OVERVIEW OF COMPUTER GRAPHICS AND OPENGL

Syllabus:

Overview: *Computer Graphics hardware and software and OpenGL: Computer Graphics: Video Display Devices, Raster-Scan Systems Basics of computer graphics, Application of Computer Graphics.* ***OpenGL:*** *Introduction to OpenGL, coordinate reference frames, specifying two-dimensional world coordinate reference frames in OpenGL, OpenGL point functions, OpenGL line functions, point attributes, line attributes, curve attributes, OpenGL point attribute functions, OpenGL line attribute functions, Line drawing algorithms(DDA, Bresenham's).*

Resources:

1. *Donald D Hearn, M Pauline Baker and Warren Carithers: Computer Graphics with OpenGL 4th Edition, Pearson, 2014*
2. *https://www.geeksforgeeks.org/*
3. *https://www.javatpoint.com/*

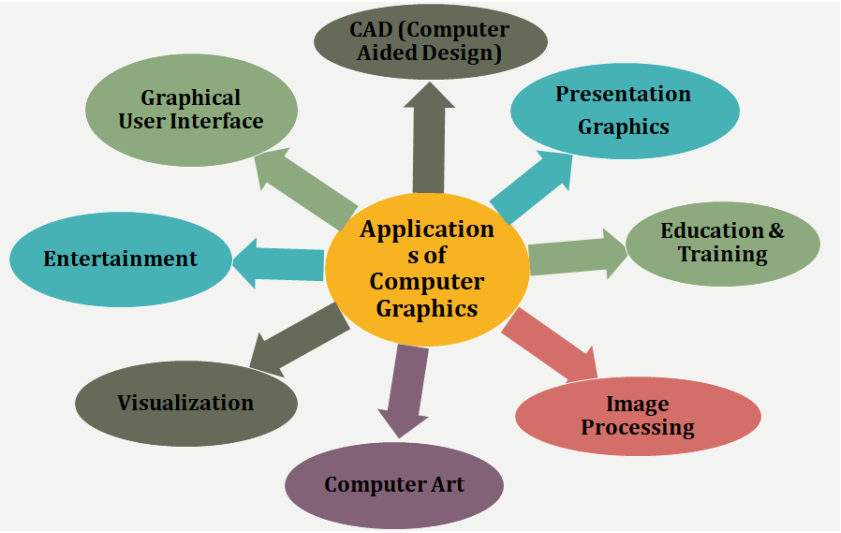
What is Computer Graphics?

- **Computer graphics** deals with all aspects of creating images with a computer.-H/W, S/W and Application.
- 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.
- Use of computer to define, store, manipulate, interrogate and present pictorial output.
- In other words, **Computer graphics** is concerned with all aspects of producing pictures or images using a computer.

Applications of Computer Graphics

The development of computer graphics has been driven both by the needs of the user community and by advances in hardware and software.

Some of the applications of computer graphics are:



1. Computer Aided Design (CAD)

- Computer graphics are mainly used in the design process, these applications are especially used in the engineering field. For example - Making buildings and other structural designs,

designing automobiles and aircraft, etc.

- Computer-Aided Design (CAD) is a type of computer-based tool used for drafting and designing.
- CAD is useful in various designing fields such as architecture, mechanical and electrical, automobiles, aircraft, spacecraft, computers, textiles and much more.
- Software packages for CAD applications typically provide the designer with the multi-window environment, i.e., a different view of objects.

2. Presentation Graphics

- Presentation graphics is used to summarize financial, statistical, mathematical, scientific, data for research reports and other types of reports.
- Typical examples of presentation graphics are bar charts, line graphs, surface graphs, pie charts, and other displays showing relationships between multiple parameters

3. Education & Training

- Today, graphics are being used extensively for education. Education is being visualized through graphics so that students are getting more interested in reading.
- Computer-generated models of physical, financial, and economic systems are often used, which can help students to understand the operation of the system.
- Graphics are also used in the field of training so that the trainee can get real experience of learning.
- **For example** - A simulator is a device that is used for training. The simulator gives you a real driving experience through graphics.
- You will not feel that you are not driving a real car, but you will feel that you are driving a real car. There are many types of simulators like - flight simulators, four wheeler simulators.

4. Computer Art

- Computer graphics methods are widely used in both fine art and commercial art applications.
- Artists use a variety of computer methods, including special- purpose hardware, artist's paintbrush programs specially developed software, symbolic mathematics packages.
- Fine artists use a variety of other computer technologies to produce images.
- Using graphics designing, you can create any type of graphics image, such as - Graphics templates, YouTube thumbnails, post feature images, Facebook post images, birthday cards & many more.

5. Entertainment

- Computer graphics methods are now commonly used in making motion pictures, music videos, and television shows.
- Computer graphics methods are used regularly in many movies, TV series for generating some graphics scene. Sometimes the graphics objects are combined with the actors and live scenes.

6. Graphical User Interface

- Our interaction with computers has become dominated by a visual paradigm that includes windows, icons, menus, and a pointing device, such as a mouse.

- More recently, millions of people have become users of the Internet. Their access is through graphical network browsers, such as Firefox, Chrome, Safari, and Internet Explorer that use these same interface tools.

7. Image processing

- Processing of existing images into refined ones for better interpretation is one of the many applications of computer graphics.
- Computer graphics are also used in the field of photography so that different types of pictures can be improved or their shortcomings can be overcome, which is called image processing.
- You can also edit the image using graphics software.
- Edit Means - You can modify any image according to your need, such as - you can increase or decrease the brightness of the image and you can do many other changes.

8. Visualization

- The use of visualization is used by big scientists, professional doctors, and experienced engineers to study a large amount of information about data.
- The weather department also uses visualization to obtain weather information. So that the information about the data of a field can be studied properly.

A Graphics System

A computer graphics system is a computer system with all the components of a general-purpose computer system as shown in the block diagram in Figure 1.1.

There are six major elements in our system:

1. Input devices
2. Central Processing Unit
3. Graphics Processing Unit
4. Memory
5. Frame buffer
6. Output devices

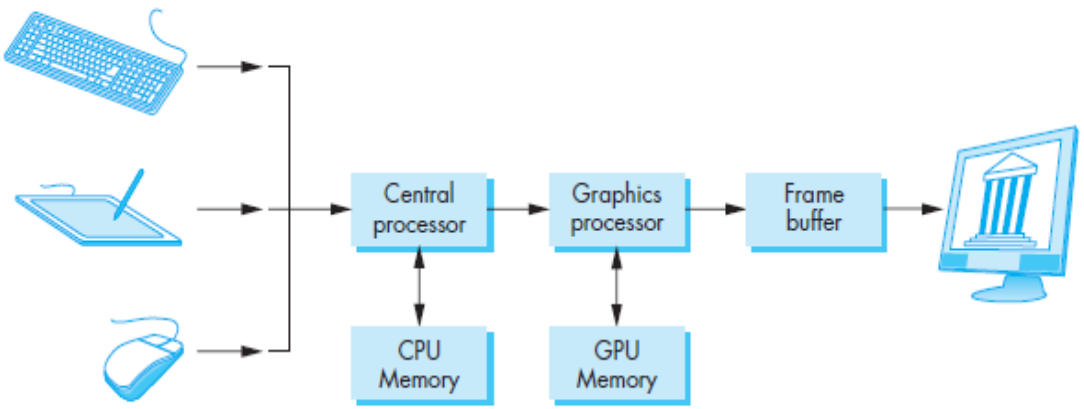


FIGURE 1.1 A graphics system.

Input Devices

- Most graphics systems provide a keyboard and at least one other input device. The most common input devices are the mouse, the joystick, and the data tablet.
- Each provides positional information to the system, often called **pointing devices**. Each such **pointing device** usually is equipped with one or more buttons to provide signals to the processor.
- 3D location on a real- world object can be obtained by **laser range finders** & **acoustic sensors**
- Higher-dimensional data can be obtained by devices such as data gloves, which may include many sensors, & computer vision systems.

The CPU and the GPU

In a simple system, there may be only one processor, the **central processing unit** (**CPU**) of the system, which must do both the normal processing and the graphical processing. The main graphical function of the processor is to take specifications of graphical primitives (such as lines, circles, and polygons) generated by application programs and to assign values to the pixels in the frame buffer that best represent these entities. The conversion of geometric entities to pixel colors and locations in the frame buffer is known as **rasterization**, or **scan conversion**.

In early graphics systems, the frame buffer was part of the standard memory that could be directly addressed by the CPU. Today, virtually all graphics systems are characterized by special-purpose **graphics processing units** (**GPUs**), custom-tailored to carry out specific graphics functions. The GPU can be either on the mother board of the system or on a graphics card. The frame buffer is accessed through the graphics processing unit.

Pixels and the Frame Buffer

- Now almost all modern graphics systems are raster based. The image we see on the output device is an array—the **raster**—of picture elements, or **pixels**, produced by the graphics system.
- Collectively, the pixels are stored in a part of memory called the **frame buffer**.
- The **resolution**—the number of pixels in the frame buffer—determines the detail that you can see in the image.
- The **depth**, or **precision**, of the frame buffer, defined as the number of bits that are used for each pixel, determines properties such as how many colors can be represented on a given system. For example,
    - a 1-bit-deep frame buffer allows 2^1=2 colors
    - whereas an 8-bit-deep frame buffer allows 2^8 = 256 colors.
    - In **full-color** systems, there are 24 (or more) bits per pixel. They are also called **true-color** systems, or **RGB-color** systems,
- In a very simple system, the frame buffer holds only the colored pixels that are displayed on the screen. In most systems, the frame buffer holds far more information, such as depth information needed for creating images from three-dimensional data.

- We can use the terms ***frame buffer*** and ***color buffer*** synonymously without confusion.

Output Devices

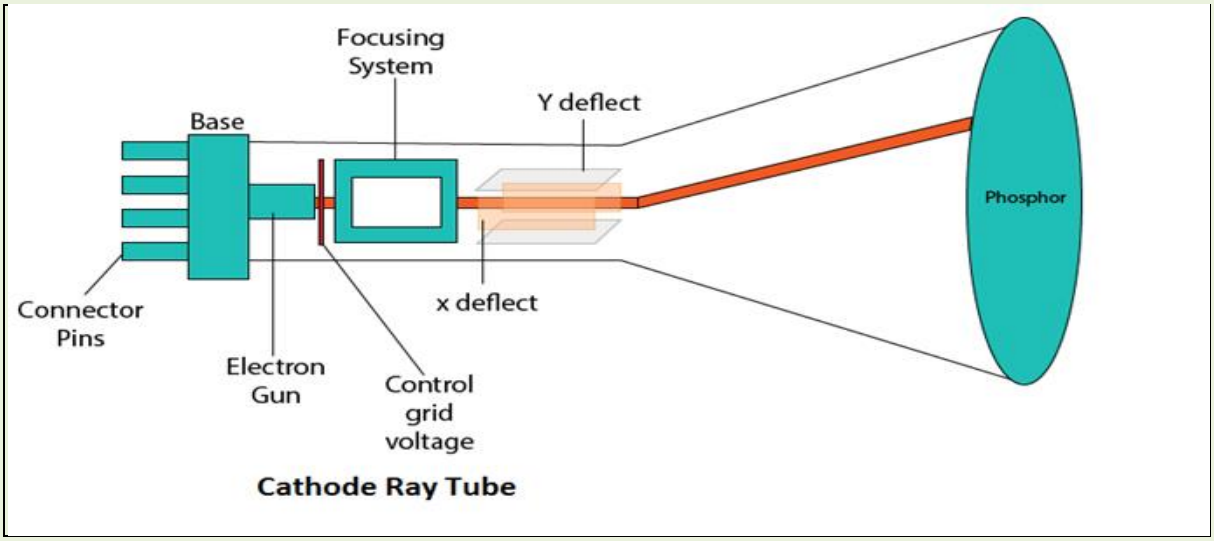
For many years, the type of display (or **monitor**) has been the **cathode-ray tube** (**CRT**).

Video Display Devices

For many years, the type of display (or **monitor**) has been the **cathode-ray tube** (**CRT**). Other popular display types:

- Liquid Crystal Display Plasma display
- Field Emission Displays
- Digital Meromirror Devices
- Light Emitting Diodes
- 3D display devices (hologram or page scan methods)

Refresh Cathode-Ray Tube (CRT):



Components of CRT:

1. Electron Gun:

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

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

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

4. Deflection Yoke:

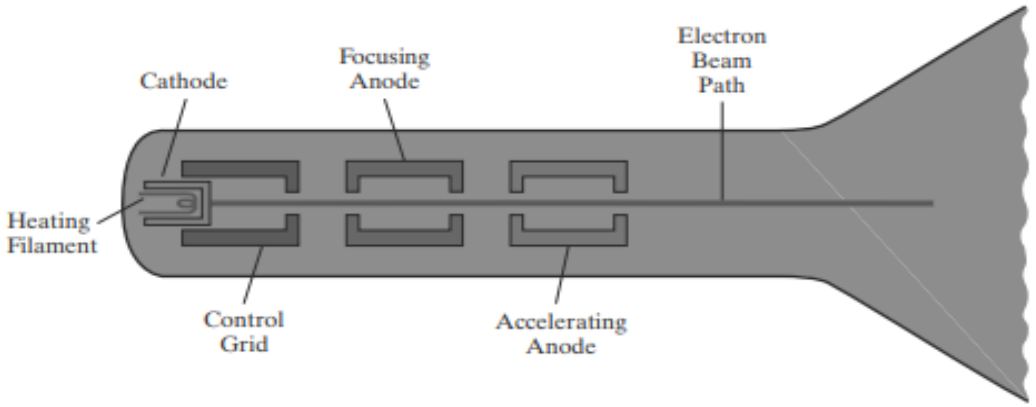
- It is used to control the direction of the electron beam.
- It creates an electric or magnetic field which will bend the electron beam as it passes through the area. In a conventional CRT, the yoke is linked to a sweep or scan generator.
- The deflection yoke which is connected to the sweep generator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

5. Phosphorus-coated screen:

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

Operation:

1. A beam of electrons (cathode rays) is emitted by an electron gun.
2. It then passes through focusing and deflection systems that direct the beam toward specified positions on the phosphor-coated screen.
3. The phosphor then emits a small spot of light at each position contacted by the electron beam. The color you view on the screen is produced by a blend of red, blue and green light.
4. Because the light emitted by the phosphor fades very rapidly, some method is needed for maintaining the screen picture.
5. One way to do this is to store the picture information as a charge distribution within the CRT, which is used to keep the phosphors activated.
6. However, the most common method now employed for maintaining phosphor glow is to redraw the picture repeatedly by quickly directing the electron beam back over the same screen points. This type of display is called a **refresh CRT**.
7. The frequency at which a picture is redrawn on the screen is referred to as the **Refresh Rate.**



Electron Gun With An Accelerating Anode

- The idea behind an electron gun is to create electrons and then accelerate them to a very high speed.
- The primary components of an electron gun in a CRT are:
  - The Heated Metal Cathode and
  - A Control Grid .

Heated Metal Cathode:

- Heat is supplied to the cathode by directing a current through a coil of wire, called the filament, inside the cylindrical cathode structure.
- The electron gun starts with a small heater, which is a lot like the hot, bright filament of a regular light bulb. It heats a cathode, which emits a cloud of electrons. Two anodes turn the cloud into an electron beam:

- The accelerating anode attracts the electrons and accelerates them toward the screen.
- The focusing anode turns the stream of electrons into a very fine beam.

Control Grid:

- Control grid is used to surround the cathode. Grid is cylindrical in shape. It is made up of metal.
- Grid has hole at one end, through which electrons get escaped.
- The control grid is kept at lower potential as compared to cathode, so that a electrostatic field can be created.
- It will direct that electrons through point source, so process of focusing will be simplified.

Focusing System

- The focusing system is to create a clear picture by focusing the electrons into a narrow beam. Otherwise, electrons would repel each other and beam would spread out as it reaches the screen.
- Focusing is accomplished with either electric or magnetic fields.

Deflection System

- Deflection of the electron beam can be controlled by either electric fields or magnetic fields.
- In case of magnetic field, two pairs of coils are used, one for horizontal deflection and other for vertical deflection.
- In case of electric field, two pairs of parallel plates are used, one for horizontal deflection and second for vertical deflection as shown in figure above.

CRT Screen

- The inside of the large end of a CRT is coated with a fluorescent material that gives off light when struck by electrons.
- When the electrons in the beam is collides with phosphor coating screen, they stopped and their kinetic energy is absorbed by the phosphor.
- Then a part of beam energy is converted into heat energy and the remainder part causes the electrons in the phosphor atom to move up to higher energy levels.

Persistence

- It is defined as the time they continue to emit light after the CRT 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 without flicker.
- A phosphor with low persistence is useful for animation; a high-persistence phosphor is useful for displaying highly complex, static pictures.

Resolution

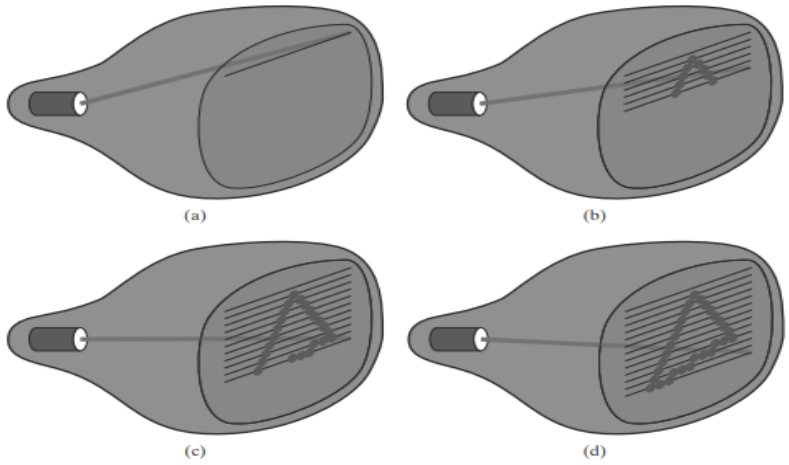
- The number of points per centimeter that can be used be plotted horizontally and vertically. Or Total number of points in each direction.
- The resolution of a CRT is depend on
- type of phosphor

- intensity to be displayed

Aspect Ratio

- It is ratio of horizontal to vertical points.
- Example: An aspect ratio of 3/4 means that a vertical line plotted with three points has same length as horizontal line plotted with four points.

Raster-Scan Displays



- The most common type of graphics monitor employing a CRT is the **raster-scan display**, based on television technology.
- In a **raster-scan system**, the electron beam is swept across the screen, **one row at a time, from top to bottom.** Each row is referred to as a **scan line**. As the electron beam moves across a **scan line**, the beam intensity is turned on and off to create a pattern of illuminated spots.
- Picture definition is stored in a memory area called the **refresh buffer or frame buffer**, where the term **frame** refers to the total screen area.
- This memory area holds the set of color values for the screen points which are then retrieved from the refresh buffer and used to control the intensity of the electron beam as it moves from spot to spot across the screen.
- In this way, the picture is "painted" on the screen one scan line at a time, as demonstrated in Figure above.
- Each screen spot that can be illuminated by the electron beam is referred to as a **pixel** or **pel** (shortened forms of **picture element**).
- Since the refresh buffer is used to store the set of screen color values, it is also sometimes called a **color buffer.** Also, other kinds of pixel information, besides color, are stored in buffer locations, so all the different buffer areas are sometimes referred to collectively as the **"frame buffer."**
- Beam refreshing is of two types. First is **horizontal retracing** and second is **vertical retracing**.
- When the beam starts from the top left corner and reaches the bottom right scale, it will again return to the top left side called at **vertical retrace.** Then it will again more horizontally from top to bottom call as **horizontal retracing** shown in figure.

Types of Scanning or travelling of beam in Raster Scan

1. Interlaced Scanning
2. Non-Interlaced Scanning

1. In **Interlaced scanning**, each horizontal line of the screen is traced from top to bottom. Due to which fading of display of object may occur. This problem can be solved by Non-Interlaced scanning.
2. In **Non-Interlaced scanning,**
   - First all odd numbered lines are traced or visited by an electron beam, then in the next circle, even number of lines are located.
   - For non-interlaced display refresh rate of 30 frames per second used. But it gives flickers.
   - For interlaced display refresh rate of 60 frames per second is used. This is an effective technique for avoiding flicker—provided that adjacent scan lines contain similar display information.

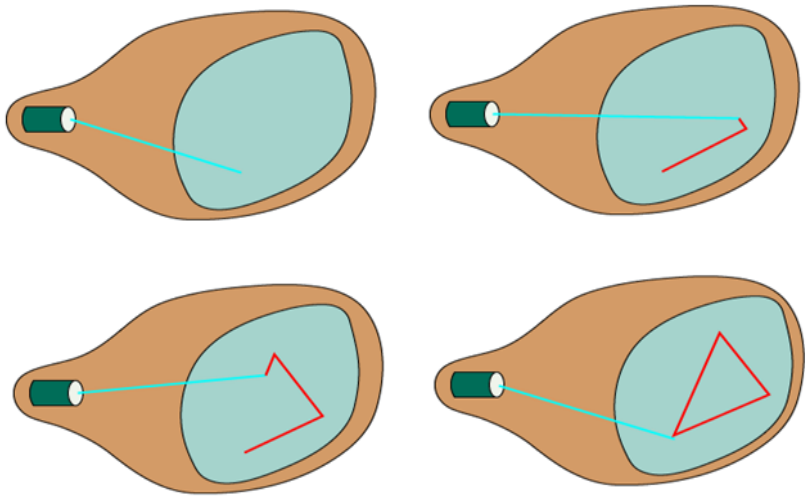
Important Definitions:

- Raster systems are commonly characterized by their **resolution**, which is the number of pixel positions that can be plotted.
- Another property of video monitors is **aspect ratio**, which is now often defined as the number of pixel columns divided by the number of scan lines that can be displayed by the system.
- **Aspect ratio** can also be described as the number of horizontal points to vertical points (or vice versa) necessary to produce equal-length lines in both directions on the screen. In other words, it can be defined to be the width of the rectangle divided by its height.
- The range of colors or shades of gray that can be displayed on a raster system depends on both the types of phosphor used in the CRT and the number of bits per pixel available in the frame buffer.
  - black and white: 1 bit per pixel.
  - gray scale: 1 byte per pixel (256 gray levels)
  - true color(R,G,B): 3 bytes=24 pits per pixel (224 colors)
- The number of bits per pixel in a frame buffer is sometimes referred to as either the **depth of the buffer** area or the **number of bit planes**.
- A frame buffer with one bit per pixel is commonly called a **bitmap**, and a frame buffer with multiple bits per pixel is a **pixmap.**
- In raster scan systems refreshing is done at a rate of 60-80 frames per second. Refresh rates are also sometimes described in units of **cycles per second / Hertz (Hz).**

Random Scan Display

- Random Scan System uses an electron beam which operates like a pencil to create a line image on the CRT screen.
- The picture is constructed out of a sequence of straight-line segments.
- Each line segment is drawn on the screen by directing the beam to move from one point on the screen to the next, where its x & y coordinates define each point.
- After drawing the picture. The system cycles back to the first line and design all the lines of the image 30 to 60 time each second. The process is shown in figure above.
- Random-scan monitors are also known as vector displays or stroke-writing displays or

calligraphic displays.



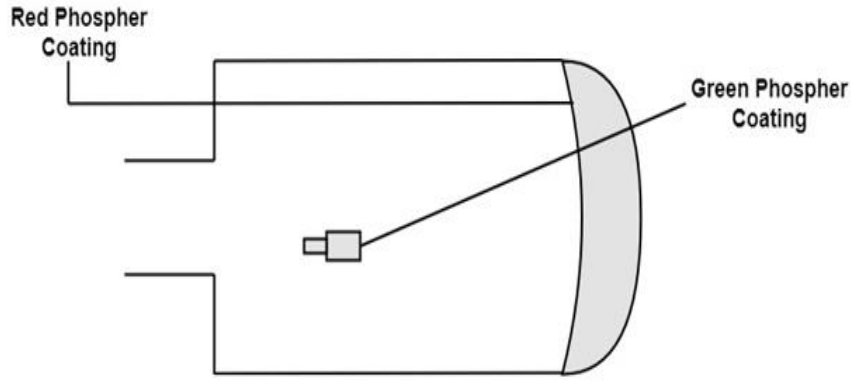
Color CRT Monitors

- The CRT Monitor display by using a combination of phosphors. The phosphors are different colors. There are two popular approaches for producing color displays with a CRT are:
  - Beam Penetration Method
  - Shadow-Mask Method
- Color CRTs in graphics systems are designed as RGB monitors.
- These monitors use shadow-mask methods and take the intensity level for each electron gun (red, green, and blue) directly from the computer system without any intermediate processing.
- High-quality raster-graphics systems have 24 bits per pixel in the frame buffer, allowing 256 voltage settings for each electron gun and nearly 17 million color choices for each pixel.
- An RGB color system with 24 bits of storage per pixel is generally referred to as a full-color system or a true-color system.

Advantage:	Disadvantage:
1.Realistic image	1.Relatively expensive compared with the monochrome CRT.
2.Million different colors to be generated	2.Relatively poor resolution
3.Shadow scenes are possible	3.Convergence Problem

Beam Penetration Method:

- The Beam-Penetration method has been used with random-scan monitors.
- In this method, the CRT screen is coated with **two layers of phosphor, red and green** and the displayed color depends on how far the electron beam penetrates the phosphor layers.
- This method produces four colors(2^2) only, **red, green, orange and yellow**.
- A beam of slow electrons excites the **outer red layer only**; hence screen shows red color only.



- A beam of high-speed electrons excites the inner green layer. Thus screen shows a green color.
- The speed of the electrons, and hence the screen color at any point, is controlled by the beam acceleration voltage.

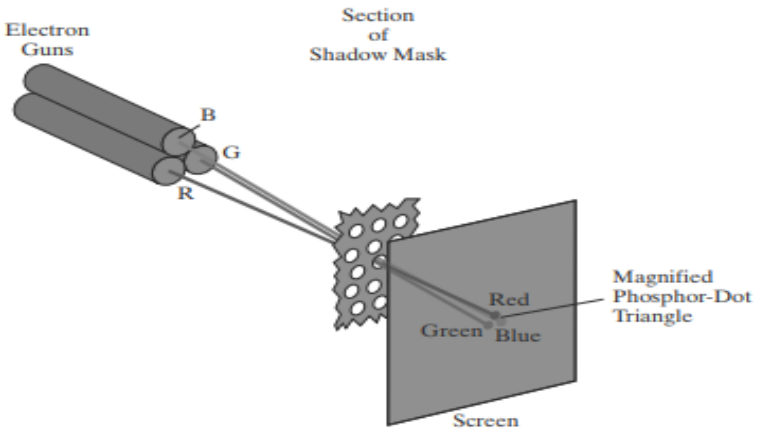
Advantages:

1. Inexpensive

Disadvantages:

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

Shadow-Mask Method:



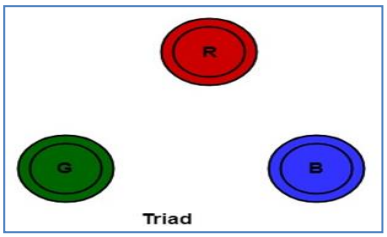
- Shadow Mask Method is commonly used in Raster-Scan System because they produce a much wider range of colors than the beam-penetration method.
- It is used in the majority of color TV sets and monitors.

Construction:

- A shadow mask CRT has 3 phosphor color dots at each pixel position.
  - One phosphor dot emits: red light
  - Another emits: green light
  - Third emits: blue light
- This type of CRT has 3 electron guns, one for each color dot and a shadow mask grid just behind the phosphor coated screen.
- Shadow mask grid is pierced with small round holes in a triangular pattern.
- Figure shows the delta-delta shadow mask method commonly used in color CRT system.

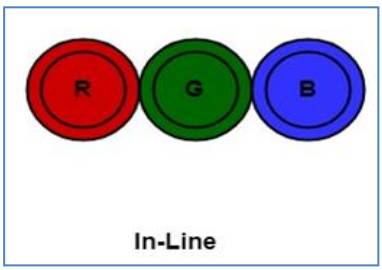
Working:

1. **Triad arrangement of red, green, and blue guns.**



- The deflection system of the CRT operates on all 3 electron beams simultaneously; the 3 electron beams are deflected and focused as a group onto the shadow mask, which contains a sequence of holes aligned with the phosphor- dot patterns.
- When the three beams pass through a hole in the shadow mask, they activate a dotted triangle, which occurs as a small color spot on the screen.
- The phosphor dots in the triangles are organized so that each electron beam can activate only its corresponding color dot when it passes through the shadow mask.

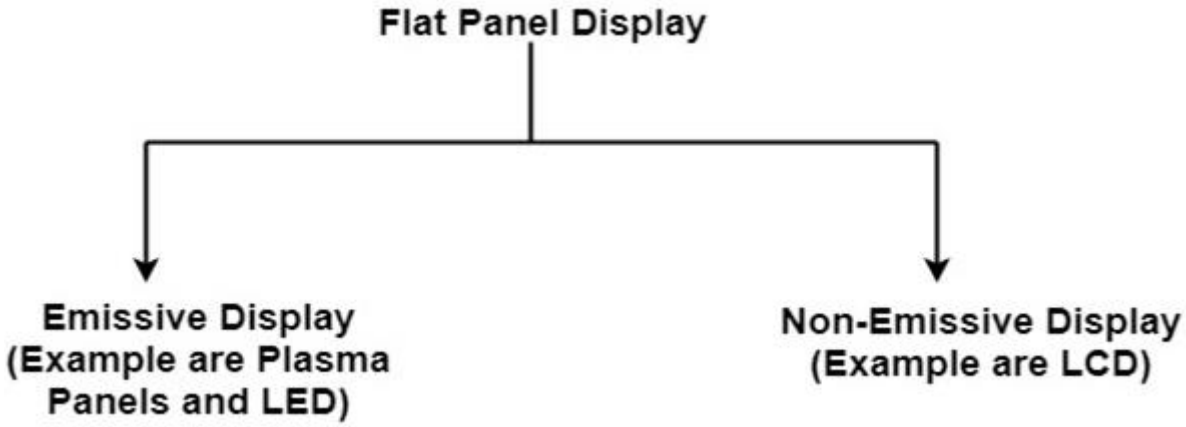
2. **Inline arrangement:**



- Another configuration for the 3 electron guns is an Inline arrangement in which the 3 electron guns and the corresponding red-green-blue color dots on the screen, are aligned along one scan line rather of in a triangular pattern.
- This inline arrangement of electron guns in easier to keep in alignment and is commonly used in high-resolution color CRT's.

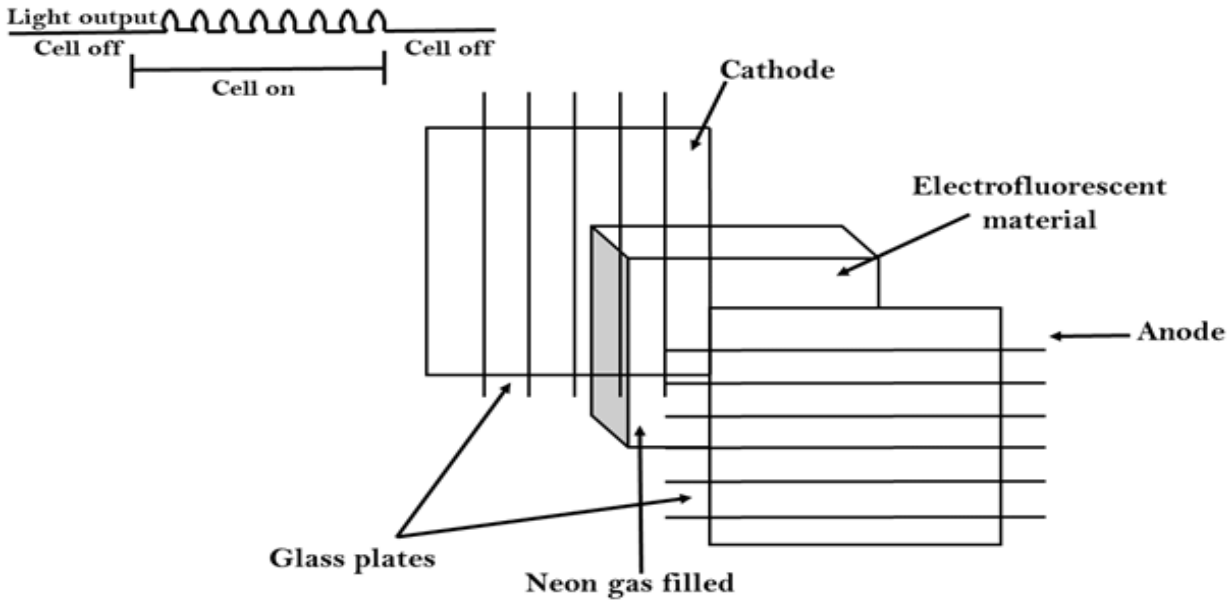
Flat-Panel Displays

- The Flat-Panel display refers to a class of video devices that have reduced volume, weight and power requirement compare to CRT.
- Example: Small T.V. monitor, calculator, pocket video games, laptop computers, an advertisement board in elevator.



1. **Emissive Display:** The emissive displays are devices that convert electrical energy into light. Examples are **Plasma Panel**, thin film electroluminescent display and **LED** (Light Emitting Diodes).
2. **Non-Emissive Display:** The Non-Emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns. Examples are **LCD** (Liquid Crystal Device).

Plasma Panel Display:



- Plasma-Panels are also called as Gas-Discharge Display. It consists of an array of small lights. Lights are fluorescent in nature. The essential components of the plasma-panel display are:
  1. **Cathode:** It consists of fine wires. It delivers negative voltage to gas cells. The voltage is released along with the negative axis.
  2. **Anode:** It also consists of line wires. It delivers positive voltage. The voltage is supplied along positive axis.
  3. **Fluorescent cells:** It consists of small pockets of gas liquids when the voltage is

applied to this liquid (neon gas) it emits light.

4. **Glass Plates:** These plates act as capacitors. The voltage will be applied, the cell will glow continuously.

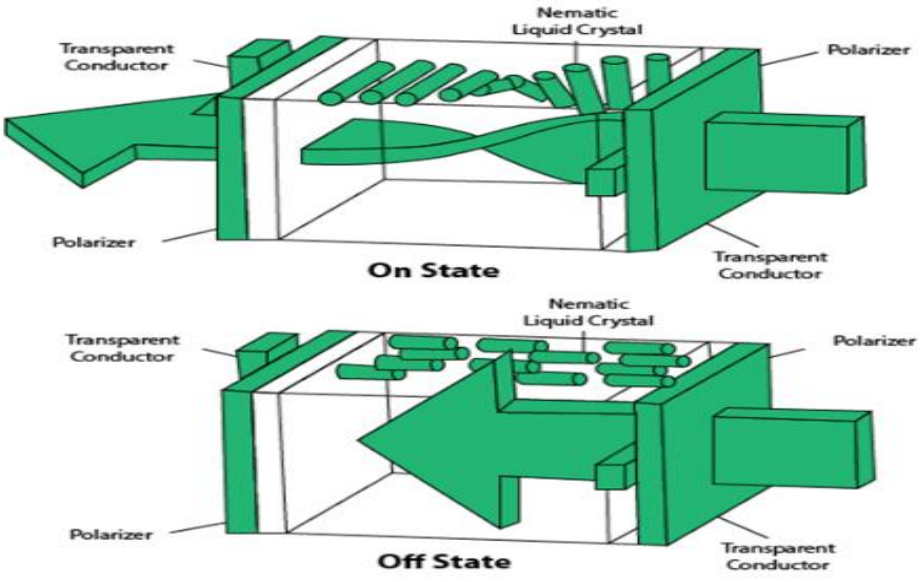
- The gas will slow when there is a significant voltage difference between horizontal and vertical wires. The voltage level is kept between 90 volts to 120 volts. Plasma level does not require refreshing. Erasing is done by reducing the voltage to 90 volts.
- Each cell of plasma has two states, so cell is said to be stable. Displayable point in plasma panel is made by the crossing of the horizontal and vertical grid. The resolution of the plasma panel can be up to 512 * 512 pixels.

Advantage:	Disadvantage:
1.High Resolution	1.Poor Resolution
2.Large screen size is also possible.	2.Wiring requirement for anode and the cathode is complex.
3.Less Volume	3.Its addressing is also complex.
4.Less weight	
5.Flicker Free Display	

LED (Light Emitting Diode):

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

LCD (Liquid Crystal Display):



Liquid Crystal Display

- Liquid Crystal Displays are the devices that produce a picture by passing polarized light from the surroundings or from an internal light source through a liquid-crystal material that transmits the light.
- LCD uses the liquid-crystal material between two glass plates; each plate is the right angle to each other between plates liquid is filled. One glass plate consists of rows of conductors arranged in vertical direction. Another glass plate is consisting of a row of conductors arranged in horizontal direction. The pixel position is determined by the intersection of the vertical & horizontal conductor. This position is an active part of the screen.
- Liquid crystal display is temperature dependent. It is between zero to seventy degree Celsius. It is flat and requires very little power to operate.

Advantage:	Disadvantage:
1.Low power consumption. 2.Small Size 3.Low Cost	1.LCDs are temperature-dependent (0-70°C) 2.LCDs do not emit light; as a result, the image has very little contrast. 3.LCDs have no color capability. 4.The resolution is not as good as that of a CRT.

Three-Dimensional Viewing Devices

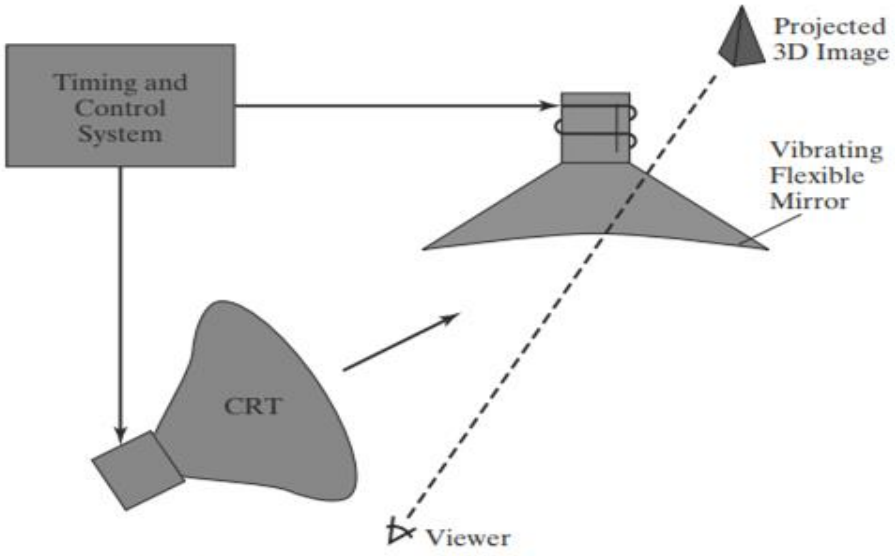


FIGURE 14
Operation of a three-dimensional display system using a vibrating mirror that changes focal length to match the depths of points in a scene.

- Graphics monitors for the display of 3-D scenes have been devised using a technique that reflects a CRT image from a vibrating, flexible mirror (Fig. 14).
- As the varifocal mirror vibrates, it changes focal length. These vibrations are synchronized with the display of an object on a CRT so that each point on the object is reflected from the mirror into a spatial position corresponding to the distance of that point from a specified viewing location.
- This allows us to walk around an object or scene and view it from different sides.
- In addition to displaying 3-D images, these systems are often capable of displaying 2-D cross-sectional "slices" of objects selected at different depths, such as in medical applications to analyze data from ultrasonography.

Stereoscopic system:

- Stereoscopic views does not produce 3-D images, but it produce 3D effects by presenting different view to each eye of an observer so that it appears to have depth.
- To obtain this we first need to obtain two views of object generated from viewing direction corresponding to each eye.
- We can construct the two views as computer generated scenes with different viewing positions or we can use stereo camera pair to photograph some object or scene.
- When we see simultaneously both the view as left view with left eye and right view with right

eye then two views is merge and produce image which appears to have depth.

- One way to produce stereoscopic effect is to display each of the two views with raster system on alternate refresh cycles.
- The screen is viewed through glasses with each lance design such a way that it act as a rapidly alternating shutter that is synchronized to block out one of the views.

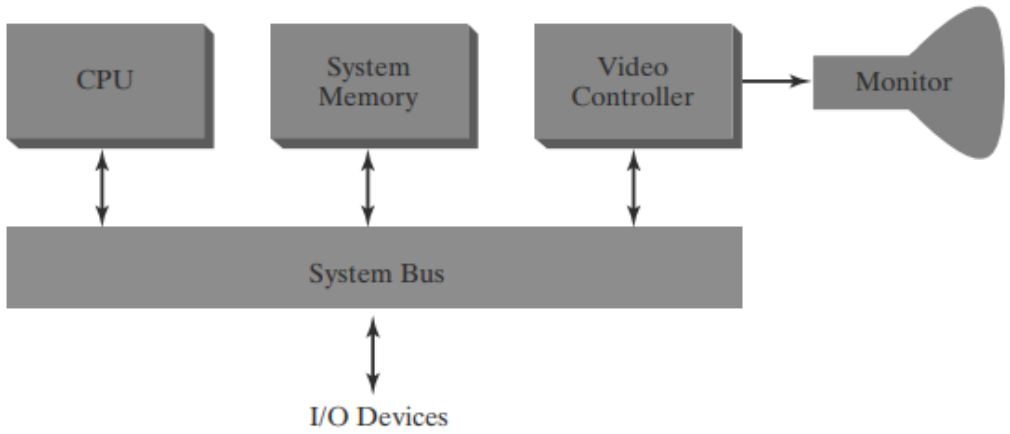
Virtual-reality:

- Virtual reality is the system which produce images in such a way that we feel that our surrounding is what we are set in display devices but in actually it does not.
- In virtual reality user can step into a scene and interact with the environment.
- A head set containing an optical system to generate the stereoscopic views is commonly used in conjunction with interactive input devices to locate and manipulate objects in the scene.
- Sensor in the head set keeps track of the viewer's position so that the front and back of objects can be seen as the viewer "walks through" and interacts with the display.
- Virtual reality can also be produce with stereoscopic glass and video monitor instead of head set. This provides low cost virtual reality system.
- Sensor on display screen track head position and accordingly adjust image depth.

Raster graphics Systems

- Interactive raster-graphics systems typically employ several processing units.
- In addition to the central processing unit (CPU), a special-purpose processor, called the video controller or display controller, is used to control the operation of the display device.

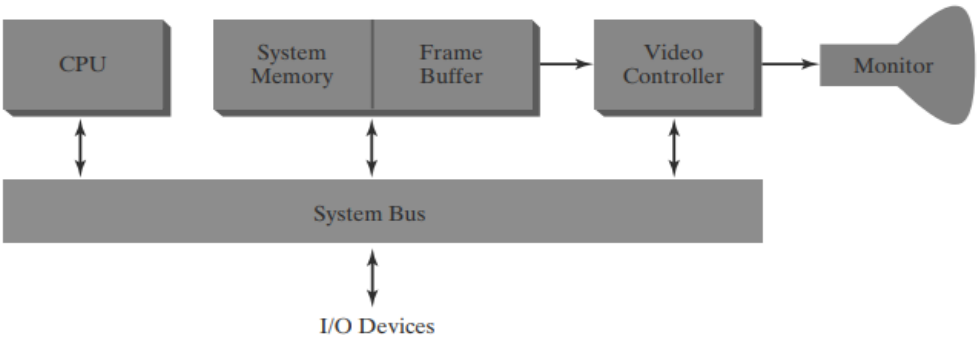
Simple raster graphics system:



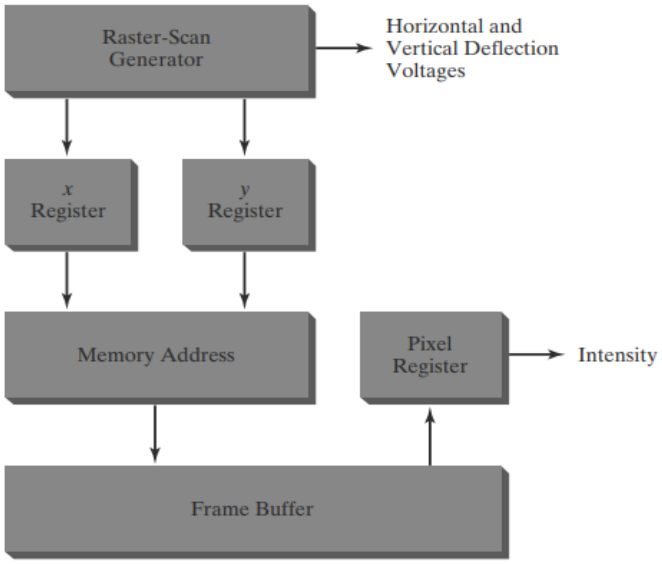
- Organization of a simple raster system is shown in Figure.
- Here, **the frame buffer can be anywhere** in the system memory, and the video controller accesses the frame buffer to refresh the screen.
- In addition to the video controller, more sophisticated raster systems employ other processors as coprocessors and accelerators to implement various graphics operations.

Raster graphics system with a fixed portion of the system memory reserved for the frame buffer:

A fixed area of the system memory is reserved for the frame buffer and the video controller can directly access that frame buffer memory. Frame buffer location and the screen position are referred in Cartesian coordinates. For many graphics monitors the coordinate origin is defined at the lower left screen corner. Screen surface is then represented as the first quadrant of the two dimensional systems with positive Xvalue increases as left to right and positive Y-value increases bottom to top.



Basic refresh operation of video controller:



Two registers are used to store the coordinates of the screen pixels which are X and Y. Initially the X is set to 0 and Y is set to Ymax. The value stored in frame buffer for this pixel is retrieved and used to set the intensity of the CRT beam. After this X register is incremented by one. This procedure is repeated till X becomes equals to Xmax. Then X is set to 0 and Y is decremented by one pixel and repeat above procedure.

This whole procedure is repeated till Y is become equals to 0 and complete the one refresh cycle. Then controller reset the register as top –left corner i.e. X=0 and Y=Ymax and refresh process start for next refresh cycle. Since screen must be refreshed at the rate of 60 frames per second the simple procedure illustrated in figure cannot be accommodated by typical RAM chips.

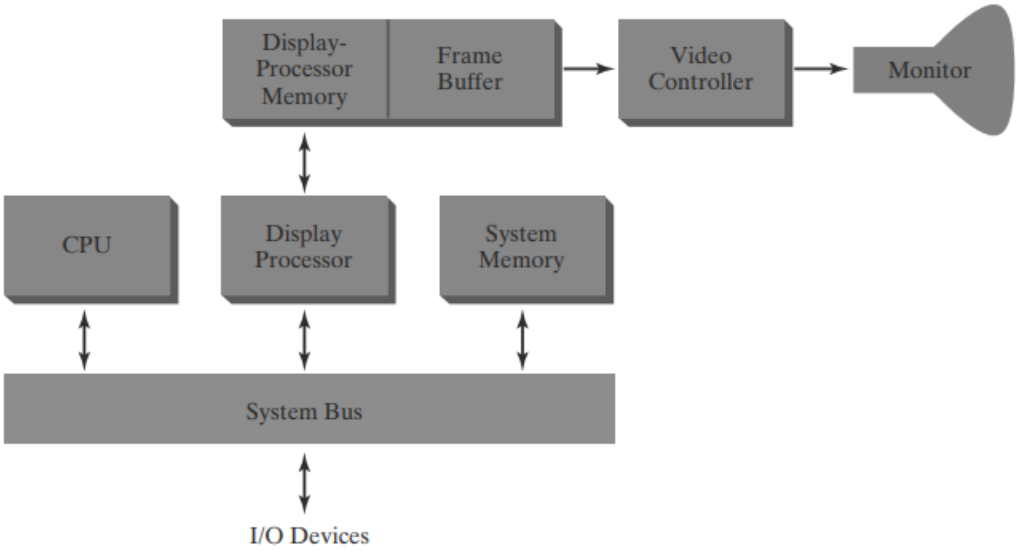
To speed up pixel processing video controller retrieves multiple values at a time using more numbers of registers and simultaneously refresh block of pixel. Such a way it can speed up and accommodate refresh rate more than 60 frames per second.

Raster-graphics system with a display processor:

One way to design a raster system is having separate display coprocessor. Purpose of display processor is to free CPU from graphics work.

Display processors have their own separate memory for fast operation. Main work of display processor is digitalizing a picture definition given into a set of pixel intensity values for store in frame buffer. This digitalization process is **scan conversion.**

Display processor also performs many other functions such as generating various line styles (dashed, dotted, or solid), displaying color areas, and applying transformations to the objects in a scene. It also interfaces with interactive input devices such as mouse.



For reduce memory requirements in raster scan system, methods have been devised for organizing the frame buffer as a linked list and encoding the color information.

One way to do this is to store each scan line as a set of integer pair. The first number in each pair can be a reference to a color value, and the second number can specify the number of adjacent pixels on the scan line that are to be displayed in that color. This technique is called **run-length encoding**. A similar approach is when pixel colors changes linearly. Another approach is to encode the raster as a set of rectangular areas (**cell encoding**).

The disadvantages of encoding runs are that color changes are difficult to record and storage requirements increase as the lengths of the runs decrease.

It is also difficult for display controller to process the raster when many short runs are involved.

Graphics Software and Standard

There are mainly two types of graphics software:

1. General programming package
2. Special-purpose application package

1. General Programming Package:

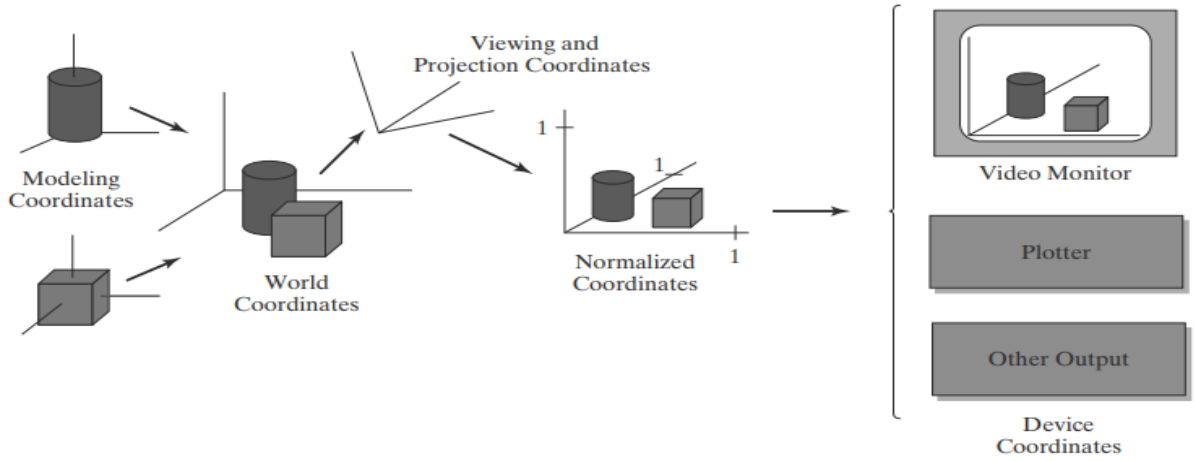
- A general programming package provides an **extensive set of graphics function that can be used in high level programming language such as C, C++, Java, Python etc.**
- It includes basic drawing element shape like **line, curves, polygon, color of element transformation etc.**
- Example: - GL (Graphics Library), , OpenGL, VRML (Virtual-Reality Modeling Language), Java 2D, and Java 3D.
- A set of graphics functions is often called a **computer-graphics application programming interface (CG API)** because the library provides a software interface between a programming language (such as C++) and the hardware.

2. Special-purpose Application Package:

- Special-purpose packages are **designed for nonprogrammers** who want to generate pictures, graphs, or charts in some application area without worrying about the graphics procedures that might be needed to produce such displays.
- The **interface to a special-purpose package is typically a set of menus that allow users to communicate with the programs in their own terms.**
- User can simply use it by interfacing with application.
- Example: - **CAD, medical and business systems**.

Coordinate Representations

- To generate a picture using a programming package, we first need to give the geometric descriptions of the objects that are to be displayed. These descriptions determine the locations and shapes of the objects.
- Except few all other general graphics packages require geometric descriptions to be specified in a standard, right-handed, Cartesian-coordinate reference frame. If coordinate values for a picture are given in some other reference frame, they must be converted to Cartesian coordinates before they can be input to the graphics package.
- Some packages that are designed for specialized applications may allow use of other coordinate frames that are appropriate for those applications.
- In general, several different Cartesian reference frames are used in the process of constructing and displaying a scene.
- First, we can define the shapes of individual objects, within a separate reference frame for each object. These reference frames are called **modeling coordinates**, or sometimes **local coordinates** or **master coordinates**. Once the individual object shapes have been specified, we can construct ("**model**") a scene by placing the objects into appropriate locations within a scene reference frame called **world coordinates**.
- Generally a graphic system first converts the world-coordinates position to normalized device coordinates where each coordinate value is in the range from −1 to 1 or in the range from 0 to 1, depending on the system.
- Finally, the picture is scan-converted into the refresh buffer of a raster system for display. The coordinate systems for display devices are generally called **device coordinates**, or **screen coordinates** in the case of a video monitor.



- Figure above briefly illustrates the sequence of coordinate transformations from modeling coordinates to device coordinates for a display that is to contain a view of two three-dimensional (3D) objects.
- An initial modeling-coordinate position (x_{mc} , y_{mc} , z_{mc}) in this illustration is transferred to world coordinates, then to viewing and projection coordinates, then to left-handed normalized coordinates, and finally to a device-coordinate position (x_{dc} , y_{dc}) with the sequence:

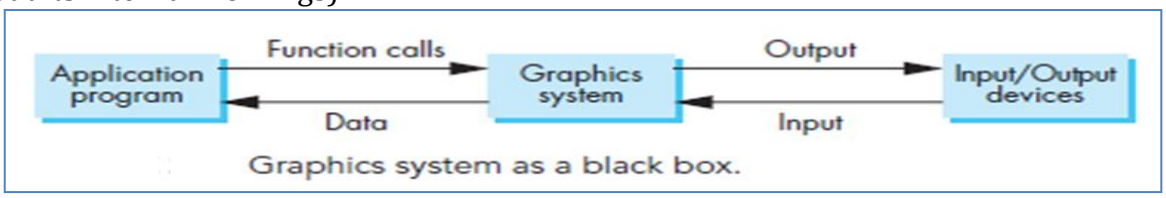
$$(x_{mc}, y_{mc}, z_{mc}) \rightarrow (x_{wc}, y_{wc}, z_{wc}) \rightarrow (x_{vc}, y_{vc}, z_{vc}) \rightarrow (x_{pc}, y_{pc}, z_{pc})$$
$$\rightarrow (x_{nc}, y_{nc}, z_{nc}) \rightarrow (x_{dc}, y_{dc})$$

Graphics Functions

- A general-purpose graphics package provides users with a variety of functions for creating and manipulating pictures.
- These routines can be broadly classified according to whether they deal with graphics output, input, attributes, transformations, viewing, subdividing pictures, or general control.
- The basic building blocks for pictures are referred to as **graphics output primitives**. They include character strings and geometric entities, such as points, straight lines, curved lines, filled color areas (usually polygons), and shapes defined with arrays of color points.
- In addition, some graphics packages provide functions for displaying more complex shapes such as spheres, cones, and cylinders.
- **Attributes** are properties of the output primitives; that is, an attribute describes how a particular primitive is to be displayed. This includes color specifications, line styles, text styles, and area-filling patterns.
- We can change the size, position, or orientation of an object within a scene using **geometric transformations**.
- Some graphics packages provide an additional set of functions for performing **modeling transformations**. Such packages usually provide a mechanism for describing complex objects.
- **Viewing transformations** are used to select a view of the scene, the type of projection to be used, and the location on a video monitor where the view is to be displayed.
- Interactive graphics applications use various kinds of input devices, including a mouse, a tablet, and a joystick. **Input functions** are used to control and process the data flow from these interactive devices.
- Finally, a graphics package contains a number of housekeeping tasks, such as clearing a screen display area to a selected color and initializing parameters. We use **control functions** for these operations.

Introduction to OpenGL

- A basic library of functions is provided in OpenGL for specifying graphics primitives, attributes, geometric transformations, viewing transformations, and many other operations.
- Our basic model of a graphics package is a black box, a term that engineers use to denote a system whose properties are described only by its inputs and outputs (we may know nothing about its internal workings).



Graphics system as a black box.

- We can take the simplified view of inputs as function calls and outputs as primitives displayed on our monitor, as shown in Figure.

An API for interfacing with this system can contain hundreds of individual functions. It will be helpful to divide these functions into **seven major groups**:

1. Primitive functions	**5. Input functions**
2. Attribute functions	**6. Control functions**
3. Viewing functions	**7. Query functions**
4. Transformation functions	

1. Primitive functions:

The primitive functions define the low-level objects or atomic entities that our system can display. Depending on the API, the primitives can include points, line segments, polygons, pixels, text, and various types of curves and surfaces.

2. Attribute functions

If primitives are the **what of** an API, then attributes are the **how**. That is, the attributes govern the way that a primitive appears on the display.

Attribute functions allow us to perform operations ranging from choosing the color with which we display a line segment, to picking a pattern with which to fill the inside of a polygon, to selecting a typeface for the titles on a graph.

3. Viewing functions

The viewing functions allow us to specify various views, although APIs differ in the degree of flexibility they provide in choosing a view.

4. Transformation functions

Transformation functions that allows her to carry out transformations of objects, such as rotation, translation, and scaling.

5. Input functions

For interactive applications, an API must provide a set of input functions to allow us to deal with the diverse forms of input that characterize modern graphics systems. We need functions to deal with devices such as keyboards, mouse, and data tablets.

6. Control functions

The control functions enable us to communicate with the window system, to initialize our programs, and to deal with any errors that take place during the execution of our programs.

7. Query functions

Within our applications we can often use other information within the API, including camera parameters or values in the frame buffer. A good API provides this information through a set of query functions.

The OpenGL Interface

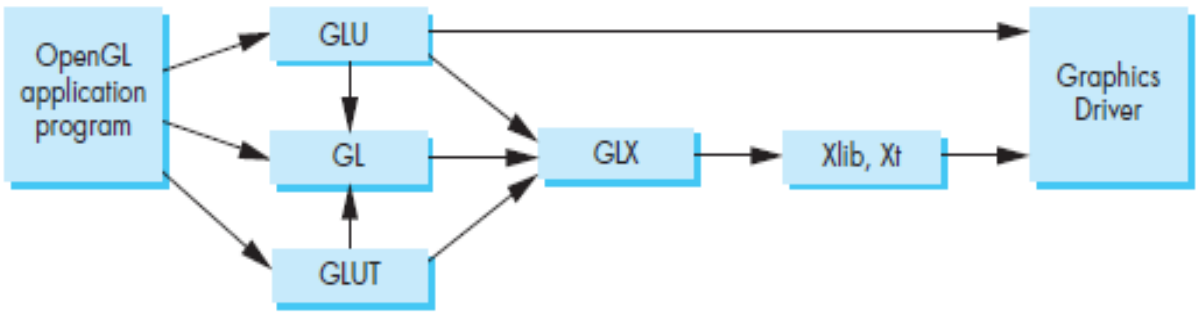


FIGURE 2.4 Library organization.

Most of our applications will be designed to access openGL directly through functions in 3 libraries:

- **OpenGL core library(GL)**
  -OpenGL32 on Windows
  -GL on most unix/linux systems
  -All funtions in the GL library begin with **gl.**
- **OpenGL Utility Library (GLU)**
  -Contains the code for creating common objects & simplifying viewing
  -All functions in this library are created from the core GL library
  -All functions in the GLU library begin with **glu.**
- **OpenGL Utility Toolkit (GLUT)**
  -To interface with the window system and to get input from external devices into our programs, we need at least one more library.
  -For the X Window System, this library is called **GLX**, for Windows, it is **wgl**, and for the Macintosh, it is **agl**. Rather than using a different library for each system, we use two readily available library called the OpenGL Utility Toolkit **(GLUT).**

Figure 2.4 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

OpenGL makes heavy use of defined constants to increase code readability and are defined in header (.h) files. In most implementations, one of the include lines

#include <GL/glut.h>
or
#include < glut.h> is sufficient to read in **glu.h** and **gl.h**.

Basic OpenGL Syntax

- Function names in the **OpenGL basic library** (also called the OpenGL core library) are prefixed with **gl,** and each component word within a function name has its first letter capitalized.
- The following examples illustrate this naming convention:

```
glBegin,   glClear,   glCopyPixels,   glPolygonMode
```

- Certain functions require that one (or more) of their arguments be assigned a symbolic constant.
- All such constants begin with the uppercase letters **GL**. In addition, component words within a constant name are written in capital letters, and the underscore (_) is used as a separator between all component words in the name.
  Example:

```
GL_2D,  GL_RGB,  GL_CCW,  GL_POLYGON,  GL_AMBIENT_AND_DIFFUSE
```

- The OpenGL functions also expect specific data types. To indicate a specific data type, OpenGL uses special built-in, data-type names, such as:

```
GLbyte,  GLshort,  GLint,  GLfloat,  GLdouble,  GLboolean
```

- Each data-type name begins with the capital letters **GL**, and the remainder of the name is a standard data-type designation written in lowercase letters.
- In addition to the OpenGL basic (core) library, the **OpenGL Utility (GLU) provides** routines for setting up viewing and projection matrices, describing complex objects with line and polygon approximations, displaying quadrics and other complex tasks.
- Every OpenGL implementation includes the GLU library, and all GLU function names start with the prefix **glu.**

Display-Window Management Using GLUT:

- To create a graphics display using OpenGL, we first need to set up a **display window** on our video screen. This is simply the rectangular area of the screen in which our picture will be displayed.
- We cannot create the display window directly with the basic OpenGL functions since this library contains only device independent graphics functions, and window-management operations depend on the computer we are using.
- However, there are several window-system libraries that support OpenGL functions for a variety of machines.
- The **GLX to the X Window System**, Apple systems can use the **Apple GL** (**AGL**), For Microsoft Windows systems, the **WGL** routines provide a **Windows-to-OpenGL** interface and so on
- The **OpenGL Utility Toolkit** (**GLUT**) provides a library of functions for interacting with any screen-windowing system.
- The GLUT library functions are prefixed with **glut**, and this library also contains methods for describing and rendering quadric curves and surfaces.
- Since GLUT is an interface to other device-specific window systems, we can use it so that our programs will be device-independent.

To get started, we can consider a simplified, minimal number of operations for displaying a picture.

Step-1: Initialization of GLUT.

Since we are using the OpenGL Utility Toolkit, This is done using the statement:

```
glutInit (&argc, argv);
```

Step-2: Window Creation.

we can state that a **display window is to be created** on the screen with a given caption for the title bar. This is accomplished with the function:

```
glutCreateWindow ("An Example OpenGL Program");
```

Step-3: Specification of the display window

- **Then** we need to specify what the display window is to contain.
- For this, we create a picture using OpenGL functions and pass the picture definition to the GLUT routine **glutDisplayFunc,** which assigns our picture to the display window.
- As an example, suppose we have the OpenGL code for describing a line segment in a procedure called **lineSegment.** Then the following function call passes the line-segment description to the display window:

```
glutDisplayFunc (lineSegment);
```

Step-4: Last Line of Main

But the display window is not yet on the screen. We need one more GLUT function to complete the window-processing operations.

```
glutMainLoop ( );
```

- This function must be the last one in our program. It displays the initial graphics and puts the program into an infinite loop that checks for input from devices such as a mouse or keyboard.
- Our first example will not be interactive, so the program will just continue to display our picture until we close the display window.

Step-5: Additional Glut Funtions:

- Although the display window that we created will be in some default **location and size**, we can set these parameters using additional **GLUT functions.**

GLUT Function-1→ glutInitWindowSize(640, 480);
//specifies a 640 width × 480 height window
GLUT Function-2→ glutInitWindowPosition(0, 0);
//specifies a 640 width × 480 height window in the top-left corner of the display.

- We can also set a number of other options for the display window, such as buffering and a choice of color modes:
  GLUT Function-3→ glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);

A Complete OpenGL Program

Event-driven GLUT program structure

1. Configure and open window
2. Initialize OpenGL state, program variables
3. Register callback functions
   - Display (where rendering occurs) Resize
   - User input: keyboard, mouse clicks, motion, etc.
4. Enter event processing loop

```
#include <glut.h> // GLUT, includes glu.h and gl.h
/* Handler for window-repaint event. Call back when the window first appears
and whenever the window needs to be re-painted. */
void display( )
{
 glClearColor(0,0,0,0); // Set background color to black and opaque
glClear(GL_COLOR_BUFFER_BIT);  // Clear the color buffer
// Draw a Red 1x1 Square centered at origin
glBegin(GL_QUADS);
// Each set of 4 vertices form a quad
glColor3f(1,0,0); // Red
glVertex2f(-0.5, -0.5); // x, y
glVertex2f( 0.5, -0.5);
glVertex2f( 0.5, 0.5);
glVertex2f(-0.5, 0.5);
glEnd();
glFlush(); // Render now
}
```

```
/* Main function: GLUT runs as a console application starting at main() */
 void main(int argc, char** argv)
{
glutInit(&argc, argv); // Initialize GLUT
glutCreateWindow("OpenGL Setup Test"); // Create a window with the given title
glutInitWindowSize(320, 320); // Set the window's initial width & height
glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
glutDisplayFunc(display); // Register display callback handler for window re-paint
glutMainLoop(); // Enter the infinitely event-processing loop
}
```

Coordinate Reference Frames

- To describe a picture, we first decide upon a convenient Cartesian coordinate system, called the **world-coordinate reference frame**, which could be either 2D or 3D. We then describe the objects in our picture by giving their geometric specifications in terms of positions in **world coordinates**.
  Example: We define a **straight-line segment** with **two endpoint positions**, and a **polygon** is specified with **a set of positions for its vertices.**
- These coordinate positions are stored in the scene description along with other info about the objects, such as their color and their **coordinate extents**.
  Co-ordinate extents :Co-ordinate extents are the minimum and maximum x, y, and z values for each object. A set of coordinate extents is also described as **a bounding box** for an object. **Ex:** For a 2D figure, the coordinate extents are sometimes called **its bounding rectangle.**
- Objects are then displayed by passing the scene description to the viewing routines which identify visible surfaces and map the objects to the frame buffer positions and then on the video monitor.

Screen Coordinates

- Locations on a video monitor are referenced in integer screen coordinates, which correspond to the integer pixel positions in the frame buffer.
- Scan-line algorithms for the graphics primitives use the coordinate descriptions to determine the locations of pixels
  Example: given the endpoint coordinates for a line segment, a display algorithm must calculate the positions for those pixels that lie along the line path between the endpoints.
- Since a pixel position occupies a finite area of the screen, the finite size of a pixel must be taken into account by the implementation algorithms.
- For the present, we assume that each integer screen position references the centre of a pixel area. Once pixel positions have been identified the color values must be stored in the frame buffer.

Absolute and Relative Coordinate Specifications

- **Absolute Coordinate** values means that the values specified are the actual positions within the coordinate system in use.
- However, some graphics packages also allow positions to be specified using **Relative Coordinates**. This method is useful for various graphics applications, such as producing drawings with pen plotters, artist's drawing and painting systems, and graphics packages for publishing and printing applications. Taking this approach, we can specify a coordinate position as an offset from the last position that was referenced (called the current position).

Specifying a Two-Dimensional World-Coordinate Reference Frame in OpenGL

- The **gluOrtho2D** command is a function we can use to set up any **2D Cartesian reference frames.** Since the **gluOrtho2D function** specifies an orthogonal projection, we need also to be sure that the coordinate values are placed in the OpenGL projection matrix.
- In addition, we could assign the identity matrix as the projection matrix before defining the world-coordinate range.
- This would ensure that the coordinate values were not accumulated with any values we may have previously set for the projection matrix.

- Thus, for our initial two-dimensional examples, we can define the coordinate frame for the screen display window with the following statements:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D (xmin, xmax, ymin, ymax);
```

- The display window will then be referenced by coordinates (xmin, ymin) at the lower-left corner and by coordinates (xmax, ymax) at the upper-right corner, as shown in Figure 2.

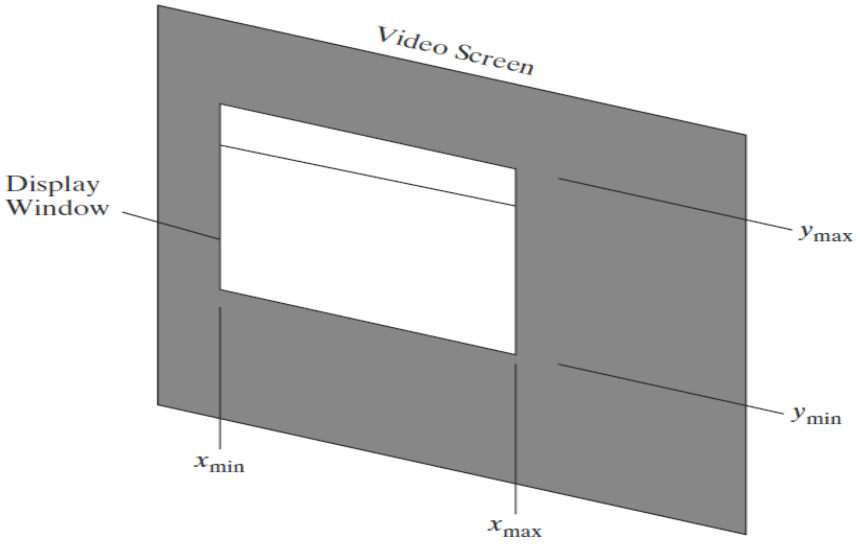


FIGURE 2
World-coordinate limits for a display window, as specified in the `glOrtho2D` function.

- We can then designate one or more graphics primitives for display using the coordinate reference specified in the gluOrtho2D statement.
- If the coordinate extents of a primitive are within the coordinate range of the display window, all of the primitive will be displayed. Otherwise, only those parts of the primitive within the display-window coordinate limits will be shown.

OpenGL Functions for Geometric primitives

- **Geometric primitives** include points, line segments, polygons, curves, and surfaces.
- These primitives pass through a geometric pipeline, where they are subject to a series of geometric operations that determine whether a primitive is visible.
- The basic OpenGL geometric primitives are specified by sets of vertices. Thus, the programmer defines the objects with sequences of the following:

```
glBegin(Type);
  glVertex*(...);
  glVertex*(...);
  glVertex*(...);
glEnd();
```

Where,
- **glVertex*** is used to specify a vertex of an object,
- ***** = **n t v**
  - **n=** no. of dimensions (parameters)(2,3,4)
  - **t=** data type(i, f or d)
  - **v=** arguments are in vector format.
- The value of **Type** (in **glBegin**) specifies how OpenGL assembles the vertices to define geometric objects.

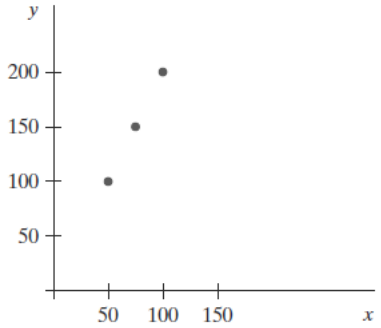
OpenGL Point Functions

- The **type** within **glBegin(type)** specifies the type of the object and its value can be: **GL_POINTS**
- Each vertex is displayed as a point.
- The size of the point would be of at least **one pixel.**
- Then this coordinate position, along with other geometric descriptions we may have in our scene, is passed to the viewing routines.
- Unless we specify other attribute values, OpenGL primitives are displayed with a default size and color.
- The default color for primitives is white, and the default point size is equal to the size of a single screen pixel.

Syntax:

Case 1:

```
glBegin (GL_POINTS);
  glVertex2i (50, 100);
  glVertex2i (75, 150);
  glVertex2i (100, 200);
glEnd ( );
```



Case 2: (Produces the same output as above)

Alternatively, we could specify the coordinate values for the preceding points in arrays such as

```
int point1 [ ] = {50, 100};
int point2 [ ] = {75, 150};
int point3 [ ] = {100, 200};
```

and call the OpenGL functions for plotting the three points as

```
glBegin (GL_POINTS);
  glVertex2iv (point1);
  glVertex2iv (point2);
  glVertex2iv (point3);
glEnd ( );
```

Case 3:

- Specifying two point positions in a three dimensional world reference frame. In this case, we give the coordinates as explicit floating-point values:

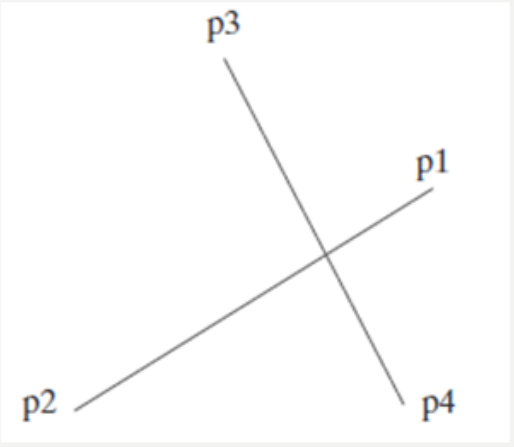
```
glBegin (GL_POINTS);
  glVertex3f (-78.05, 909.72, 14.60);
  glVertex3f (261.91, -5200.67, 188.33);
glEnd ( );
```

OpenGL Line Functions

- Primitive type is **GL_LINES**
- Successive pairs of vertices are considered as endpoints and they are connected to form an individual line segments.
- We obtain one line segment between the first and second coordinate positions and another line segment between the third and fourth positions.
- If the number of specified endpoints is odd, so the last coordinate position is ignored.

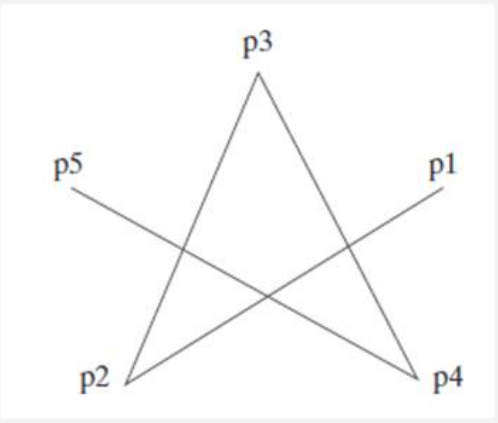
Case-1:

```
glBegin (GL_LINES);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd ( );
```



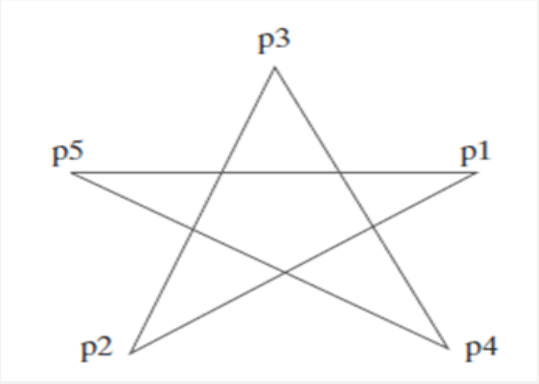
Case-2: With the OpenGL primitive constant **GL LINE STRIP,** we obtain a polyline.

```
glBegin (GL_LINE_STRIP);
   glVertex2iv (p1);
   glVertex2iv (p2);
   glVertex2iv (p3);
   glVertex2iv (p4);
   glVertex2iv (p5);
glEnd ( );
```



Case-3: With the OpenGL primitive constant **GL LINE LOOP,** we obtain a closed polyline.

```
glBegin (GL_LINE_LOOP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd ( );
```

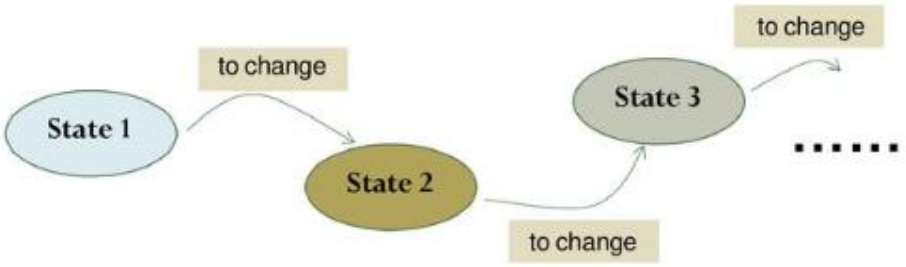


Attributes of Graphics Primitives

- Properties that describe how an object should be rendered are called **attributes.**
- Attributes may be associated with, or bound to, geometric objects, in the modeling and rendering pipeline.
- Each geometric type has a set of attributes. For example,
  - A point has a **color** attribute and a **size** attribute.
  - Line segments can have **color, thickness, and pattern** (solid, dashed, or dotted).
  - Filled primitives, such as polygons, have more attributes –We can fill with a solid color or a pattern. We can decide not to fill the polygon and to display only its edges. If we fill the polygon, we might also display the edges in a color different from that of the interior.
  - Stroke text as a primitive, there is a variety of attributes such as direction, the height, and width of the characters, the font, and the style (bold, italic, underlined).

OpenGL State Machine

- A graphics system that maintains a list for the current values of attributes and other parameters is referred to as a **state system** or **state machine.**
- Attributes of output primitives and some other parameters, such as the current frame-buffer position, are referred to as **state variables** or **state parameters**.
- When we assign a value to one or more state parameters, we put the system into a particular state, and that state remains in effect until we change the value of a state parameter.



- OpenGL is a finite state machine:
  - A predetermined and countable number of different states
  - The graphics system behaviors are determined by these system state, which can be modified by calling OpenGL functions.
- The OpenGL state includes:
  - The current color or other attributes
  - The current model & viewing transformations
  - The current camera model & clipping
  - The current lighting & reflectance model
- All have default values, remaining until a new setting on it.

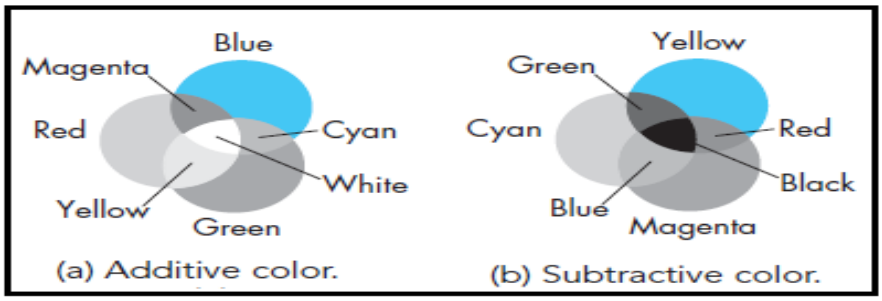
COLOR

Color is one of the most interesting aspects of both human perception and computer graphics.

A visible color can be characterized by a function $C(\lambda)$ that occupies wavelengths from about 350 to 780 nm.

The human visual system has three types of cones responsible for color vision. Hence, our brains do not receive the entire distribution $C(\lambda)$ for a given color but rather three values—the **tristimulus values**—that are the responses of the three types of cones to the color. This leads to the **basic tenet of three-color theory.**

A consequence of this tenet is that, in principle, a display needs only three primary colors to produce the three tristimulus values needed for a human observer.



The CRT is one example of the use of **additive color**, where the primary colors add together to give the perceived color. Other examples that use additive color include projectors and slide (positive) film. In such systems, the primaries are usually red, green, and blue. With additive color, primaries add light to an initially black display, yielding the desired color.

For processes such as commercial printing and painting, a **subtractive color model** is more appropriate. Here we start with a white surface, such as a sheet of paper. Colored pigments remove color components from light that is striking the surface. In subtractive systems, the primaries are usually the **complementary colors**: cyan, magenta, and yellow (CMY; Figure above).

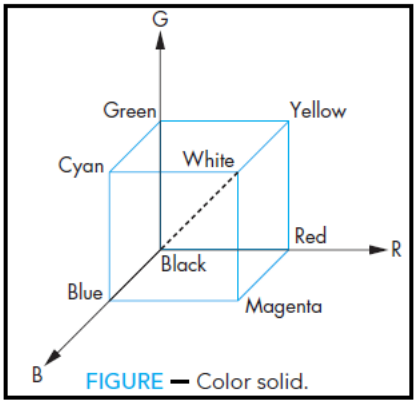


FIGURE — Color solid.

We can view a color as a point in a color solid, as shown in Figure. We draw the solid using a coordinate system corresponding to the three primaries. The vertices of the cube correspond to black (no primaries on); red, green, and blue (one primary fully on); the pairs of primaries, cyan (green and blue fully on), magenta (red and blue fully on), and yellow (red and green fully on); and white (all primaries fully on). The principal diagonal of the cube connects the origin (black) with white.

There are two different approaches to handle color in a graphics system from the programmer's perspective—that is, through the API. They are

1. **RGB-Color Model**
2. **Indexed-Color Model**

RGB-Color Model:

In modern systems RGB color has become the norm. In a three-primary-color, additive-color RGB system, there are conceptually separate buffers for red, green, and blue images. Each pixel has separate red, green, and blue components that correspond to locations in memory as shown in the figure below.

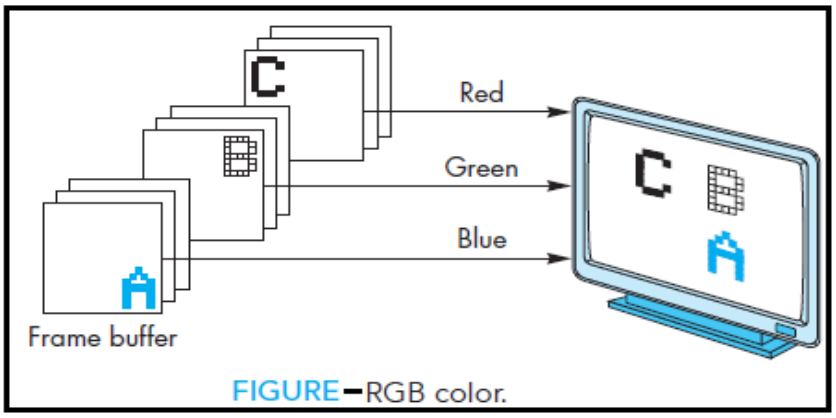


FIGURE – RGB color.

In a typical system, there might be a 1280 × 1024 array of pixels, and each pixel might consist of 24 bits (3 bytes): 1 byte for each of red, green, and blue. With present commodity graphics cards having up to 12GB of memory, there is no longer a problem of storing and displaying the contents of the frame buffer at video rates.

For our 24-bit example, there are 2^{24} possible colors, other systems may have as many as 12 (or more) bits per color or as few as 4 bits per color. Because our API should be independent of the particulars of the hardware, we would like to specify a color independently of the number of bits in the frame buffer. A natural technique is to use the color cube and to specify color components as numbers between 0.0 and 1.0, where 1.0 denotes the maximum (or saturated value) of the corresponding primary and 0.0 denotes a zero value of that primary.

In OpenGL, for **RGB-Color Model**, we use the following function call:

glColor* (...)- specifies vertex colors

glClearColor(r, g, b, a)-sets current color for cleaning color buffer

glutInitDisplayMode(mode)-specify either an RGB window (GLUT_RGB|), or a color indexed window (GLUT_RGB |GLUT_INDEX)

Example: Set color display mode to RGB

```
glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB );
// The 1st argument states we are using a single buffer
// The 2nd argument sets the RGB mode (default)
// To use a color table set GLUT_INDEX | GLUT_RGB

glColor3f( 0.0, 1.0, 1.0 );
glColor3fv( colorArray);
glColor3i( 0, 255, 255);
```

Indexed- Color Model

Input	Red	Green	Blue
0	0	0	0
1	$2^m - 1$	0	0
.	0	$2^m - 1$	0
.	.	.	.
.	.	.	.
$2^k - 1$	.	.	.
	m bits	*m* bits	*m* bits

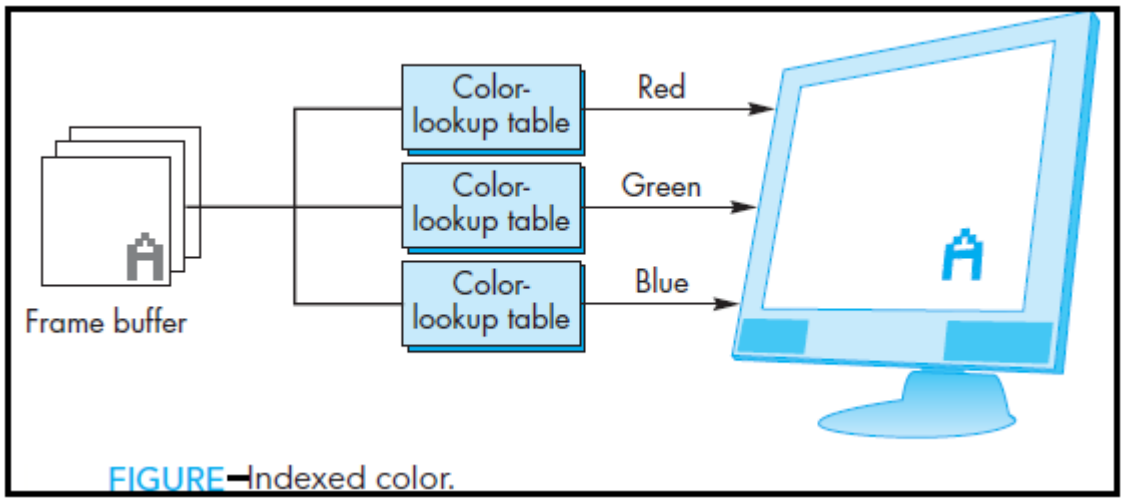
FIGURE – Color-lookup table.

Early graphics systems had frame buffers that were limited in depth. For example, we might have had a frame buffer with a spatial resolution of 1280 × 1024, but each pixel was only 8 bits deep. We could divide each pixel's 8 bits into smaller groups of bits and assign red, green, and blue to each. Although this technique was adequate in a few applications, it usually did not give us enough flexibility with color assignment.

Indexed color provided a solution that allowed applications to display a wide range of colors. This technique can be created within an application.

We can select colors by interpreting our limited-depth pixels as indices into a table of colors rather than as color values. Suppose that our frame buffer has k bits per pixel. Each pixel value or index is an integer between 0 and $2^k - 1$. Suppose that we can display each color component with a precision of $m$bits; that is, we can choose from 2^m reds, 2^m greens, and 2^m blues. Hence, we can produce any of 2^{3m} colors on the display, but the frame buffer can specify only 2^k of them. We handle the specification through a user-defined **color-lookup table** that is of size $2^k \times 3^m$as shown in the figure.

Once the user has constructed the table, she can specify a color by its index, which points to the appropriate entry in the color-lookup table shown in the figure below.



FIGURE–Indexed color.

One difficulty arises if the window system and underlying hardware support only a limited number of colors because the window system may have only a single color table that must be used for all its windows, one for each window on the screen.

OpenGL supports this mode, called **color index mode**. The functions available are:

glIndex*(...)- specifies vertex colors

glClearIndex(Glfloat index)- sets current color for cleaning color buffer.

glutSetColor(int color, GLfloat r, GLfloat g, GLfloat b)

sets the entries in a color table for window.

Example: We set the color by specifying an index into a color table

glutSetColor(196, 1,1,0);

Color parameters R,G,B are assigned floating-point values

glIndexi (196);

Point Attributes

- Basically, we can set two **attributes for points**: **color and size.**
- **In a state system:** The displayed color and size of a point is determined by the current values stored in the attribute list.
- **For a raster system:** Point size is an integer multiple of the pixel size, so that a large point is displayed as a square block of pixels.

OpenGL Point-Attribute Functions

1. Color components are set with RGB values or an index into a color table.
glColor3f(r,g,b);
glColor3ub(r,g,b)[0-255]
2. Point Size: we can set size of our rendered point by using
glPointSize(float size)- default is 1 pixel wide

Attribute functions may be listed inside or outside of a **glBegin/glEnd pair.** For **example**, the following code segment plots **3 points** in varying colors and sizes. The **first** is a **standard-size red point**, the **second is a double-size green point**, and the **third is a triple-size blue point:**

```
glColor3f (1.0, 0.0, 0.0);
glBegin (GL_POINTS);
   glVertex2i (50, 100);
   glPointSize (2.0);
   glColor3f (0.0, 1.0, 0.0);
   glVertex2i (75, 150);
   glPointSize (3.0);
   glColor3f (0.0, 0.0, 1.0);
   glVertex2i (100, 200);
glEnd ( );
```

Line Attributes

- A straight-line segment can be displayed with three basic attributes:
  1. **Color**
  2. **Width**
  3. **Style**

OpenGL Line-Attribute Functions

1. Line color: Color components are set with RGB values or an index into a color table.
glColor3f(r,g,b);
glColor3ub(r,g,b)[0-255]

2. Line Width:

- Line width is set in OpenGL with the function:
  glLinewidth (width);
- We assign a floating-point value to parameter width, and this value is rounded to the nearest nonnegative integer.
- The line is displayed with a standard width of **1.0**, which is the **default width**.

3. Line Style:

- By default, a straight-line segment is displayed as a solid line.
- However, we can also display **dashed lines, dotted lines, or a line with a combination of dashes and dots,** and we can vary the length of the dashes and the spacing between dashes or dots.
- We set a current display style for lines with the OpenGL function:

  glLineStipple(int *factor*,short *pattern*);

  - The **pattern** is a 16 bit sequence of 1s and 0s
    e.g. 1110111011101110
  - The **factor** is a bit multiplier for the pattern **(it enlarges it)**
    e.g. factor = 2 turns the above pattern into:
    1111111001111110011111110011111100
  - The pattern can be expressed in hexadecimal notation
  - e.g. 0xEECC = 1110111011001100
- **Line Stippling-** Stippling means to add a pattern to a simple line or the filling of a polygon.
- OpenGL allows stippling to be performed using bit patterns.
- To Enter into and exit from the stipple mode we use:

  Turn stippling on with**: glEnable(GL_LINE_STIPPLE);**
  Turn off with**: glDisable(GL_LINE_STIPPLE);**

Example: To Draw Dashed Lines, with line width=2 and red line

```
glColor2f(1,1,0);
glLineWidth(2);
glEnable (GL_LINE_STIPPLE)
glLineStipple (1,0xF0F0);
glBegin(GL_LINE_LOOP);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glVertex2f(x3,y3);
glEnd( )
glDisable(GL_LINE_STIPPLE);
```

Curve Attributes

- Parameters for curve attributes are the same as those for straight-line segments.
- We can display curves with varying colors, widths, dot-dash patterns, and available pen or brush options.
- OpenGL does not consider curves to be drawing primitives in the same way that it considers points and lines to be primitives.
- Curves can be drawn in several ways in OpenGL.
- Perhaps the simplest approach is to approximate the shape of the curve using short line segments.
- Alternatively, curved segments can be drawn using *splines.*

Line-Drawing Algorithms

- A straight-line segment in a scene is defined by the coordinate positions for the endpoints of the segment.
- To display the line, first project the endpoints to integer screen coordinates and determine the nearest pixel positions along the line path between the two endpoints.
- This process digitizes the line into a set of discrete integer positions that, in general, only approximates the actual line path.
- A computed line position of (10.48, 20.51), for example, is converted to pixel position (10, 21). This rounding of coordinate values to integers causes all but horizontal and vertical lines to be displayed with a stair-step appearance (known as "the jaggies"), as shown in Figure.

- Need algorithm to figure out which intermediate pixels are on line path which is more effective technique for smoothing a raster line that are based on adjusting pixel intensities along the line path.

Properties of Good Line Drawing Algorithm:

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

Fig: O/P from a poor line generating algorithm

2. **Lines should terminate accurately:** Unless lines are plotted accurately, they may terminate at the wrong place.
3. **Lines should have constant density:** Line density is proportional to the no. of dots displayed divided by the length of the line. To maintain constant density, dots should be equally spaced.

Fig: Uneven line density caused by bunching of dots.

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

Line Equations:

- The Cartesian slope-intercept equation for a straight line is:

$$y = m \cdot x + b$$

- With **m** as the slope of the line and **b** as the **y** intercept. Given that the two endpoints of a line segment are specified at positions (x0, y0) and (xend, yend), as shown in Figure, we can determine values for the slope **m** and **y** intercept **b** with the following calculations:

$$m = \frac{dy}{dx} = \frac{y_{end} - y_0}{x_{end} - x_0}$$

$$b = y_0 - m \cdot x_0$$

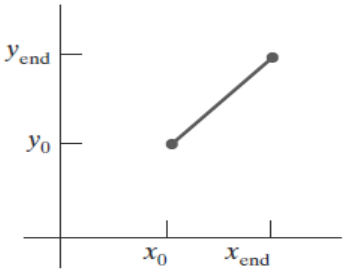


FIGURE 2
Line path between endpoint positions (x_0, y_0) and (x_{end}, y_{end}).

- For drawing a line, the following algorithms are used:
  1. **DDA (Digital Differential Analyzer) Line Drawing Algorithm**
  2. **Bresenham's Line Drawing Algorithm**

DDA Line Drawing Algorithm

- **DDA (Digital Differential Analyzer)** is a line drawing algorithm used in computer graphics to generate a line segment between two specified endpoints.
- It is a simple and efficient algorithm that works by using the incremental difference between the x-coordinates and y-coordinates of the two endpoints to plot the line.

The steps involved in DDA line generation algorithm are:

Step-1: Input the two endpoints of the line segment, **(x0,y0) and (x1,y1).**

Step-2: Calculate the difference between the two endpoints as **dx** and **dy** respectively.

$$dx = x_1 - x_0$$

$$dy = y_1 - y_0$$

Step-3: Based on the calculated difference in **step-2**, you need to identify the number of steps to put pixel.

If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))
   Steps = absolute(dx);
else
   Steps = absolute(dy);
```

Step-4: Calculate the increment in x coordinate and y coordinate.

$$X_{increment} = dx / (float)\ steps; \qquad Y_{increment} = dy / (float)\ steps;$$

Step-5: Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

Initialize **x=x0,y=y0, putpixel(Round(x), Round(y)**

```
for(int v=0; v < Steps; v++)
{
   x = x + Xincrement;
   y = y + Yincrement;
   putpixel(Round(x), Round(y));
}
```

Illustration:

If a line is drawn from (2, 3) to (6, 8) with use of DDA, How many points will needed to generate such line?

Solution:

Step1: Given (x0,y0)=(2,3) and (x1,y1)=(6,8)

Step2: **dx**=x1-x0=6-2=**4**

dy=y1-y0=8-3=**5**

Step3: here, dy>dx, therefore, **steps=5**

Step4: xincrement=dx/steps=4/5=**0.8**

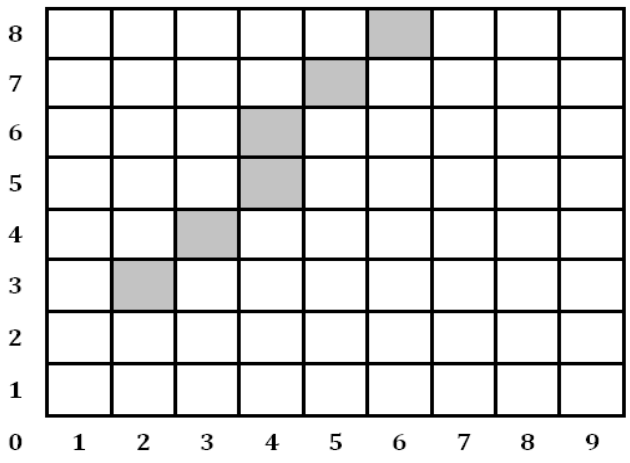
yincrement=dy/steps=5/5=**1**

Step5: x=x0=**2**,y=y0=**3**

p0=(2,3)

```
for(int v=0; v < Steps; v++)
{
    x = x + Xincrement;
    y = y + Yincrement;
    putpixel(Round(x), Round(y));
}
```

Iter-v=0: x=2+0.8=2.8
y=3+1=4
p1=(3,4)
Iter-v=1: x=2.8+0.8=3.6
y=4+1=5
p2=(4,5)
Iter-v=2: x=3.6+0.8=4.4
y=5+1=6
p3=(4,6)
Iter-v=3: x=4.4+0.8=5.2
y=6+1=7
p4=(5,7)
Iter-v=4: x=5.2+0.8=6
y=7+1=8
p5=(6,8)



Advantages:

- It is a simple algorithm.
- It is easy to implement.
- It avoids using the multiplication operation which is costly in terms of time complexity.

Disadvantages:

- There is an extra overhead of using **round off( )** function.
- Using round off( ) function increases time complexity of the algorithm.
- Resulted lines are not smooth because of round off( ) function.
- The points generated by this algorithm are not accurate.

Bresenham's Line Drawing Algorithm

- It was developed by Jack Bresenham in 1965.
- Bresenham's algorithm approximates a continuous straight line with discrete pixels, ensuring that the line appears straight and smooth on a pixel-based display.
- Since it works exclusively with integer arithmetic, it avoids the need for costly floating-point calculations and is better suited for hardware-constrained environments.
- It's also more accurate than simply rounding coordinates to the nearest pixel.

Explanation:

- Assuming that we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column $x_{k+1} = x_k + 1$.
- Our choices are the pixels at positions (x_k+1, y_k) and (x_k+1, y_k+1).
- At sampling position x_k+1, we label vertical pixel separations from the mathematical line path as d_{lower} and d_{upper} (Figure 7).
- The **y** coordinate on the mathematical line at pixel column position $x_k + 1$ is calculated as :

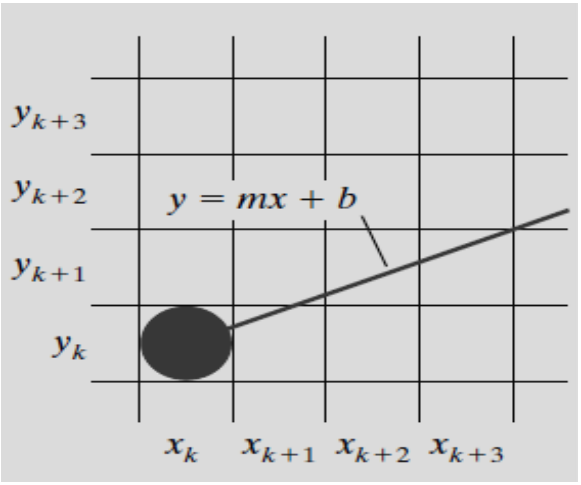
$$y = m(x_k + 1) + b \rightarrow (1)$$

Then

$$d_{lower} = y - y_k$$
$$= m(x_k + 1) + b - y_k \rightarrow (2)$$

and

$$d_{upper} = (y_k + 1) - y$$
$$= y_k + 1 - m(x_k + 1) - b \rightarrow (3)$$



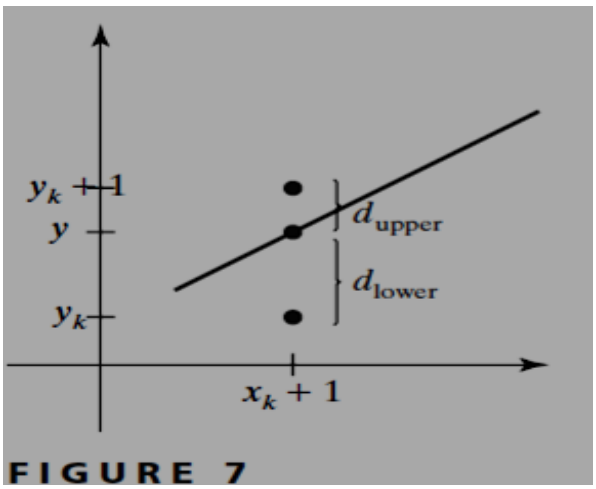


FIGURE 7

- To determine which of the two pixels is closest to the line path, we can set up an efficient test that is based on the difference between the two pixel separations as follows:

$$d_{lower} - d_{upper} = (m(x_k + 1) + b - y_k) - (y_k + 1 - m(x_k + 1) - b)$$
$$= 2m(x_k + 1) - 2y_k + 2b - 1$$
$$d_{lower} - d_{upper} = 2(dy/dx)(x_k + 1) - 2y_k + 2b - 1 \rightarrow (4)$$

By rearranging Equation **4** (by multiplying ***dx*** on both sides) so that it involves only integer calculations.

Thus the decision parameter $p_k = dx(d_{lower} - d_{upper})$, We have,

$$dx(d_{lower} - d_{upper}) = 2(dy)(x_k + 1) - 2*dx*y_k + 2*dx*b - dx$$
$$= 2*dy*x_k + 2*dy - 2*dx*y_k + 2*dx*b - dx$$
$$dx(d_{lower} - d_{upper}) = 2*dy*x_k - 2*dx*y_k + c$$

Where C=2*dy+ 2*dx*b – dx ,which is independent of the pixel position and will be eliminated in the recursive calculations.

Thus the decision parameter p_k= 2*dy* x_k - 2*dx*y_k + c

If the pixel at y_k *is "closer" to the line path than the pixel at* $y_k + 1$*(that is,* $d_{lower} < d_{upper}$*), then decision parameter* p_k *is negative. In that case, we plot the lower pixel; otherwise, we plot the upper pixel.*

At step ***k + 1****, the decision parameter is evaluated from* ***Equation 6*** *as*

P_{k+1}= 2*dy* x_{k+1} - 2*dx*y_{k+1} + c →(6)

Subtracting Equation 6 from the preceding equation 5, we have

P_{k+1} -P_k= (2*dy* x_{k+1} - 2*dx*y_{k+1} + c)- (2*dy* x_k - 2*dx*y_k + c)

= 2*dy* x_{k+1} - 2*dx*y_{k+1} + c- 2*dy* x_k + 2*dx*y_k - c

= 2*dy($x_{k+1} – x_k$) -2*dx(y_{k+1} - y_k)

However x_{k+1}= x_k+1, so we have

P_{k+1} =P_k+ 2*dy(x_k+1 – x_k) -2*dx(y_{k+1} - y_k)

P_{k+1} =P_k+ 2*dy -2*dx(y_{k+1} - y_k) →(6)

This recursive calculation of decision parameters is performed at each integer *x position, starting at the left coordinate endpoint of the line.*

The first parameter, p_0, *is evaluated from Equation 5 at the starting pixel position* ***(x0, y0)*** *and with* ***m*** *evaluated as* ***dy/dx*** *as follows:*

p_k= 2*dy* x_k - 2*dx*y_k + c

= 2*dy* x_k - 2*dx*y_k +2*dy+ 2*dx*b – dx

p_0 = 2*dy* x_0 - 2*dx*y_0 +2*dy+ 2*dx*b – dx →(7)

Wkt, y_0 =m x_0 +b

=(dy/dx) x_0 +b

b=y_0 -(dy/dx) x_0 →(8)

Put (8) in (7)

p_0 = 2*dy* x_0 - 2*dx*y_0 +2*dy+ 2*dx*[y_0 -(dy/dx) x_0] – dx

p_0 = 2*dy* x_0 - 2*dx*y_0 +2*dy+ 2*dx*y_0 -2*dx* (dy/dx) x_0 – dx

p_0 = 2*dy* x_0 - 2*dx*y_0 +2*dy+ 2*dx*y_0 -2*dy* x_0 – dx

p_0 = 2*dy+ 2*dx*y_0 -2*dy* x_0 – dx

Summary:

Initially at (x_{k+1}, y_{k+1}) decision parameter is,

$$p_0 = 2*dy - dx$$

If $y_{k+1}=y_k$, decision parameter is,

$$P_{k+1} = P_k + 2*dy$$

If $y_{k+1}=y_k+1$, decision parameter is,

$$P_{k+1} = P_k + 2*dy - 2*dx$$

Algorithm

```
bresenham (x1,y1,x2,y2)
{
x=x1;
y=y1;
dx=x2-x1;dy=y2-y1;
p=2*dy-dy
while(x<=x2)
{
putpixel(x,y); x++;
if(p<0)
p=p+2*dy;
else
{
p=p+2*dy-2dx;
y++;
    }
}
}
```

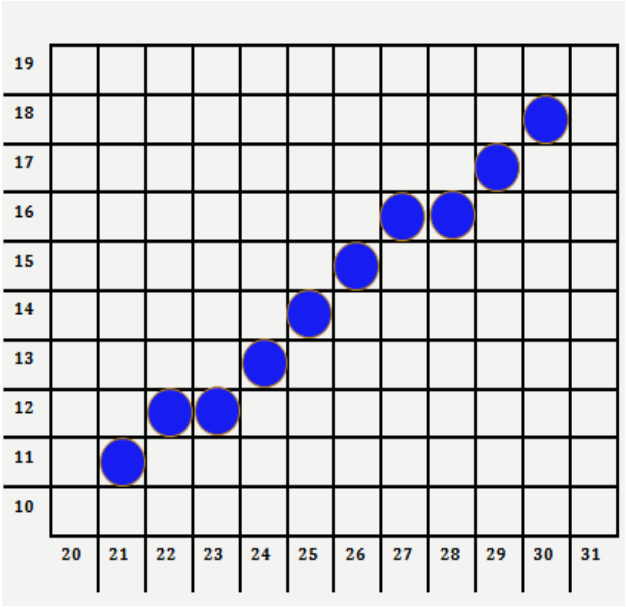
Illustration

Illustrate Bresenham's Algm for a line with endpoints (20, 10) to (30, 18).

Solution:

x=20, y=10
dx=30-20=10; dy=18-10=8
p=2*8-10=6
twody=2*8=16
twodyminusdx=2*(8-10)=-4
plot(**20,10**)

k	p	(x_k,y_k)
0(p>0)	2	**(21,11)**
1(p>0)	-2	**(22,12)**
2(p<0)	14	**(23,12)**
3(p>0)	10	**(24,13)**
4(p>0)	6	**(25,14)**
5(p>0)	2	**(26,15)**
6(p>0)	-2	(27,16)
7(p<0)	14	(28,16)
8(p>0)	10	(29,17)
9(p>0)	6	(30,18)



Questions on Unit-1

1) Define the following:

a) Pixel	b) Resolution	c) Bit Plane
d) Raster	e) Depth of the frame Buffer	f) Refresh Rate
g) Frame Buffer	h) Rasterization	i) Aspect Ratio

2) Discuss the applications of computer graphics.
3) With the help of a diagram, describe the open GL interface.
4) Describe the working of CRT with a neat diagram.
5) Differentiate Raster Scan and Random Scan Displays.
6) Differentiate DDA and Bresenham's line drawing algorithm.
7) Explain DDA Line Algorithm with an Example.
8) Define scan conversion. Write an algorithm of DDA line drawing.
9) Write an algorithm of DDA Line drawing. Consider a line AB with A(0,0) and B(8,4). Apply a simple DDA to calculate the pixels of this line.
10) Consider a line from (0,0) to B(6,6). Using simple DDA to calculate the points of this line.
11) Consider a line from (0,0) to (5,5). Using simple DDA to calculate the points of this line.
12) Use DDA to draw pixels of the line AB with A(1,1) and B(5,3).
13) Use Bresenham's line drawing algorithm to draw pixels of the line XY(5,5) and Y (13,9)
14) Use Bresenham's line drawing algorithm to draw pixels of the line XY(0,0) and Y (8,4)
15) Write the applications of line drawing algorithm.
16) Explain the architecture of raster scan system with suitable diagrams.
17) Rasterize the line segment from pixel coordinate (1, 1) to (8,5) using Bresenham's line drawing algorithm.
18) Scan convert the line segment with end points (0,0) and (10,5) using DDA line drawing algorithm. Find out and discuss the advantages and disadvantages of this method.
19) Describe the working of a beam penetration and shadow mask CRT.
20) Derive the expression for decision parameter used in Bresenham's line drawing algorithm.
21) Explain Random scan and Raster scan display system.
22) Explain Bresenham's Line Drawing Algorithm to draw line between 2 end points.
23) For 10×10 frame buffer, interpret the Bresenham's algorithm to find which pixels are turned on for the line segment (1, 2) and (7, 6).
24) Explain OpenGL Line Primitive functions with examples.
25) What are attribute functions? Explain OpenGL Point-Attribute Functions.
26) Explain OpenGL Line-Attribute Functions.
27) Write explanatory notes on: i) RGB color model; ii) indexed color model.
28) Explain the additive and subtractive colors, indexed color and RGB color concept.

Important Definitions

A. Pixel: A pixel is a short form of picture element.

- It is a single point in a graphic image.
- A pixel has two properties: a color and a position.
- Color is expressed in RGB (Red-Green-Blue) components - typically 8 bits per component or 24 bits per pixel (or true color).
- The position is expressed in terms of (x, y) coordinates.

B. Frame Buffer:

- Picture definition is stored in a memory area called **Frame Buffer.**
- It holds the color information of the pixels, hence also called **Color buffer.**

C. Aspect Ratio:

The aspect ratio of an image is the ratio of its width to its height. It is usually expressed as two numbers separated by a colon, such as 16:9 (width:height).

D. Resolution:

- Resolution is the amount of detail that can be seen in an image.
- It is usually measured in pixels per inch (PPI) or lines per inch (LPI).
- The higher the resolution, the sharper and clearer the image will be.

E. Scan Conversion/Rasterization:

It is the process of converting a vector image into a raster image.

F. Depth of the frame Buffer/Bit Plane:

It is number of bits per pixel in the frame buffer.

G. Raster:

It is an array of Pixels in the frame buffer.

H. Refresh Rate:

The frequency at which a picture (content of frame buffer) is redrawn on the screen is referred to as the **Refresh Rate.**

DDA Vs. Bresenham's line drawing algorithm		
Basis of comparison	**DDA**	**Bresenham's Algorithm**
Method	Multiplication and division are the only operations used.	Only addition and subtraction are used in this process
Efficiency	Less efficient.	More effective than the DDA algorithm.
Speed	Slower	Faster
Precision	It does not have a high degree of accuracy or precision.	It's extremely accurate and precise.
Complexity	In order to complete its tasks, it relies on complex calculations.	In order to do its tasks, it performs basic calculations.
Price	It comes at a high cost.	It is on the lower end of the price range.

Raster Scan Vs. Random Scan Displays	
Random Scan	**Raster Scan**
The resolution of random scan is higher in comparison raster scan.	The resolution of raster scan is lower in comparison to random scan.
It is expensive in comparison to raster scan.	It is inexpensive in comparison to random scan.
Any alterations can be done easily.	Alterations are difficult to make.
The concept of interweaving is not used.	The concept of interweaving is used.
A mathematical function is used to render an image or a picture.	To render image or picture, pixels are used.
It is suited for applications that require polygon drawings.	It is suitable to create realistic scenes.
An example of random scan is a pen plotter.	A TV set is an example of raster scan.
Random scan has lower refresh rate about 30 to 60 times per second.	Raster scan has higher refresh rate about 60 to 80 times per second.

Solved Example:

1. Consider three different raster systems with resolutions of 640 x 480, 1280 x 1024, and 2560 x 2048.

What size is frame buffer (in bytes) for each of these systems to store 12 bits per pixel?

Solution:

Because eight bits constitute a byte, frame-buffer sizes of the systems are as follows:

640 x 480 x 12 bits / 8 = 460800 bytes

1280 x 1024 x 12 bits / 8 = 1966080 bytes

2560 x 2048 x 12 bits / 8 = 7864320 bytes

2. Consider two raster systems with the resolutions of 640 x 480 and 1280 x 1024.

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?

Solution:

Since 60 frames are refreshed per second.

For the 640 x 480 System, - The access rate of such a system = (640 x 480) * 60 = 1.8432 x 107 pixels/second.

For the 1280 x 1024 System, - The access rate is (1280 x 1024) * 60 = 7.86432 x 107 pixels/second.

3. Suppose an RGB raster system is to be designed using an 8-inch by 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 bytes do we need for the frame buffer?

Solution:

The size of frame buffer is (8 x 10 x 100 x 100 x 6)/ 8= 600000bytes

4. Find out the aspect ratio of the raster system using 8 x 10 inches screen and 100 pixel/inch.

We know that,

$$\text{Aspect ratio} = \frac{\text{Width}}{\text{Height}}$$

$$= \frac{8 \times 100}{10 \times 100} = 4/5$$

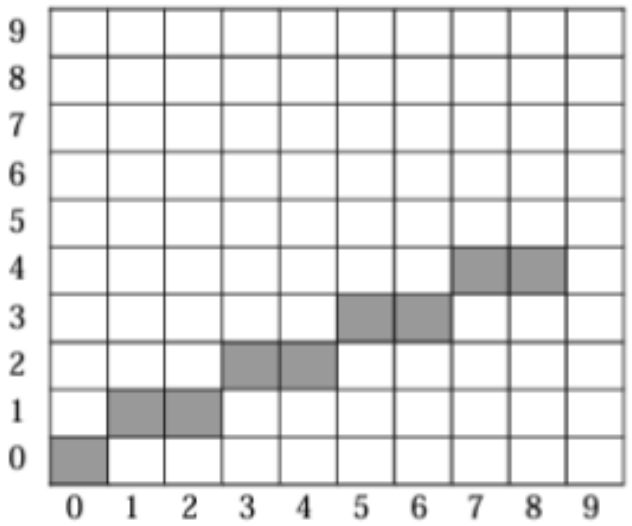
Aspect ratio = 4 : 5

Q.1. Consider a line AB with A(0,0) and B(8,4) apply a simple DDA algorithm to calculate the pixels on this line.

Solution:

1. A(0,0) B(8,4) x1=0 y1=0 x2=8 y2=4
2. dx = 8-0 = 8, dy = 4-0 = 4
3. Steps=8, dx>dy
4. xinr =1
   yinr = 0.5

v	x	y	plot
	0	0	(0,0)
0	1	0.5	(1,1)
1	2	1	(2,1)
2	3	1.5	(3,2)
3	4	2	(4,2)
4	5	2.5	(5,3)
5	6	3	(6,3)
6	7	3.5	(7,4)
7	8	4	(8,4)



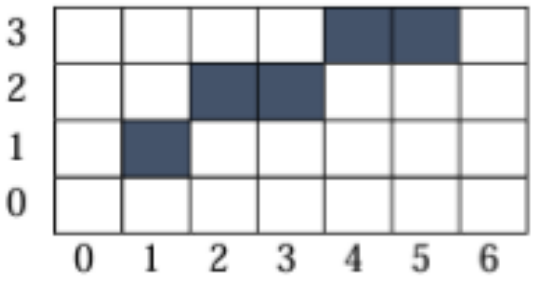
Plot of the Line AB

Q.2. Use DDA Line drawing algorithm draw a line AB for the endpoints A (1,1) and B(5,3)

Solution:

1. A(1,1) and B(5,3), x1=1 y1=1 x2= 5 y2=3
2. dx=5-1 = 4 , dy =3-1 =2
3. Steps= 4, dx>dy
4. xincr =1
   yincr = 0.5

v	x	y	plot
	1	1	(1,1)
0	2	1.5	(2,2)
1	3	2	(3,2)
2	4	2.5	(4,3)
3	5	3	(5,3)



Q.3. Consider a line AB with A (2,3) and B (6,8). Apply a simple DDA algorithm and calculate the pixels on the line

Solution:

1. A(2,3) and B(6,8) x1=2, y1 =3 x2=6 y2 =8
2. dx = x2-x1 = 6-2 =4 dy = y2-y1 = 8-3 =5
3. Steps=5,dy>dx
4. xincr =0.8
   yincr = 1

v	x	y	plot
	2	3	(2,3)
0	2.8	4	(3,4)
1	3.6	5	(4,5)
2	4.4	6	(4,6)
3	5.2	7	(5,7)
4	6	8	(6,8)