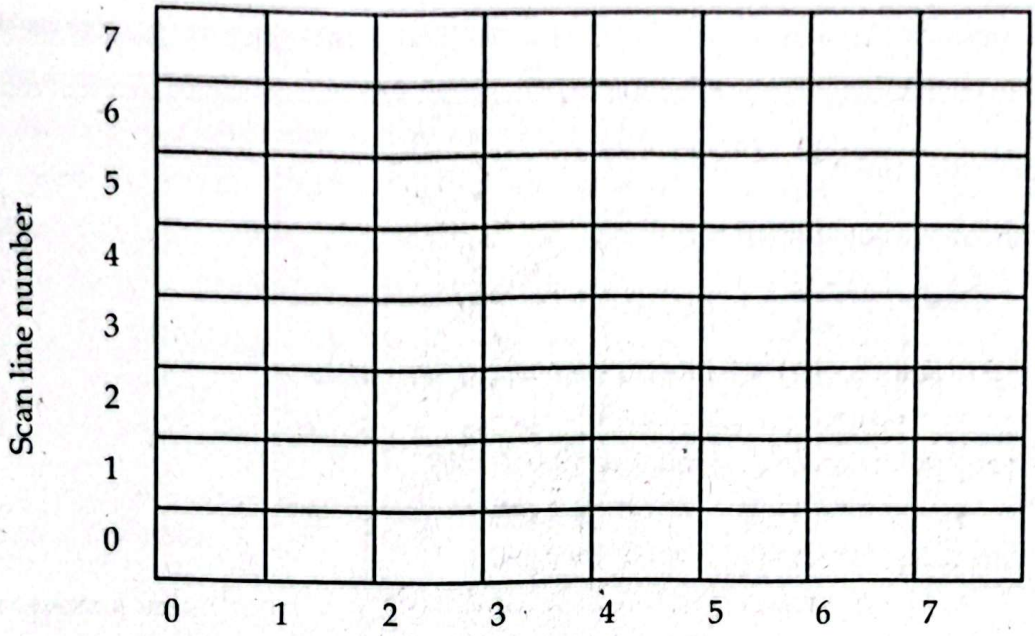


Figure 1.34: Pixel column number

LINE DRAWING ALGORITHM

A line drawing algorithm is a graphical algorithm for approximating a line segment on discrete graphical media. On discrete media, such as pixel-based displays and printers, line drawing requires such an approximation (in nontrivial cases). Basic algorithms rasterizing lines in one color. There are mainly three most widely used line drawing algorithms:

1. Direct use of Line Equation
2. Digital Differential Analyzer Algorithm (DDA)
3. Bresenham Line Drawing Algorithm (BSA)

Direct use of Line Equation

In this method simply calculate the corresponding y coordinate for each unit x coordinate. The slope-intercept equation of a straight line is:

$$y = mx + b \quad \ldots\ldots\ldots\ldots \text{(i)}$$

Where, m = slope of line and, b = y-intercept.

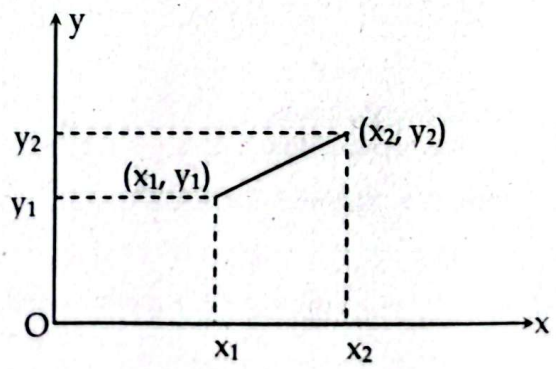


Figure 1.35: Line path between two end points (x_1, y_1), and (x_2, y_2)

Suppose (x_1, y_1) and (x_2, y_2) and are the two end points of a line segment as shown above.

Then slope $(m) = \frac{y_2 - y_1}{x_2 - x_1}$

From above equation (i) becomes,

$$\therefore \quad b = y - \frac{y_2 - y_1}{x_2 - x_1} x \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(ii)}$$

For any point (x_k, y_k) equation (i) becomes,

$$y_k = mx_k + b \quad \ldots\ldots\ldots\ldots \text{(iii)}$$

For any point (x_{k+1}, y_{k+1}) equation (i) becomes,

$$y_{k+1} = mx_{k+1} + b \quad \ldots\ldots\ldots\ldots \text{(iv)}$$

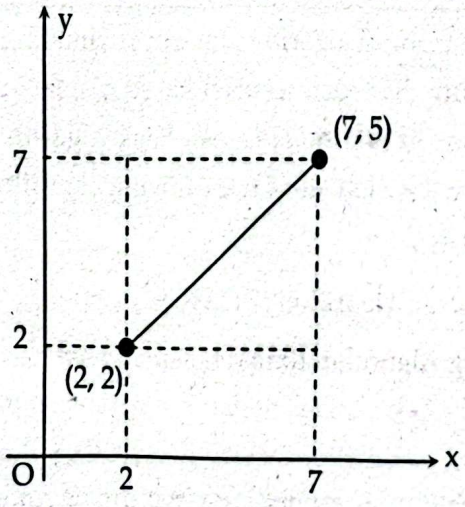
Subtracting equation (iii) from (iv) we get,

$$y_{k+1} - y_k = m(x_{k+1} - x_k) \quad \ldots\ldots\ldots\ldots \text{(v)}$$

So, $\Delta y = m.\Delta x$, where Δx and Δy are x and y-increment respectively.

$$m = \Delta y / \Delta x \quad \ldots\ldots\ldots\ldots \text{(vi)}$$

Example 1.10: Let's consider the following example:



First workout m and b as,

$$m = \frac{5-2}{7-2} = \frac{3}{5}$$

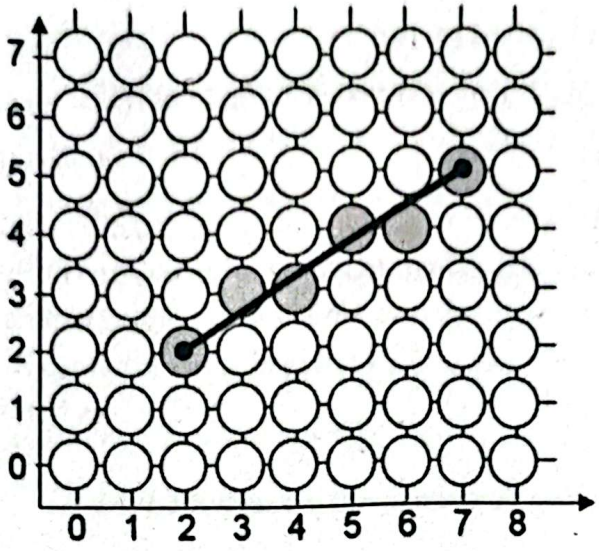
$$b = 2 - \frac{3}{2} \times 2 = \frac{4}{5}$$

Now for each x-values workout the y values,

$$y(3) = \frac{3}{5} \times 3 + \frac{4}{5} = \frac{13}{5} \approx 3 \qquad y(4) = \frac{3}{5} \times 4 + \frac{16}{5} \approx 3$$

$$y(5) = \frac{3}{5} \times 5 + \frac{4}{5} = \frac{19}{5} \approx 4 \qquad y(6) = \frac{3}{5} \times 6 + \frac{4}{5} = \frac{22}{5} \approx 4$$

Thus, we have co-ordinates: (2, 2), (3, 3), (4, 3), (5, 4), (6, 4), (7, 5)

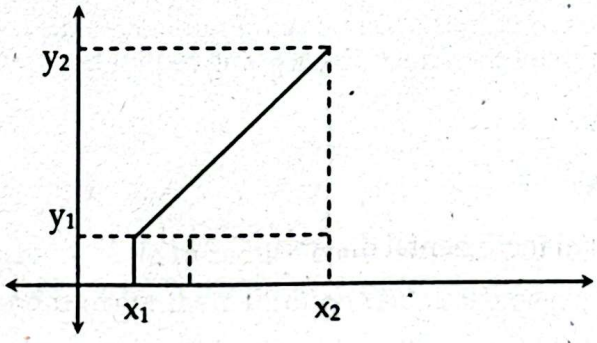


DDA line drawing Algorithm

The digital differential analyzer (DDA) is a scan conversion line drawing algorithm based on calculating either Δx or Δy form the equation,

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

We sample the line at unit intervals in one co-ordinate and determine the corresponding integer values nearest to the line path for the other co-ordinates. Here following cases occur:



$|m|>1$

Case 1: If m is positive (m>0) and $|m| <= 1$,

Let (x_k, y_k) be the plotted pixel then next pixels going to plot be,

$x_{k+1} = x_k + \Delta x$

$y_{k+1} = y_k + \Delta y$ Where, Δx and Δy are small incremental distance.

We have $m = \frac{\Delta y}{\Delta x}$

$\Delta y = m\Delta x$..... (1)

Also, here $|m| <= 1$, sampling be done along x-axis with Δx=1.

Then,

$x_{k+1} = x_k + 1$

$y_{k+1} = y_k + m$

Here *k* takes value from starting point and increase by 1 until final end point. *m* can be any real value between 0 and 1.

Case 2: If m is positive (m > 0) and |m|>=1,

Let (x_k, y_k) be the plotted pixel then next pixels going to plot be,

$$x_{k+1} = x_k + \Delta x$$

$$y_{k+1} = y_k + \Delta y$$

Where Δx and Δy are small incremental distance

We have $m = \frac{\Delta y}{\Delta x}$

$$\Delta y = m\Delta x \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Also, here |m|> =1, sampling be done along y-axis with $\Delta y=1$.

Then, $\Delta x=1/m$

$$x_{k+1} = x_k + \frac{1}{m}$$

$$y_{k+1} = y_k + 1$$

The above equations are under the assumption that the lines are processed from left to right i.e., left end point is starting.

If the processing is from right to left

Case 3: if |m|<=1

Let (x_k, y_k) be the plotted pixel then next pixels going to plot be,

$$x_{k+1} = x_k + \Delta x$$

$$y_{k+1} = y_k + \Delta y$$

Where Δx and Δy are small incremental distance

We have $m = \frac{\Delta y}{\Delta x}$

$$\Delta y = m\Delta x \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Also, here |m|<=1, sampling be done along x-axis with $\Delta x = -1$.

Then,

$$x_{k+1} = x_k - 1$$

$$y_{k+1} = y_k - m$$

Case 4: If |m|>1

Let (x_k, y_k) be the plotted pixel then next pixels going to plot be,

$$x_{k+1} = x_k + \Delta x$$

$$y_{k+1} = y_k + \Delta y$$

Where Δx and Δy are small incremental distance

We have $m = \frac{\Delta y}{\Delta x}$

$$\Delta y = m\Delta x \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Also, here $|m| >= 1$, sampling be done along y-axis with $\Delta y = -1$.

Then,

$$x_{k+1} = x_k - \frac{1}{m}$$

$$y_{k+1} = y_k - 1$$

Therefore, in general,

$$\therefore \quad y_{k+1} = y_k \pm m$$

For $|m| < 1$ and

$$x_{k+1} = x_k \pm \frac{1}{m}$$

For $|m| > 1$

Algorithm for DDA line drawing Algorithm

Step 1: Start

Step 2: Input the line endpoints and store the left endpoint in (x_1, y_1) and right end point in (x_2, y_2).

Step 3: Calculate the values of dx and dy,

dx = $x_2 - x_1$

dy = $y_2 - y_1$.

Step 4: if (abs(dx) > abs(dy)) then

steplength = abs(dx)

else

steplength= abs(dy)

Step 5: C alculate the values of x-increment and y-increment as,

xIncrement = dx/ steplength

yIncrment = dy/ steplength

Step 6: set $x=x_1$ and $y=y_1$;

Step 7: draw the pixel (round(x), round(y));

Step 8: for k=0 to steplength do

x = x + xIncrement

y = y + yIncrement

Draw the pixel (round(x), round(y));

Step 9: End

Advantages of DDA line drawing algorithm

- It is the simplest method, involves only integer additions and it does not require special skills for implementation.
- It is a faster method for calculating pixel positions than the direct use of equation $y = mx + b$.
- It eliminates the multiplication in the equation by making use of raster characteristics, so that appropriate increments are applied in the x or y direction to find the pixel positions along the line path.

Disadvantages of DDA line drawing algorithm

- Floating point arithmetic in DDA algorithm is still time consuming.
- The algorithm is orientation dependent. Hence end point accuracy is poor.
- Although DDA is fast, the accumulation of round-off error in successive additions of floating-point increment however can cause the calculation pixel position to drift away from the true line path for long line segment.
- Rounding-off in DDA is time consuming.

Example 1.11: Digitize the line with endpoints (1, 5) and (7, 2) using DDA algorithm

Solution:

Here, dx = 7-1=6 and dy = 2-5 = -3

So, steplength = 6 (since abs (dx)>abs (dy))

$$\text{Therefore, xIncrement} = \frac{dx}{\text{steplength}}$$

$$= \frac{6}{6} = 1$$

$$\text{yIncrement} = \frac{dy}{\text{steplength}}$$

$$= \frac{-3}{6} = \frac{-1}{2} = -0.5$$

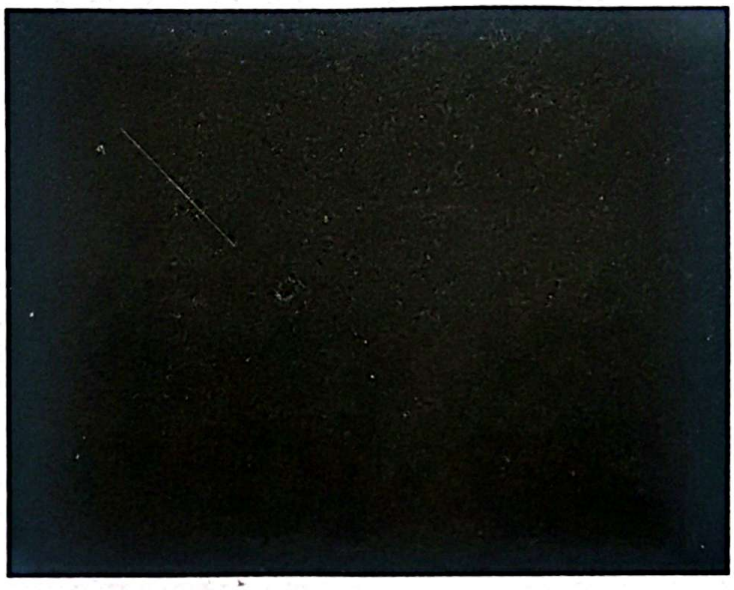
Based on these values the intermediate pixel calculation is shown in the table below.

k	x_{k+1}	y_{k+1}	(x_{k+1}, y_{k+1})	Plot in screen (x_{k+1}, y_{k+1})
1	2	4.5	(2, 4.5)	(2, 5)
2	3	4	(3, 4)	(3, 4)
3	4	3.5	(4, 3.5)	(4, 4)
4	5	3	(5, 3)	(5, 3)
5	6	2.5	(6, 2.5)	(6, 3)
6	7	2	(7, 2)	(7, 2)

Lab problem 1.1: First program to draw a line by using build in function line().

```
#include<stdio.h>
#include<graphics.h>//must be included for every graphics program
#include<conio.h>
#include<dos.h> //for including delay function
void main()
{
        int gd=DETECT, gm;
        //gd=detects best available graphics driver, gm =graphics mode
        initgraph(&gd, &gm, "C:\\TurboC3\\BGI");// for initializing graph mode
        /*above 2 steps are must for every graphics program. Declaration of any variables must
        be done before calling initgraph() function. */
        line(100,100,200,200);//draws a line segment.
        getch();
}
```

Output:

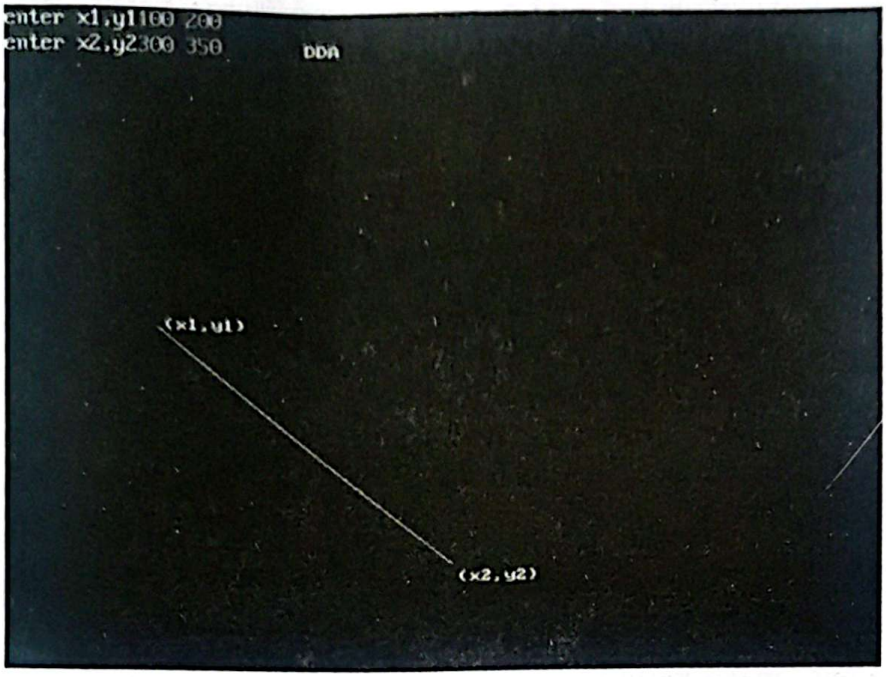


Lab problem 1.2: To implement DDA algorithm for drawing a line segment between two given end points A(x1, x2) and B(x2, y2).

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
float round(float a);
void main()
{
```

```
        int gd=DETECT,gm;
        int x1,y1,x2,y2,steps,k;
        float xincr,yincr,x,y,dx,dy;
        printf("enter x1,y1");
        scanf("%d%d",&x1,&y1);
        printf("enter x2,y2");
        scanf("%d%d",&x2,&y2);
        initgraph(&gd,&gm,"c:\\turboc3\\BGI");
        dy=y2-y1;
        if(abs(dx)>abs(dy))
                steps=abs(dx);
        else
                steps=abs(dy);
        xincr=dx/steps;
        yincr=dy/steps;
        x=x1;
        y=y1;
        for(k=1;k<=steps;k++)
        {
                delay(100);
                y+=yincr;
                putpixel(round(x),round(y),WHITE);
        }
        outtextxy(200,20,"DDA");
        outtextxy(x1+5,y1-5,"(x1,y1)");
        outtextxy(x2+5,y2+5,"(x2,y2)");
        getch();
        closegraph();
}
float round(float a)
{
        int b=a+0.5;
        return b;
}
```

Output



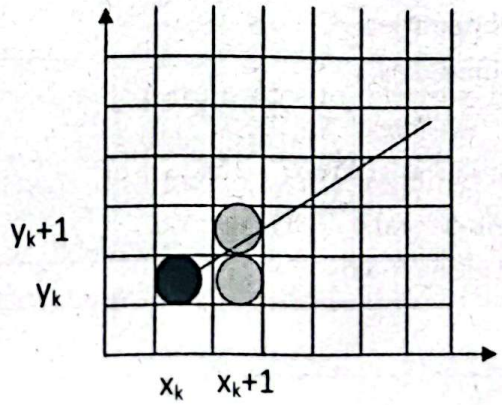
Bresenham's Line Drawing Algorithm (BSA)

The Bresenham's algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates. For example, as shown in the following illustration, from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.DDA algorithm follow the floating-point calculation such that calculated pixels can be drift from actual line path due to accumulation of floating-point error. Hence an integer calculation be made by using a parameter called decision parameter (P-value) that is use to decide the pixel position for next plot and such a method is called Bresenham's line drawing algorithm.

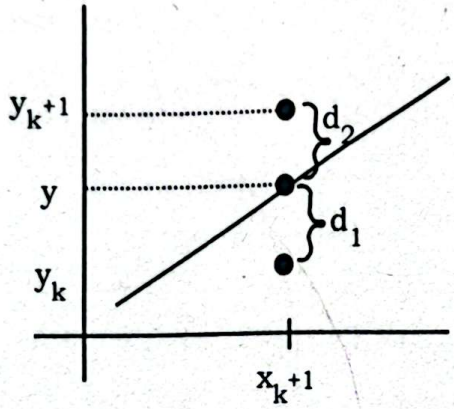
Derivation of P- value for positive slope less than 1 (0<m<1)

Since m<1 hence pixel position along the line path is determined by sampling at unit x intervals. Starting from left end point, we step to each successive column and plot the pixel closest to line path.

Assume that (x_k, y_k) is pixel at k^{th} step then next point to plot may be either $(x_k + 1, y_k)$ or $(x_k + 1), y_k + 1)$.



At sampling position $x_k + 1$, we label vertical pixel separation from line path as d_1 & d_2 as in figure below.



The y-coordinate on the mathematical line path at pixel column x_k+1 is

$$y = m(x_{k+1}) + b$$

Then $d_1 = y - y_k$

$$= m(x_k + 1) + b - y_k$$

$$= \frac{\Delta y}{\Delta x}(x_k + 1) + b - y_k$$

Also, $d_2 = (y_k + 1) - y$

$$= (y_k + 1) - m(x_k + 1) - b = (y_k + 1) - \frac{\Delta y}{\Delta x}(x_k + 1) - b$$

Now, $d_1 - d_2 = \frac{\Delta y}{\Delta x}(x_k + 1) + b - y_k - [(y_k + 1) - \frac{\Delta y}{\Delta x}(x_k + 1) - b]$

$$= 2\frac{\Delta y}{\Delta x}(x_k + 1) - (y_k + 1) - y_k + 2b$$

or, $\Delta x(d_1 - d_2) = 2\Delta y x_k + 2\Delta y - \Delta x y_k - \Delta x - \Delta x y_k + 2b\Delta x$

$$= 2\Delta y.x_k - 2\Delta x.y_k + 2\Delta y + 2b\Delta x - \Delta x$$

$$= 2\Delta y.x_k - 2\Delta x.y_k + c$$

Where $c = 2\Delta y + b\Delta x - \Delta x$ is a constant.

Let's say Δx be always positive then we can introduce a parameter for decision parameter p_k for the k^{th} step as,

$$P_k = \Delta x(d_1 - d_2)$$

$$P_k = 2\Delta y.x_k - 2\Delta x.y_k + c$$

At step k+1, P_{k+1} is evaluated as,

$$P_{k+1} = 2\Delta y.x_{k+1} - 2\Delta x y_{k+1} + c$$

Now, $P_{k+1} - P_k = 2\Delta y.x_{k+1} + c - (2\Delta y.x_k - 2\Delta x.y_k + c)$

$\Rightarrow$ $P_{k+1} - P_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$

Since sapling is done along x-axis

So, $x_{k+1} = x_k + 1$

$\therefore$ $P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$

The term $y_{k+1} - y_k$ is either 0 or 1 depending upon the sign of P_k.

If $P_k \leq 0$ then $y_{k+1} = y_k$

$\therefore \quad P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_k - y_k)$

or, $\quad P_{k+1} = P_k + 2\Delta y$

If $P_k > 0$ then $y_{k+1} = y_k + 1$

$\therefore \quad P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_k + 1 - y_k)$

$\quad P_{k+1} = P_k + 2\Delta y - 2\Delta x$

Derivation for initial decision parameter

We know, $P_k = 2\Delta y.x_k - 2\Delta x.y_k + c$

$= 2\Delta y.x_k - 2\Delta x.y_k + 2\Delta y + 2b\Delta x - \Delta x$

When k = 0,

$P_0 = 2\Delta y.x_0 - 2\Delta x.y_0 + 2\Delta y + 2b\Delta x - \Delta x$

Since y =m.x + b

$y_0 = mx_0 + b$

or, $\quad y_0 = \frac{\Delta y}{\Delta x} x_0 + b$

So $\quad P_0 = 2\Delta y.x_0 - 2\Delta x\left(\frac{\Delta y x_0 + \Delta x b}{\Delta x}\right) + 2\Delta y + 2b\Delta x - \Delta x$

$\Rightarrow \quad P_0 = 2\Delta y - \Delta x$

BSA quick summary

1. **For |m| < 1 i.e., |dy| < |dx|**

X_{k+1}=Increment or decrement by 1

Y_{k+1}= According to decision parameter (P_k) looped till |dx|.

$P_0 = 2 \times |dy| - |dx|$

Case	Conditions		X_{k+1}	If $P_k < 0$ $P_{k+1} = P_k + 2 \times \|dy\|$	Else ($P_k >= 0$) $P_{k+1} = P_k + 2 \times \|dy\| - 2 \times \|dx\|$	Example Coordinate
	dx	dy		Y_{k+1}	Y_{k+1}	
I	Postive (+ve)	Positive (+ve)	$X_k + 1$	Y_k	$Y_k + 1$	(10, 10) and (18, 17)
II		Negative (-ve)			$Y_k - 1$	(10, 10) and (14, 7)
III	Negative (+ve)	Positive (+ve)	$X_k - 1$		$Y_k + 1$	(10, 10) and (6, 13)
IV		Negative (-ve)			$Y_k - 1$	(10, 10) and (6, 7)

2. **For |m| > 1 i.e., |dy| > |dx|**

X_{k+1} = According to decision parameter (P_k) looped till |dy|.

Y_{k+1} = Increment or decrement by 1

$P_0 = 2 \times |dx| - |dy|$

Case	Conditions		If $P_k < 0$ $P_{k+1} = P_k + 2 \times \lvert dx \rvert$	Else ($P_k >= 0$) $P_{k+1} = P_k + 2 \times \lvert dy \rvert - 2 \times \lvert dx \rvert$	Y_{k+1}	Example Coordinate
	dx	dy	X_{k+1}	X_{k+1}		
I	Positive (+ve)	Postive (+ve)	X_k	$X_k + 1$	$Y_k + 1$	(10, 10) and (16, 17)
II	Negative (-ve)			$X_k - 1$		(10, 10) and (4, 17)
III	Positive (+ve)	Negative (+ve)		$X_k + 1$	$Y_k - 1$	(6, 12) and (10, 5)
IV	Negative (-ve)			$X_k - 1$		(10, 10) and (7, 5)

Advantages of Bresenham's Line Drawing Algorithm

- Involves simple integer calculations, so does not need to perform round off operation.
- Bresenham's line algorithm is a highly efficient incremental method over DDA
- It can be used to generate circles and other curves.
- It produces mathematically accurate results using only integer addition, subtraction, and multiplication by 2, which can be accomplished by a simple arithmetic shift operation.

Disadvantages of Bresenham's Line Drawing Algorithm

- Additional parameter i.e., decision parameter must be calculated at each step

Algorithm of Bresenham's Line Drawing Algorithm

Step 1: Start

Step 2: Input the two-line endpoint and store the left endpoint at (x_0, y_0) and right endpoint at (x_1, y_1).

Step 3: calculate $\Delta x = abs(x_1 - x_0)$, $\Delta y = abs(y_1 - y_0)$

Step 4: calculate initial decision parameter as,

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

Step 4: if $x_1 > x_0$ then

Set $x = x_0$

Set $y = y_0$

Set $xend = x_1$;

Otherwise

Set $x = x_1$

Set $y = y_1$

Set $xend = x_0$;

Step 5: draw pixel at (x, y)

Step 6: while(x<xend)

x ++;

if P<0 then

P=P+2Δy

Otherwise

y++

P=P+2Δy-2Δx

Draw pixel at (x, y)

Step 7: Stop

Example 1.12: Digitize a line with end points (10, 15) and (15, 18) using BSA.

Solution:

Here, Δy = |18 - 15| = 3

Δx = |15-10| = 5

Since, |m| = Δy/Δx ≤ 1, so we sample at x direction i.e., at each step we simply increment x-coordinate by 1 and find appropriate y-coordinate.

First pixel to plot is (10, 15), which is the starting endpoint.

Now, initial decision parameter, $P_0 = 2\Delta y - \Delta x = 2 \times 3 - 5 = 1$

We know that, if $P_k < 0$, then we need to set $P_{k+1} = P_k + 2\Delta y$ and plot pixel (x_k+1, y_k)

And if $P_k \geq 0$, then we need to set $P_{k+1} = P_k + 2\Delta y - 2\Delta x$ then plot pixel (x_k+1, y_k+1)

Here, $P_0 = 1$, we have i.e., $P_k \geq 0$

So, $P_1 = P_0 + 2\Delta y - 2\Delta x = 1 + 2 \times 3 - 2 \times 5 = -3$

$P_2 = P_1 + 2\Delta y = -3 + 2 \times 3 = 3.$

$P_3 = P_2 + 2\Delta y - 2 \times \Delta x = 3 + 6 - 10 = -1.$

$P_4 = P_3 + 2\Delta y = -1 + 6 = 5$

Successive pixel positions and decision parameter calculation is shown in the table below.

k	x	y	P_k	(x_{k+1}, y_{k+1})
0	10	15	1(>=0)	(11, 16)
1	11	16	-3(<0)	(12, 16)
2	12	16	3	(13, 17)
3	13	17	-1	(14, 17)
4	14	17	5	(15, 18)

Example 1.13: Go through the steps of the Bresenham's line drawing algorithm for a line going from (20,10) to (30,18).

Solution

Here, Δy = |18-10| = 8

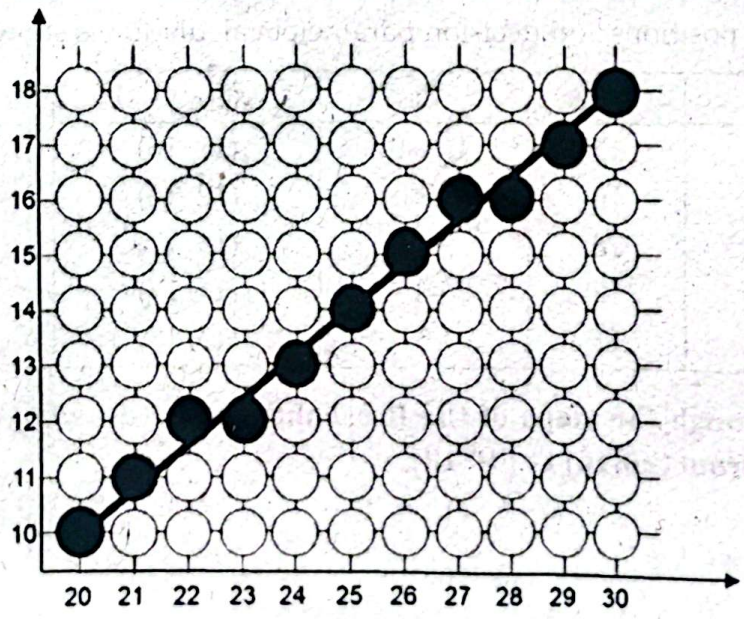
Δx = |30 - 20| = 10

Since, $|m| = \Delta y/\Delta x \leq 1$, so we sample at x direction i.e., at each step we simply increment x-coordinate by 1 and find appropriate y-coordinate.
First pixel to plot is (20, 10), which is the starting endpoint.
Now, initial decision parameter, $P_0 = 2\Delta y - \Delta x = 2 \times 8 - 10 = 6$
We know that, If $P_k < 0$, then we need to set $P_{k+1} = P_k + 2\Delta y$ and plot pixel (x_k+1, y_k)
And if $P_k \geq 0$, then we need to set $P_{k+1} = P_k + 2\Delta y - 2\Delta x$ then plot pixel (x_k+1, y_k+1)
Here, $P_0 = 6$, we have i.e., $P_k \geq 0$

So, $P_1 = P_0 + 2\Delta y - 2\Delta x = 6 + 2 \times 8 - 2 \times 10 = 2$
$P_2 = P_1 + 2\Delta y - 2\Delta x = 2 + 2 \times 8 - 2 \times 10 = -2.$
$P_3 = P_2 + 2\Delta y = -2 + 2 \times 8 = 14.$
$P_4 = P_3 + 2\Delta y - 2\Delta x = 14 + 2 \times 8 - 2 \times 10 = 10$
$P_5 = P_4 + 2\Delta y - 2\Delta x = 10 + 2 \times 8 - 2 \times 10 = 6$
$P_6 = P_5 + 2\Delta y - 2\Delta x = 6 + 2 \times 8 - 2 \times 10 = 2$
$P_7 = P_6 + 2\Delta y - 2\Delta x = 2 + 2 \times 8 - 2 \times 10 = -2$
$P_8 = P_7 + 2\Delta y = -2 + 2 \times 8 = 14$
$P_9 = P_8 + 2\Delta y - 2\Delta x = 14 + 2 \times 8 - 2 \times 10 = 10$

Successive pixel positions and decision parameter calculation is shown in the table below.

k	x	y	P_k	(x_{k+1}, y_{k+1})
0	20	10	6(>=0)	(21, 11)
1	21	11	2(>=0)	(22, 12)
2	22	12	-2(<0)	(23, 12)
3	23	12	14(>=0)	(24, 13)
4	24	13	10(>=0)	(25, 14)
5	25	14	6(>=0)	(26, 15)
6	26	15	2(>=0)	(27, 16)
7	27	16	-2(<0)	(28, 16)
8	28	16	14(>=0)	(29, 17)
9	29	17	10(>=0)	(30, 18)



Difference between DDA line drawing algorithm and Bresenham's line drawing algorithm

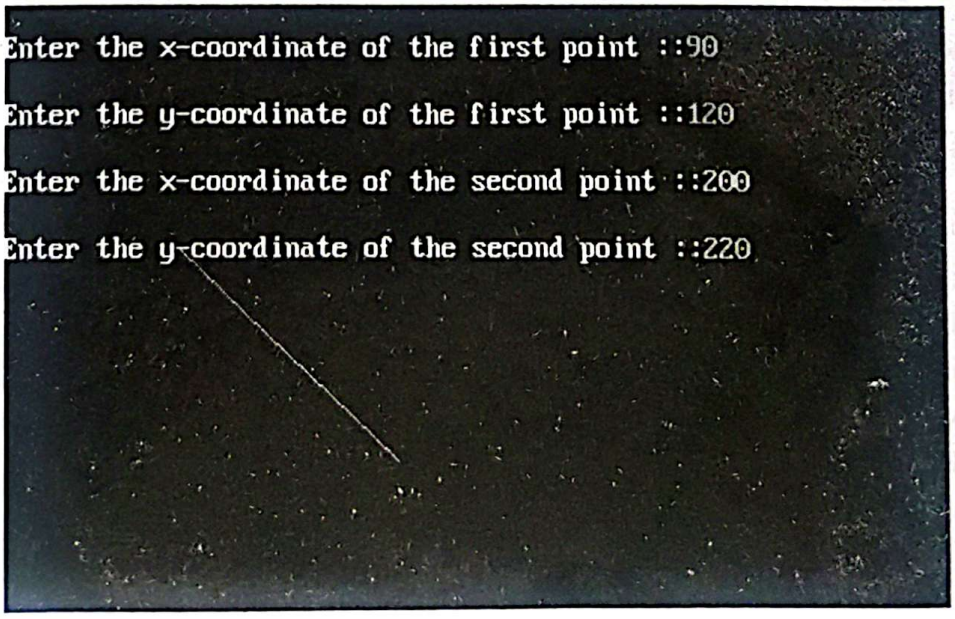
	DDA Line Drawing Algorithm	Bresenham's Line Drawing Algorithm
Arithmetic	DDA algorithm uses floating points i.e., Real Arithmetic.	Bresenham's algorithm uses fixed points i.e., Integer Arithmetic.
Operations	DDA algorithm uses multiplication and division in its operations.	Bresenham's algorithm uses only subtraction and addition in its operations.
Speed	DDA algorithm is rather slowly than Bresenham's algorithm in line drawing because it uses real arithmetic (floating-point operations).	Bresenham's algorithm is faster than DDA algorithm in line drawing because it performs only addition and subtraction in its calculation and uses only integer arithmetic so it runs significantly faster.
Accuracy & Efficiency	DDA algorithm is not as accurate and efficient as Bresenham's algorithm.	Bresenham's algorithm is more efficient and much accurate than DDA algorithm.
Drawing	DDA algorithm can draw circles and curves but that are not as accurate as Bresenham's algorithm.	Bresenham's algorithm can draw circles and curves with much more accuracy than DDA algorithm.
Round Off	DDA algorithm round off the coordinates to integer that is nearest to the line.	Bresenham's algorithm does not round off but takes the incremental value in its operation.
Expensive	DDA algorithm uses an enormous number of floating-point multiplications so it is expensive.	Bresenham's algorithm is less expensive than DDA algorithm as it uses only addition and subtraction.

Lab problem 1.3: To implement Bresenham's line drawing algorithm for drawing a line segment between two given endpoints A(x1, y2) and B(x2, y2).

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
    int x,y,x1,y1,x2,y2,p,dx,dy;
    int gd=DETECT,gm;
    initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
    printf("\nEnter the x-coordinate of the first point ::");
    scanf("%d",&x1);
    printf("\n Enter the y-coordinate of the first point ::");
    scanf("%d",&y1);
```

```
printf("\n Enter the x-coordinate of the second point ::");
scanf("%d",&x2);
printf("\n Enter the y-coordinate of the second point ::");
scanf("%d", &y2);
x=x1;
y=y1;
dx=x2-x1;
dy=y2-y1;
putpixel(x,y,2);
p=(2*dy-dx);
while(x<=x2)
{
  if(p<0)
  {
      x=x+1;
      p=p+2*dy;
  }
  else
  {
          x=x+1;
          y=y+1;
          p=p+(2*dy)-(2*dx);
  }
  putpixel(x,y,7);
}
getch();
closegraph();
}
```

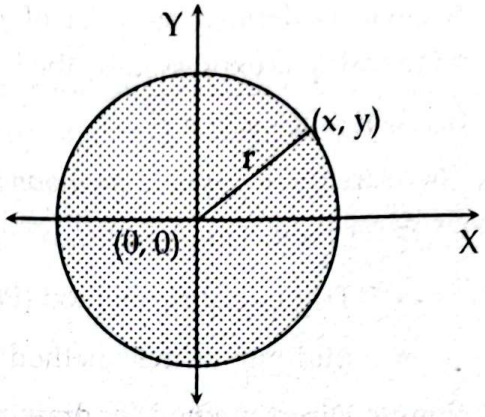
Output:



CIRCLE DRAWING ALGORITHM

A circle is defined as a set of points that are all at a given distance r from the center position (x_c, y_c). The general circle equation can be written as;

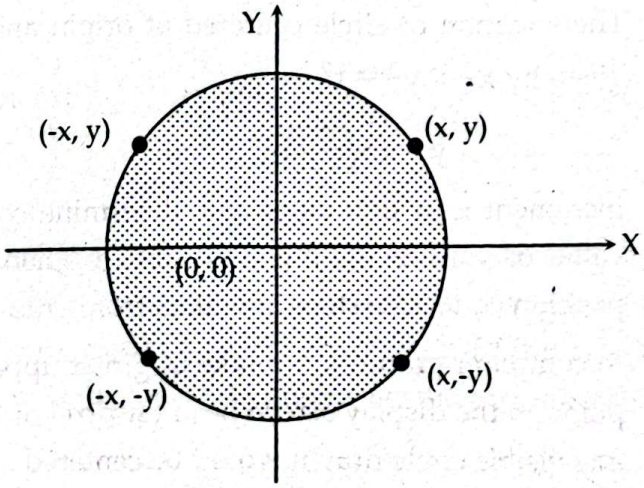
$$(x - x_c)^2 + (y - y_c)^2 = r^2.$$



Properties of circle

Symmetry in quadrants

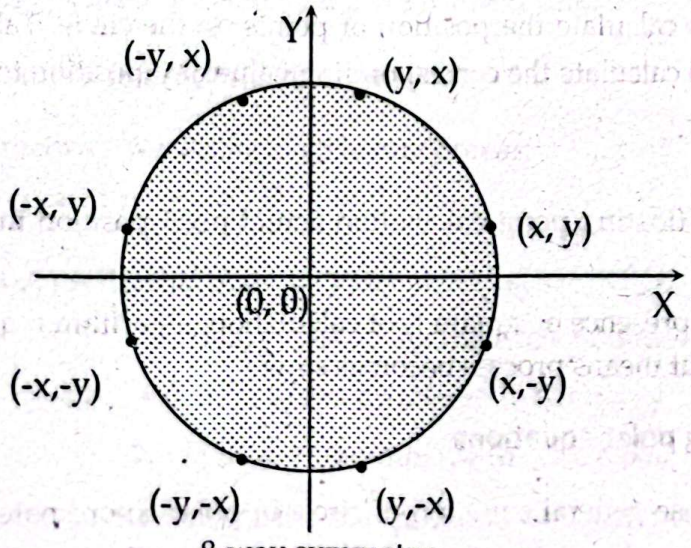
The shape of the circle is similar in each quadrant. Thus, by calculating points in one quadrant we can calculate points in other three quadrants.



4-way symmetry

Symmetry in octants

The shape of the circle is similar in each octant. Thus, by calculating points in one octant we can calculate points in other seven octants. If the point (x, y) is on the circle, then we can trivially compute seven other points on the circle. Therefore, we need to compute only one 45° segment to determine the circle, as shown in figure below. By taking advantage of circle symmetry in octants, we can generate all pixel positions around a circle by calculating only the points within the sector from x = 0 to y = x.



8-way symmetry

METHODS TO DRAW CIRCLE IN COMPUTER GRAPHICS

A circle is defined as a set of points that are all the given distance (x_c, y_c). This distance relationship is expressed by the Pythagorean Theorem in Cartesian coordinates as,

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

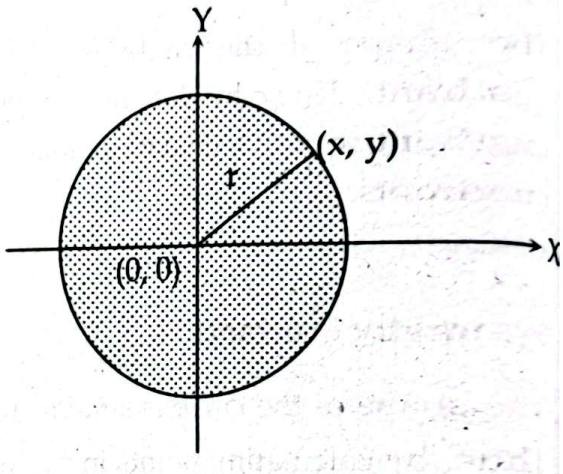
There are three popular methods for drawing circle in computer graphics,

- Simple Direct method
- Trigonometric method (Polar form)
- Mid-point Circle method

Simple Direct method for drawing circle

The equation of circle centered at origin and radius r is given by $x^2 + y^2 = r^2$

$$\Rightarrow \quad y = \pm\sqrt{r^2 - x^2}$$



Increment x in unit steps and determine corresponding value of y from the equation above. Then set pixel at position *(x, y)*. The steps are taken from *–r* to *+r*.

In computer graphics, we take origin at upper left corner point on the display screen i.e., first pixel of the screen so any visible circle drawn would be centered at point other than (0,0).

If center of circle is (x_c, y_c) then the calculated point from origin center should be moved to pixel position by $(x + x_c, y + y_c)$. In general, the equation of circle centered at (x_c, y_c) and radius r is,

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

$$\Rightarrow \quad y = y_c \pm \sqrt{r^2 - (x - x_c)^2} \qquad \ldots\ldots\ldots (i)$$

Use this equation to calculate the position of points on the circle. Take unit step from $x_c - r$ to $x_c + r$ for x value and calculate the corresponding value of y-position for pixel position (x, y).

Drawbacks

- Due to floating point calculation actual pixel position may be away from circular path.
- Due to presence of square root calculation, algorithm requires more computational time that means process becomes slow.

Drawing circle using polar equations

In this method we use general equation of circle in polar co-ordinate as below. If (x, y) be any point on the circle boundary with center (0, 0) and radius r, then

$x = r \cos\theta$

$y = r \sin\theta$

i.e. $(x, y) = (r \cos\theta, r \sin\theta)$

To draw circle using these co-ordinates approach, just increment angle starting from 0 to 360. Here we compute (x, y) position corresponding to increment angle. Which draws circle centered at origin, but the circle centered at origin is not visible completely on the screen since (0, 0) is the starting pixel of the screen. If center of circle is given by (x_c, y_c) then the pixel position (x, y) on the circle path will be computed as,

$x = x_c + r \cos\theta$

$y = y_c + r \sin\theta$

Drawback

- Follow the floating-point calculation.

Midpoint circle Algorithm

In midpoint circle algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step. For a given radius r, and screen center position (x_c, y_c) we can set up our algorithm to calculate pixel positions around a circle path centered at (0, 0) and then each calculated pixel position (x, y) is moved to its proper position by adding x_c to x and y_c to y.

$x = x + x_c,$

$y = y + y_c.$

It uses the symmetric property of circle to reduce computational time. Pixel positions are only calculated within an octant and transfer them through remaining seven octant to get a complete circle.

Here a decision parameter is used to decide the pixel for next plot.

Derivation of p-value for mid-point circle algorithm

To apply the midpoint method, we define a circle function as:

$$f_{circle} = x^2 + y^2 - r^2$$

To summarize the relative position of point (x, y) by checking sign of f_{circle} function,

$$f_{circle}(x, y) \begin{cases} < 0, \text{ if } (x, y) \text{ lies inside the circle boundary} \\ = 0, \text{ if } (x, y) \text{ lies on the circle boundary} \\ > 0, \text{ if } (x, y) \text{ lies outside the circle boundary.} \end{cases}$$

The circle function tests are performed for the mid positions between pixels near the circle path at each sampling step. Thus, the circle function is decision parameter in mid point algorithm.

Example 1.14: Calculate the points to draw a circle having radius 10 and center at (0, 0).

Solution:

we have r = 10

Centre (h, k) = (0, 0)

Set x = 0, y = r = 10

Now the first pixel to be drawn= (0, 10)

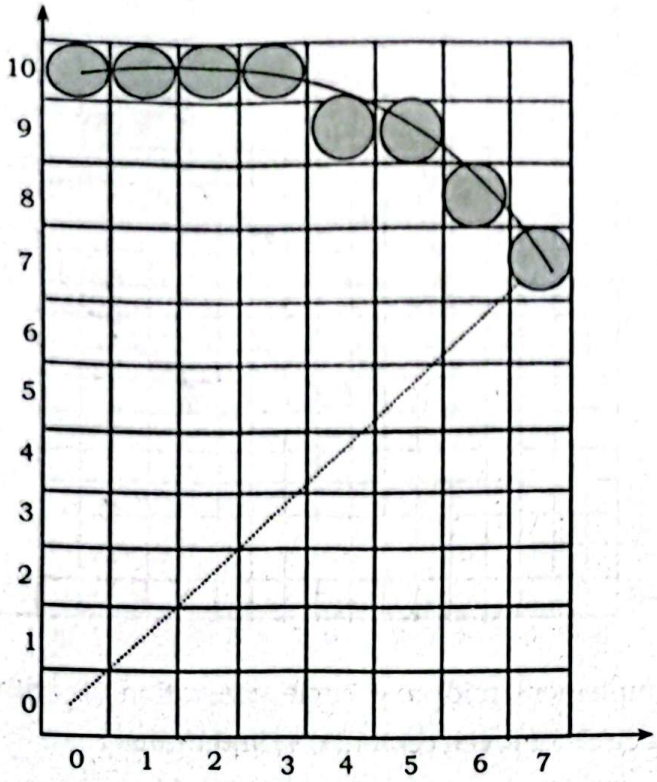
Initial decision parameter (p) = 1 - r = 1 – 10 = -9

Now points on the octant are given by,

K	P_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
1	-9	(1, 10)	2	20
2	-6	(2, 10)	4	20
3	-1	(3, 10)	6	20
4	6	(4, 9)	8	18
5	-3	(5, 9)	10	18
6	8	(6, 8)	12	16
7	5	(7, 7)	14	14

Now using 8 points symmetry we get all the points on the circle as below:

(x, y)	(0, 10)	(1, 10)	(2, 10)	(3, 10)	(4, 9)	(5, 9)	(6, 8)	(7, 7)
(-x, y)	(0, 10)	(-1, 10)	(-2, 10)	(-3, 10)	(-4, 9)	(-5, 9)	(-6, 8)	(-7, 7)
(x, -y)	(0, -10)	(1, -10)	(2, -10)	(3, -10)	(4, -9)	(5, -9)	(6, -8)	(7, -7)
(-x, -y)	(0, -10)	(-1, -10)	(-2, -10)	(-3, -10)	(-4, -9)	(-5, -9)	(-6, -8)	(-7, -7)
(y, x)	(10, 0)	(10, 1)	(10, 2)	(10, 3)	(9, 4)	(9, 5)	(8, 6)	(7, 7)
(-y, x)	(10, 0)	(-10, 1)	(-10, 2)	(-10, 3)	(-9, 4)	(-9, 5)	(-8, 6)	(-7, 7)
(y, -x)	(10, 0)	(10, -1)	(10, -2)	(10, -3)	(9, -4)	(9, -5)	(8, -6)	(7, -7)
(-y, -x)	(10, 0)	(-10, -1)	(-10, -2)	(-10, -3)	(-9, -4)	(-9, -5)	(-8, -6)	(-7, -7)



Example 1.15: Digitize a circle $(x-2)^2 + (y-3)^2 = 25$

Solution:

Step 0: We have center C (h, k) = (2, 3)

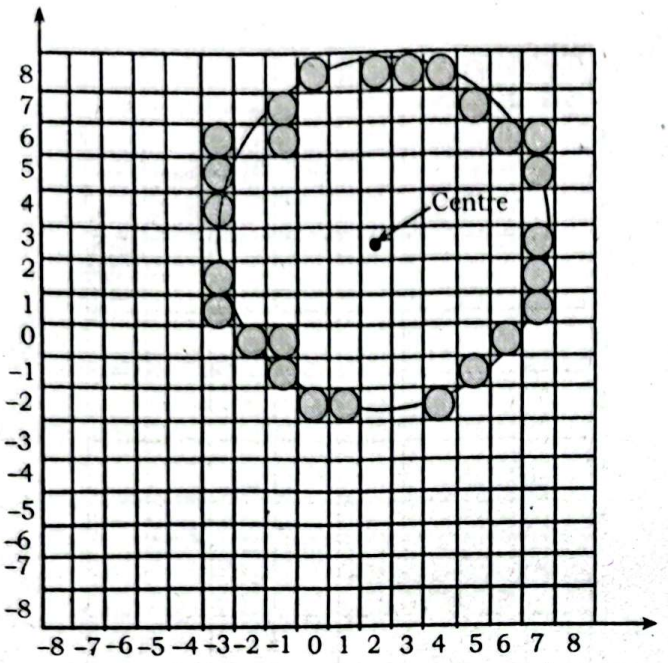
Radius (r) = 5

First pixel= (0, 5)

Initial decision parameter (P_0) = 1 – r = 1 – 5 = – 4

Now points on the octant are given by,

k	P_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$	Actual pixel (x_{k+1}, y_{k+1})
1	-4	(1, 5)	2	10	(3, 8)
2	-1	(2, 5)	4	10	(4, 8)
3	4	(3, 4)	6	8	(5, 7)
4	3	(4, 3)	8	6	(6, 6)
5	6	(5, 2)	10	4	(7, 5)
6	13	(6, 1)	12	2	(8, 4)
7	24	(7, 0)	14	0	(9, 3)
8	39	(8, -1)	16	-2	(10, 2)
9	.....				
.....					

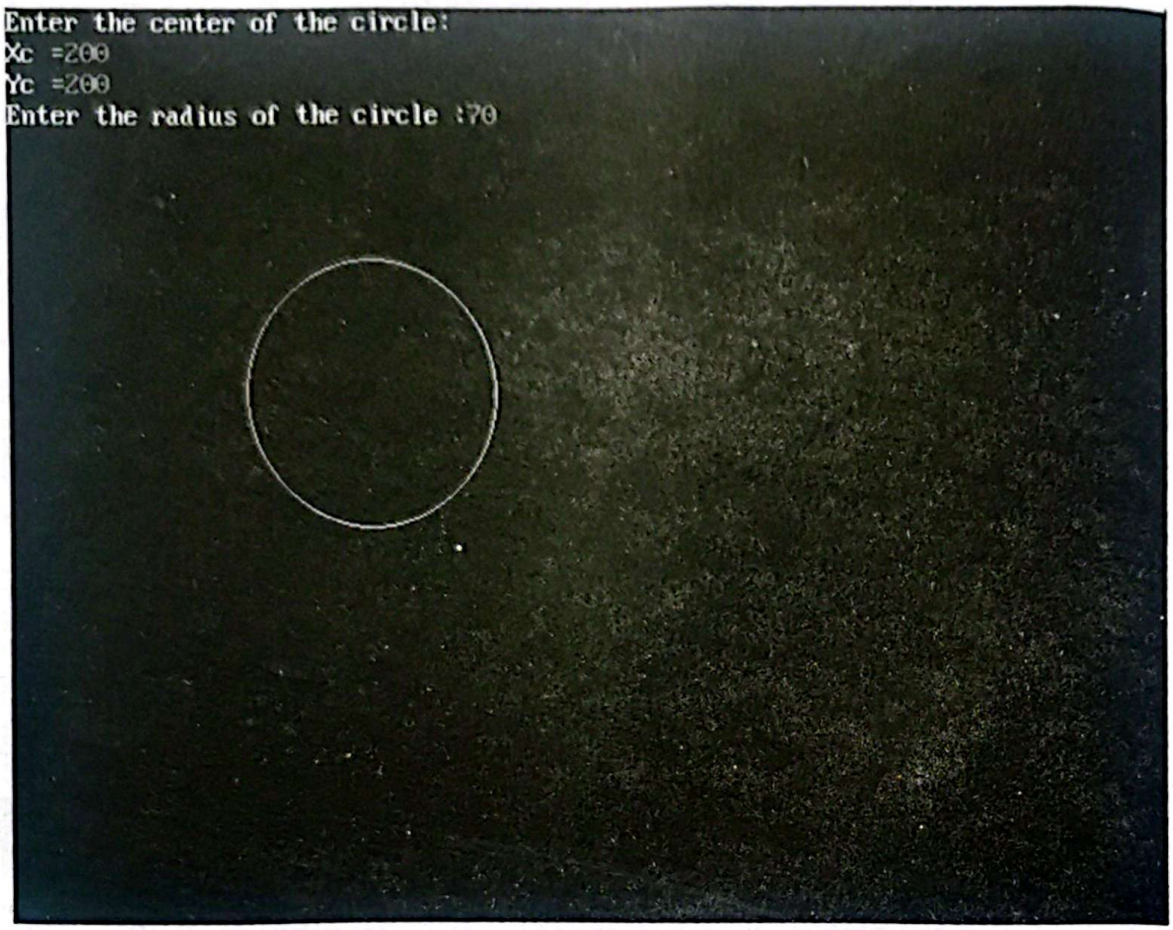


Lab problem 1.4: To implement midpoint circle generation algorithm or bresenham's circle algorithm for drawing a circle of given center (x, y) and radius r.

```
#include<dos.h>
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void draw_circle(int, int, int);
void symmetry(int, int, int, int);
void main()
{
        int xc,yc,R;
        int gd=DETECT,gm;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        printf("Enter the center of the circle:\n");
        printf("Xc =");
        scanf("%d",&xc);
        printf("Yc =");
        scanf("%d", &yc);
        printf("Enter the radius of the circle :");
        scanf("%d",&R);
        draw_circle(xc,yc,R);
        getch();
        closegraph();
}
```

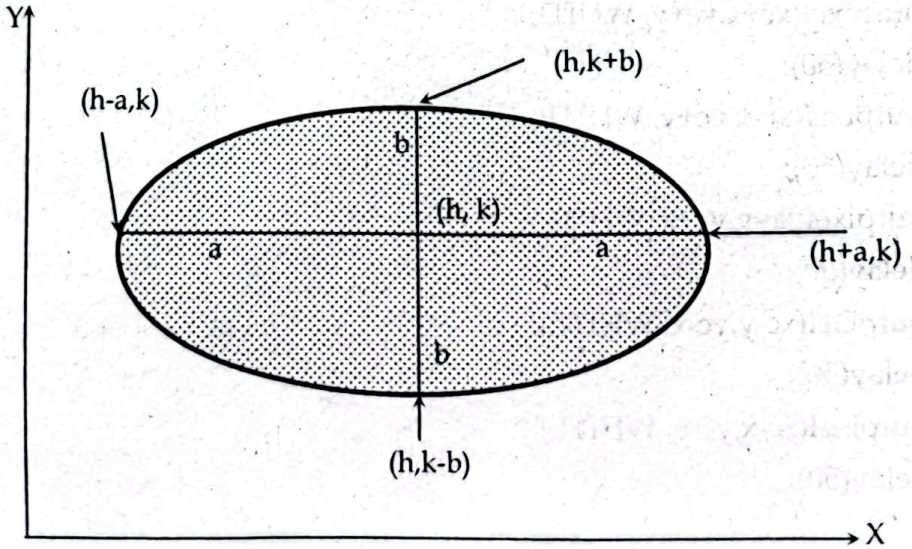
```
void draw_circle(int  xc, int yc, int rad)
{
        int x = 0;
        int y = rad;
        int p = 1-rad;
        symmetry(x,y,xc,yc);
        for(x=0; y>x; x++)
        {
                if(p<0)
                        p += 2*x + 3;
                else
                {
                        p += 2*(x-y) + 5;
                        y--;
                }
                symmetry(x,y,xc,yc);
                delay(50);
        }
}
void symmetry(int x, int y, int xc, int yc)
{
        putpixel(xc+x,yc-y, WHITE);
        delay(50);
        putpixel(xc+y,yc-x, WHITE);
        delay(50);
        putpixel(xc+y,yc+x, WHITE);
        delay(50);
        putpixel(xc+x,yc+y, WHITE);
        delay(50);
        putpixel(xc-x,yc+y, WHITE);
        delay(50);
        putpixel(xc-y,yc+x, WHITE);
        delay(50);
        putpixel(xc-y,yc-x, WHITE);
        delay(50);
        putpixel(xc-x,yc-y, WHITE);
        delay(50);
}
```

Output



ELLIPSE ALGORITHM

An ellipse is an elongated circle. Therefore, elliptical curves can be generated by modifying circle drawing procedures. The ellipse, like the circle, shows symmetry. An ellipse is symmetric in quadrants. So, if one quadrant is generated then other three parts can be easily generated The basic algorithm for drawing ellipse is same as circle computing x and y position at the boundary of the ellipse from the equation of ellipse directly.



Ellipse centered at (h, k) with semi-major axis 'a' and semi-minor axis 'b'.

An ellipse can be represented as:

$$\frac{(x - h)^2}{a^2} + \frac{(x - k)^2}{b^2} = 1$$

Where, (h, k) = ellipse center

a = length of semi-major axis.

b = length of semi-minor axis.

An ellipse centered at origin (0, 0) can be represented as:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Ellipse generating algorithms

There are three popular methods for drawing ellipse:

- Direct Method
- Trigonometric Method
- Midpoint Ellipse Algorithm

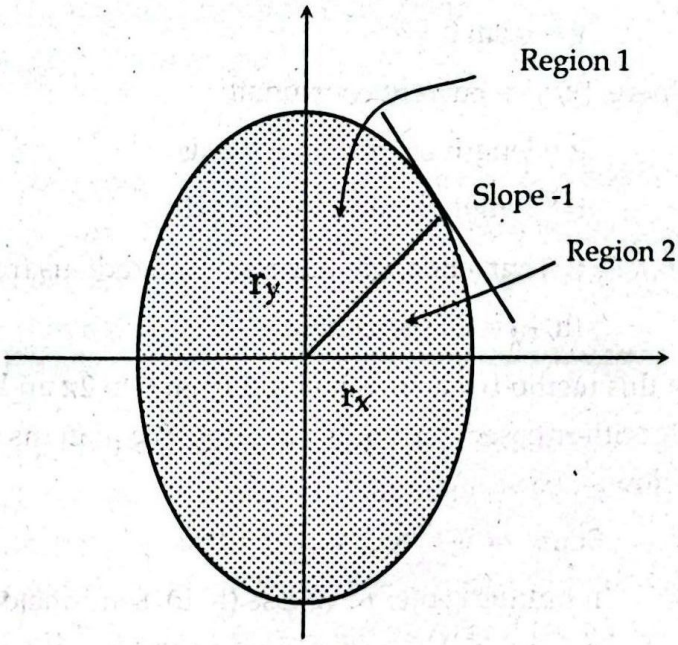
Direct Method for generating ellipse

The basic algorithm for drawing ellipse is same as circle computing x and y position at the boundary of the ellipse from the equation of ellipse directly. We have equation of ellipse centered at origin (0, 0) is,

$$\frac{x^2}{rx^2} + \frac{y^2}{r_y^2} = 1$$

This gives,

$$y = \pm \frac{r_y}{r_x}\sqrt{(rx)^2 - x^2} \quad \ldots\ldots (1)$$



Stepping unit interval in x direction from $-r_x$ to r_x we can get corresponding y value at each x position which gives the ellipse boundary co-ordinates. Plotting these computed points, we can get the ellipse. If center of ellipse is any arbitrary point (x_c, y_c) then the equation of ellipse can be written as,

$$\frac{(x - x_c)^2}{r_x^2} + \frac{(y - y_c)^2}{r_y^2} = 1$$

$$y = y_c \pm \frac{r_y}{r_x}\sqrt{r_x^2 - (x - x_c)^2} \quad \ldots\ldots\ldots\ldots (2)$$

For any point (x, y) on the boundary of the ellipse If major axis of ellipse with major axis alon X-axis the algorithm based on the direct computation of ellipse boundary points can b summarized as,

1. Start
2. Input the center of ellipse (x_c, y_c) x-radius x_r and y-radius y_r.
3. For each x position starting from x_c-r and stepping unit interval along x-direction compute corresponding y positions as

$$y = y_c \pm \frac{r_y}{r_x}\sqrt{r_x^2 - (x - x_c)^2}$$

4. Plot the point (x, y)
5. Repeat step 3 to 4 until $x >= x_c + x_r$
6. Stop

Computation of ellipse using polar co-ordinates

Using polar coordinates an ellipse can be represented as;

$$x = a \cos\theta + h$$

$$y = b \sin\theta + k$$

Where, (x, y) = current coordinate

a = length of semi-major axis

b = length of semi-minor axis

θ = current angle, measured in radians from 0 to 2π.

(h, k) = ellipse center.

In this method, θ is incremented from 0 to 2π and we compute successive values of x and y. Th algorithm based on these parametric equations on polar co-ordinates can be summarized a below,

1. Start
2. Input the center of ellipse (h, k), semi-major and semi-minor axis 'a' and 'b' respectively.
3. Starting from angle $\theta = 0$ compute boundary point of ellipse as

$$x = a \cos\theta + h$$

$$y = b \sin\theta + k$$

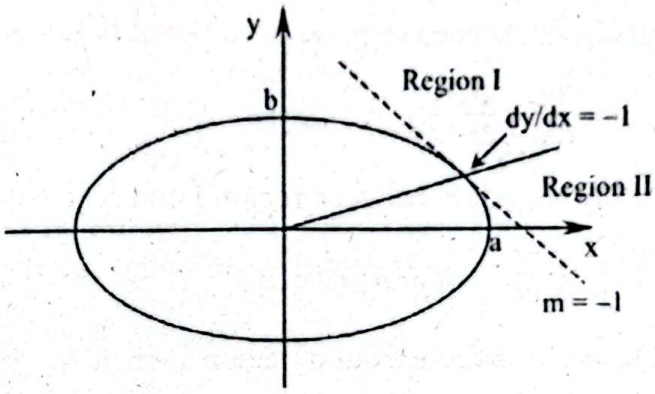
4. Plot the point at position (round(x), round(y))
5. Repeat until $\theta \geq 2\pi$ i.e., 360^0.
6. Stop

The methods of drawing ellipses explained above are not efficient. The method based on dir equation of ellipse must perform the square and square root operations due to which there m be floating point number computation which cause rounding off to plot the pixels. Also,

square root causes the domain error. Due to the changing slope of curve along the path of ellipse, there may be un-uniform separation of pixel when slope changes. To eliminate this problem, extra computation is needed. Although, the method based on polar co-ordinate parametric equation gives the uniform spacing of pixel due to uniform increment of angle but it also takes extra computation to evaluate the trigonometric functions. So, these algorithms are not efficient to construct the ellipse. We have another algorithm called mid- point ellipse algorithm similar to mid-point circle algorithm which is efficient algorithm for computing ellipse.

Mid-Point Ellipse Drawing Algorithm

Midpoint ellipse algorithm is a method for drawing ellipses in computer graphics. This method is modified from Bresenham's algorithm. The advantage of this modified method is that only addition operations are required in the program loops. This leads to simple and fast implementation in all processors.



Let us consider one quarter of an ellipse. The curve is divided into two regions. In region I, the slope on the curve is greater than -1 while in region II less than -1.

Consider the general equation of an ellipse,

$b^2x^2 + a^2y^2 - a^2b^2 = 0$ where 'a' is the horizontal radius and b is the vertical radius, we can define a function f(x, y) by which the error due to a prediction coordinate (x, y) can be obtained. The appropriate pixels can be selected according to the error so that the required ellipse is formed. The error can be confined within half a pixel.

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

The mid-point ellipse algorithm decides which point near the boundary (i.e., path of the ellipse) is closer to the actual ellipse path described by the ellipse equation. That point is taken as next point. It is applied to the first quadrant in two parts as in figure. Region 1 and Region 2. We process by taking unit steps in x-coordinates direction and finding the closest value for y for each x-step in region 1. In first quadrant at region 1, we start at position $(0, r_y)$ and incrementing x and calculating y closer to the path along clockwise direction. When slope becomes -1 then shift unit step in x to y and compute corresponding x closest to ellipse path at Region 2 in same direction.

Alternatively, we can start at position $(r_x, 0)$ and select point in counterclockwise order shifting unit steps in y to unit step in x when slope becomes greater than -1. Here, to implement mid-point ellipse algorithm, we take start position at $(0, r_y)$ and step along the ellipse path in clockwise position throughout the first quadrant.

We define ellipse function center at origin i.e., $(x_c, y_c) = (0,0)$ as,

$$f_{ellipse}(x, y) = r_y^2x^2 + r_x^2y^2 - r_x^2r_y^2$$

$$f_{ellipse}(x, y) \begin{cases} < 0, \text{ if } (x, y) \text{ lies inside boundary of ellipse} \\ = 0 \text{ if } (x, y) \text{ lies on the boundary of ellipse} \\ > 0 \text{ if } (x, y) \text{ lies outside the boundary of ellipse} \end{cases}$$

So $f_{ellipse}$ function serves as decision parameter in ellipse algorithm at each sampling position. We select the next pixel position according to the sign of decision parameter.

Starting at $(0, r_y)$, we take unit step in x-direction until we reach the boundary between the region 1 and region 2. Then we switch unit steps in y over the remainder of the curve in first quadrant. At each step, we need to test the slope of curve. The slope of curve is calculated as;

$$\frac{dy}{dx} = \frac{2r_y^2x}{2r_x^2y}$$

At the boundary between region 1 and region 2,

$$\frac{dy}{dx} = -1 \text{ and } 2r_y^2 = 2r_x^2$$

Therefore, we move out of region 1 when $2r_y^2 \geq 2r_x^2$

Assuming the position (x_k, y_k) is filled, we move x_{k+1} to determine next pixel. The corresponding y value for x_{k+1} position will be either y_k or y_k-1 depending upon the sign of decision parameter. So, the decision parameter for region 1 is tested at midpoint of (x_k+1, y_k) and $(x_k + 1, y_k - 1)$ i.e.

$$P_{1k} = f_{ellipse}\left(x_{k+1}, y_k - \frac{1}{2}\right)$$

$$P_{1k} = r_y^2 (x_{k+1})^2 + r_x^2\left(y_k - \frac{1}{2}\right)^2 - r_x^2r_y^2$$

$$P_{1k} = r_y^2 (x_{k+1})^2 + r_x^2r_y^2 - r_y^2y_k + \frac{r_x^2}{4} - r_x^2r_y^2 \quad \ldots\ldots\ldots (1)$$

If $P_{1k} < 0$ the midpoint lies inside boundary, so next point to plot is (x_k+1, y_k) otherwise, next point to plot will be (x_k+1, y_k-1). The successive decision parameter is computed as

$$P_{1k+1} = f_{ellipse}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$= r_y^2(x_{k+1} + 1)^2 + r_x^2\left(y_{k+1} - \frac{1}{2}\right)^2 - r_x^2r_y^2$$

$$P_{1k+1} = r_y^2 (x_{k+1}^2 + 2x_{k+1} + 1) + r_x^2\left(y_{k+1}^2 - y_{k+1} + \frac{1}{4}\right)^2 - r_x^2r_y^2$$

$$P_{1k+1} = r_y^2x_{k+1}^2 + 2r_y^2x_{k+1} + r_y^2 + r_x^2y_{k+1}^2 - r_x^2y_{k+1} + \frac{r_x^2}{4} - r_x^2r_y^2 \ldots. (2)$$

Subtracting (2) - (1)

$$p_{1k+1} - p_{1k} = 2r_y^2x_{k+1} + r_y^2 + r_x^2 (y_{k+1}^2 - y_k^2) - r_x^2 (y_{k+1} - y_k)$$

If $P_{1k} < 0$, $y_{k+1} = y_k$ then,

$\therefore \quad P_{1k+1} = P_{1k} + 2r_y^2 x_{k+1} + r_y^2$

Otherwise $y_{k+1} = y_k - 1$ then we get,

$$P_{1k+1} = P_{1k} + 2r_y^2 x_{k+1} + r_x^2 - 2r_x^2 y_{k+1}$$

At the initial position, $(0, r_y)$ $2r_x^2 x = 0$ and $2r_y^2 y = 2r_x^2 r_y$

In region 1, initial decision parameter is obtained by evaluating ellipse function at $(0, r_y)$ as

$$P_{10} = f_{ellipse}\left(1, r_y - \frac{1}{2}\right)$$

$$P_{10} = f_{ellipse}\left(1, r_y - \frac{1}{2}\right)$$

$$= r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

Similarly, over the region 2, the decision parameter is tested at midpoint of $(x_k, y_k - 1)$ and $(x_k + 1, y_k - 1)$ i.e.

$$P_{2k} = f_{ellipse}\left(x_k + \frac{1}{2}, y_k - 1\right)$$

$$= r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2$$

$$\therefore \quad P_{2k} = r_y^2 x_k^2 + r_y^2 x_k + \frac{r_y^2}{4} + r_x^2(y_k - 1)^2 - r_x^2 r_y^2 \ldots\ldots\ldots (3)$$

If $p_{2k} > 0$, the midpoint lies outside the boundary, so next point to plot is $(x_k, y_k - 1)$ otherwise, next point to plot will be $(x_k + 1, y_k - 1)$.

The successive decision parameter is computed as evaluating ellipse function at midpoint of

$$P_{2k+1} = f_{ellipse}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right) \text{ with } y_{k+1} = y_{k-1}$$

$$P_{2k+1} = r_y^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x^2\left[(y_k - 1) - 1\right]^2 - r_x^2 r_y^2$$

$$\text{or,} \quad P_{2k+1} = r_y^2 x_{k+1}^2 + r_y^2 x_{k+1} + \frac{r_y^2}{4} + r_x^2(y_k - 1)^2 - 2r_x^2(y_k - 1) + r_x^2 - r_x^2 r_y^2 \quad \ldots\ldots\ldots (4)$$

Subtracting (4) - (3)

$$P_{2k+1} - P_{2k} = r_y^2 (x_{k+1}^2 - x_k^2) + r_y^2 (x_{k+1} - x_k) - 2r_x^2(y_k - 1) + r_x^2$$

$$\text{or,} \quad P_{2k+1} = P_{2k} + r_y^2 (x_{k+1}^2 - x_k^2) + r_y^2 (x_{k+1} - x_k) - 2r_x^2(y_k - 1) + r_x^2$$

if $P_{2k} > 0$, $x_{k+1} = x_k$ then

$$P_{2k+1} = P_{2k} - 2r_x^2(y_k - 1) + r_x^2$$

Otherwise $x_{k+1} = x_k + 1$ then

$$P_{2k+1} = P_{2k} + r_y^2[(x_k + 1)^2 - x_k^2)] + r_y^2(x_k + 1 - x_k) - 2r_x^2(y_k - 1) + r_x^2$$

or, $P_{2k+1} = P_{2k} + r_y^2(2x_k + 1) + r_y^2 - 2r_x^2(y_k - 1) + r_x^2$

or, $P_{2k+1} = P_{2k} + r_y^2(2x_k + 2) - 2r_x^2(y_k - 1) + r_x^2$

or, $P_{2k+1} = P_{2k} + 2r_y^2x_{k+1} - 2r_x^2y_{k+1} + r_x^2$ where $x_{k+1} = x_k + 1$ and $y_{k+1} = y_k - 1$.

The initial position for region 2 is taken as last position selected in region 1 say which is (x_0, y_0) then initial decision parameter in region 2 is obtained by evaluating ellipse function at midpoint of $(x_0, y_0 - 1)$ and $(x_0 + 1, y_0 - 1)$ as

$$P_{20} = f_{ellipse}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

$$= r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2r_y^2$$

Now the midpoint ellipse algorithm is summarized as;

1. Input center (x_c, y_c) and r_x and r_y for the ellipse and obtain the first point as
2. $(x_0, y_0) = (0, r_y)$
3. Calculate initial decision parameter value in Region 1 as

   $$P_{10} = r_y^2 - r_x^2r_y + \frac{1}{4}r_x^2$$

4. At each x_k position, in Region 1, starting at $k = 0$, compute

   $x_{k+1} = x_k + 1$
5. If $P_{1k} < 0$, then the next point to plot is

   $P_{1k+1} = P_{1k} + 2r_y^2x_{k+1} + r_y^2$

   $y_{k+1} = y_k$

   Otherwise, next point to plot is

   $y_{k+1} = y_k - 1$

   $P_{1k+1} = P_{1k} + 2r_y^2x_{k+1} + r_y^2 - 2r_x^2y_{k+1}$

   With $x_{k+1} = x_k + 1$ and $y_{k+1} = y_k - 1$
6. Calculate the initial value of decision parameter at region 2 using last calculated point say (x_0, y_0) in region 1 as

   $$P_{20} = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2r_y^2$$
7. At each y_k position in Region 2 starting at $k = 0$, perform computation

   $y_{k+1} = y-1;$

   if $P_{2k} > 0$ then

   $x_{k+1} = x_k$

   $P_{2k+1} = P_{2k} - 2r_x^2(y_k - 1) + r_x^2$

   Otherwise

   $x_{k+1} = x_k + 1$

$P_{2k+1} = P_{2k} + 2r_y^2x_{k+1} - 2r_x^2y_{k+1} + r_x^2$ Where $x_{k+1} = x_k + 1$ and $y_{k+1} = y_k - 1$

8. Determine the symmetry points in other 3 quadrants.
9. Move each calculated point (x_k, y_k) on to the centered (x_c, y_c) ellipse path as

   $x_k = x_k + x_c$;

   $y_k = y_k + y_c$
10. Repeat the process for region 1 until $2r_y^2x_k \geq 2r_x^2y_k$ and region until $(x_k, y_k) = (r_x, 0)$.

Example 1.16: Input ellipse parameters r_x=8 and r_y=6 the midpoint ellipse algorithm by determining raster position along the ellipse path is the first quadrant. Initial values and increments for the decision parameter calculate are:

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

$2r_x^2y = 2r_x^2 y = 2r_x^2r_y$ (with increment $-2rx^2 = -128$)

For region 1 the initial point for the ellipse centered on the origin is $(x_0, y_0) = (0, 6)$ and the initial decision parameter value is:

$$P_{10} = r_y^2 - r_x^2 r_y^2 + \frac{1}{4} r_x^2 = -332$$

Successive midpoint decision parameter values and the pixel positions along the ellipse are listed in the following table:

k	P_{1k}	(x_{k+1}, y_{k+1})	$2r_y^2x_{k+1}$	$2r_x^2y_{k+1}$
0	-332	(1, 6)	72	768
1	-224	(2, 6)	144	768
2	-44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	-108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

Move out of region 1, $2r_y^2x > 2r_x^2y$

For a region 2, the initial point is $(x_0, y_0) = (7, 3)$ and the initial decision parameter is

$$P_{20} = f_{ellipse}\left(7 + \frac{1}{2}, 2\right) = -151$$

The remaining positions along the ellipse path in the first quadrant are then calculated as,

k	P_{2k}	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	-151	(8, 2)	576	256
1	233	(8, 1)	576	128
2	745	(8, 0)	---	---

Lab problem 1.5: To implement the Ellipse Generation Algorithm for drawing an ellipse of given center (x, y) and radius r_x and r_y.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void disp();
float x, y;
int xc, yc;
void main()
{
        int gd=DETECT, gm;
        int rx,ry;
        float p1,p2;
        clrscr();
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        printf("Enter the center point :");
        scanf("%d%d",&xc,&yc);
        printf("Enter the value for Rx and Ry :");
        scanf("%d%d", &rx,&ry);
        x=0;
        y=ry;
        disp();
        p1=(ry*ry)-(rx*rx*ry)+(rx*rx)/4;
        while((2.0*ry*ry*x)<=(2.0*rx*rx*y))
        {
                x++;
                if(p1<=0)
                        p1=p1+(2.0*ry*ry*x)+(ry*ry);
                else
                {
                        y--;
                        p1=p1+(2.0*ry*ry*x)-(2.0*rx*rx*y)+(ry*ry);
                }
```

```
            disp();
            x=-x;
            disp();
            x=-x;
        }
        x=rx;
        y=0;
        disp();
        p2=(rx*rx)+2.0*(ry*ry*rx)+(ry*ry)/4;
        while((2.0*ry*ry*x)>(2.0*rx*rx*y))
        {
            y++;
            if(p2>0)
                p2=p2+(rx*rx)-(2.0*rx*rx*y);
            else
            {
                x--;
                p2=p2+(2.0*ry*ry*x)-(2.0*rx*rx*y)+(rx*rx);
            }
            disp();
            y=-y;
            disp();
            y=-y;
        }
        getch();
        closegraph();
}
void disp()
{
        delay(50);
        putpixel(xc+x,yc+y,7);
        putpixel(xc-x,yc+y,7);
        putpixel(xc+x,yc-y,7);
        putpixel(xc-x,yc-y,7);
}
```

Output

AREA FILLING

In this section we are going to study different types of polygons, their representation and filling algorithms for them. A polyline is a chain of connected line segments. It is specified by giving the vertices (nodes) P_0, P_1, P_2 ... and so on. The first vertex is called the initial or starting point and the last vertex is called the final or terminal point, as shown in the figure 3.1. When starting point and terminal point of any polyline is same, i.e., when polyline is closed then it is called polygon. This is illustrated in Figure 3.1.

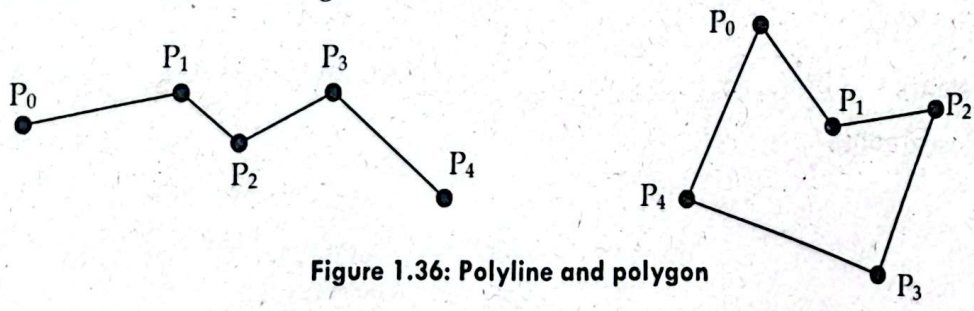


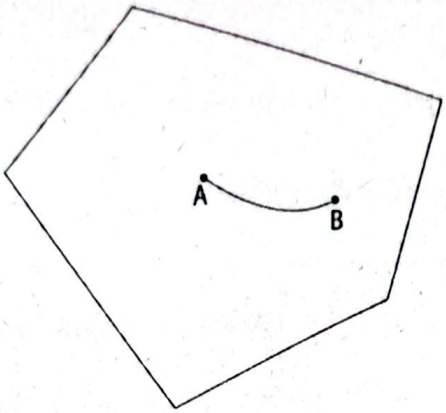
Figure 1.36: Polyline and polygon

Types of Polygons

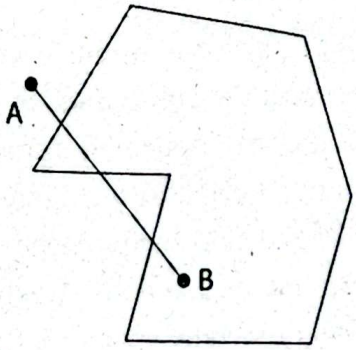
The classification of polygons is based on where the line segment joining any two points within the polygon is going to lie. There are two types of polygons:

- Convex and
- Concave

A convex polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon. The figure shown below is convex polygons.



A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon. The figure below illustrates concave polygons.



FILLED AREA PRIMITIVES

A standard output primitive in general graphics package is solid color or patterned polygon area. Other kinds of area primitives are sometimes available, but polygons are easier to process since they have linear boundaries. There are two basic approaches to area filling in raster systems. One way to fill an area is to determine the overlap intervals for scan lines that cresses the area. Another method for area filling is to start from a given interior position and point outward from this until a specified boundary is met.

There are three popular methods for are fill algorithms:

- Scan line polygon fill algorithms
- Boundary fill algorithms and
- Flood fill algorithms

Scan Line Polygon Fill Algorithm

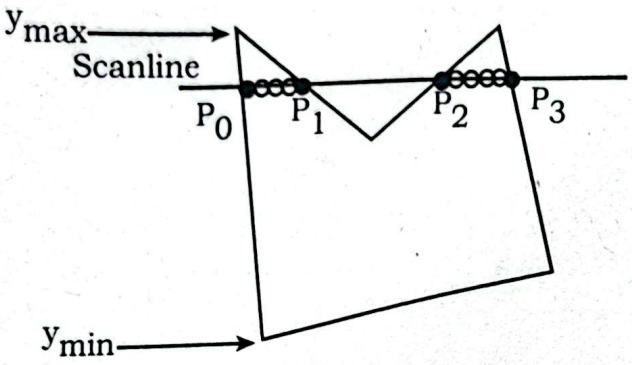
This algorithm works by intersecting scan line with polygon edges and fills the polygon between pairs of intersections. The following steps depict how this algorithm works.

Step 1 – Find out the Y_{min} and Y_{max} from the given polygon.

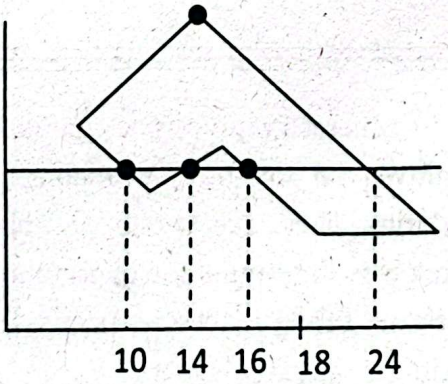
Step 2 – Scan Line intersects with each edge of the polygon from Y_{min} to Y_{max}. Name each intersection point of the polygon. As per the figure shown below, they are named as P_0, P_1, P_2, P_3.

Step 3 – Sort the intersection point in the increasing order of X coordinate i.e. (P_0, P_1), (P_1, P_2), and (P_2, P_3).

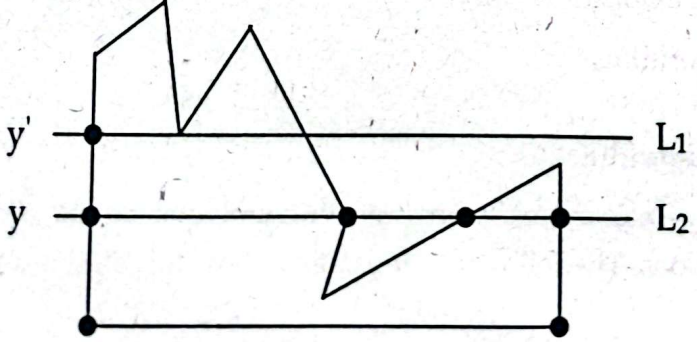
Step 4 – Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.



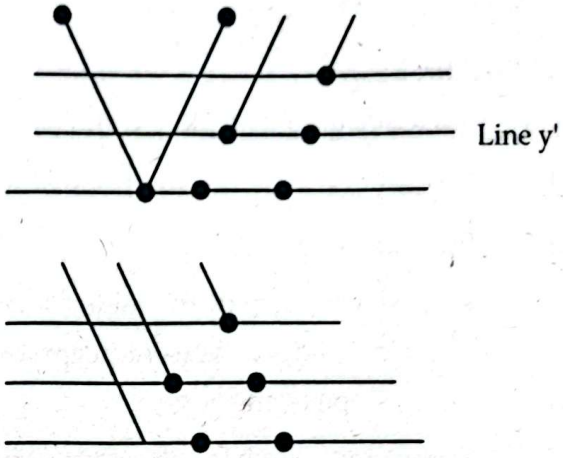
In scan-line polygon fill algorithm, for each scan-line crossing a polygon, it locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified color. In the figure below, the four-pixel intersection positions with the polygon boundaries defined two stretches of interior pixel from x=10 to x=14 and from x=16 to x=24. Some scan-line intersections at polygon vertices require extra special handling. A scan-line passing through a vertex intersects two polygon edges at that position, adding two points to the list of intersection for the scan-line.



The figure below shows two scan lines at position y and y' that intersect the edge points. Scan line at y intersects five polygon edges. Scan line at y' intersects 4 (even numbers) of edges though it passes through vertex.



Intersection points along scan line y' correctly identify the interior pixel spans. But with scan line y, we need to do some additional processing to determine the correct interior points. For scan line y, the two edges sharing the intersecting vertex are on opposite side of the scan-line. But for scan-line y' the two edges sharing intersecting vertex are on the same side (above) the scan line position. So, the vertices those are on opposite side of scan line require extra processing.



We can identify these vertices by tracing around the polygon boundary either in clockwise or counter clockwise order and observing the relative changes in vertex y coordinates as we move from one edge to next. If the endpoint y values of two consecutive edges monotonically increases or decrease, we need to count the middle vertex as a single intersection point for any scan line passing through that vertex. Otherwise, the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan-line passing through that vertex.

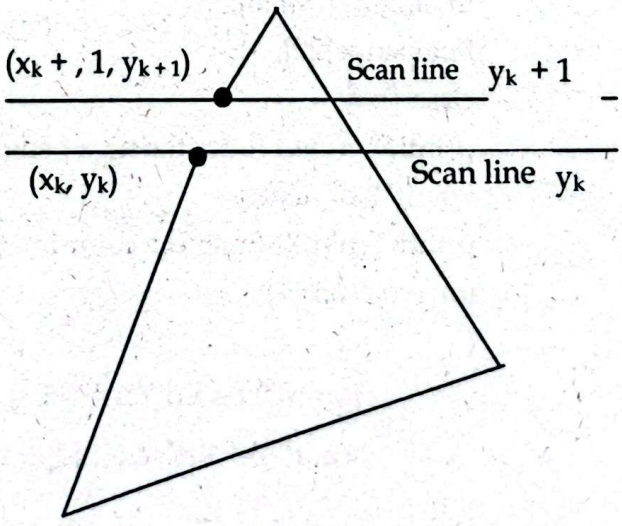
We can identify these vertices by tracing around the polygon boundary either in clockwise or counter clockwise order and observing the relative changes in vertex y coordinates as we move from one edge to next. If the endpoint y values of two consecutive edges monotonically increases or decrease, we need to count the middle vertex as a single intersection point for any scan line passing through that vertex. Otherwise, the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan-line passing through that vertex.

In successive scan lines crossing a left edge of a polygon, the slope of this polygon boundary line can be expressed in terms of scan-line intersection co-ordinates:

$$m = \frac{y_k + 1 - y_k}{x_k + 1 - x_k}$$

Since the change between two scan line in y co-ordinates is 1,

$$y_k + 1 - y_k = 1$$



The x-intersection value $x_k + 1$, on the upper scan line can be determined from the x-intersection value x_k, on the preceding scan line as

$$x_k + 1' = x_k + \frac{1}{m}$$

Each successive xx intercept can thus be calculated by x values by the amount of $\frac{1}{m}$ along an edge can be accomp0lished with integer operation by recalling that the slope m is the ratio to two integers

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

Where, Δx & Δy are the differences between the edge endpoint x and y co-ordinate values. Thus, incremental calculations of x intercepts along an edge for successive scan lines can be expressed as

$$x_k + 1 = x_k + \frac{\Delta x}{\Delta y}$$

Algorithm Steps

1. the horizontal scanning of the polygon from its lowermost to its topmost vertex
2. identify the edge intersections of scan line with polygon
3. Build the edge table
   - Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.
4. Determine whether any edges need to be splitted or not. If there is need to split, split the edges.
5. Add new edges and build modified edge table.
6. Build Active edge table for each scan line and fill the polygon based on intersection of scan line with polygon edges.

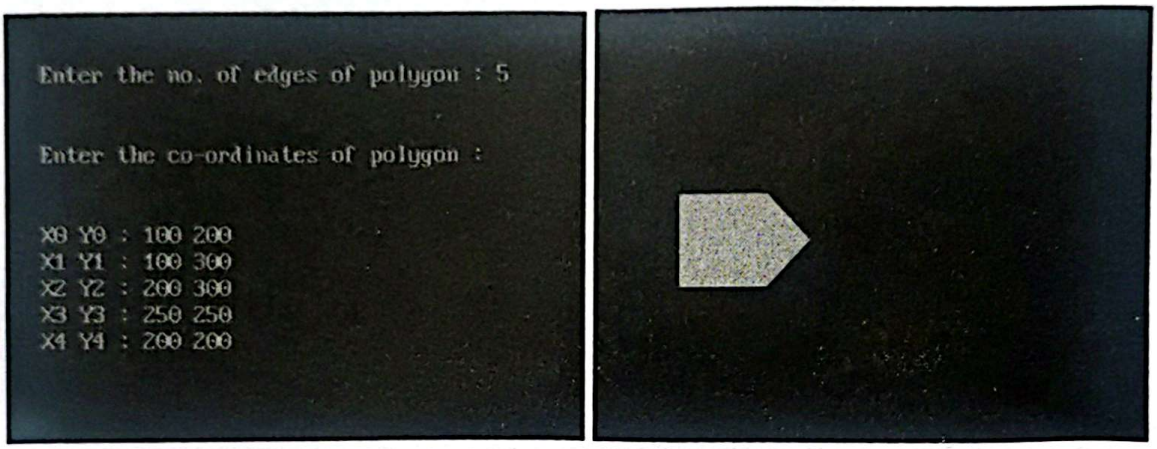
Lab problem 1.6: Program for scan line polygon fill algorithm

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void main()
{
        int n, i, j, k, gd, gm, dy, dx;
        int x, y, temp;
        int a[20][2], xi[20];
        float slope[20];
        clrscr();
        printf("\n\n\t Enter the no. of edges of polygon : ");
        scanf("%d", &n);
        printf("\n\n\t Enter the co-ordinates of polygon :\n\n\n ");
        for(i=0;i<n;i++)
        {
                printf("\t X%d Y%d : ",i ,i);
                scanf("%d %d", &a[i][0],&a[i][1]);
        }
        a[n][0]=a[0][0];
        a[n][1]=a[0][1];
```

```
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
for(i=0;i<n;i++)
{
        line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);
}
getch();
for(i=0;i<n;i++)
{
        dy=a[i+1][1]-a[i][1];
        dx=a[i+1][0]-a[i][0];
        if(dy==0)
                slope[i]=1.0;
        if(dx==0)
                slope[i]=0.0;
        if((dy!=0)&&(dx!=0))
        {
                slope[i]=(float) dx/dy;
        }
}
for(y=0;y< 480;y++)
{
        k=0;
        for(i=0;i<n;i++)
        {
                if( ((a[i][1]<=y)&&(a[i+1][1]>y)) | |((a[i][1]>y)&&(a[i+1][1]<=y)))
                {
                        xi[k]=(int)(a[i][0]+slope[i]*(y-a[i][1]));
                        k++;
                }
        }
        for(j=0;j<k-1;j++) /*- Arrange x-intersections in order -*/
        for(i=0;i<k-1;i++)
        {
                if(xi[i]>xi[i+1])
                {
                        temp=xi[i];
                        xi[i]=xi[i+1];
                        xi[i+1]=temp;
                }
        }
```

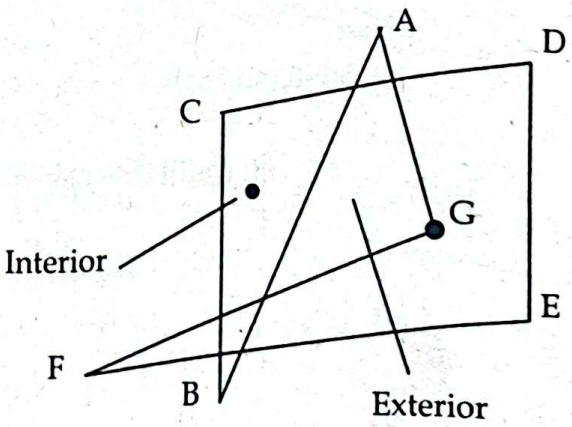
```
            setcolor(7);
            for(i=0;i<k;i+=2)
            {
                  line(xi[i],y,xi[i+1]+1,y);
                  getch();
            }
      }
}
```

Output



Inside-Outside Test

One simple way of finding whether the point is inside or outside a simple polygon is to test how many times a ray, starting from the point and going in any fixed direction, intersects the edges of the polygon. If the point is on the outside of the polygon the ray will intersect its edge an even number of times. If the point is on the inside of the polygon, then it will intersect the edge an odd number of times. Unfortunately, this method won't work if the point is on the edge of the polygon.

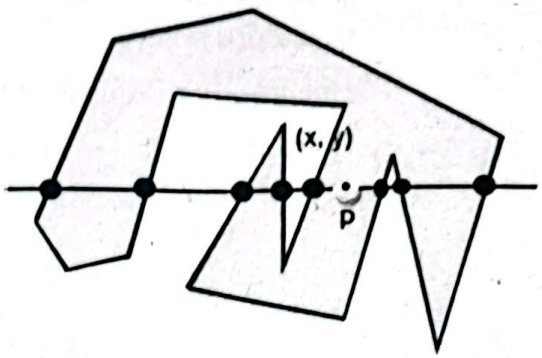


Area filling algorithms and other graphics package often need to identify interior and exterior region for a complex polygon in a plane. For example, in figure below, it needs to identify interior and exterior region.

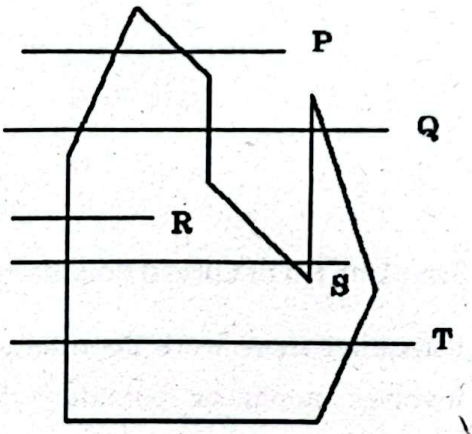
This method is also known as **counting number method**. While filling an object, we often need to identify whether particular point is inside the object or outside it. There are two methods by which we can identify whether particular point is inside an object or outside.

- Odd-Even method
- Nonzero winding number method

Odd-Even method



In this technique, we will count the edge crossing along the line from any point P(x, y) to infinity. If the number of interactions is odd, then the point P(x, y) is an interior point; and if the number of interactions is even, then the point P(x, y) is an exterior point. The following example depicts this concept. From the above figure, we can see that from the point P(x, y), the number of interactions point on the left side is 5 and on the right side is 3. From both ends, the number of interaction points is odd, so the point is considered within the object.



In the above figure the points P, Q, T lays outside of the polygon since the crosses the side of the polygon in even times i.e., 2 times, 4-times and 2-times respectively.

Also, the points R and S lies inside of the polygon since they cross the side of the polygon in odd times i.e., in 1-times and 3-times respectively.

Non-zero Winding Number Rule

This method is also used with the simple polygons to test the given point is interior or not. It can be simply understood with the help of a pin and a rubber band. Fix up the pin on one of the edges of the polygon and tie-up the rubber band in it and then stretch the rubber band along the edges of the polygon. When all the edges of the polygon are covered by the rubber band, check out the pin which has been fixed up at the point to be test. If we find at least one wind at the point consider it within the polygon, else we can say that the point is not inside the polygon. In non-zero winding number method we need to know the direction of each edge in the polygon, i.e., whether the edge is clockwise or counter-clockwise. The winding number is number of times the polygon edges wind around a particular point in the counterclockwise direction.

Draw a scan line from the point to be test towards the left most of X direction.

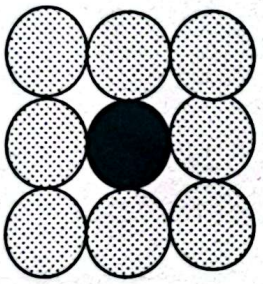
- Give the value 1 to all the edges which are going to upward direction and all other -1 as direction values.
- Check the edge direction values from which the scan line is passing and sum up them.
- If the total sum of this direction value is non-zero, then this point to be tested is an interior point, otherwise it is an exterior point.

```
            boundary_fill(x + 1, y, fcolor, bcolor);
            boundary_fill(x - 1, y, fcolor, bcolor);
            boundary_fill(x, y + 1, fcolor, bcolor);
            boundary_fill(x, y - 1, fcolor, bcolor);
        }
}
void main()
{
        int x, y, fcolor, bcolor;
        int gd=DETECT,gm;
        initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
        printf("Enter the seed point (x, y) for a circle(200,200,45): ");
        scanf("%d%d", &x, &y);
        printf("Enter boundary color : ");
        scanf("%d", &bcolor);
        printf("Enter new color : ");
        scanf("%d", &fcolor);
        circle(200,200,45);
        boundary_fill(x, y, fcolor, bcolor);
        getch();
}
```

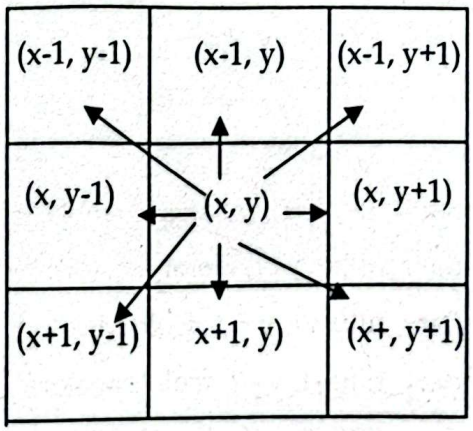
Output:

8-connected pixels

More complex figures are filled using this approach. The pixels to be tested are the 8 neighboring pixels, the pixel on the right, left, above, below and the 4 diagonal pixels. Areas filled by this method are called 8-connected. In this technique 8-connected pixels are used as shown in the figure below. We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique. In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.



8- Connected



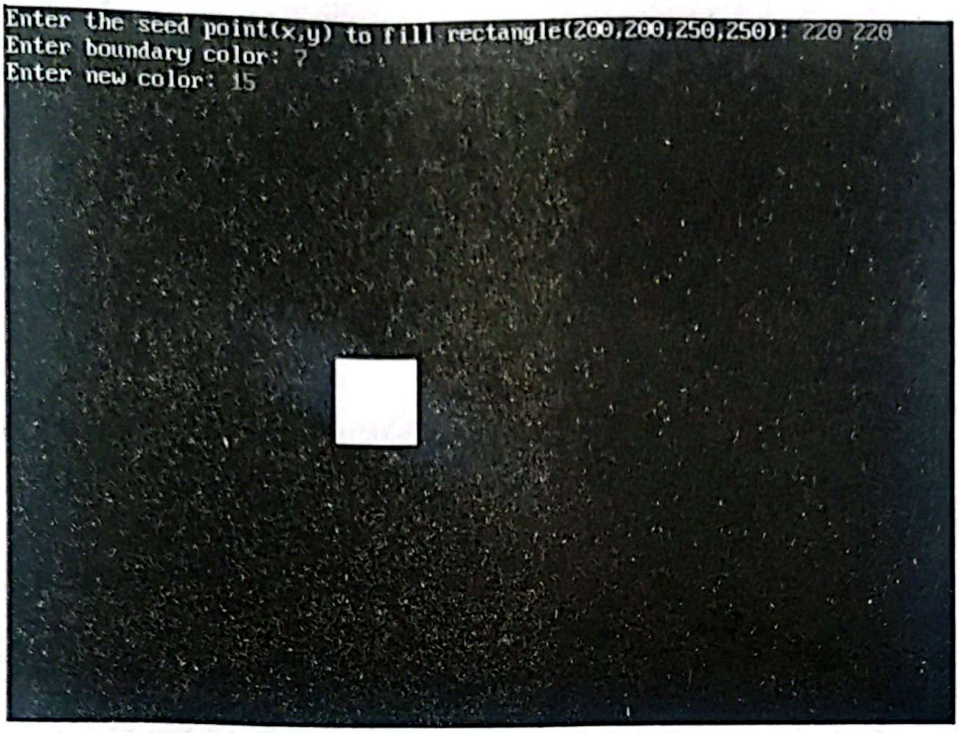
The outline of this algorithm is given below

```
void Boundary-fill8(int x, int y, int b_color, int fill_color)
{
        int current;
        current=getpixel (x, y);
        if (current !=b_color&&current!=fill_color)
        (       putpixel (x,y,fill_color);
                Boundary_fill8(x-1, y, b_color,fill_color);
                Boundary_fill8(x+1, y, b_color,fill_color);
                Boundary_fill8(x, y-1, b_color,fill_color);
                Boundary_fill8(x, y+1, b_color,fill_color);
                Boundary_fill8(x-1, y-1, b_color,fill_color);
                Boundary_fill8(x-1, y+1, b_color,fill_color);
                Boundary_fill8(x+1, y-1, b_color,fill_color);
                Boundary_fill8(x+1, y+1, b_color,fill_color);
        )
)
```

Lab problem 1.8: Program to implement the 8-connected Boundary fill algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void boundary_fill(int x, int y, int fcolor, int bcolor)
{
   if ((getpixel(x, y) != bcolor) && (getpixel(x, y) != fcolor))
   {
         delay(10);
         putpixel(x, y, fcolor);
         boundary_fill(x + 1, y, fcolor, bcolor);
         boundary_fill(x , y+1, fcolor, bcolor);
         boundary_fill(x+1, y + 1, fcolor, bcolor);
         boundary_fill(x-1, y - 1, fcolor, bcolor);
         boundary_fill(x-1, y, fcolor, bcolor);
         boundary_fill(x , y-1, fcolor, bcolor);
         boundary_fill(x-1, y + 1, fcolor, bcolor);
         boundary_fill(x+1, y - 1, fcolor, bcolor);
   }
}
void main()
{
      int x, y, fcolor, bcolor;
      int gd=DETECT, gm;
      initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
      printf("Enter the seed point(x,y) to fill rectangle(200,200,250,250) : ");
      scanf("%d%d", &x, &y);
      printf("Enter boundary color: ");
      scanf("%d", &bcolor);
      printf("Enter new color: ");
      scanf("%d", &fcolor);
      rectangle(200,200,250,250);
      boundary_fill(x, y, fcolor, bcolor);
      getch();
}
```

Output



Flood-fill Algorithm

Flood fill algorithm is applicable when we want to fill an area that is not defined within a single-color boundary. If fill area is bounded with different color, we can paint that area by replacing a specified interior color instead of searching of boundary color value. This approach is called flood fill algorithm. We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with desired fill color. Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior. There are some cases where the boundary color is different than the fill color. For situations like this Flood fill algorithm is used. Here the process is started in a similar way by examining the colors of neighboring pixels. But instead of matching it with a boundary color a specified color is matched.

The algorithm fills the area by 4- connected region by desired color.

```
void flood_fill4(int x, int y, int fill_color, int old_color)
{
        int current;
        current=getpixel (x, y);
        if (current==old_color)
        {
                putpixel (x, y, fill_color);
                flood_fill4(x-1,y, fill_color, old_color);
                flood_fill4(x,y-1, fill_color, old_color);
```

```
            flood_fill4(x,y+1, fill_color, old_color);
            flood_fill4(x+1,y, fill_color, old_color);
        }
}
```

Procedure for filling an 8- connected region

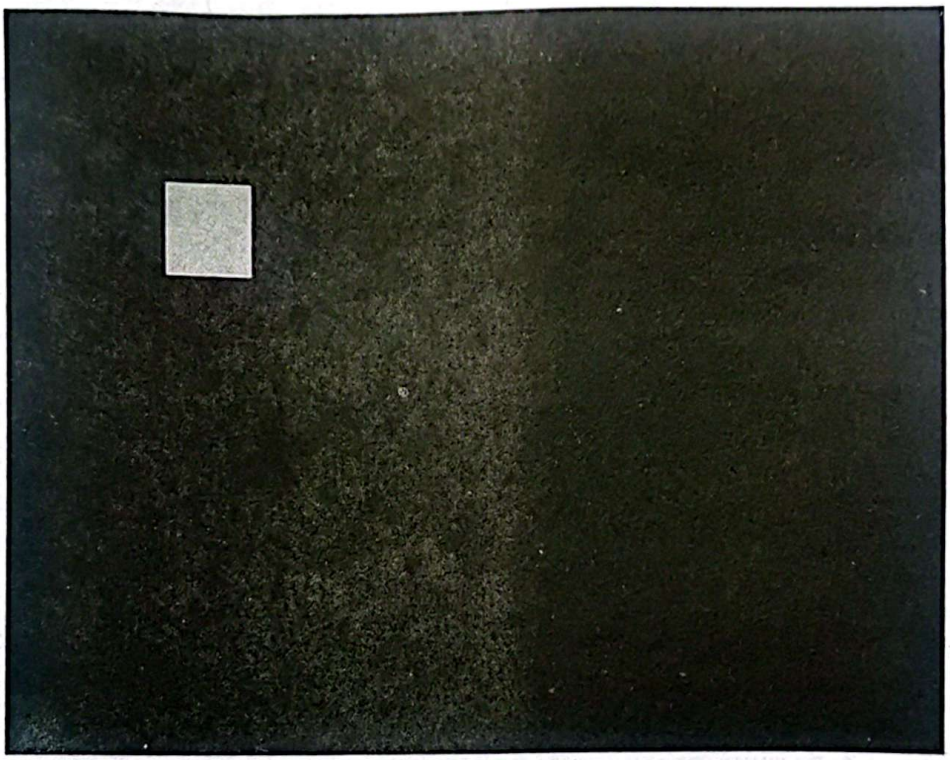
```
flood_ fill (x, y, old_color, new_color)
{
        putpixel(x,y,new_color)
        flood_ fill (x+1, y, old_color, new_color)
        flood_ fill (x-1, y, old_color, new_color)
        flood_ fill (x, y+1, old_color, new_color)
        flood_ fill (x, y-1, old_color, new_color)
        flood_ fill (x+1, y+1, old_color, new_color)
        flood_ fill (x-1, y-1, old_color, new_color)
        flood_ fill (x+1, y-1, old_color, new_color)
        flood_ fill (x-1, y+1, old_color, new_color)
}
```

Lab problem 1.9: Program for 4-connected flood fill

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void flood(int,int,int,int);
void main()
{
        int gd,gm=DETECT;
        clrscr();
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        rectangle(100,100,150,150);
        flood(110,110,7,0);
        getch();
  }
void flood(int x,int y, int fill_col, int old_col)
{
        if(getpixel(x,y)==old_col)
        {
```

```
            delay(10);
            putpixel(x,y,fill_col);
            flood(x+1,y,fill_col,old_col);
            flood(x-1,y,fill_col,old_col);
            flood(x,y+1,fill_col,old_col);
            flood(x,y-1,fill_col,old_col);
        }
}
```

Output:

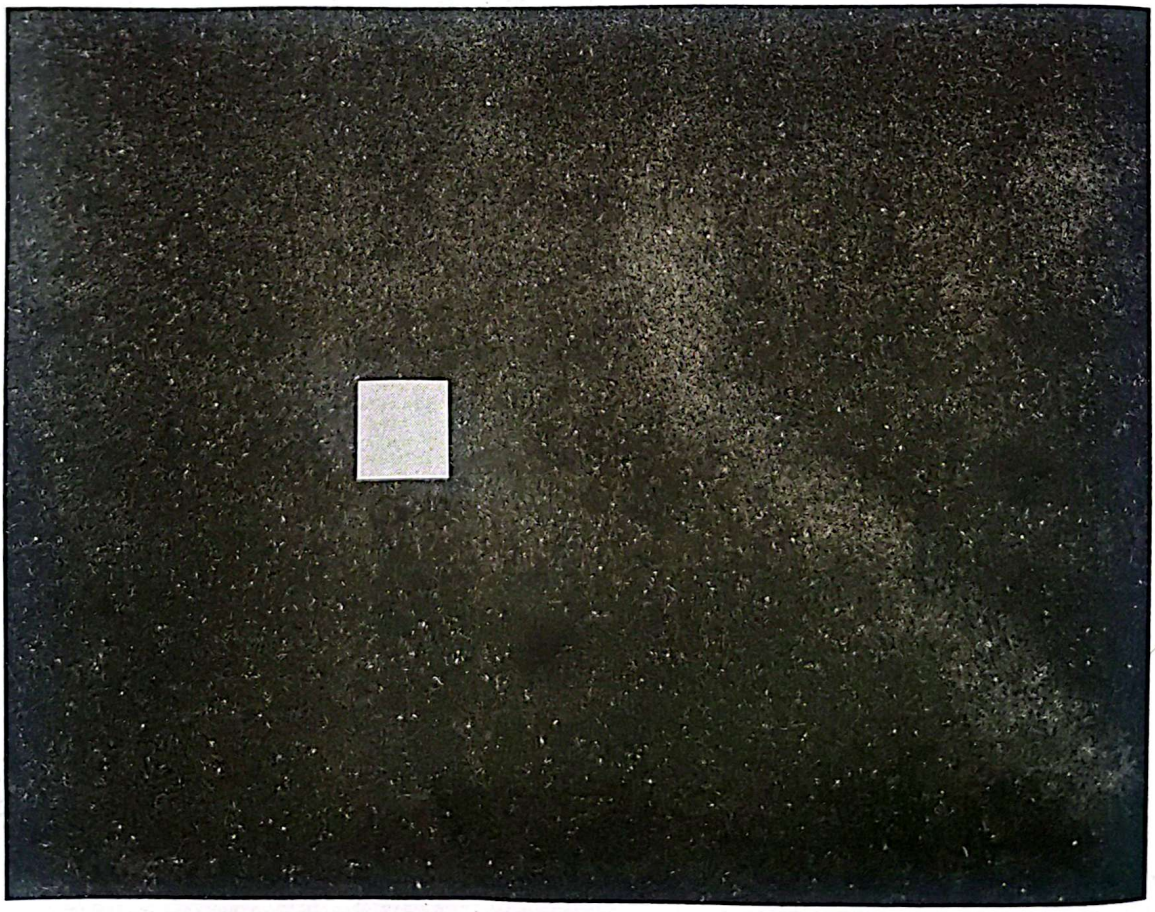


Lab problem 1.10: Program for 8-connected flood fills

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void flood(int, int, int, int);
void main()
{
        int gd,gm=DETECT;
        clrscr();
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        rectangle(200,200,250,250);
        flood(220,220,7,0);
        getch();
```

```
}
void flood(int x, int y, int fill_col, int old_col)
{
        if(getpixel(x,y)==old_col)
        {
                delay(10);
                putpixel(x,y,fill_col);
                flood(x+1,y,fill_col,old_col);
                flood(x-1,y,fill_col,old_col);
                flood(x,y+1,fill_col,old_col);
                flood(x,y-1,fill_col,old_col);
                flood(x + 1, y - 1, fill_col, old_col);
                flood(x + 1, y + 1, fill_col, old_col);
                flood(x - 1, y - 1, fill_col, old_col);
                flood(x - 1, y + 1, fill_col, old_col);
        }
}
```

Output



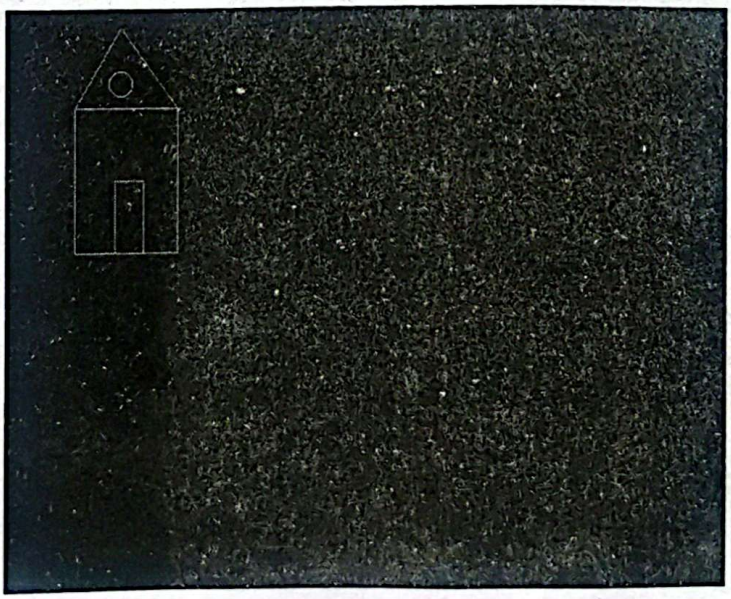
Differences between Food fill and boundary fill algorithms

Flood Fill Algorithm	Boundary Fill Algorithm
Flood fill colors an entire area in an enclosed figure through inters- connected pixels using a single color.	Here, area gets colored with pixels of a chosen color as boundary this giving the technique its name
Flood Fill is one in which all connected pixels of a selected color get replaced by a fill color.	Boundary Fill is very similar with the difference being the program stopping when a given color boundary is found.
A flood fill may use an unpredictable amount of memory to finish because it isn't known how many sub-fills will be spawned	Boundary fill is usually more complicated but it is a linear algorithm and doesn't require recursion
Time Consuming	It is less time Consuming

Lab problem 1.11: Program for creating House shape:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
    setcolor(7);
    rectangle(60,80,150,200);
    rectangle(95,140,120,200);
    line(60,80,100,15);
    line(100,15,150,80);
    circle(100,60,10);
    getch();
    closegraph();
}
```

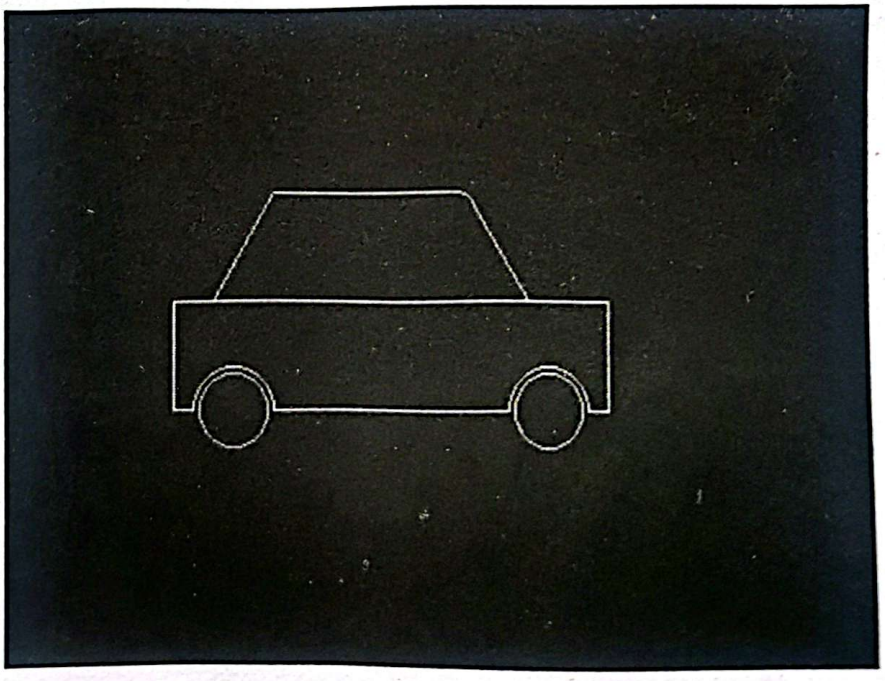
Output



Lab problem 1.12: Program for creating simple car shape:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void main()
{
        int gd = DETECT, gm;
        initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
        cleardevice();
        line( 150, 100,  242, 100);
        ellipse(242, 105, 0, 90, 10, 5);
        line(150, 100,  120, 150);
        line(252, 105,  280, 150);
        line(100, 150,  320, 150);
        line(100, 150,  100, 200);
        line(320, 150,  320, 200);
        line(100, 200,  110, 200);
        line( 320, 200,   310, 200);
        arc(130, 200, 0, 180, 20);
        arc( 290, 200, 0, 180, 20);
        line( 270, 200,  150, 200);
        circle(130, 200, 17);
        circle(290, 200, 17);
        getch();
}
```

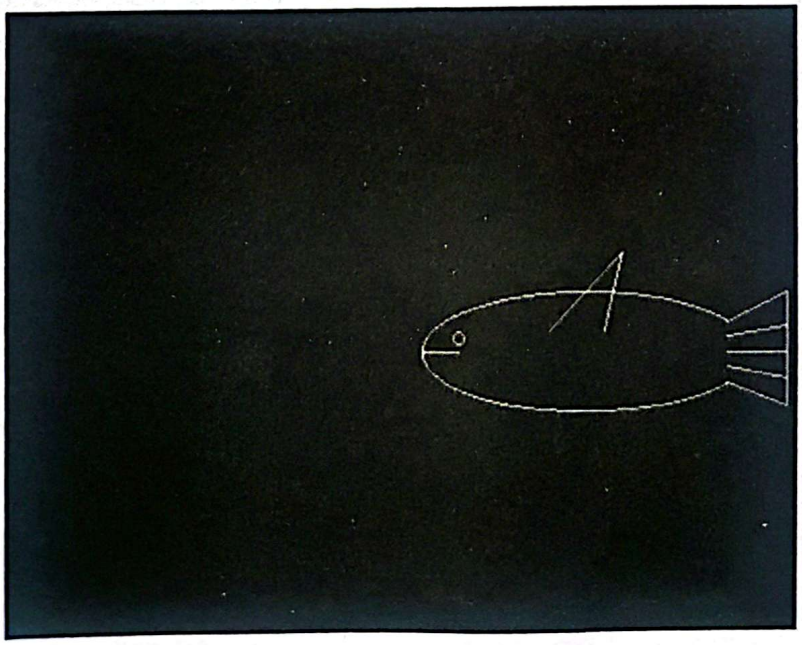
Output



Lab problem 1.13: Program for creating fish:

```
#include<stdlib.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>
#include<ctype.h>
void main()
{
        int gd=DETECT, gm;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        cleardevice();
        ellipse(520,200,30,330,90,30);
         circle(450,193,3);
         line(430,200,450,200);
         line(597,185,630,170);
         line(597,215,630,227);
         line(630,170,630,227);
         line(597,200,630,200);
         line(597,192,630,187);
         line(597,207,630,213);
         line(500,190,540,150);
         line(530,190,540,150);
        getch();
}
```

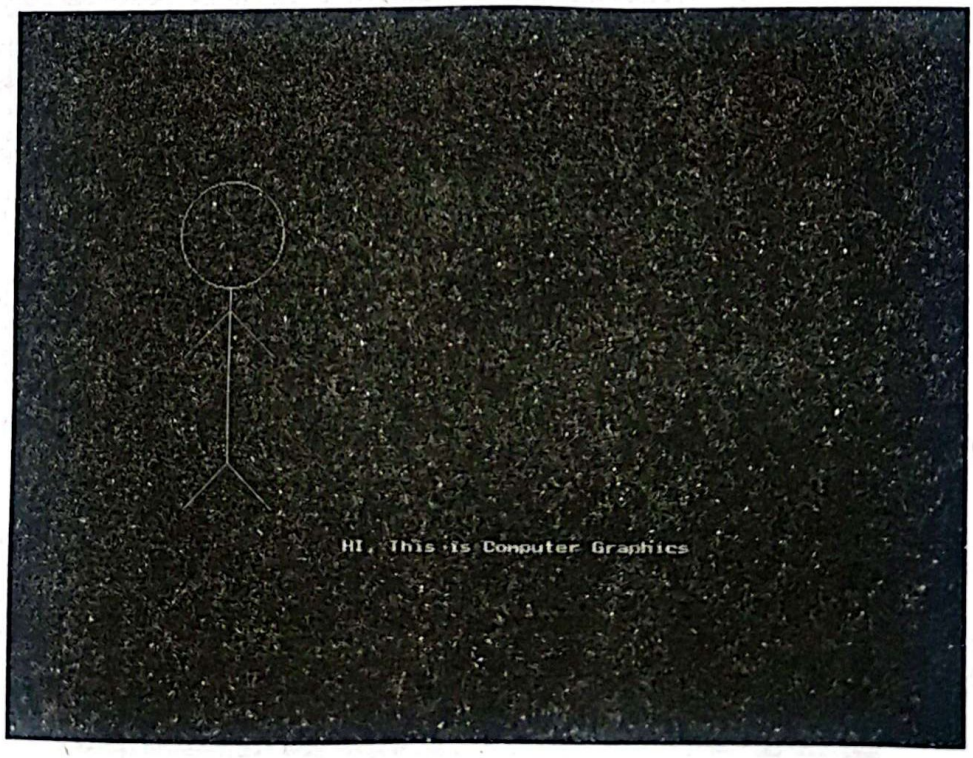
Output

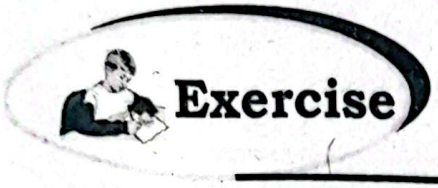


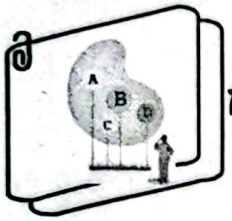
Lab problem 1.14: Program for creating man object

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
void main()
{
        int gd=DETECT,gm;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        setcolor(7);
        circle(150,150,35);
        line(150,185,150,300);
        line(150,200,120,230);
        line(150,200,180,230);
        line(150,300,120,330);
        line(150,300,180,330);
        outtextxy(230,350,"HI, This is Computer Graphics");
        getch();
}
```

Output







Multiple Choice Questions

1. GUI stands for.....
   a) Graphics uniform interaction
   b) Graphical user interaction
   c) Graphical user interface
   d) none of the above
2. What is a pixel mask?
   a) a string containing only 0's
   b) a string containing only 1's
   c) a string containing two 0's
   d) a string containing both 1's and 0's
3. Aspect Ratio can be defined as.....
   a) The ratio of the vertical points to horizontal points
   b) collection of pixels
   c) Both (a) & (b)
   d) None of the above
4. DDA stands for.....
   a) Direct differential analyzer
   b) Data differential analyzer
   c) Direct difference analyzer
   d) Digital differential analyzer
5. Which is not the input device?
   a) Impact printers
   b) Trackball
   c) Mouse
   d) Keyboard
6. Which of the following is an example of the impact device?
   a) Laser printer
   b) Inkjet printer
   c) Line printer
   d) none of the above
7. Random scan systems are used for.....
   a) Color drawing application
   b) Pixel drawing application
   c) Line drawing application
   d) None of the above
8. Shadow mask method is used in.....
   a) Raster scan system
   b) Random scan system
   c) Both (a) & (b)
   d) None of the above
9. Which of the following uses the Beam penetration method?
   a) Raster scan system
   b) Random scan system
   c) Both (a) & (b)
   d) None of the above
10. Which of the following algorithm is used to fill the interior of a polygon?
    a) Boundary fill algorithm
    b) Scan line polygon fill algorithm
    c) Flood fill algorithm
    d) All of the above

Subjective Questions

1. Define Computer Graphics. Why study this course?
2. What is Point in the Computer Graphics System? What are the applications of computer graphics?
3. What do you mean by interactive computer Graphics? What do you mean by GUI? What is the use of graphics in GUI? Explain. What Is Rasterization?
4. What is meant by refreshing of the screen? Write down brief history of computer graphics.
5. What do you mean by image processing? How it is differ from computer graphics?
6. Describe types of computer graphics with suitable example. Describe uses of computer graphics in computer field.
7. Define pixel. What is the use of pixel in display devices?
8. What is the main objective of computer graphics in your study? Plotting is the application of computer graphics. Justify.
9. What is interactive computer system? Explain. What is resolution? What is the impact of resolution in display devices?
10. What is image processing? What are the applications of image processing in computer science?
11. What is a digital differential analyzer (DDA)? How can you draw the line using this algorithm?
12. How computer graphics used in entertainments? Explain. How computer graphics used in education and training? Explain.
13. What is CAD? Describe with suitable example.
14. Difference between Raster Scan Display and Random Scan Display.
15. What is Color CRT Display? Explain Beam-penetration and Shadow-mask method.
16. What are frame grabbers? Are frame buffers different from frame grabbers? Draw a neat block diagram, to explain the architecture of a raster display.
17. Give the logical organization of the video controller in a raster display system.
18. What is refresh buffer? Identify the contents and the organization of the refresh buffer for the case of raster display and vector display.
19. Describe the function of an image scanner. What role does CCD play in an image scanner?
20. How are different shades of color generated on the RGB monitors? What is the role of shadow masks in graphics monitors?
21. What do you understand by VGA and SVGA monitors? What is computer graphics? Indicate five practical applications of computer graphics.
22. Discuss in brief different interactive picture construction techniques. How is color depth and resolution of an image related to the video memory requirement?
23. What is the fundamental difference in the method of operation of a monochrome CRT and a color CRT?
24. Compare storage type CRT against refresh type CRT display. List the important properties of phosphor being used in CRTs.
25. Compare and contrast the operating characteristics of raster refresh systems, plasma panels and LCDs. Briefly explain two main classes of hardware device for user interaction.
26. Discuss DDA line drawing algorithm in detail. Also give its advantages and disadvantages.
27. Describe briefly Bresenham's line drawing algorithm. Why we prefer incremental algorithm over DDA?
28. How DDA line drawings differ from Bresenham's line drawing algorithm?
29. What is scan conversion? Explain Bresenham's incremental circle algorithm for first quadrant assuming all variables as integers.

30. Write an algorithm for scan conversion of circle using Bresenham's method.
31. Differentiate between flood fill and boundary fill algorithms. Describe boundary fill algorithm with suitable example.
32. Describe the architecture of raster scan display. Explain about sweep, octree and boundary representations for solid modeling.
33. Derive the equation to draw a line using DDA algorithm when slope is greater than 1.
34. What is raster scan display system? Explain with its architecture.
35. Derive the expression for Bresenham Line Drawing Application.

Numerical Problems

1. Consider three different raster systems with resolutions of 640 × 480, 1280 × 1024, and 2560 × 2048.
2. If a monitor screen has 525 scan lines and an aspect ratio of 3:4 and if each pixel contains 8 bits worth of intensity information, how many bits per second are required to show 30 frames each second?
3. What size is frame buffer (in bytes) for each of these systems to store 12 bits per pixel?
4. How much storage (in bytes) is required for each system if 24 bits per pixel are to be stored?
5. On an average it takes 20 nano seconds for a Raster Graphics system to access the pixel value from the frame buffer and glow the phosphor dot on the screen. If the total resolution of the screen is 640 x 480 will this access rate produce a flickering effect?
6. Suppose RGB raster system is to be designed using on 8 inch × 10 inch screen with a resolution of 100 pixels per inch in each direction. If we want to store 6 bits per pixel in the frame buffer, how much storage (in bytes) do we need for frame buffer?
7. If you are supposed to create an animated movie of 20 minutes and your video is of 30fps (frames/second). Calculate the number of frames should be in this video.
8. Find out the time required to scan one row of screen if the screen resolution is 20MP (mega pixel), aspect ratio $\frac{5}{4}$ and refresh frequency is 30Hz.
9. How much time is spent scanning across each row of pixels during screen refresh on a raster system with resolution of 1280 × 1024 and a refresh rate of 60 frames per second?
10. Consider three different raster systems with resolutions of 640 × 480, 1280 × 1024, and 2560 × 2048.
    a) What size is frame buffer (in bytes) for each of these systems to store 12 bits per pixel?
    b) How much storage (in bytes) is required for each system if 24 bits per pixel are to be stored?
11. Consider two raster systems with the resolutions of 640 × 480 and 1280 × 1024.
    a) How many pixels could be accessed per second in each of these systems by a display controller that refreshes the screen at a rate of 60 frames per second?
    b) What is the access time per pixel in each system?
12. Given a 25cm × 20cm display operating in 1024 × 768 × 16 color mode which is refreshed 30 times per second, and for which 10% of the refresh cycle is spent in retrace, calculate
    a) the pixel aspect ratio,
    b) the size of the frame buffer, and
    c) The required data transfer rate in kilobytes per second.
13. What is the size of a pixel on a 21-inch diagonal screen with physical aspect ratio 8:5 operating in 1152 × 800 modes?

14. A raster display system operating in 1280 × 1024 mode displays 60 frames per second, and 8% of operating time is spent in retrace. In what amount of time must the video controller be able to wr a pixel to the screen?
15. Consider a raster system with the resolution of 1024 × 768 pixels and the color palette calls f 65,536 colors. What is the minimum amount of video RAM that the computer must have to suppo the above-mentioned resolution and number of colors?
16. How Many k bytes does a frame buffer need in a 600 × 400 pixel?
17. How much time is spent scanning across each row of pixels during screen refresh on a rast system with resolution of 1280 × 1024 and a refresh rate of 60 frames per second?
18. Suppose RGB raster system is to be designed using on 8 inch × 10 inch screen with a resolution o 100 pixels per inch in each direction. If we want to store 6 bits per pixel in the frame buffer, ho much storage (in bytes) do we need for frame buffer?
19. Consider two raster systems with the resolutions of 640 × 480 and 1280 × 1024. How many pixel could be accessed per second in each of these systems by a display controller that refreshes th screen at a rate of 60 frames per second?
20. How long would it take to load a 24-bit per pixel frame buffer with a resolution of 1280 by 102 using this same transfer rate?
21. Consider a non-interlaced raster monitor of resolution 1280 × 1024. If horizontal and vertical retrac times are 20 microseconds each, then calculate the time available to display a pixel. Assume refres rate of 60 frames per second.
22. How much time is spent in scanning across each row of pixels during screen refresh on a raste system with are solution of 1280 by 1024 and a refresh rate of 60 frames per second? Assum horizontal and vertical retrace times are negligible.
23. Consider a raster scan system having 12 inch by 10 inch screen with resolution of 100 pixels pe inch in each direction. If the display controller of this system refreshes the screen at the rate of 5 frames per second, how many pixels could be accessed per second and what is the access time pe second and what is the access time of pre pixels of the system?
36. Digitize the line with endpoints (1, -6) and (4, 4) using BSA/BLA algorithm.
37. Digitize the line with endpoints (1, 6), (6, 10) using BSA algorithm.
38. Trace BSA for line with endpoints (15, 15), (10, 11).
39. Trace BSA for line with endpoints (20, 10), (30, 18).
40. Trace BSA for endpoints (5, 10), (10, 7).
41. Trace BSA for endpoints (20, 30), (30, 22)
42. Digitize the octant of the circle with radius r = 7 and center (20, 30).
43. Digitize the line with endpoints (1, -6) and (4, 4) using DDA algorithm.
44. Digitize the line with endpoints (1, 6), (6, 10) using DDA algorithm.
45. Trace DDA algorithm for line with endpoints (1, 2), (5, 6).
46. Trace DDA algorithm for endpoints (1, 7), (6, 3).
47. Rasterizing the line from (10, 5) to (15, 9) using Bresenham's line drawing Algorithm.
48. Rasterizing the circle with radius r = 5 and center = (100,100) with midpoint circle generatio algorithm.
49. Find the points on a circle one of its octants with the circle centered at (5, 5) and has a radius of 8 units.
50. Digitize an ellipse with center (20, 20) and x-radius = 8 and y-radius = 6.

❑❑❑