Computer Graphics

Practical File

BCA Semester VI

GHANSHYAM

Class Roll no. 22200013

University Roll no. 6220840008

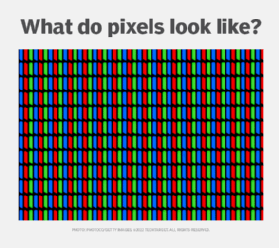
**INDEX**

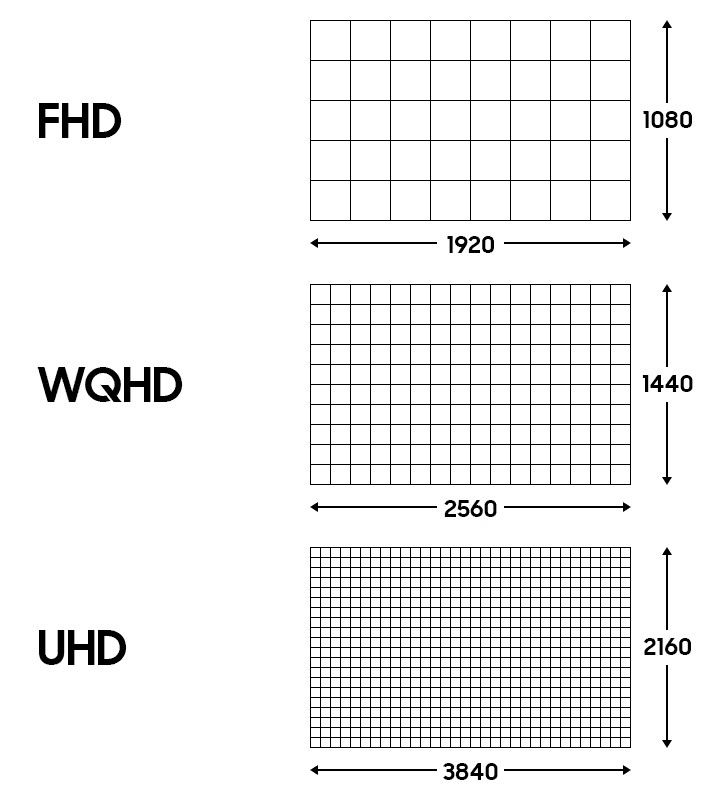
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No.** | **Practical Name / Title** | **Page No.** | **Date** | **Remarks** |
| 1. | Computer Graphics | 1 | 06/03/2025 |  |
| 2. | Applications of Computer Graphics | 2 | 06/03/2025 |  |
| 3. | Cathode Ray Tube | 6 | 08/03/2025 |  |
| 4. | Raster Scan Display | 8 | 09/03/2025 |  |
| 5. | Random Scan Display | 10 | 10/03/2025 |  |
| 6. | Flat Panel Displays | 12 | 11/03/2025 |  |
| 7. | Input Devices | 18 | 14/03/2025 |  |
| 8. | Bresenham’s Line Drawing Algorithm | 22 | 16/03/2025 |  |
| 9. | Mid-Point Circle Drawing algorithm | 24 | 18/03/2025 |  |
| 10. | Ellipse Generating algorithm | 27 | 21/03/2025 |  |

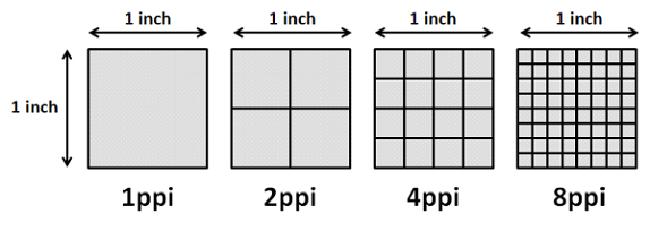
What are Computer Graphics?

**Graphics** refers to visual representations or images, such as drawings, pictures, and designs. It can include anything from sketches and paintings to printed images on paper or digital images on screens. Graphics are used to convey information, ideas, or emotions through visuals rather than words.

So, **Computer Graphics** are those graphics that are created using computers. These can be simple things like drawings or photos, or more complex things like 3D models and animations. Computer graphics are used in many things we see every day, like in video games, movies, websites, and apps.

All computer graphics, whether simple images or complex 3D models, are made up of pixels. A pixel is the smallest unit of a digital image or display, like a tiny dot on the screen. When you zoom in on a graphic, you can see the individual pixels, but from a normal viewing distance, they all blend together to form a clear image.

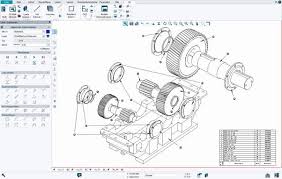
The resolution of an image refers to the number of pixels it contains. A higher resolution means more pixels, and this results in a clearer and more detailed image. For example, an image with a resolution of 1920x1080 has 1920 pixels in width and 1080 pixels in height. More pixels allow for finer details and less pixelation.

PPI, or pixels per inch, is a measure of how many pixels are packed into one inch of a screen. A higher PPI means that more pixels are packed into each inch, making the image appear sharper. For instance, a screen with a high PPI (like 400 PPI) will display images in greater detail than one with a lower PPI (like 100 PPI), which might appear more pixelated.

**1. Computer-Aided Design (CAD)**

CAD is used in engineering, architecture, and product design for creating, modifying, and analysing models.

**Key Features:**

* **Wireframe Models:** Initial 3D designs for structure and shape.
* **Circuit & Network Design:** Laying out electrical circuits, networks, and mechanical systems.
* **Animation & Simulation:** Real-time testing of performance and system behaviour.

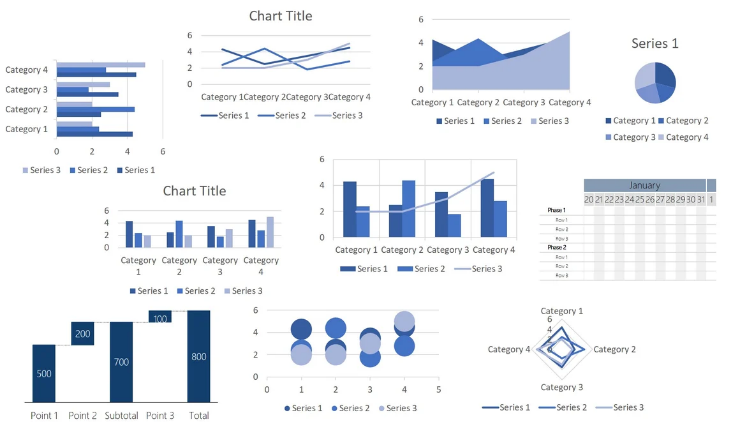
**Applications:**

* Architecture, Automobile & Aerospace, Electronics, Industrial Design.

**2. Presentation Graphics**

Used to simplify data through computer-generated visuals like charts and graphs.

**Key Features:**

* **Bar Charts:** Compare quantities across categories.
* **Line Graphs:** Show data trends over time.
* **Pie Charts:** Represent proportions of a whole.
* **Surface Graphs:** Display relationships between three variables in 3D.
* **2D vs. 3D Graphics:** Simple vs. enhanced data representation.
* **Geographical Integration:** Combines visuals with geographical data.

**Applications:**

* Business Reports, Scientific/Research, Education, Geographical Data.

**3. Computer Graphics in Fine Art & Commercial Art**

Digital tools help create and manipulate art, from traditional simulations to 3D modelling.  
**Fine Art Applications:**

* **Paintbrush Programs:** Simulate traditional brush strokes with styluses.
* **Traditional Medium Simulations:** Mimic watercolour, oil, and pastel effects.

**Commercial Art Applications:**

* **Page Layout:** Combine text and graphics for marketing.
* **Photorealistic Techniques:** Used in lifelike product renderings.
* **Animation in Advertising:** Frame-by-frame rendering for motion ads.
* **Morphing:** Transform objects for creative product demos.

**Examples:**

* Cartoon Drawing, Van Gogh-style Painting, 3D Design & Texture Mapping, Logo Design.

**4. Computer Graphics in Entertainment**

Enhance visual storytelling in movies, TV, music videos, and virtual reconstructions.

**TV Series & Music Videos:**

* **Sci-Fi CGI:** Used for creating space environments and alien planets.
* **Live Action + Graphics:** Blends live-action with CGI for seamless effects.
* **Animates Shows and Cartoons:** Creating a moving picture out of creative shots of different images and illustration.

**Virtual Reconstructions:**

* Recreate historical sites and structures for accuracy.

**Examples:**

* *Star Trek*, *Deep Space Nine*, *David Byrne’s She’s Mad*, cartoons like Tom &Jerry.

**5. Computer Graphics in Education & Training**

Graphics enhance learning through interactive models and simulations.

**Applications:**

* **Educational Models:** Simplified diagrams for complex concepts (e.g., nuclear reactor).
* **Flight & Tank Simulators:** Realistic training tools for pilots and military.
* **Simulated Environments:** Train astronauts and study driver behaviour.

**Viewing Systems in Simulators:**

* **External Viewing Systems:** 360-degree views for realism.
* **Instructor’s Area:** Instructors monitor and adjust simulations.

**6. Data Visualization in Science, Engineering, & Business**

Graphical representations of complex data help identify trends and patterns.

**Types & Techniques:**

* **Color-Coding:** Visual representation of scalar data.
* **Contour Plots:** Show changes in values across surfaces.
* **Surface Renderings:** Visualize 3D data.
* **Volume Visualization:** Techniques for 3D volumetric data.

**Applications:**

* Scientific (e.g., Weather, Aerodynamics), Business (e.g., Traffic, Agriculture).

**7. Image Processing Overview**

Focuses on improving image quality or extracting useful information.

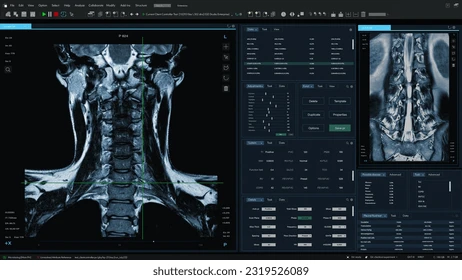
**Techniques:**

* **Quality Enhancement:** Noise reduction, sharpening.
* **Machine Perception:** Helps robots recognize objects and navigate.

**Applications:**

* Medical Imaging (CT, MRI), Robotics.

**8. Medical Applications of Image Processing & Computer Graphics**

Used for diagnosis and surgery planning.

**Applications:**

* **Medical Imaging:** CT, PET scans for detailed body images.
* **Computer-Aided Surgery:** Enhanced images for surgery planning.

**9. Graphical User Interfaces (GUIs)**

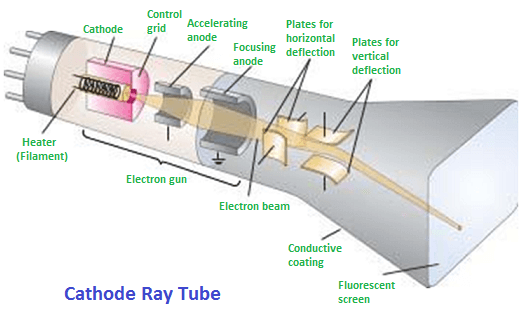
GUIs enable user-friendly interaction with software.

**Key Features:**

* **Window Manager:** Handles multiple windows for task switching.
* **Menus:** Offer options for user interaction (text and icons).
* **Icons:** Small visual representations for quick recognition.

**Example:**

* Multiple windows, menus, and icons for intuitive interaction.

**CRT (Cathode Ray Tube)** is a type of display device that was commonly used in old televisions and computer monitors. It has a large glass tube inside. At the back of the tube, there is an electron gun that shoots a beam of electrons. This beam hits the screen which is coated with special materials (phosphor). When the electrons hit the screen, it lights up and forms images. The beam moves very fast across the screen to draw pictures and text.

So, CRT consists of different components as listed in the diagram. Here is a how cathode ray tube works.

The **electron gun** is the part of a CRT that creates the stream of electrons, also known as cathode rays. It has a heated metal part called the cathode and a control grid. When the cathode is heated using a filament, it releases electrons. These electrons are negatively charged.

The **control grid** helps control how many electrons are sent toward the screen. This is done by applying voltage to the control grid. If the grid has a high negative voltage, it pushes back the electrons. If the voltage is less negative, more electrons can pass through. This controls the brightness of the screen — more electrons mean a brighter spot.

After the electrons are released, they need to be **accelerated and focused**. A high positive voltage pushes them quickly toward the screen. To make the image sharp, the electrons are focused into a small point. This can be done using electric fields (called electrostatic focusing) or magnetic fields (called magnetic focusing).

To make the beam reach different parts of the screen, it is **deflected**, or bent, using either magnetic or electric fields. Most CRTs use magnetic deflection. Magnetic or electric fields are used by placing magnets or plates that guide the beam horizontally and vertically.

When the electron beam hits the **phosphor-coated screen**, the phosphor glows and gives off light. The colour of the light depends on the phosphor material and display technology used, for example, Raster scan display uses **shadow masking,** while the Random scan display uses the **beam penetration** to display colours.

Because the light from the screen fades quickly, the image has to be **refreshed** again and again. The beam redraws the picture many times per second (usually 60 to 80 times). This makes the image look steady to our eyes. A memory area called the **frame buffer** stores the picture data, and this data is used to refresh the screen during every cycle.

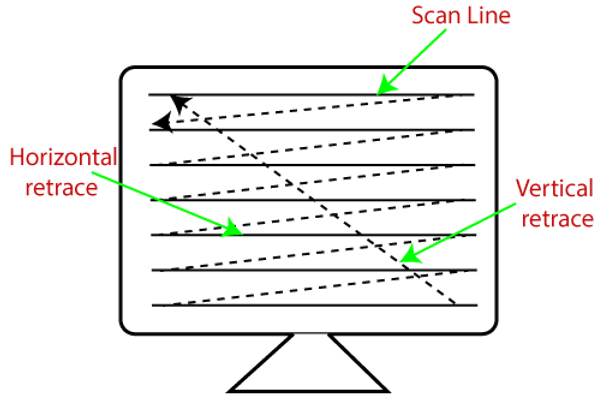
A **raster scan display** is a type of display system where the screen image is created by scanning the electron beam line by line from top to bottom. The beam moves horizontally from left to right to draw one line (called a scan line), then moves down slightly to start the next line, continuing this process until the entire screen is covered. After reaching the bottom, the beam returns to the top to refresh the image. This scanning happens very quickly, many times per second, creating the appearance of a steady and continuous image.

**Working:**

This system shows an image by scanning the screen **row by row**, refreshing pixels one after another to display an image or video. The scanning is done from **top to bottom**, where an **electron beam** moves across each row. The beam is turned **on and off** to make a pixel glow.

The **pixel intensity values** of the beam are stored in a memory area called the **frame buffer** (or **refresh buffer**). These values are retrieved and used to glow each pixel on one row (called a **scan line**) at a time. The intensity values are usually **0 and 1**, where 0 means the beam is **off** and 1 means it is **on**.

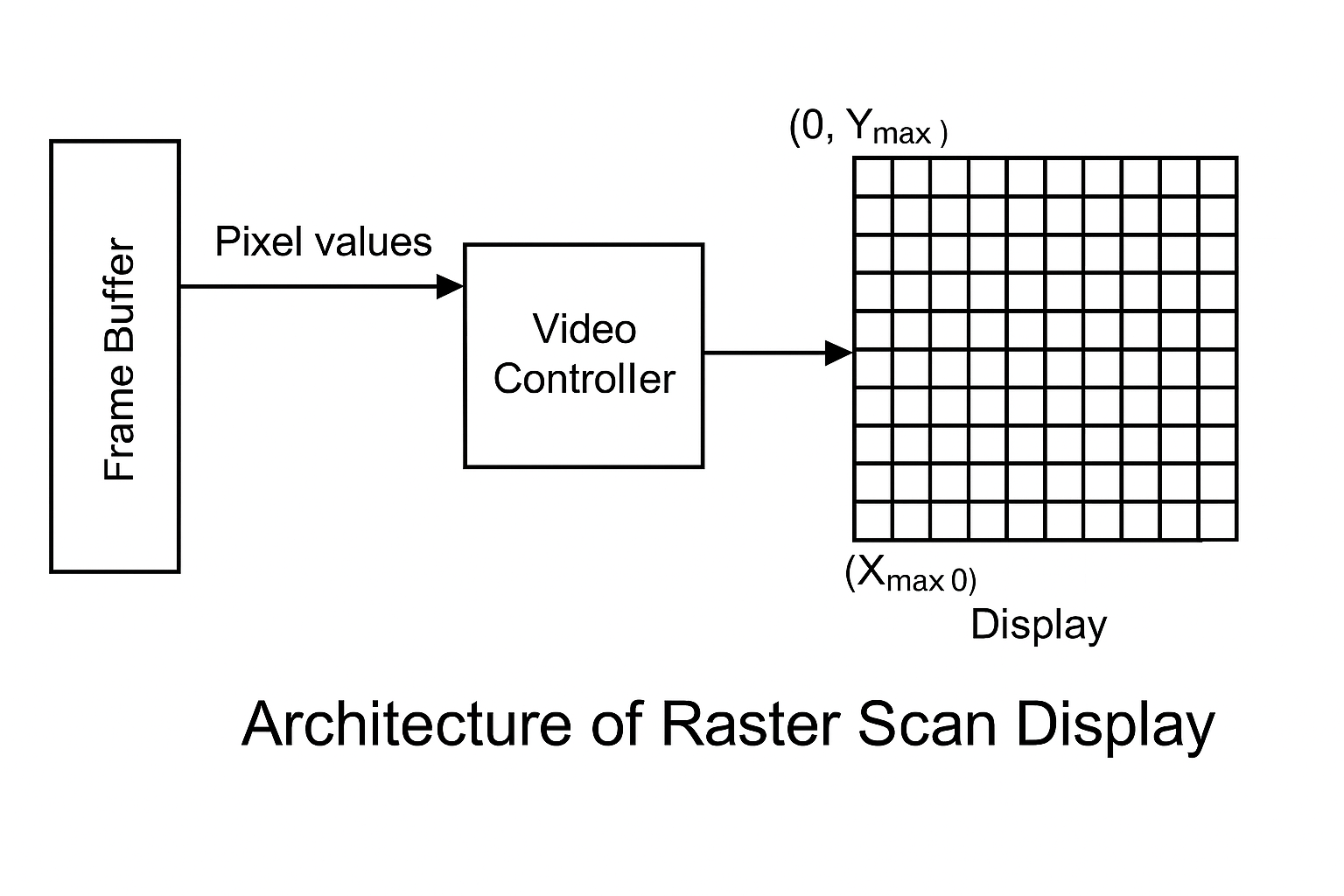
The screen is **refreshed** about **30 to 90 times per second**, which is **30 to 90 FPS (frames per second)**.

The beam scans each line from **left to right**. After scanning one line, it returns to the **left side** again and starts scanning the next line. This returning motion on the horizontal axis is called **horizontal retrace**.

At the end of each frame, the beam returns to the **top-left corner** to start the process again. This is called **vertical retrace**.

**Architecture**

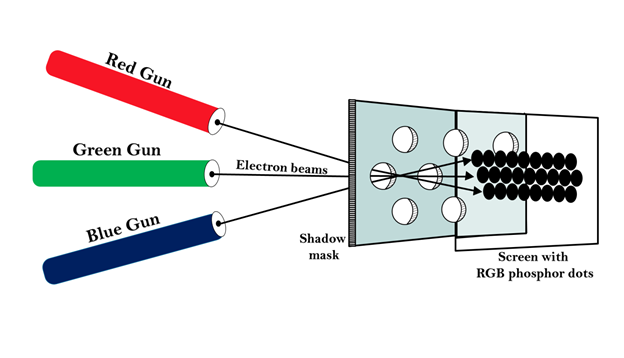
A **raster scan display** uses a **video controller/Display processor** to perform these operations. The video controller reads pixel values from the **frame buffer** and sends them to the display.

1. The video controller starts at the **top-left corner**, setting **X = 0** and **Y = maximum**.
2. The beam moves on the current scan line toward the **right**, and for each pixel, its brightness is set from the value in the frame buffer.
3. Once the scan line ends, **X is reset to 0** (horizontal retrace), and **Y is decreased by 1** to move to the next line.
4. After all scan lines are done, **Y is reset to maximum** (vertical retrace), and the process repeats for the next frame.

**Colour Raster Scan Display:**

To show colours, **raster scan displays** use a method called **shadow masking**.

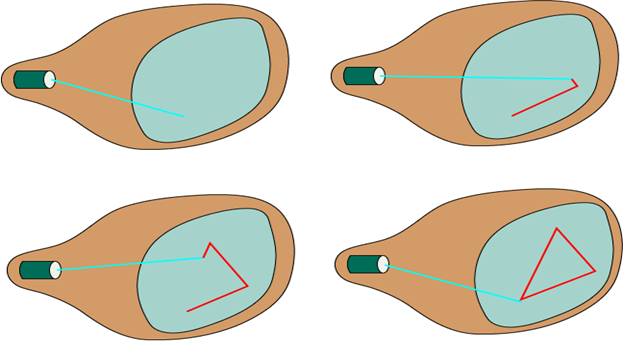
In a **CRT colour display**, each pixel has **three phosphor dots** that glow when hit by an electron beam. These three dots emit **red**, **green**, and **blue** light. There are **three separate electron guns**, one for each colour.

A **shadow mask grid** is placed just behind the screen. This grid has a series of small holes that are perfectly aligned with the coloured phosphor dots on the screen.

When the **three electron beams** pass through the holes in the shadow mask, they are directed in such a way that:

* The **red beam** hits the red dot,
* The **green beam** hits the green dot,
* And the **blue beam** hits the blue dot.

By changing the **intensity** of each of the three beams, the system can produce many different colours. Combining these three colours in different intensities allows the display to show **millions of colour shades**.

**Random Scan Display** is a type of display system where the image is drawn directly on the screen by moving the electron beam only to the parts of the screen where lines or shapes are needed. It does not scan the whole screen like raster displays. Instead, it draws pictures using straight lines from point to point, like using a pen to draw on paper.

This method is mostly used for displaying images made of lines. It is also called a **vector display**.

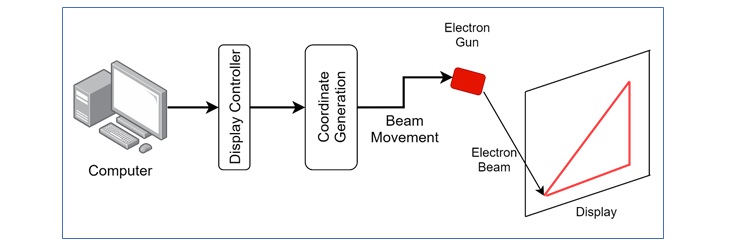
**Working**:

In **Random Scan Displays**, the system uses **line drawing commands** to create images. These commands are usually stored in a **display file** in the computer’s memory.

The **graphics package** interprets these commands and creates the display file. This file contains the data needed to draw the graphics on the screen.

A **graphics/display/video controller** reads the display file and refreshes the screen by controlling the electron beam. During each **refresh cycle**, the graphics processor loops through the drawing commands.

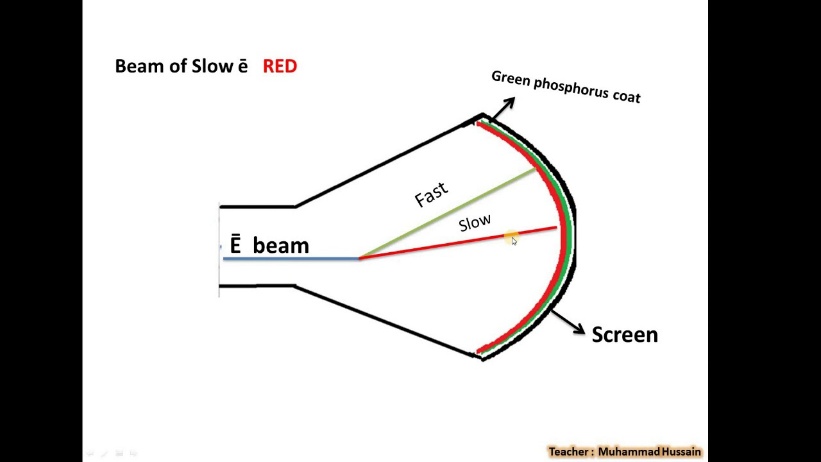
The **graphics controller/video controller/display controller** directs the electron beam on the screen using the **coordinates** of the endpoints of the lines. These **X and Y values** are converted into **deflection voltages**, which guide the beam to draw each line correctly.

****The image is drawn **one line at a time** by positioning the beam between the given endpoints, similar to how you draw a shape with a pen from point to point.

**Colour Random Scan Display:**

To show colours, **random scan displays** use a method called **Beam penetration.**

In **beam penetration**, the random scan display uses a screen coated with **two layers of phosphor**. The **first layer is red**, and the **second layer is green**. The colour that appears on the screen depends on **how deep the electron beam penetrates** these two layers.

* If the **beam has low power**, it only hits the **red layer**, so the screen shows **red colour**.
* If the **beam has high power**, it goes deeper and hits the **green layer**, showing **green colour**.
* If the beam power is **in between**, it hits both layers partially. This results in a **mix of red and green**, producing **yellow or orange**, depending on the beam’s intensity.

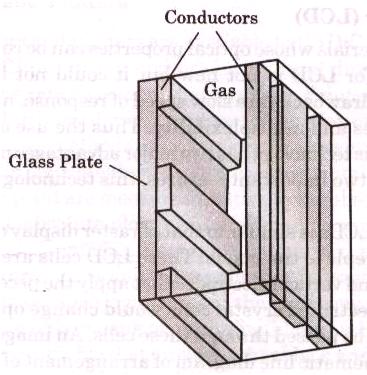
With this method, the display can produce a **maximum of four colours**: red, green, yellow, and orange.

Flat-panel displays are a modern class of video devices that have rapidly gained popularity due to their **thin profiles, lightweight construction, and lower power requirements** compared to traditional cathode ray tube (CRT) monitors. These displays have made it possible to design **portable, space-saving, and energy-efficient systems** such as laptops, tablets, handheld game consoles, calculators, and wall-mounted televisions. Their compact nature also allows for innovative applications, such as wrist-worn screens and digital notepads. Flat-panel displays can be broadly categorized into two types: **Emissive** and **Non-Emissive Displays**, based on how they handle light generation and display formation.

**Emissive Displays**

Emissive displays are a type of flat-panel display that **produce light by converting electrical energy directly into visible light**. This means they do not require any external source of illumination to form the image on the screen. The light-emitting nature of these displays allows for high brightness and excellent contrast, making them suitable for various types of digital displays. The most commonly used emissive displays include **Plasma Panels**, **Thin-Film Electroluminescent Displays**, and **Light-Emitting Diode (LED) Displays**.

**1 Plasma Panels**

Plasma panels, also known as **gas-discharge displays**, operate by filling the space between two tightly sealed glass plates with a low-pressure gas mixture, often containing neon. Embedded within the inner surfaces of these glass plates are **conductive ribbons** — one set aligned vertically on one panel and the other set aligned horizontally on the opposite panel. The crossing of a vertical and horizontal conductor creates a **pixel location**.

When a voltage is applied to a specific pair of these conductors, the gas at their intersection ionizes, forming a **plasma** — a state of matter consisting of glowing electrons and ions. This plasma emits visible light, thus illuminating the selected pixel. The entire image is controlled via a **refresh buffer**, which stores the pixel data and refreshes the screen approximately **60 times per second** to maintain the image. Over time, advancements in technology have improved plasma displays to support **colour and grayscale output**, though they were initially monochromatic.

**Features:**

* High brightness and good contrast.
* Capable of displaying high-definition images.
* Can be constructed at large sizes, suitable for public displays.

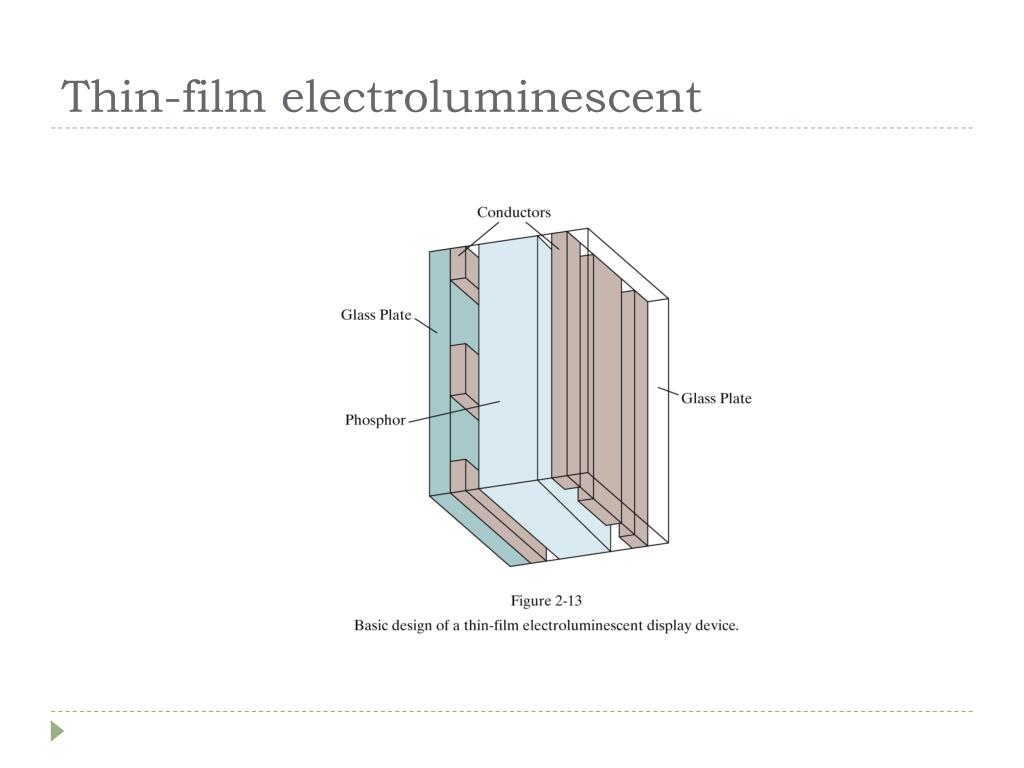
**Applications:**

* Televisions.
* Airport and railway information boards.
* Commercial advertising displays.

**Limitations:**

* Relatively high-power consumption.
* Susceptible to **burn-in** effects when static images are displayed for long periods.
* Not ideal for small devices due to size and energy needs.

**2 Thin-Film Electroluminescent Displays**

Thin-film electroluminescent displays share a similar basic structure with plasma panels but replace the gas mixture with a solid **phosphor**, typically zinc sulphide doped with manganese. The structure consists of two glass plates with horizontal and vertical electrodes embedded in them, sandwiching a thin layer of the electroluminescent material.

When a sufficiently high voltage is applied across a pair of intersecting electrodes, the material at the intersection becomes electrically conductive. This energy excites the manganese atoms within the phosphor, which in turn emit visible light as they return to a lower energy state. The light produced at each activated intersection forms the display image.

**Features:**

* Durable and rugged design suitable for harsh environments.
* Uniform brightness over a wide range of viewing angles.

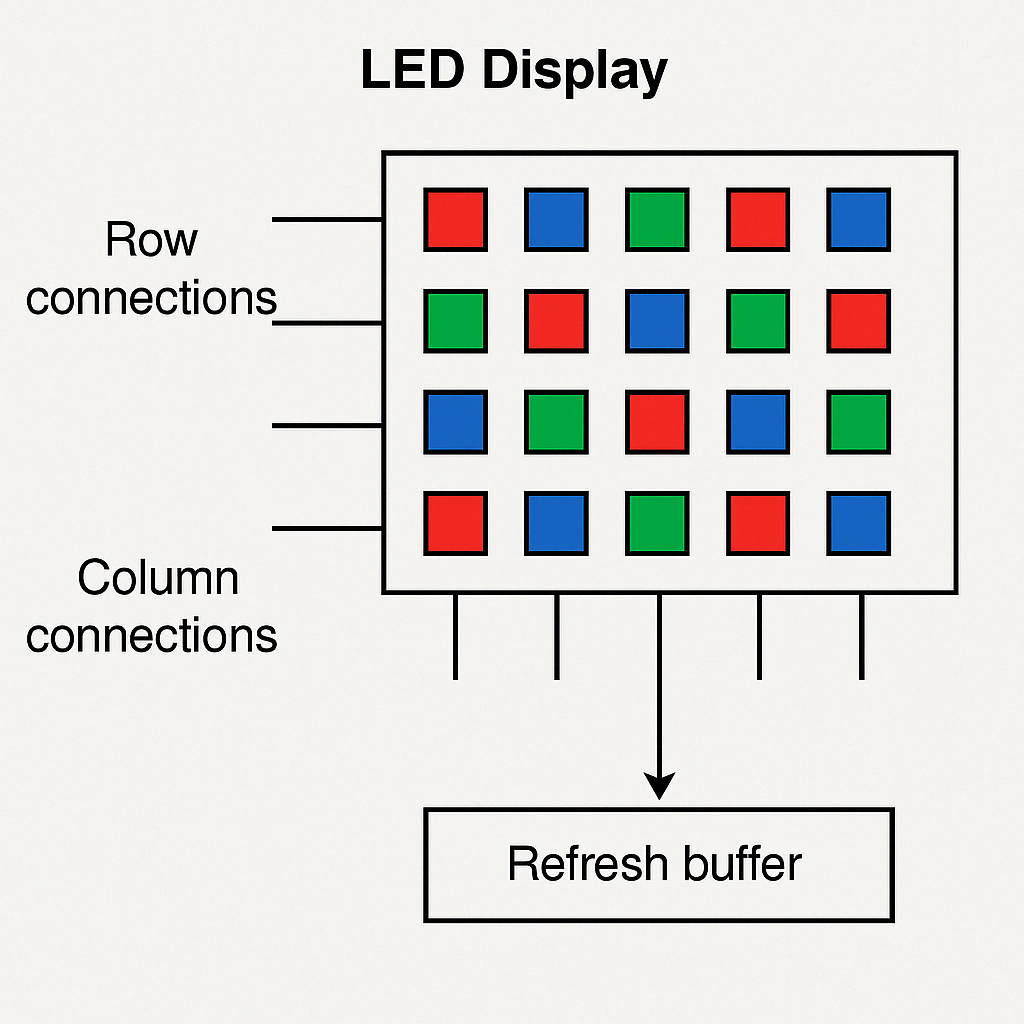
**Applications:**

* Military and industrial equipment.
* Medical instrumentation panels.
* Outdoor signs and information boards.

**Limitations:**

* Requires more power than plasma displays.
* Achieving vibrant colour and smooth grayscale remains difficult.
* Not widely used in consumer electronics due to cost and complexity.

**3 Light-Emitting Diode (LED) Displays**

LED displays are made up of a grid or matrix of **semiconductor diodes** that emit light when an electric current passes through them. Each LED functions as an individual pixel, and by arranging red, green, and blue LEDs together, full-colour displays can be achieved. These displays are controlled by a **refresh buffer** which stores the image and sends signals to the appropriate diodes to illuminate and form the visual output.

LED displays are known for their **brightness, energy efficiency, and long lifespan**. Unlike plasma and electroluminescent displays, LEDs are easier to scale both up and down in size, making them suitable for everything from massive stadium displays to tiny smartwatch screens.

**Features:**

* High energy efficiency and long life.
* Capable of producing very bright images.
* Available in both monochrome and full-colour formats.

**Applications:**

* Televisions and computer monitors.
* Smartwatches and wearable devices.
* Traffic signs and outdoor digital billboards.

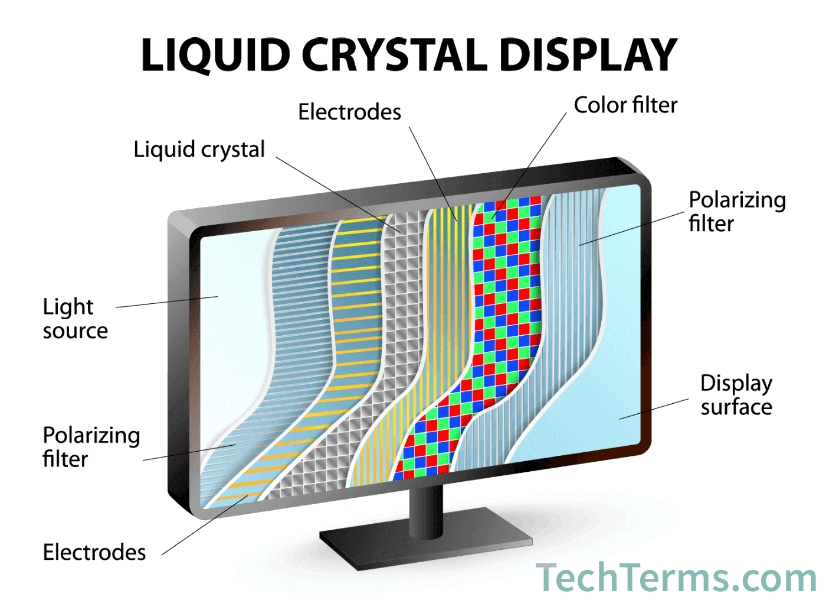
**Limitations:**

* Can suffer from heat buildup in dense or large arrays.
* Manufacturing full-colour displays requires complex circuitry.
* Slight colour variation among LEDs can affect image uniformity.

**Non-Emissive Displays**

Unlike emissive displays, non-emissive displays do not produce light on their own. Instead, they rely on **external light sources**, such as ambient light or built-in backlighting, to make the image visible. These displays work by **modulating or controlling the passage of light** using special materials. The most widely used non-emissive display technology is the **Liquid Crystal Display (LCD)**, which can be further divided into **Passive-Matrix** and **Active-Matrix (TFT)** displays.

**1 Liquid Crystal Displays (LCDs)**

Liquid Crystal Displays work using materials that exhibit both **liquid and crystalline properties**. The type commonly used in displays is called **nematic liquid crystal**, where the molecules tend to align in a particular direction but can still flow like a liquid. An LCD is built using two glass plates with a thin layer of liquid crystal between them. These plates also include **transparent electrodes** arranged in horizontal and vertical lines, forming a matrix of intersections where pixels are created.

Additionally, the inside surfaces of the plates have **light polarizers** oriented at right angles to each other. In the **default “on” state**, the liquid crystal molecules twist the incoming polarized light so it can pass through the second polarizer and reflect to the viewer, making the pixel appear bright. When a voltage is applied to the electrodes, the molecules align themselves in such a way that the light is **not twisted** and is blocked by the second polarizer, turning the pixel dark.

To ensure visibility in all lighting conditions, many LCDs use **LED backlighting**. Colour is achieved by placing **coloured filters or sub-pixel arrangements** (typically red, green, and blue) at each pixel location.

**Features:**

* Very low power consumption, especially for static images.
* Lightweight and suitable for battery-powered devices.
* Excellent for displaying sharp text and images.

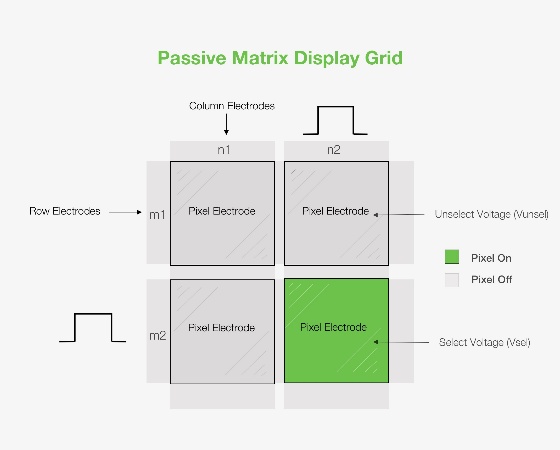
**Applications:**

* Laptop screens, desktop monitors.
* Digital watches, calculators, and thermometers.
* Mobile phones, tablets, and televisions.

**Limitations:**

* Requires an external or backlight source.
* Limited viewing angles, especially in passive models.
* Slower response time compared to emissive displays.

**2 Passive-Matrix LCD**

****Passive-Matrix LCDs are the simpler and earlier version of LCD technology. In these displays, the control of each pixel is achieved through **a basic grid of electrodes** with no switching elements at the pixel level. Each pixel is activated by the voltage applied to the corresponding row and column.

While cost-effective and easy to manufacture, passive-matrix LCDs suffer from **slow response times**, **limited brightness**, and **low contrast ratios**. Due to these shortcomings, they are mostly used in devices where **simple, low-power display output** is sufficient.

**Features:**

* Simple design with low manufacturing cost.
* Minimal power usage for static images.

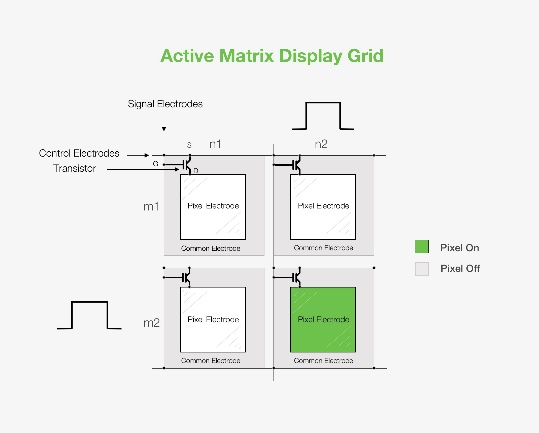
**Applications:**

* Basic calculators and clocks.
* Digital thermometers and portable meters.

**Limitations:**

* Poor image quality with slow refresh rates.
* Not suitable for dynamic or multimedia content.

**3 Active-Matrix LCD (Thin-Film Transistor - TFT LCD)**

Active-Matrix LCDs use a **thin-film transistor (TFT)** at each pixel location. These transistors act like tiny switches that control the voltage applied to the liquid crystal cell, holding the charge long enough to maintain the image until the next refresh. This results in a **much faster response time**, **better brightness**, and **higher resolution** compared to passive-matrix LCDs.

The addition of TFT technology has made active-matrix LCDs the standard choice for **high-quality colour displays** in modern devices. These displays also offer better viewing angles and smoother motion rendering.

**Features:**

* Fast refresh rate and high responsiveness.
* Supports high-resolution, full-colour output.
* Wide viewing angles and excellent image quality.

**Applications:**

* Smartphones and tablets.
* Laptops and desktop monitors.
* Televisions and digital cameras.

**Limitations:**

* More complex and expensive to manufacture.
* Slightly higher power consumption than passive LCDs.

**1. Keyboard**

A keyboard is used to type letters, numbers, and symbols into a computer. It is one of the most common input devices.

**Working:**

* Inside the keyboard, there is a grid of wires arranged in rows and columns (called a key matrix).
* Every key is located where a row and column meet.
* When you press a key, it completes a circuit at that point.
* A small chip inside the keyboard (microcontroller) checks the grid again and again.
* When it sees a connection (key press), it finds which key was pressed.
* It changes that into a digital signal and sends it to the computer.

**2. Optical Mouse**

An optical mouse is used to move the pointer (cursor) on the computer screen. It also has buttons to click and scroll.

**Working:**

* It has a tiny LED light that shines on the surface under the mouse.
* A sensor inside takes many quick pictures of the surface.
* The mouse checks how the pictures change when you move it.
* A chip calculates how far and in which direction the mouse has moved.
* It sends this data to the computer to move the pointer.
* When you click a button, a small switch is pressed inside, sending a signal to perform an action.

**3. Z-Mouse**

A Z-mouse is a special mouse that can also detect movement up and down (Z-axis), not just side to side.

**Working:**

* It has extra sensors like accelerometers and pressure sensors.
* These sensors detect if the mouse is being lifted, pressed harder, or tilted.
* This allows it to understand depth and 3D movement.
* A chip collects all this data and sends it to the computer.
* It is useful in 3D software, games, or design work.

**4. Trackball**

A trackball is like a mouse, but instead of moving the whole device, you move a ball on it with your fingers.

**Working:**

* A large ball is placed in the middle or side of the device.
* When you rotate the ball, sensors inside detect how the ball is moving.
* The sensors read the movement in left-right and up-down directions.
* The trackball sends this data to the computer to move the pointer.
* It also has buttons for clicking, just like a mouse.

**5. Joystick**

A joystick is a stick-like device used mostly in games and simulators to control movement in all directions.

**Working:**

* When you move the stick, it changes position in different directions.
* Sensors like Hall Effect or potentiometers detect this change.
  + Hall sensors read magnetic changes.
  + Potentiometers measure resistance as the stick moves.
* The joystick converts these changes into signals.
* A microcontroller processes these signals and sends them to the computer.
* The computer then moves the object or character on screen.

**6. Data Glove**

A data glove is a special glove used to control things in a virtual or 3D space using hand movements.

**Working:**

* The glove has small bend sensors along the fingers and palm.
* When you move or bend your fingers, the sensors detect the bending.
* This bending creates changes in the electrical signal (analogue signals).
* These signals are converted into digital data.
* A processor sends the data to the computer, which shows the hand movements on the screen.
* It is commonly used in virtual reality and robotics.

**7. Digitizer**

A digitizer is a flat tablet used to draw or write with a pen-like tool. It turns your handwriting or drawing into digital form.

**Working:**

* The tablet surface has a grid of tiny sensors (electromagnetic or capacitive).
* The pen has a coil or metal tip, depending on the tablet type.
* When you touch the pen on the surface, it creates a small signal.
* The tablet reads the exact position (X and Y) of the pen.
* The data is sent to the computer to draw or write what you create.
* Some digitizers can also sense pen pressure to make lines thick or thin.

**8. Light Pen**

A light pen is a pen-like device that can be used to point at or draw on the screen.

**Working:**

* The tip of the pen has a light-sensitive part (like a small camera or diode).
* When the screen flashes light (as it does while refreshing), the pen detects the light.
* The computer checks the exact time the flash happened.
* Based on timing, the computer finds the exact spot on the screen.
* It helps select or draw directly on the display.

**9. Touch Panel (Touchscreen)**

A touch panel is a screen that works when you touch it with your finger or a stylus.

**Working:**

* **Resistive Touchscreen**: Two thin layers inside the screen are separated by a gap.
  + When you press, they touch each other, and the touch point is recorded.
* **Capacitive Touchscreen**: It has a special layer that holds electric charge.
  + When you touch it, your finger changes the electric field at that spot.
* Sensors detect this change and find the X-Y coordinates of the touch.
* The processor sends this information to the computer or device to perform actions like tap, swipe, etc.

**10. OCR (Optical Character Recognition)**

OCR is used to read printed or handwritten text from images or scanned documents and turn them into editable text.

**Working:**

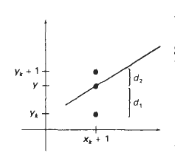
* First, a scanner or camera captures an image of the text.
* The image is cleaned to make the letters clear (removes background noise).
* OCR software looks at the shapes of letters in the image.
* It compares these shapes with a set of known characters.
* When a match is found, it turns that shape into digital text.
* The final result is text that you can edit on the computer.

**11. MICR (Magnetic Ink Character Recognition)**

MICR is mostly used in banks to read information printed with magnetic ink on checks.

**Working:**

* Special ink with iron oxide (magnetic) is used to print numbers and codes.
* A MICR machine reads the printed area with a magnetic head.
* The magnetic field from each character creates a different signal.
* These signals are read and matched with stored character patterns.
* The machine then converts the signal into digital characters.
* It helps in fast and secure check processing in bank

Bresenham’s Line Drawing Algorithm is a method used to draw straight lines on a digital grid (like a computer screen or graphics display) by calculating which pixels should be glowed to represent a straight line.

It works by choosing the nearest pixel to the ideal straight line and gradually "drawing" the line pixel by pixel from one endpoint to the other. It makes decisions about which pixels to glow based on the difference between the ideal line and the pixel, and it uses only integer math (no floating-point calculations).

**Algorithm:**

1. Input the two-line endpoints and store the left endpoint in (x₀, y₀).
2. Load (x₀, y₀) into the frame buffer; that is, plot the first point.
3. Calculate the constants:

Δx = x₁ - x₀  
Δy = y₁ - y₀  
2Δy  
2Δy - 2Δx

Then, compute the starting value for the decision parameter:  
p₀ = 2Δy - Δx

1. At each xₖ along the line, starting at k = 0, perform the following test:
   * If pₖ < 0:  
     The next point to plot is (xₖ + 1, yₖ)  
     Update decision parameter:  
     pₖ₊₁ = pₖ + 2Δy
   * Else:  
     The next point to plot is (xₖ + 1, yₖ + 1)  
     Update decision parameter:  
     pₖ₊₁ = pₖ + 2Δy - 2Δx
2. Repeat step 4 a total of Δx times.

**Example:**

*#include* <graphics.h>

*#include* <iostream>

*using* *namespace* std;

*void* drawBresenhamLine(*int* x1, *int* y1, *int* x2, *int* y2, *int* screenHeight) {

*// Flip y-coordinates*

    y1 *=* screenHeight *-* y1;

    y2 *=* screenHeight *-* y2;

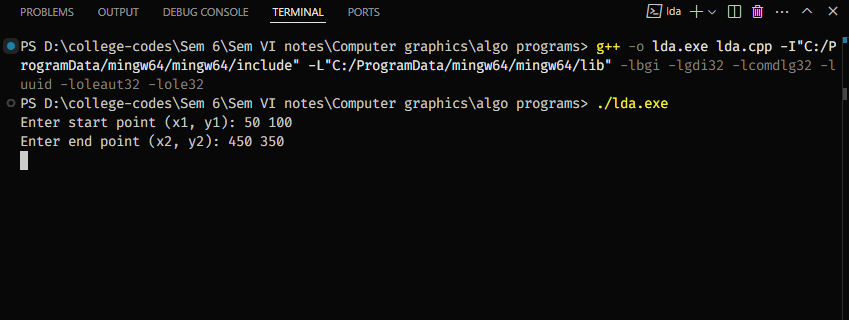
*int* dx *=* abs(x2 *-* x1), dy *=* abs(y2 *-* y1);

*int* sx *=* (x1 *<* x2) *?* 1 *:* *-*1;

*int* sy *=* (y1 *<* y2) *?* 1 *:* *-*1;

*int* err *=* dx *-* dy;

*while* (true) {

        putpixel(x1, y1, WHITE);

*if* (x1 *==* x2 *&&* y1 *==* y2) *break*;

*int* e2 *=* 2 *\** err;

*if* (e2 *>* *-*dy) { err *-=* dy; x1 *+=* sx; }

*if* (e2 *<* dx) { err *+=* dx; y1 *+=* sy; }

    }

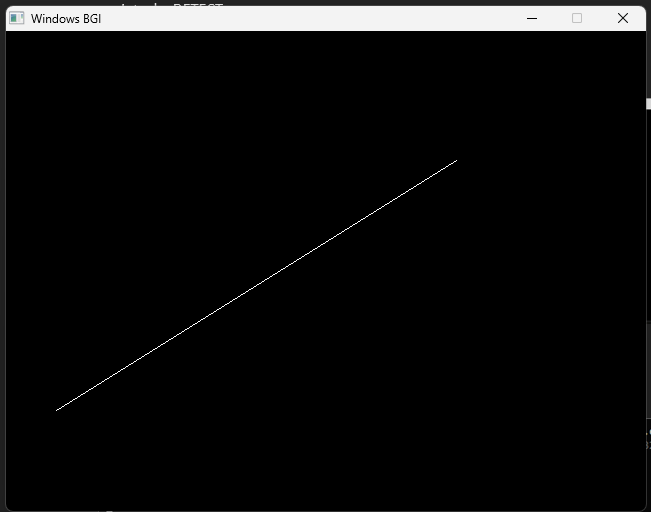
}

*int* main() {

*int* gd *=* DETECT, gm;

    initgraph(*&*gd, *&*gm, NULL);

*int* screenHeight *=* getmaxy(); *// Get window height*

    *int* x1, y1, x2, y2;

    cout *<<* "Enter start point (x1, y1): ";

    cin *>>* x1 *>>* y1;

    cout *<<* "Enter end point (x2, y2): ";

    cin *>>* x2 *>>* y2;

    drawBresenhamLine(x1, y1, x2, y2, screenHeight);

    getch();

    closegraph();

*return* 0;

}

A circle is a closed figure where each point on its boundary is at a certain distance (radius) from its center. It can be described as a set of points that are at an equal distance, R, from the center (X₀, Y₀).

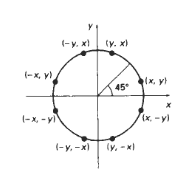
The general equation of a circle is:

(X - H)² + (Y - K)² = R²

Where (H, K) represents the centre of the circle. If the circle is cantered at the origin, then the equation becomes:

X² + Y² = R²

To draw a circle on a digital grid, we divide the circle into eight equal sections (octants) using the symmetry of the circle. This method makes use of the fact that the circle has symmetrical properties, which allow us to plot only a portion of the circle and reflect it into the other parts.

* **Symmetry of the Circle:**
  + At **0 degrees** on the circumference, the points (X, Y) will become (Y, X) at **90 degrees**.
  + The circle is symmetric across both axes, so the points in one quadrant can be reflected to the other quadrants by switching their signs.
* **Coordinate Quadrants:**
  + **First Quadrant**: Both X and Y are positive.
  + **Second Quadrant**: X is negative, while Y remains positive.
  + **Third Quadrant**: Both X and Y are negative.
  + **Fourth Quadrant**: Y is negative, while X remains positive.

To draw a circle efficiently:

* The circle is divided into 8 equal parts (octants).
* By calculating the coordinates for one octant, we can use the symmetry of the circle to determine the coordinates for the other seven parts.
* For example, once the points for the first octant are determined, they can be reflected to the other octants, making the process faster and more efficient.

By drawing these smaller sections of the circle simultaneously or one after another, we can plot the entire circle on the screen, leveraging symmetry to reduce computational effort.

**Algorithm:**

1. Input the radius **r** and circle centre **(x₀, y₀)**, and obtain the first point on the circumference of a circle cantered on the origin as:  
     **(x₀, y₀) = (0, r)**
2. Calculate the initial value of the decision parameter as:  
     **p₀ = 5⁄4 − r**
3. At each **xₖ** position, starting at **k = 0**, perform the following test:

  • If **pₖ < 0**, the next point along the circle cantered on (0, 0) is **(xₖ₊₁, yₖ)**  
    – Update decision parameter:  
      **pₖ₊₁ = pₖ + 2xₖ₊₁ + 1**

  • Otherwise, the next point along the circle is **(xₖ₊₁, yₖ₋₁)**  
    – Update decision parameter:  
      **pₖ₊₁ = pₖ + 2xₖ₊₁ + 1 − 2yₖ₊₁**

  • Where:  
    – **2xₖ₊₁ = 2x + 2**  
    – **2yₖ₊₁ = 2y − 2**

1. Determine the symmetry points in the other seven octants.
2. Move each calculated pixel position **(x, y)** onto the circular path cantered on **(xc, yc)** and plot the coordinate values:  
     **x = x + xc**  
     **y = y + yc**
3. Repeat steps 3 through 5 until **x ≥ y**

**Example:**

*#include* <graphics.h>

*#include* <iostream>

*using* *namespace* std;

*void* plotCirclePoints(*int* xc, *int* yc, *int* x, *int* y, *int* screenHeight) {

    putpixel(xc *+* x, screenHeight *-* (yc *+* y), WHITE);

    putpixel(xc *-* x, screenHeight *-* (yc *+* y), WHITE);

    putpixel(xc *+* x, screenHeight *-* (yc *-* y), WHITE);

    putpixel(xc *-* x, screenHeight *-* (yc *-* y), WHITE);

    putpixel(xc *+* y, screenHeight *-* (yc *+* x), WHITE);

    putpixel(xc *-* y, screenHeight *-* (yc *+* x), WHITE);

    putpixel(xc *+* y, screenHeight *-* (yc *-* x), WHITE);

    putpixel(xc *-* y, screenHeight *-* (yc *-* x), WHITE);

}

*void* drawBresenhamCircle(*int* xc, *int* yc, *int* r, *int* screenHeight) {

*int* x *=* 0, y *=* r;

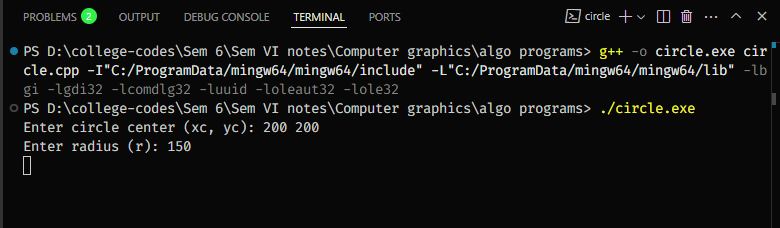
*int* d *=* 3 *-* 2 *\** r; *// Decision parameter*

*while* (x *<=* y) {

        plotCirclePoints(xc, yc, x, y, screenHeight);

        x*++*;

*if* (d *<* 0)

            d *+=* 4 *\** x *+* 6;

*else* {

            y*--*;

            d *+=* 4 *\** (x *-* y) *+* 10;

        }

    }

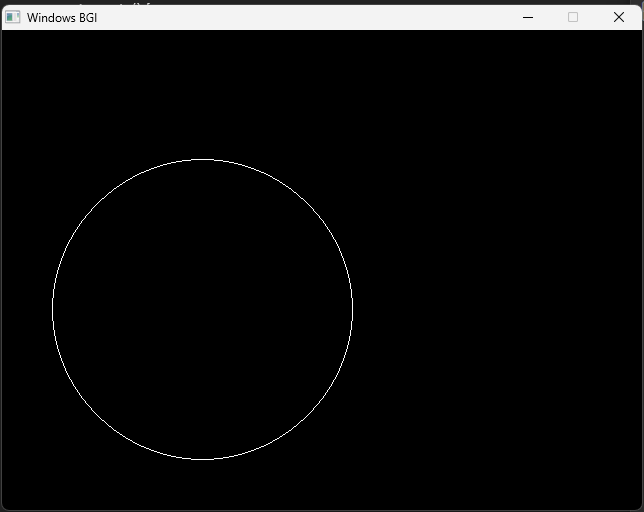
}

*int* main() {

*int* gd *=* DETECT, gm;

    initgraph(*&*gd, *&*gm, NULL);

*int* screenHeight *=* getmaxy(); *// Get screen height*

    *int* xc, yc, r;

    cout *<<* "Enter circle center (xc, yc): ";

    cin *>>* xc *>>* yc;

    cout *<<* "Enter radius (r): ";

    cin *>>* r;

    drawBresenhamCircle(xc, yc, r, screenHeight);

    getch();

    closegraph();

*return* 0;

}

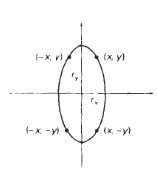
An ellipse is like a stretched-out circle. It has two important dimensions:

* **Major axis**: The longest diameter (across the widest part).
* **Minor axis**: The shortest diameter (perpendicular to the major axis).

And a unique property:

* For every point on the ellipse, the **sum of its distances to two fixed points (foci)** is constant.

**Standard Equation of an Ellipse**

**If an ellipse is cantered at origin with semi-major axis **rₓ** and semi-minor axis **rᵧ**, and aligned with the x and y axes, its equation is:

**ry2 x2 + rx2 y2 -rx2 ry2 = 0**

In Midpoint Ellipse drawing algorithm, the circle is divided into 4 equal parts or **quadrants**.

Using the symmetrical property of the ellipse, that is, there are 4 symmetrical points on the circumference of the ellipse, the ellipse is drawn as quadrants one after another using these symmetry points.

**Algorithm**

1. Input rₓ, rᵧ, and ellipse centre (xₐ, yₐ), and obtain the first point on an ellipse cantered on the origin as:  
   **(x₀, y₀) = (0, rᵧ)**
2. Calculate the initial value of the decision parameter in region 1 as:  
   **p₁₀ = rᵧ² − rₓ²rᵧ + (1/4)rₓ²**
3. At each xₖ position in region 1, starting at k = 0, perform the following test:  
     If **p₁ₖ < 0**, the next point along the ellipse cantered on (0, 0) is (xₖ₊₁, yₖ) and  
       **p₁ₖ₊₁ = p₁ₖ + 2rᵧ²xₖ₊₁ + rᵧ²**  Otherwise, the next point along the circle is **(xₖ + 1, yₖ − 1)** and  
       **p₁ₖ₊₁ = p₁ₖ + 2rᵧ²xₖ₊₁ − 2rₓ²yₖ₊₁ + rᵧ²**  with  
       **2rᵧ²xₖ₊₁ = 2rᵧ²xₖ + 2rᵧ²  
       2rₓ²yₖ₊₁ = 2rₓ²yₖ − 2rₓ²**  and continue until **2rᵧ²x ≥ 2rₓ²y**
4. Calculate the initial value of the decision parameter in region 2 using the last point **(x₀, y₀)** calculated in region 1 as:  
     **p₂₀ = rᵧ² (x₀ + 1/2)² + rₓ² (y₀ − 1)² − rₓ²rᵧ²**
5. At each **yₖ** position in region 2, starting at **k = 0**, perform the following test:  
     If **p₂ₖ > 0,** the next point along the ellipse cantered on (0, 0) is (xₖ, yₖ − 1) and  
       **p₂ₖ₊₁ = p₂ₖ − 2rₓ²yₖ₊₁ + rₓ²**  Otherwise, the next point along the circle is **(xₖ + 1, yₖ − 1)** and  
       **p₂ₖ₊₁ = p₂ₖ + 2rᵧ²xₖ₊₁ − 2rₓ²yₖ₊₁ + rₓ²**  using the same incremental calculations for **x** and **y** as in **region 1.**
6. Determine **symmetry points** in the **other three quadrants**.
7. Move each calculated pixel position (x, y) onto the elliptical path cantered on **(xₐ, yₐ)** and plot the coordinate values:  
   **x = x + xₐ  
     y = y + yₐ**
8. Repeat the steps for region **1** until **2rᵧ²x ≥ 2rₓ²y**

**Example:**

*#include <graphics.h>*

*#include <iostream>*

*using namespace std;*

*void plotEllipsePoints(int xc, int yc, int x, int y, int screenHeight) {*

*putpixel(xc + x, screenHeight - (yc + y), WHITE);*

*putpixel(xc - x, screenHeight - (yc + y), WHITE);*

*putpixel(xc + x, screenHeight - (yc - y), WHITE);*

*putpixel(xc - x, screenHeight - (yc - y), WHITE);*

*}*

*void drawMidpointEllipse(int xc, int yc, int rx, int ry, int screenHeight) {*

*float dx, dy, d1, d2;*

*int x = 0, y = ry;*

*// Region 1*

*d1 = (ry \* ry) - (rx \* rx \* ry) + (0.25 \* rx \* rx);*

*dx = 2 \* ry \* ry \* x;*

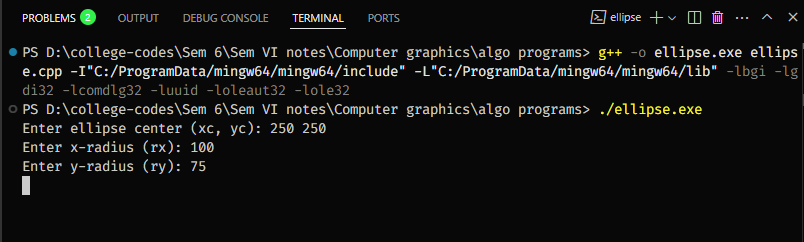
*dy = 2 \* rx \* rx \* y;*

*while (dx < dy) {*

*plotEllipsePoints(xc, yc, x, y, screenHeight);*

*x++;*

*dx += 2 \* ry \* ry;*

*        if (d1 < 0)*

*d1 += dx + (ry \* ry);*

*else {*

*y--;*

*dy -= 2 \* rx \* rx;*

*d1 += dx - dy + (ry \* ry);*

*}*

*}*

*// Region 2*

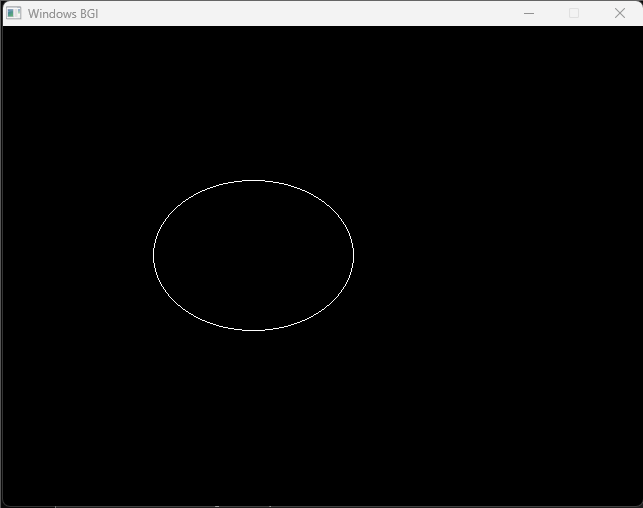
*d2 = ((ry \* ry) \* ((x + 0.5) \* (x + 0.5))) + ((rx \* rx) \* ((y - 1) \* (y - 1))) - (rx \* rx \* ry \* ry);*

*while (y >= 0) {*

*plotEllipsePoints(xc, yc, x, y, screenHeight);*

*y--;*

*dy -= 2 \* rx \* rx;*

*        if (d2 > 0)*

*d2 += (rx \* rx) - dy;*

*else {*

*x++;*

*dx += 2 \* ry \* ry;*

*d2 += dx - dy + (rx \* rx);*

*}*

*}*

*}*

*int main() {*

*int gd = DETECT, gm;*

*initgraph(&gd, &gm, NULL);*

*int screenHeight = getmaxy(); // Get screen height*

*int xc, yc, rx, ry;*

*cout << "Enter ellipse center (xc, yc): ";*

*cin >> xc >> yc;*

*cout << "Enter x-radius (rx): ";*

*cin >> rx;*

*cout << "Enter y-radius (ry): ";*

*cin >> ry;*

*drawMidpointEllipse(xc, yc, rx, ry, screenHeight);*

*getch();*

*closegraph();*

*return 0;*

*}*