Computer Graphics with OpenGL

[1] Computer Graphics with OpenGL, Donald Hearn and M. Pauline Baker, International ed., Prentice Hall Inc., 2004

[2] 컴퓨터 그래픽스 with OpenGL, 번역서, 이병래, 김원철, 김동규, 강학수 공역, 홍릉과학출판사, 2006

강의노트:
vrlab.suwon.ac.kr/mWise

COMPUTER GRAPHICS & VISUALIZATION (18CS62)

TEXT BOOKS

1. **DONALD HEARN AND PAULINE BAKER: Computer Graphics–Open GL 3rd /4th Edition, Pearson Education, 2004.**

2. **EDWARD ANGEL: Interactive Computer Graphics A top down approach with OpenGL 5th edition Pearson Education 2008.**

Module-1 Overview

In this Module we discuss about :

- **Basics of computer graphics**
- **Application**
- **Display Devices:** Random Scan and Raster Scan displays,
- **Raster-scan systems**
- **Input devices,**
- **Graphics software- OpenGL**
- **Introduction to OpenGL ,**
- **Coordinate reference frames,**
- **OpenGL functions,**
- **Line drawing algorithms(DDA, Bresenham's),**
- **Circle generation algorithms (Bresenham's).**

What is Computer Graphics

Creation, Manipulation, and Storage of geometric objects (modelling) and their images (rendering).

Display those images on screens or output/display devices.

What is Computer Graphics

- Computer Graphics refers to the tools used to create Pictures

- The hardware tools include
  - Video monitors
  - Graphics cards
  - Printer
  - Input devices (mouse, data glove & trackball)

What is Computer Graphics

- The Software tools are
  - Operating System
  - Editor
  - Complier
  - Debugger

  Along with these graphics routines are required to create a picture

  eg. Graphics Libraries

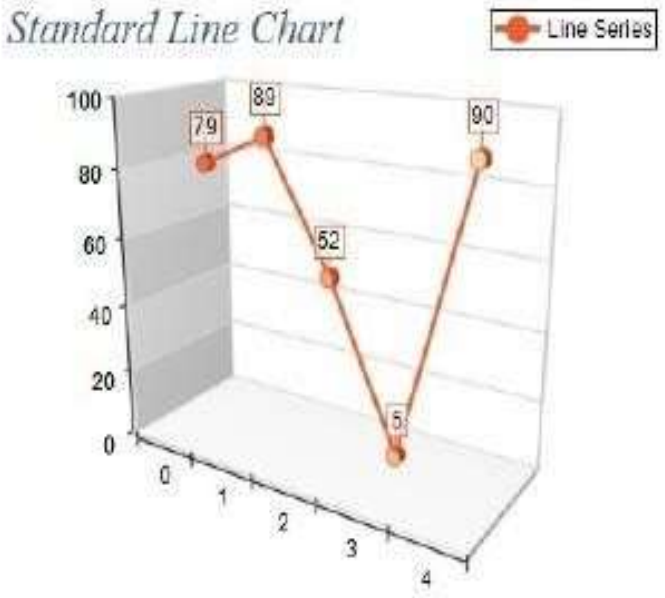
Applications of Computer Graphics

1. Graphs and Charts.
2. Entertainment
3. Computer Aided Design
4. Virtual reality environment.
5. Education and Training
6. Computer Animation
7. Data Visualization
8. Image processing
9. Graphical user interface

2.Presentation Graphics

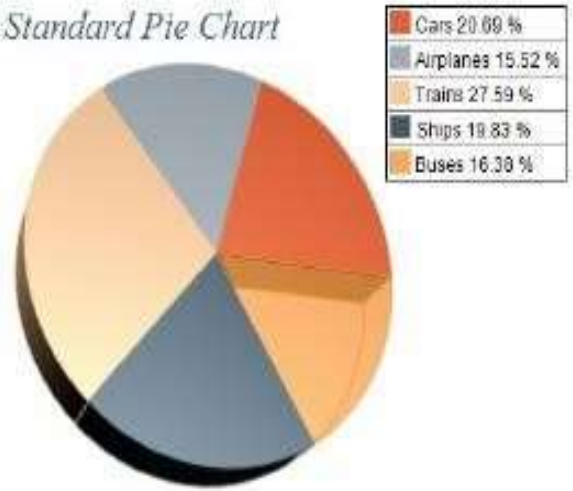
- Used to produce illustrations for reports or generate slides for use with projectors
- Commonly used to summarize financial, statistical, mathematical, scientific, economic data for research reports, managerial reports & customer information bulletins
- Examples : Bar charts, line graphs, pie charts, surface graphs, time chart

Examples of presentation graphics

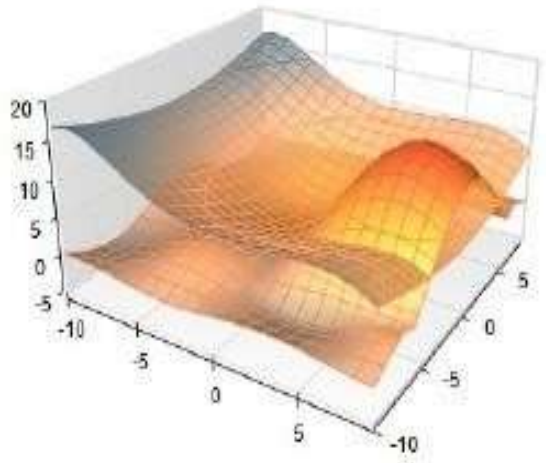


Examples of presentation graphics

Standard Pie Chart



Intersected Surfaces



3.Computer Art

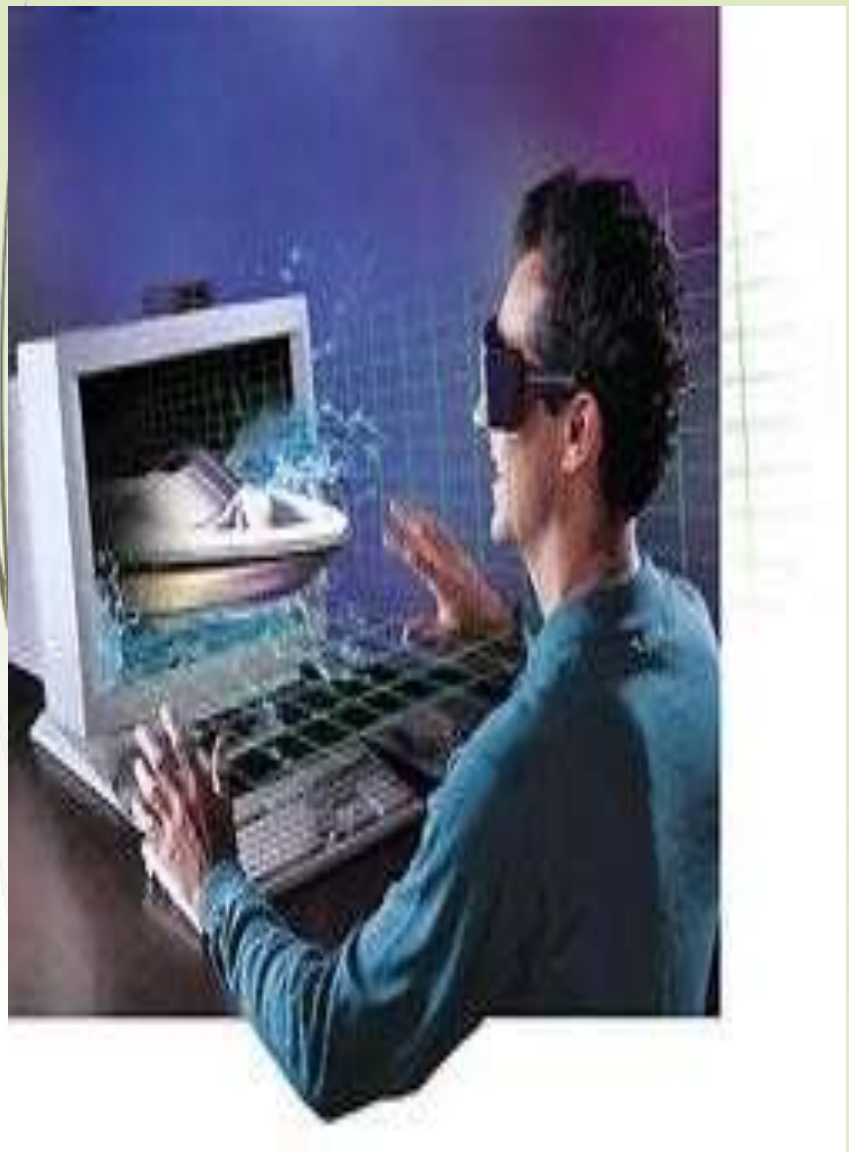
- Used in fine art & commercial art
  - Includes artist's paintbrush programs, paint packages, CAD packages and animation packages
  - These packages provides facilities for designing object shapes & specifying object motions.
  - Examples : Cartoon drawing, paintings, product advertisements, logo design

Examples :

Computer Art

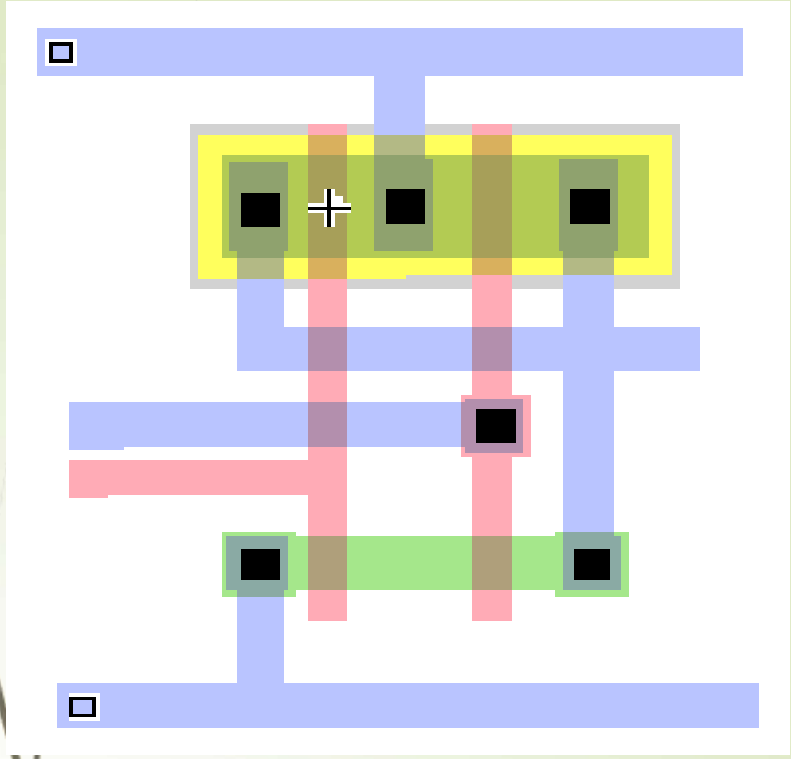
- Electronic painting
  - Picture painted electronically on a graphics tablet (digitizer) using a stylus
  - Cordless, pressure sensitive stylus
- Morphing
  - A graphics method in which one object is transformed into another

Video games

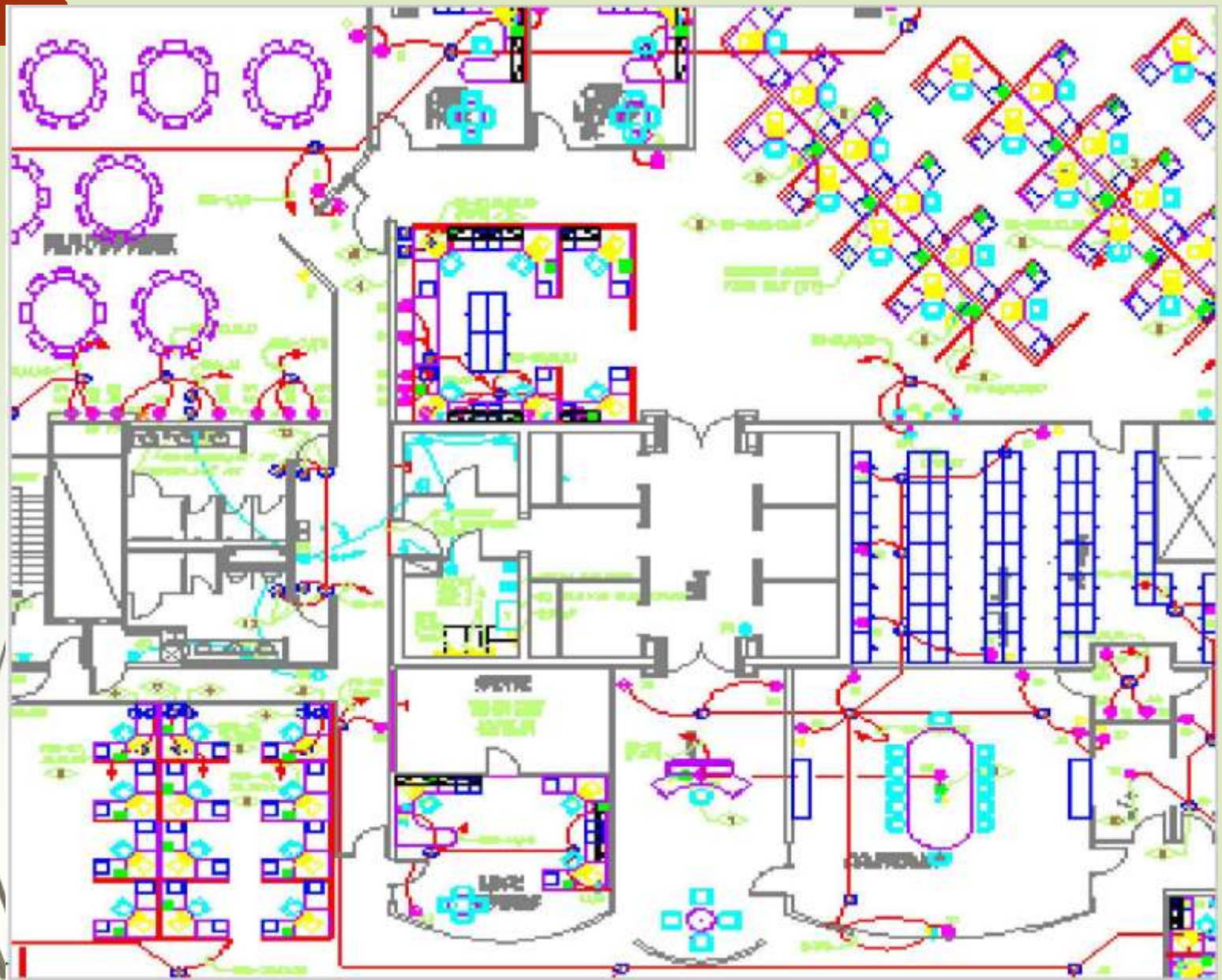


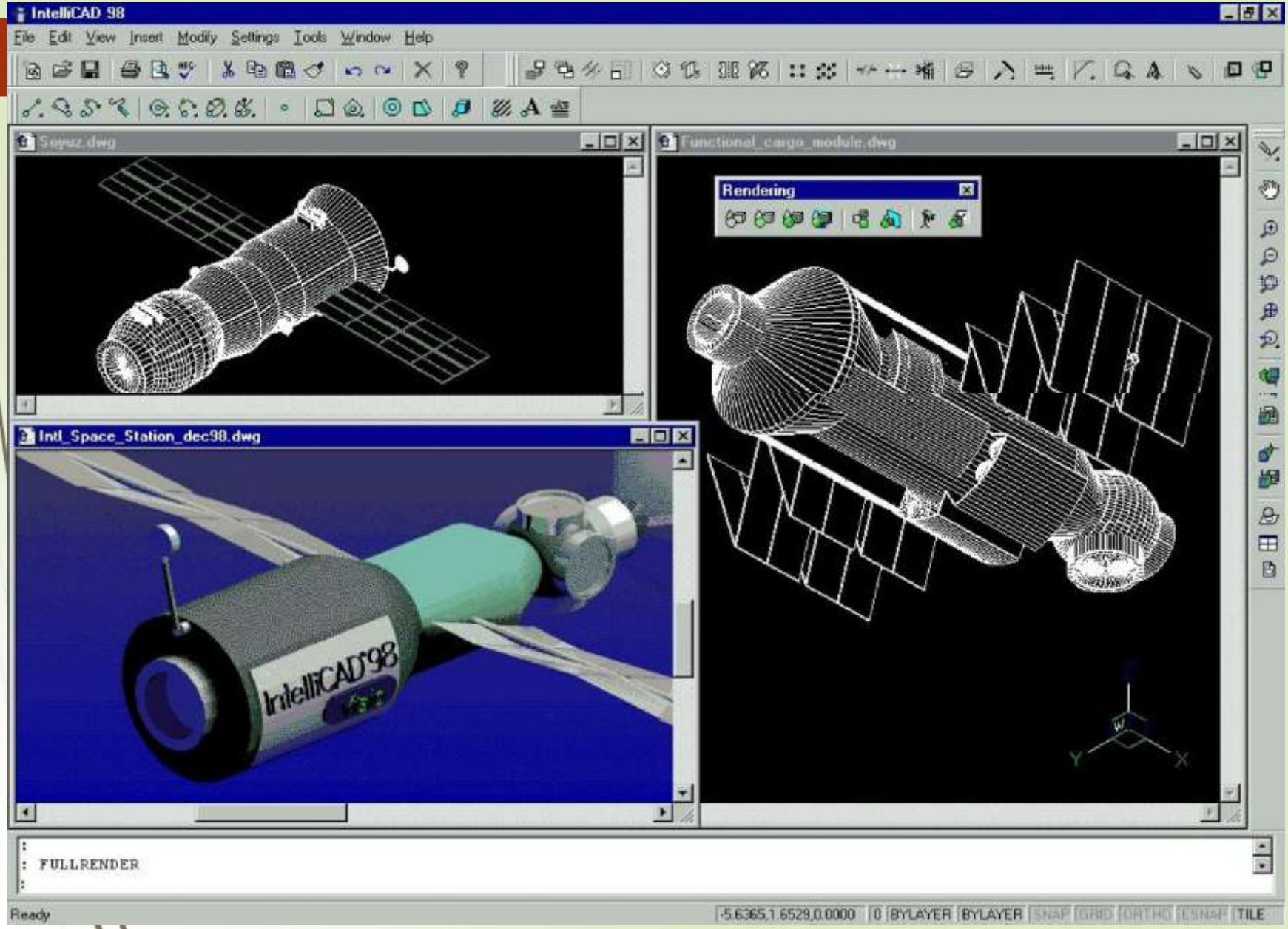
Games are very important in Computer Graphics

Computer Aided Design too

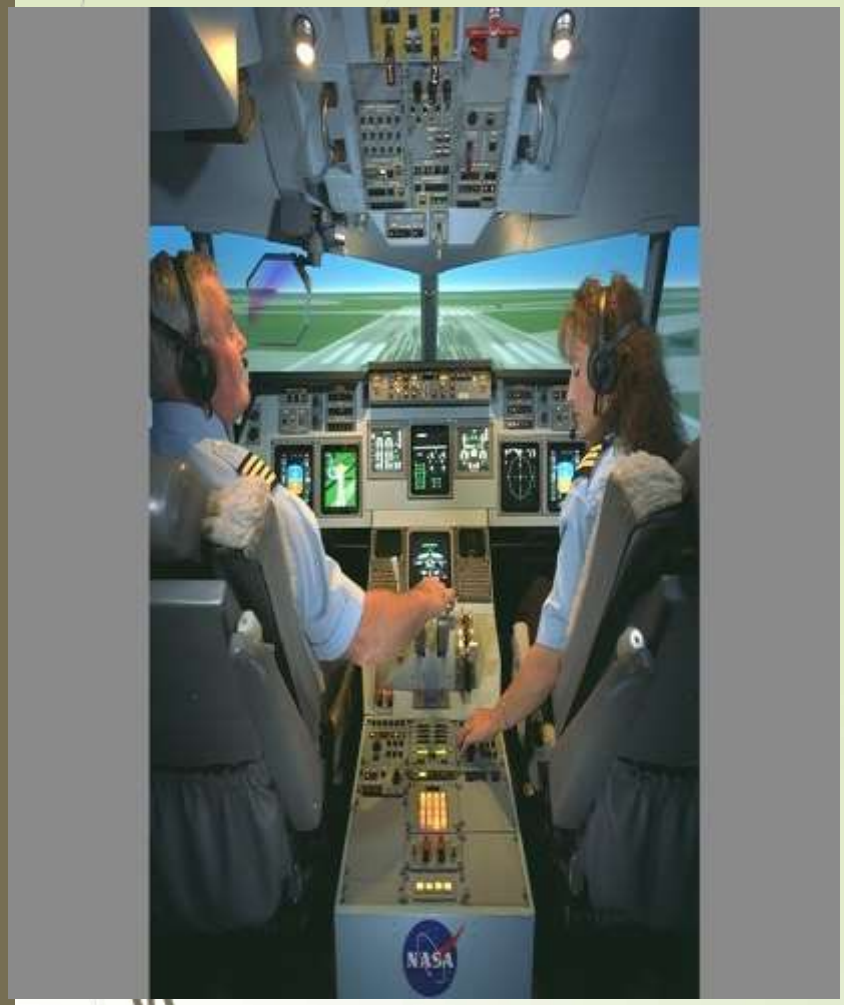


Floor plan

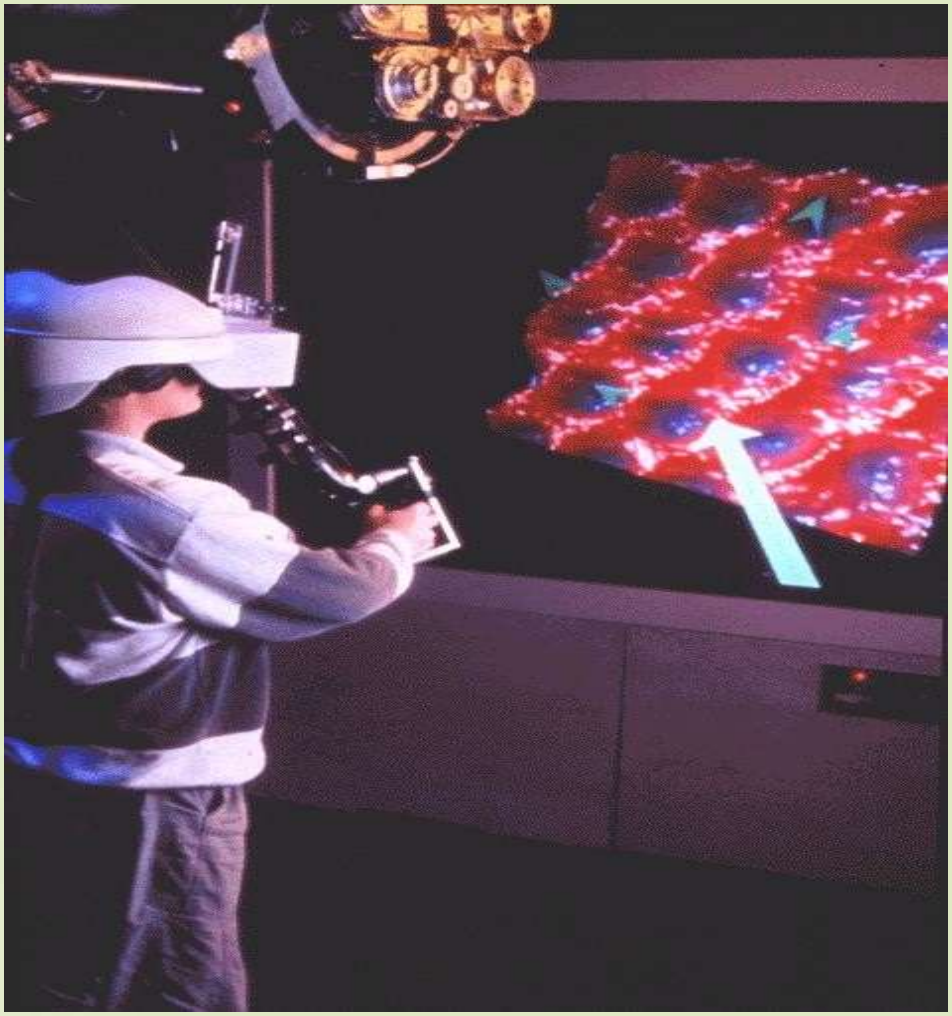



IntelliCAD 98
File Edit View Insert Modify Settings Tools Window Help
Soyuz.dwg
Functional_cargo_module.dwg
Rendering
Intl_Space_Station_dec98.dwg
IntelliCAD'98
:
: FULLRENDER
:
Ready
-5.6365,1.6529,0.0000
0 BYLAYER BYLAYER SNAP GRID ORTHO ESNAP TILE

Flight simulator



Virtual reality



Education and Training

- Graphical flight simulator has proved to increase the safety and to reduce training expenses.

- The field of virtual reality has opened many new paths.

  for ex.

  Stereoscopic vision.

  Surgical training.

  Astronauts trained in weight less environment.

  Video games use standard computer and specialized hardware boxes.

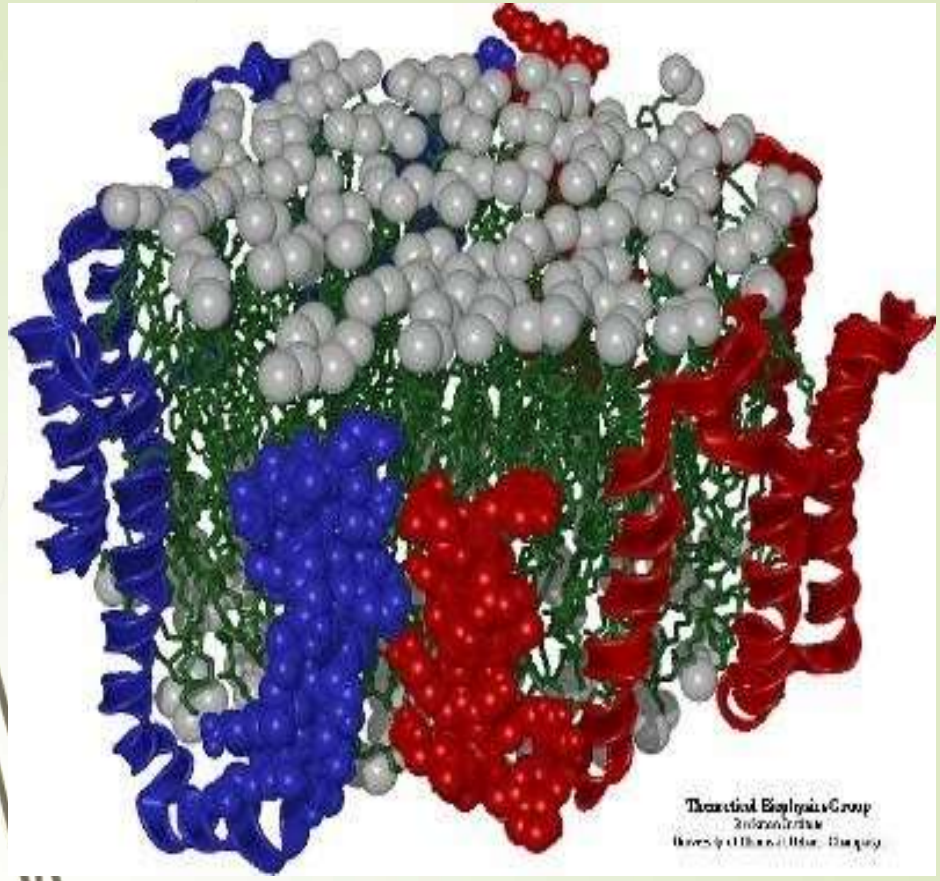
Training

Computer Graphics is about animation (films)

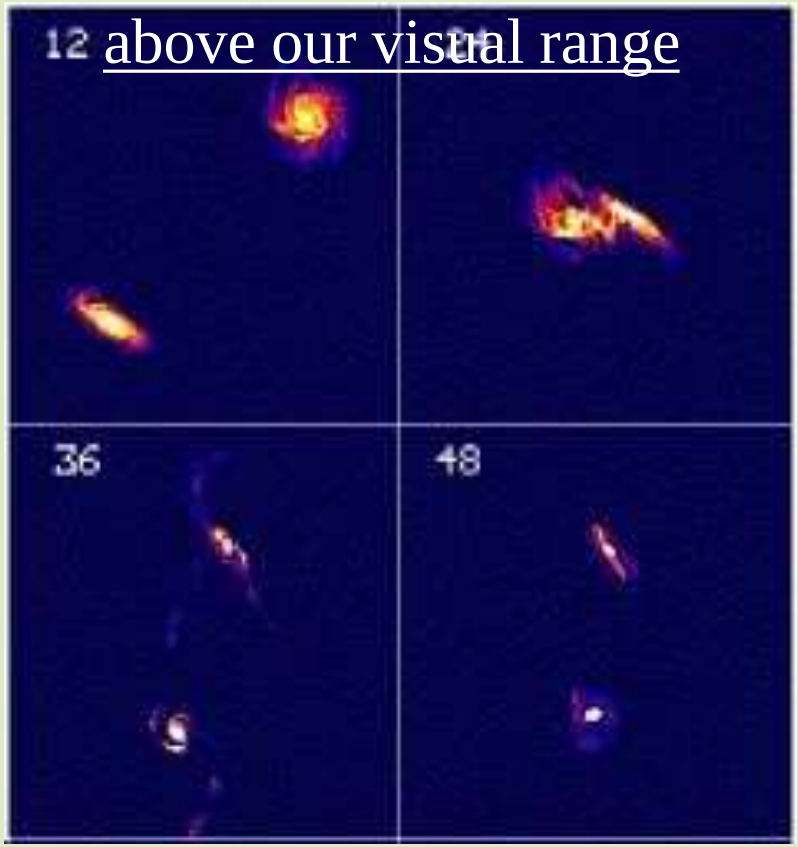
- Major driving force now

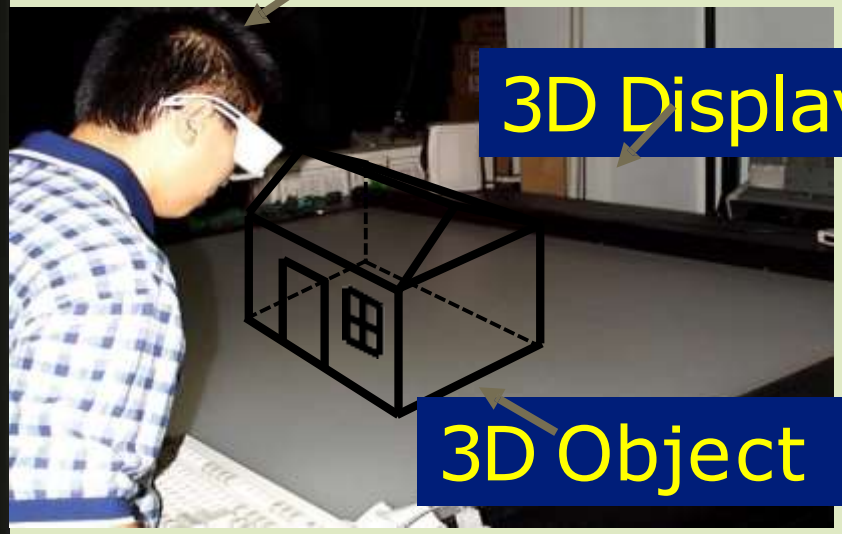
Medical Imaging is another driving force

Scientific Visualisation



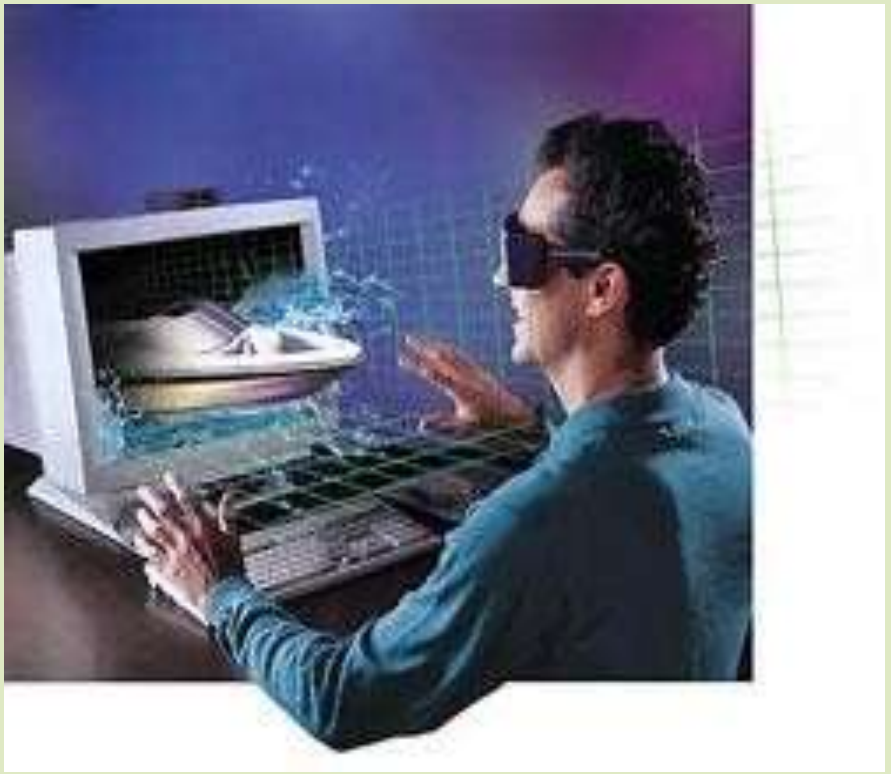
To view below and above our visual range




3D Glasses
3D Display
3D Object

Head-Mounted Displays (HMDs)

The display and a position tracker are attached to the user's head



Head-Tracked Displays (HTDs)

Display is stationary, tracker tracks the user's head relative to the display.

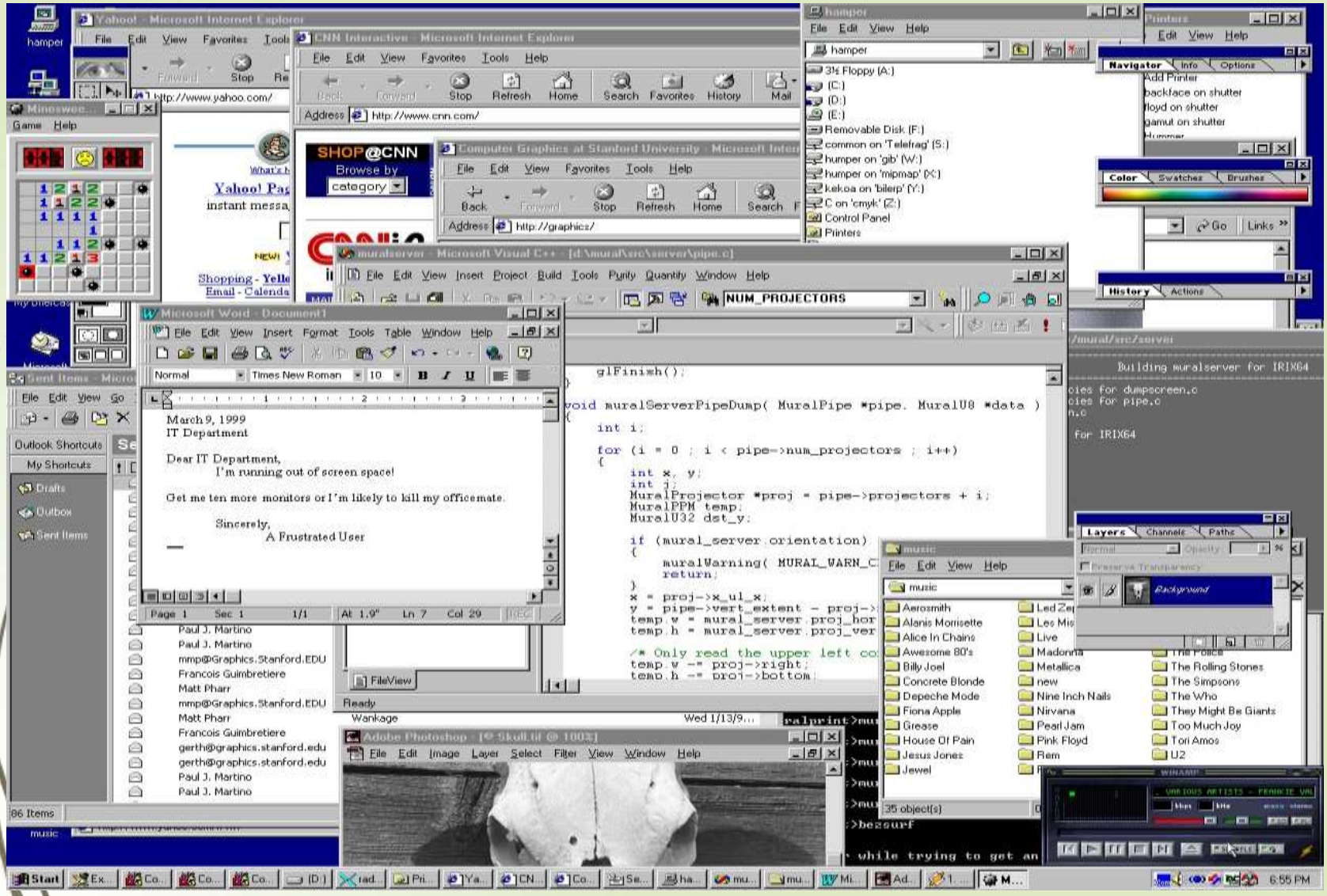
Example: CAVE, Workbench, Stereo monitor

Graphical User interfaces

Advances in computer graphics made different types of interfaces.

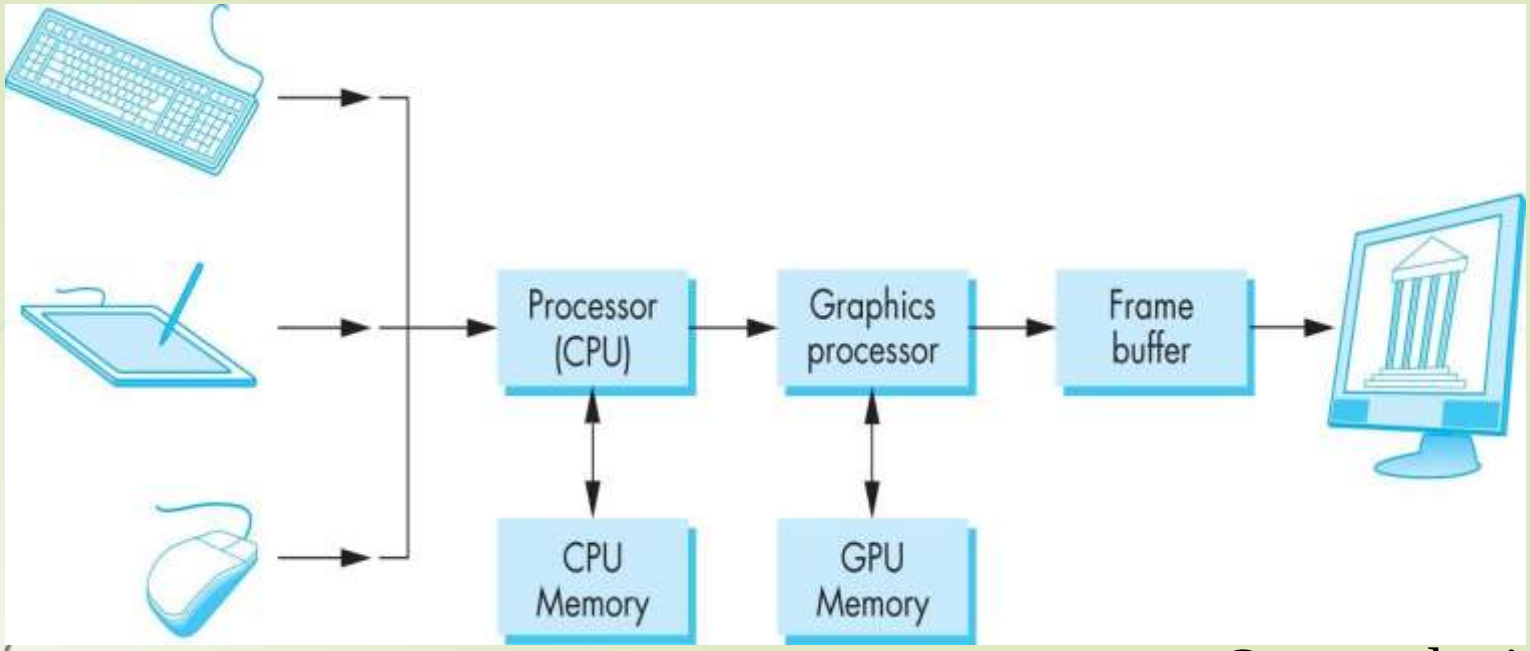
User can interact with the computer using windows icons, menus & pointing devices.

Operating system provides user interface



Window system and large-screen interaction

Graphics System



Input devices

Output device

Image formed in frame buffer

Components of Graphics System

1. **Input devices**
2. **Framebuffer & Memory**
3. **Processor**
4. **Output devices**

Input devices

- **Locator Devices**
- **Keyboard**
- **Scanner**
  - **Images**
  - **Laser**
- **Cameras**

INPUT DEVICES

keyboard

scanner

laser range finder

Data glove

camera

data tablet

Locator devices/pointing devices

Processor

CPU – both normal operation and Graphical operation.

GPU -- Graphical Processing Unit, a special purpose processor, which use Graphical Primitives (line, circle, polygon) generated by application programs and assign values to the pixels in the frame buffer that represent these entities.

For example, triangle is represented by 3 vertices and line segments connecting these vertices.

All Graphic system are raster based.

" The conversion of geometric entities to pixel colours and locations in the frame buffer is known as ***Rasterization or Scan Conversion*** *"*

Output devices: 2 types

- Graphic output without display
  - Printers, Plotters…..
- Graphic output with display
  - **Cathode Ray Tubes**
    - **Vector (Random)scan Display**
    - **Raster scan Display**
  - Flat panel display
    - Plasma panel
    - LCD
    - LED

Cathode-ray tube (CRT) Monitors

Primary output device – Video monitors

- **Standard design of video monitor: Cathode-ray tube (CRT)**

A cathode ray tube

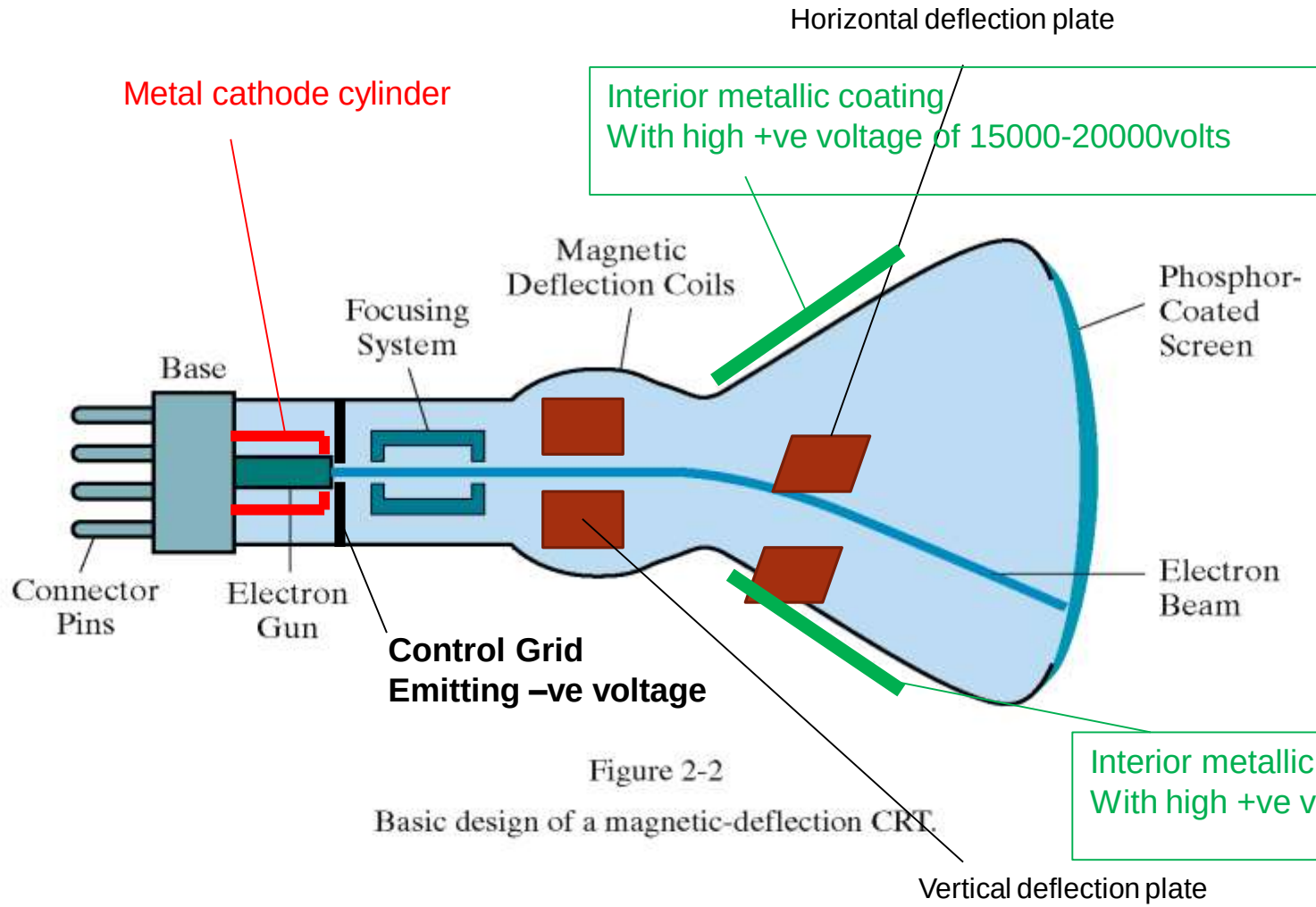
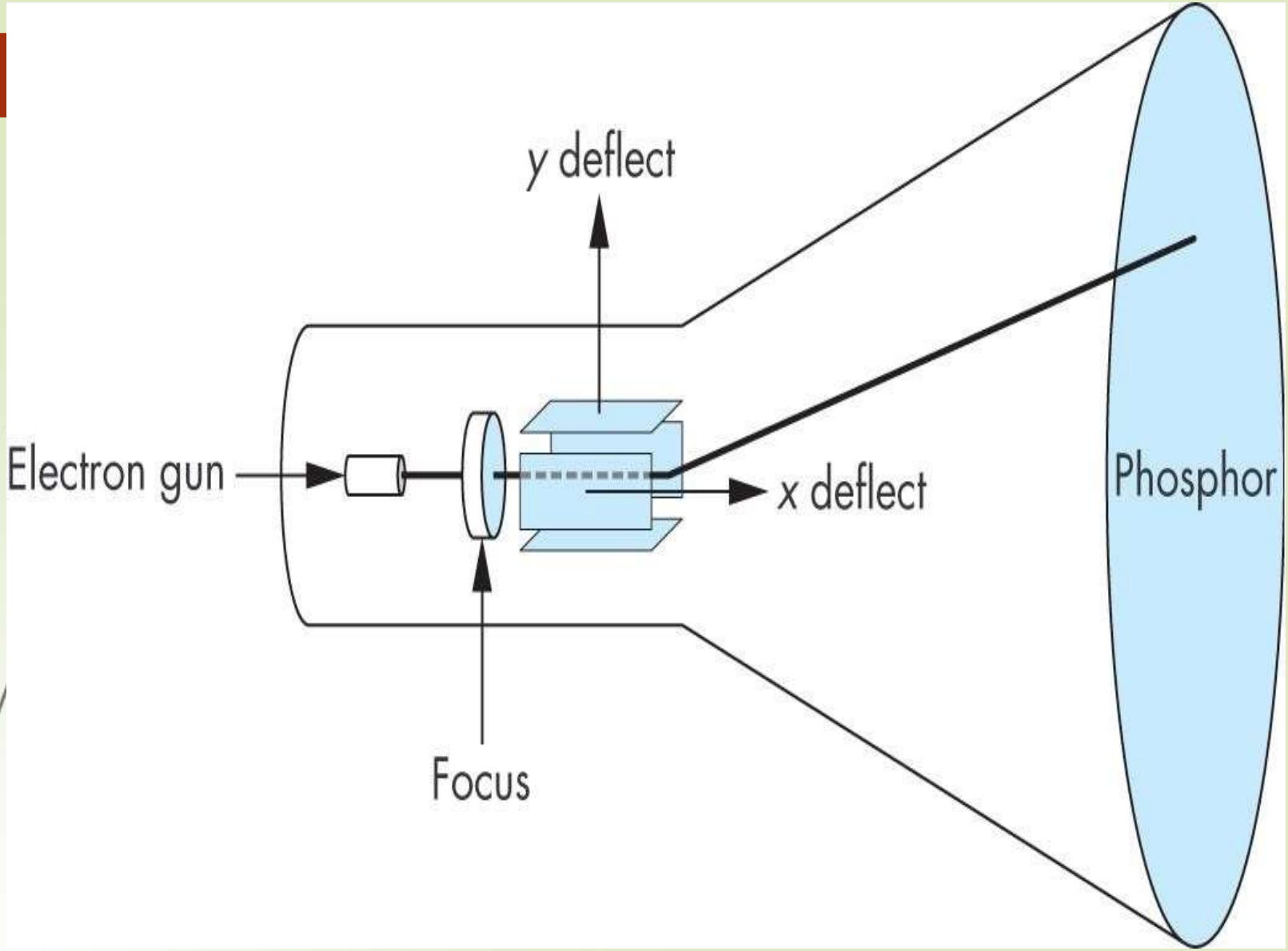


Figure 2-2

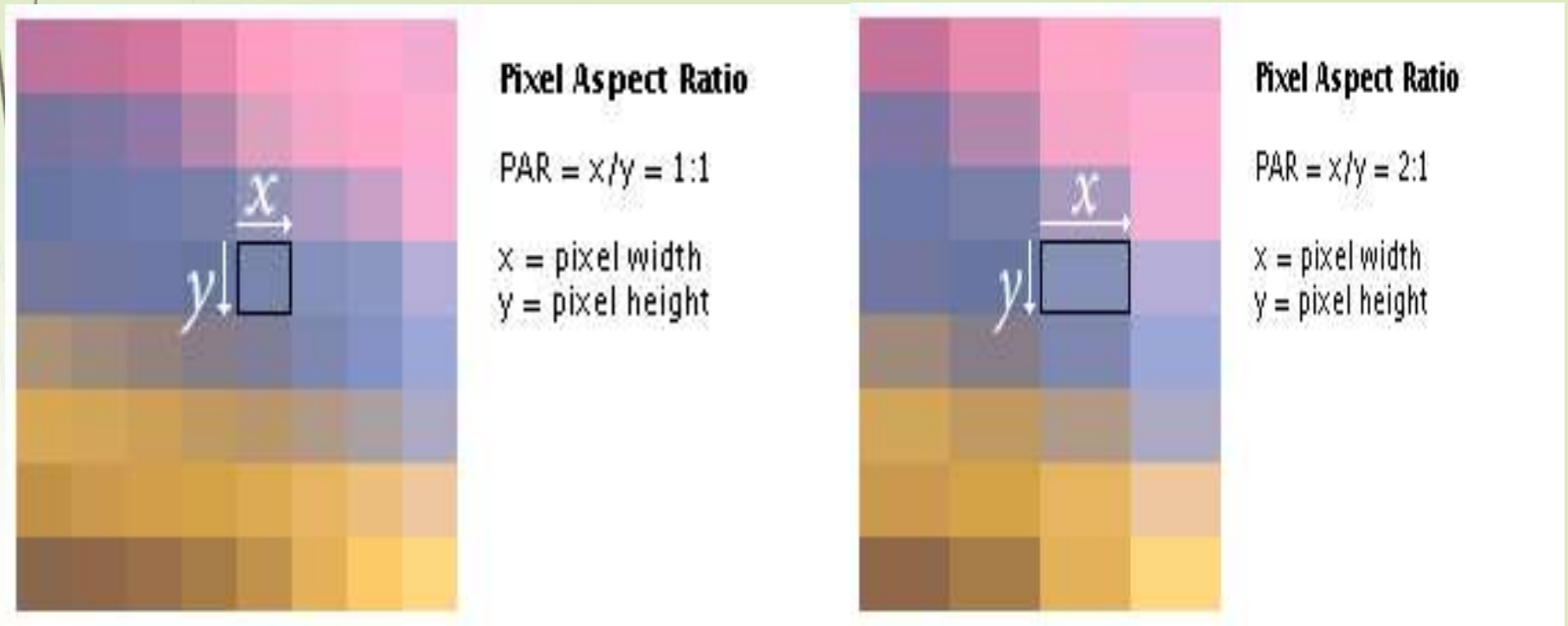
Basic design of a magnetic-deflection CRT.


y deflect
Electron gun
x deflect
Phosphor
Focus

- Beam of electrons hit phosphor-coated screen, light emitted by phosphor.
- The direction of the beam is controlled by two pairs of deflection plates.
- The output of the computer is converted by DAC, to voltages across the X and Y deflection plates.
- Light appears on the surface of CRT when sufficient beam of electrons is directed at phosphor.
- CRT will emit the light for a few milliseconds, in order to get a steady flicker free image same path must be retraced at high rate in order to keep phosphor activated.
- The frequency at which a picture is redrawn on the screen is referred to as the "**refresh rate**" in current display operate at a rate up to 50-85 HZ

Frame aspect ratio (FAR) = horizontal/vertical size

- TV 4:3
- HDTV 16:9
- Page 8.5:11 ~ 3/4



RANDOM SCAN DISPLAYS AND RASTER SCAN DISPLAYS

Refreshing techniques:

- Raster scan display
- Random scan display

Raster: A **raster scan**, or **raster scanning**, is the rectangular pattern of image capture in television.

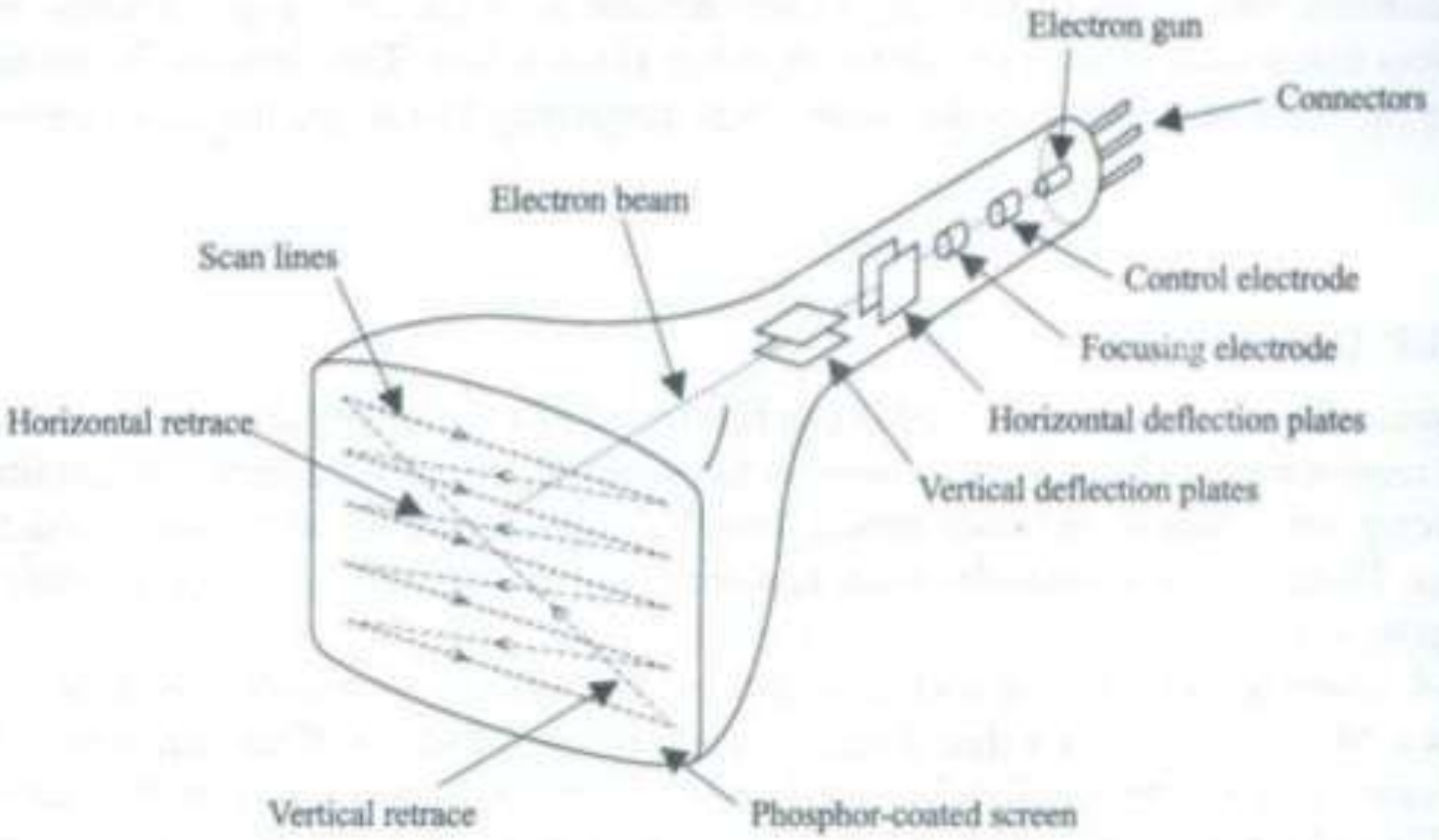
- It's a rectangular array of points or dot.
- The word *raster* comes from the Latin word *rastrum* (a rake).

- An image is subdivided into a sequence of strips known as "scan lines" which can be further divided into discrete pixels for processing in a computer system.

WORKING

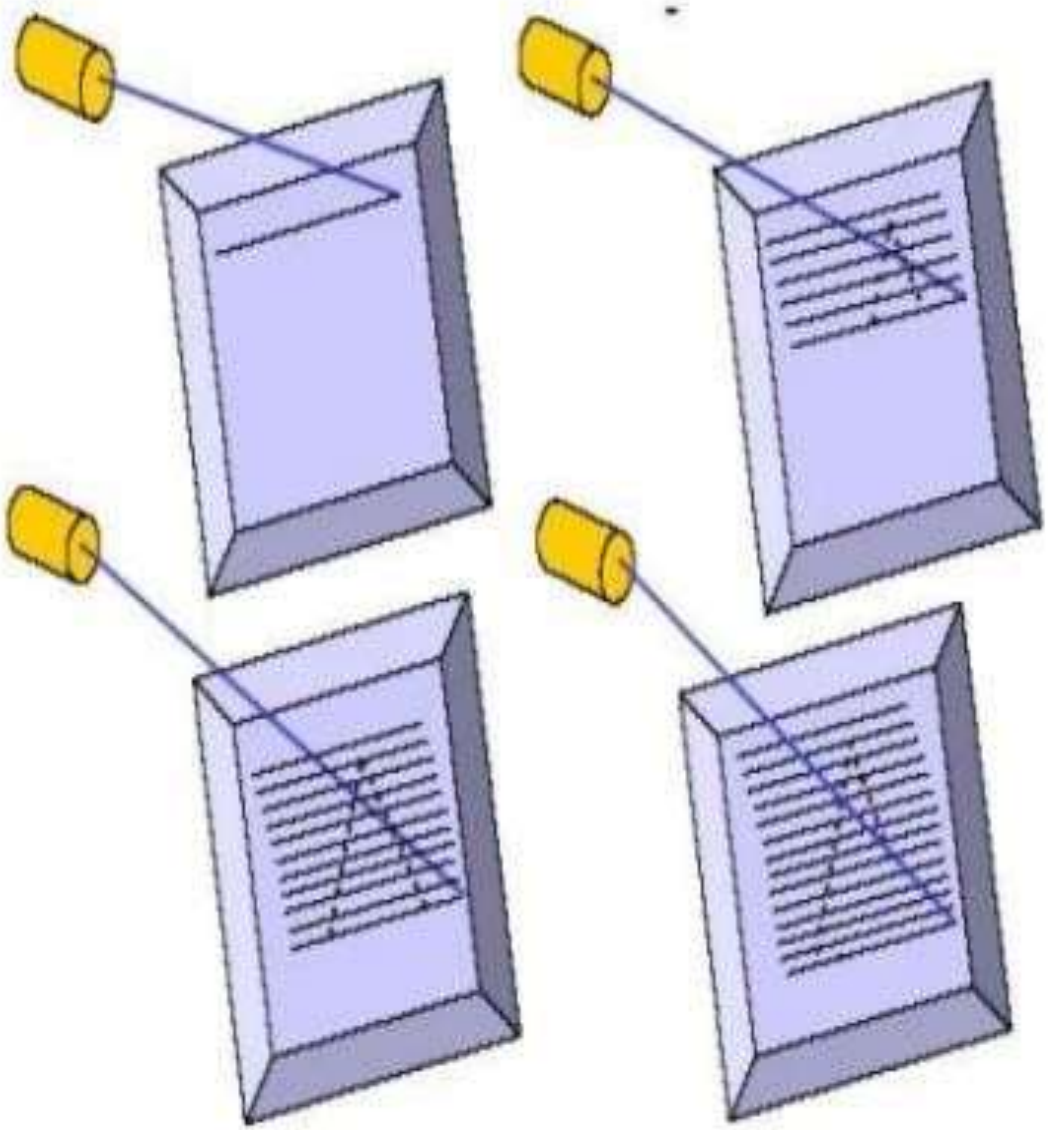
- In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom.
- As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.
- The return to the left of the screen, after refreshing each scan line is called **Horizontal retrace**.
- At the end of each frame the electron beam returns to the top left corner of the screen to begin the next frame is called **Vertical retrace**:

Raster Scan Display



WORKING

- Picture definition is stored in a memory area called the **refresh buffer** or **frame buffer**.
- **Refresh buffer** or **frame buffer** is memory area that holds the set of intensity values for all the screen points.
- Stored intensity values then retrieved from refresh buffer and "**painted**" on the screen one row (**scan line**) at a time.



Object as set of discrete points across each scan line

- The quality of a raster image is determined by the total number pixels (**resolution**), and the amount of information in each pixel (**color depth**)
- A black-and-white system: each screen point is either on or off, so only **one bit** per pixel is needed to control the intensity of screen positions. Such type of frame buffer is called Bit map
- High quality raster graphics system have **24** bits per pixel in the frame buffer (a **full color** system or a **true color** system)
- Refreshing on raster scan displays is carried out at the rate 60 to 80 frame per second.

HISTORY:

- The use of raster scanning in television was proposed in 1880 by French engineer Maurice Leblanc.
- The term *raster* was used for a halftone printing screen pattern as early as 1894.
- The first use of *raster* specifically for a television scanning pattern is often credited to Baron Manfred von Ardenne who wrote in 1933.
- in January 1930 it was proven by demonstrations that the Braun tube was prototyped in the laboratory with point sharpness and point brightness for the production of a precise, bright raster.
- In analog TV, originally it was too costly to create a simple sequential raster scan of the type just described with a fast-enough refresh rate and sufficient horizontal resolution, although the French 819-line system had better definition than other standards of its time.

Architecture of Raster System

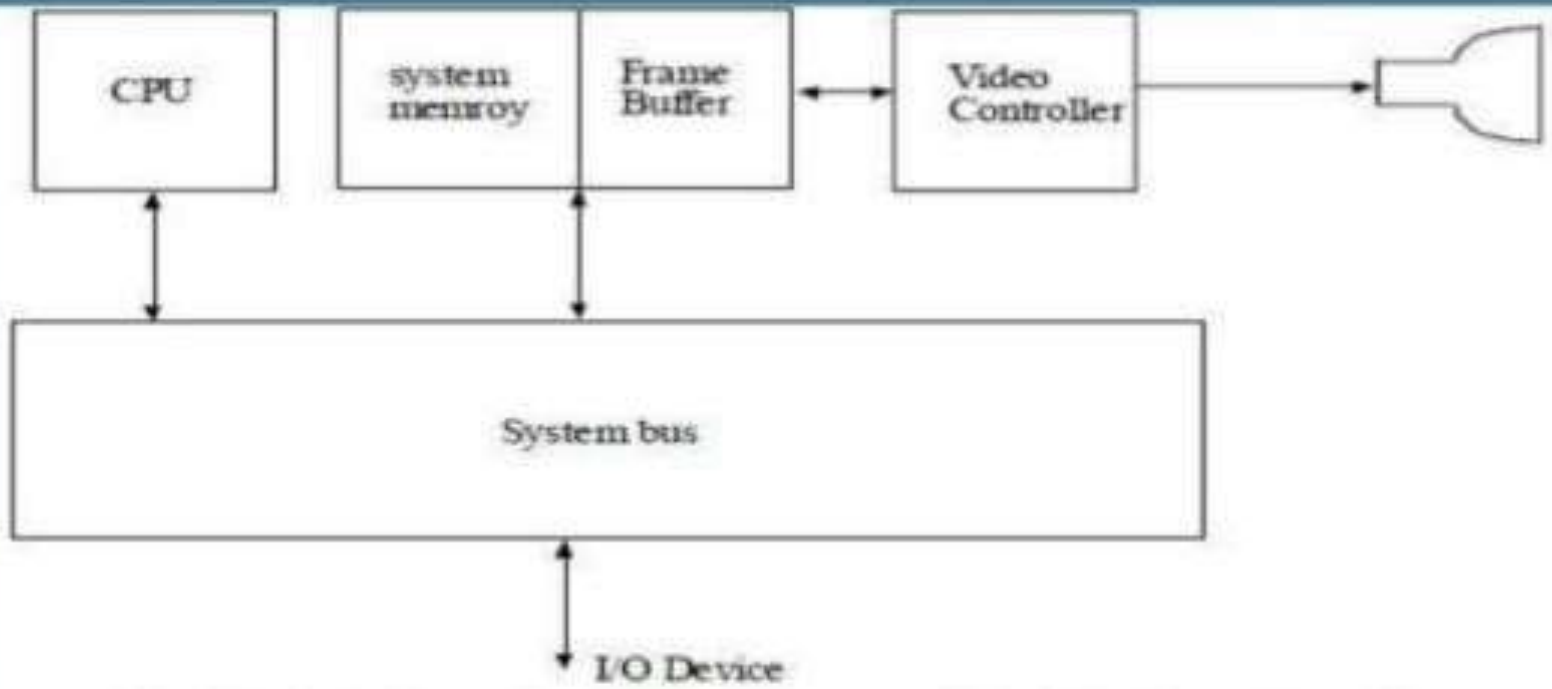


Fig. Architecture of a raster system with a fixed positon of the system memory requred for the frame buffer

FRAME BUFFER

- The image is stored in a frame buffer containing the total screen area and each memory locations corresponds to pixel.
- Frame buffer also contains the colour for each pixel.
- Examples- Television panels ,printer (99% of raster scan)

VIDEO CONTROLLER

- It is used to scan each and every line of the refresh or buffer memory
- The lines are scanned from left to right and when line is finishes ,it get to the next line and so on ..
- It has the direct access to refresh buffer to retrieve all the coordinates corresponding to each pixels.
- Two registers are used to represent the coordinates .

Raster Scan

The graphics system takes pixel from the frame buffer and display them as a points on the display in two different ways.

1. Non interlaced/Progressive displays

The pixels are displayed row by row or scan-line by scan-line, at the refresh rate.

2. Interlaced display

Odd rows and even rows are refreshed alternately. It is used in Commercial television.

Scan frame 30 times per second.

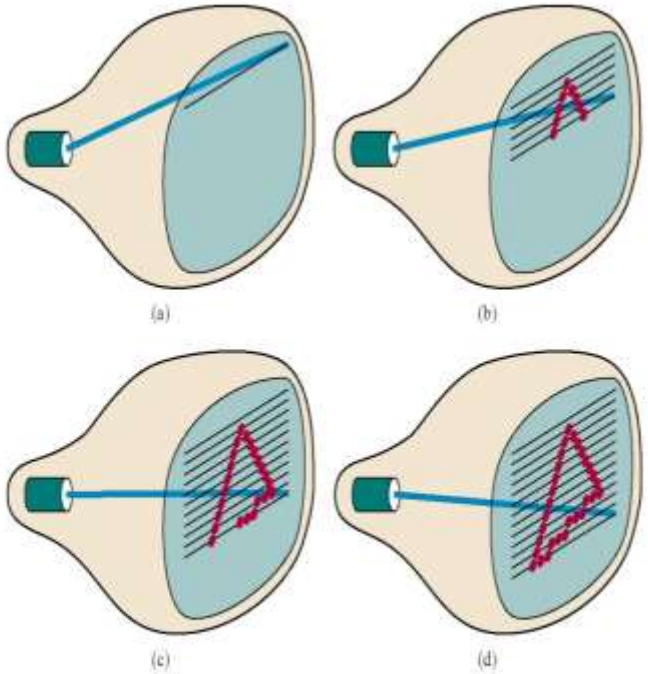
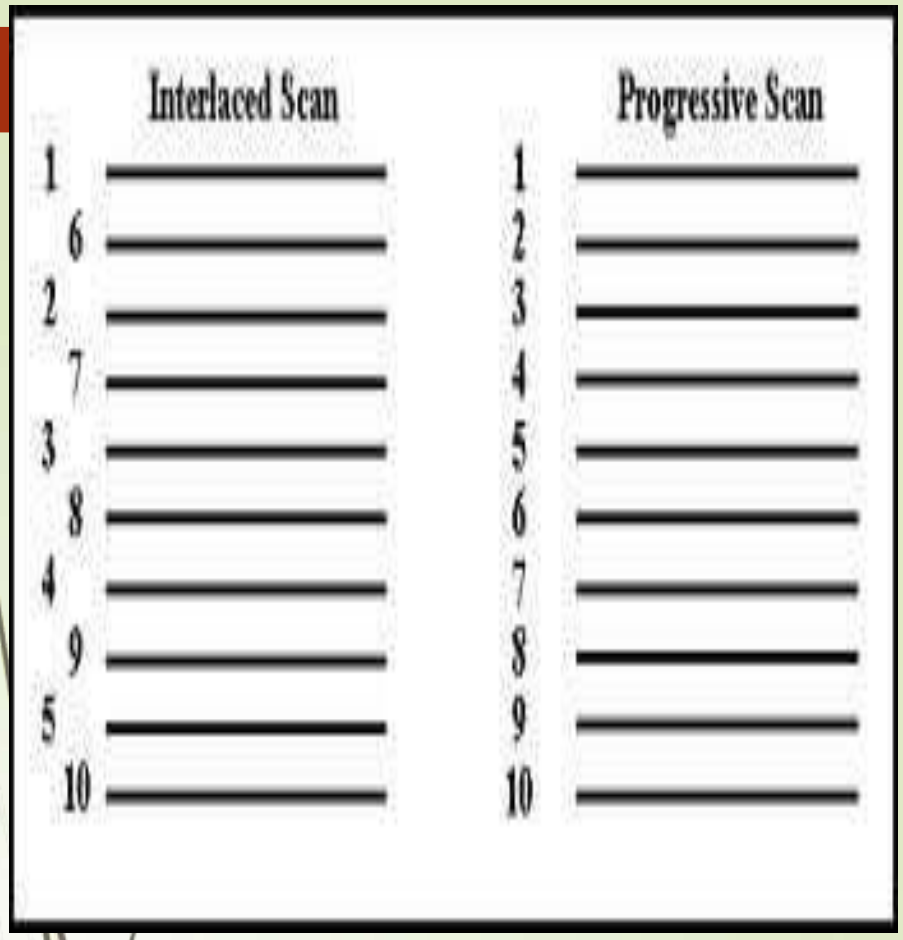
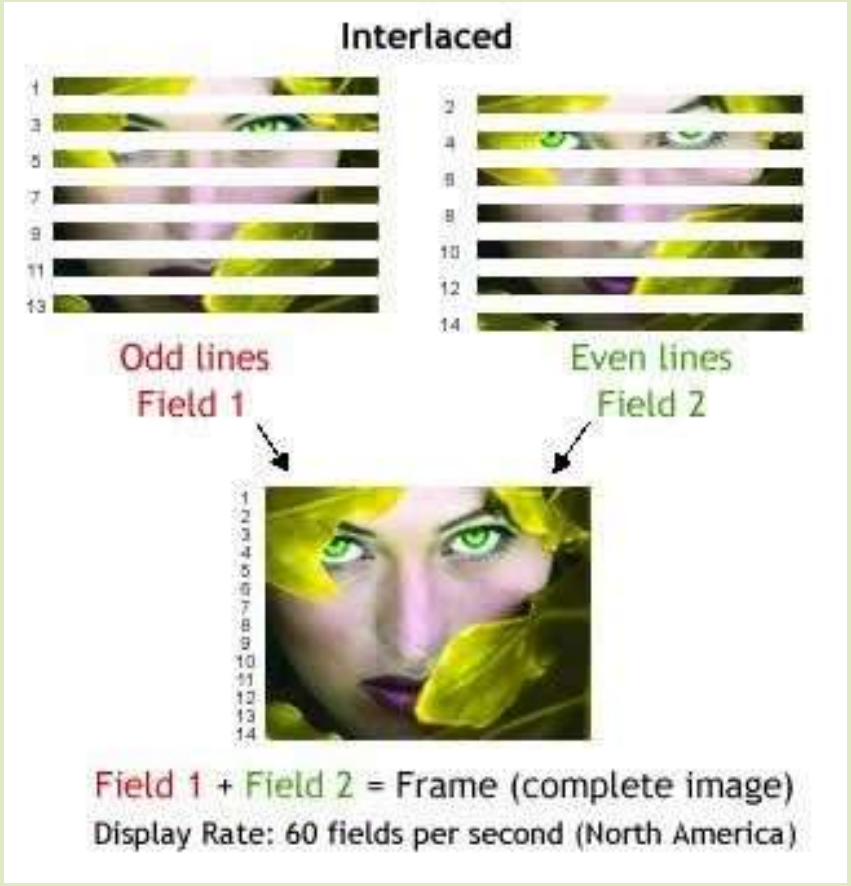


Figure 2-7
A raster-scan system displays an object as a set of discrete points across each scan line.

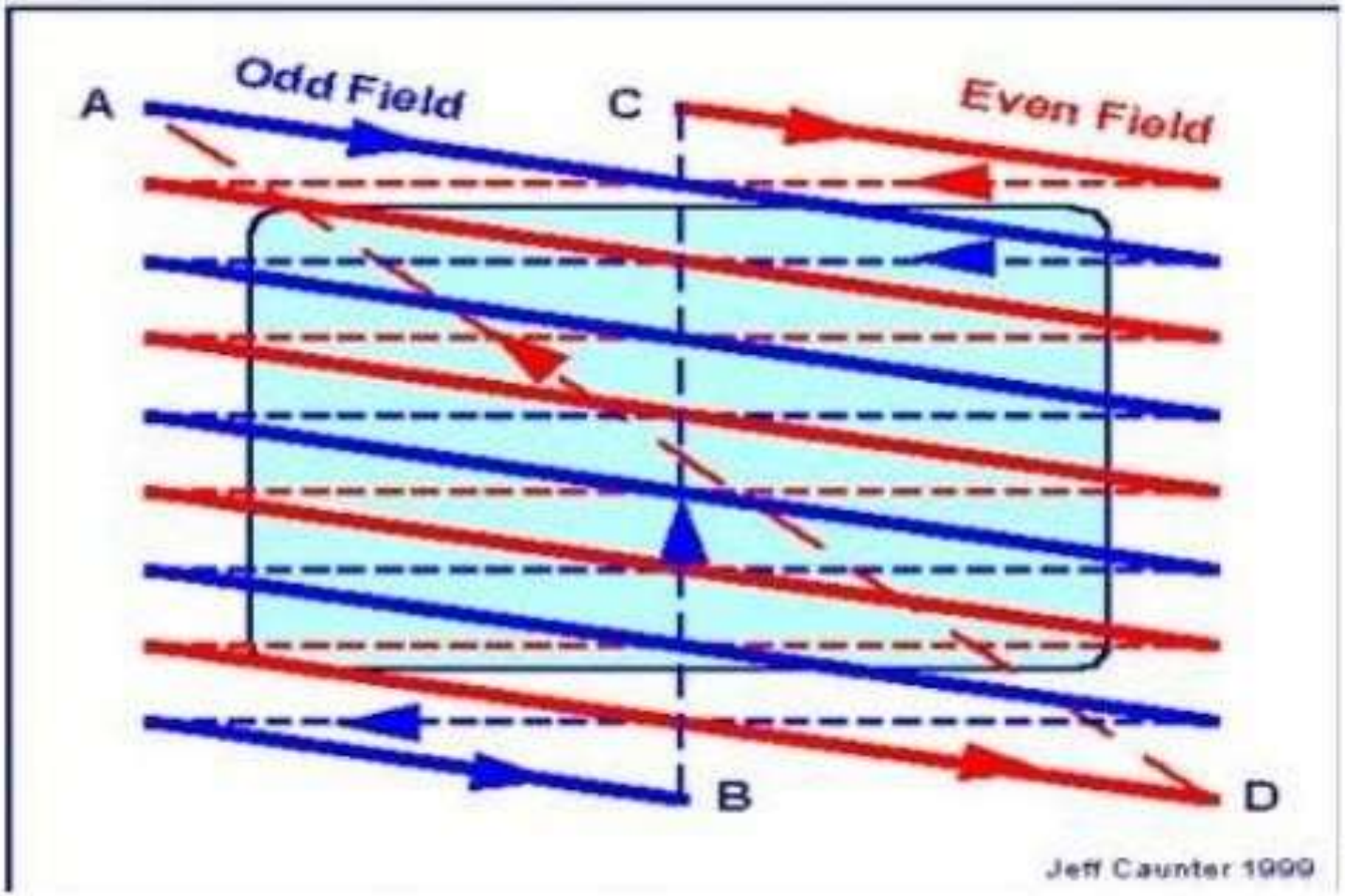
Interlacing

- On some raster systems(TV), each frame is displays in two parsing using an interlaced refresh procedure.
- Interlacing is primarily used for slower refresh rates.
- An effective technique to avoid **Flicker**. (Flicker occurs on CRTs when they are driven at a low refresh rate, allowing the brightness to drop for time intervals sufficiently long to be noticed by a human eye).

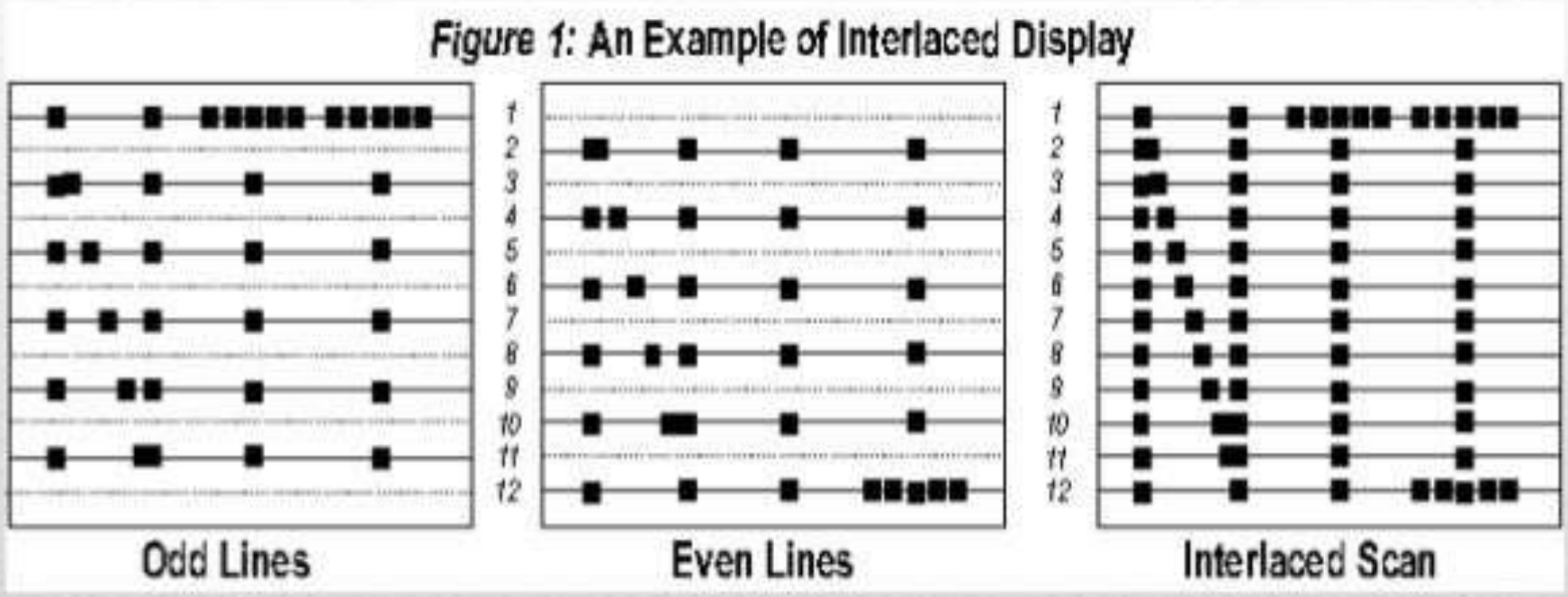

Interlaced Scan
1
6
2
7
3
8
4
9
5
10
Progressive Scan
1
2
3
4
5
6
7
8
9
10


Interlaced
Odd lines
Field 1
Even lines
Field 2
Field 1 + Field 2 = Frame (complete image)
Display Rate: 60 fields per second (North America)

INTERLACING



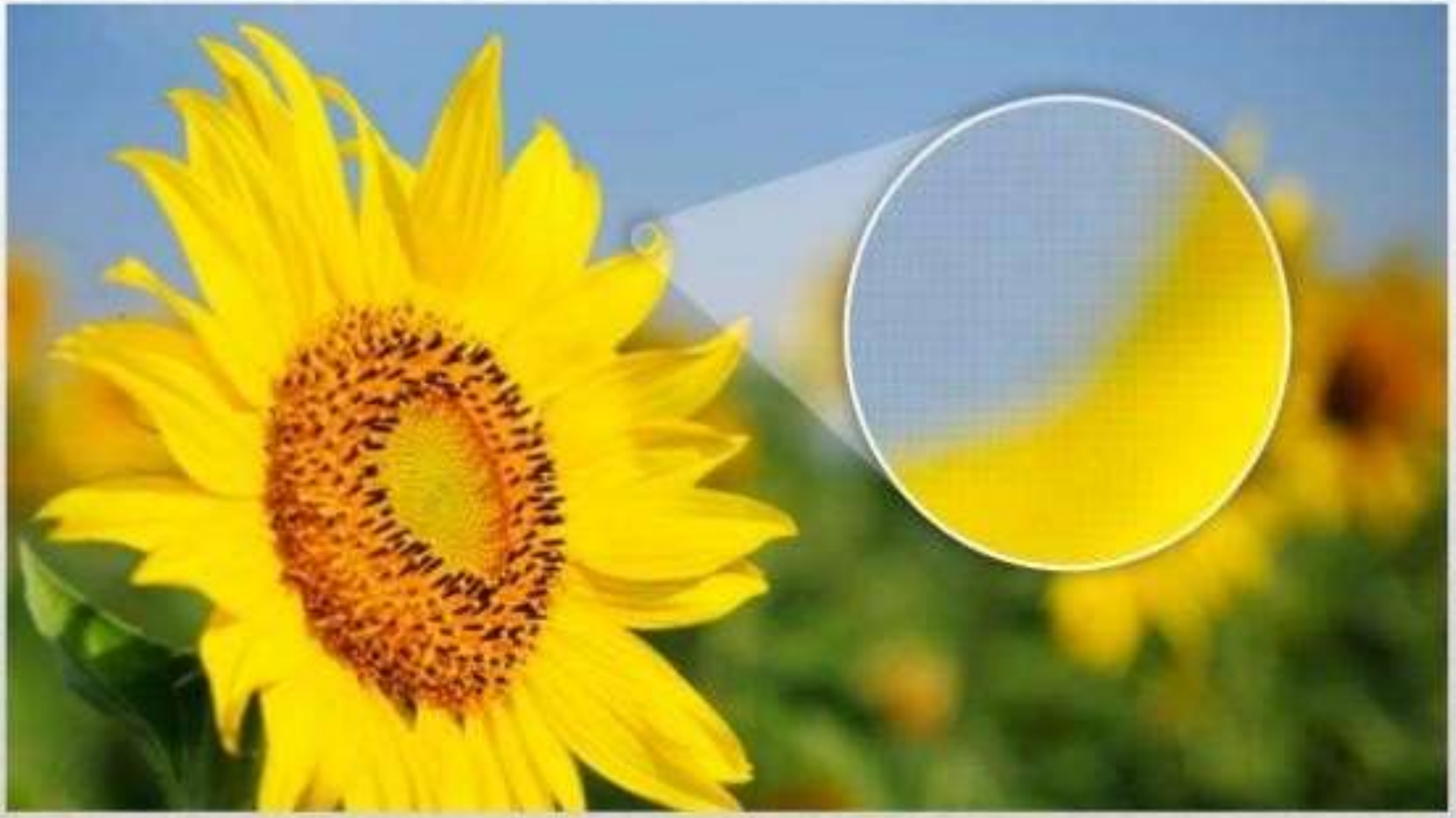
Interlacing



Raster Image

The quality of raster image is determined by total number pixels (resolution), and the amount of information in each pixel (color depth).

Raster Image

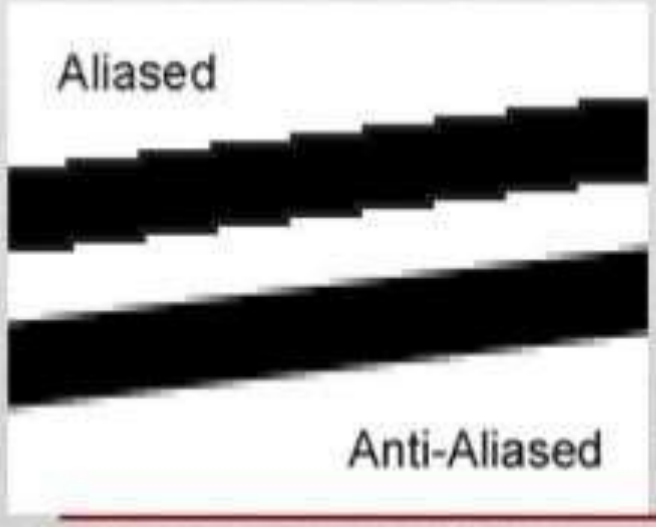


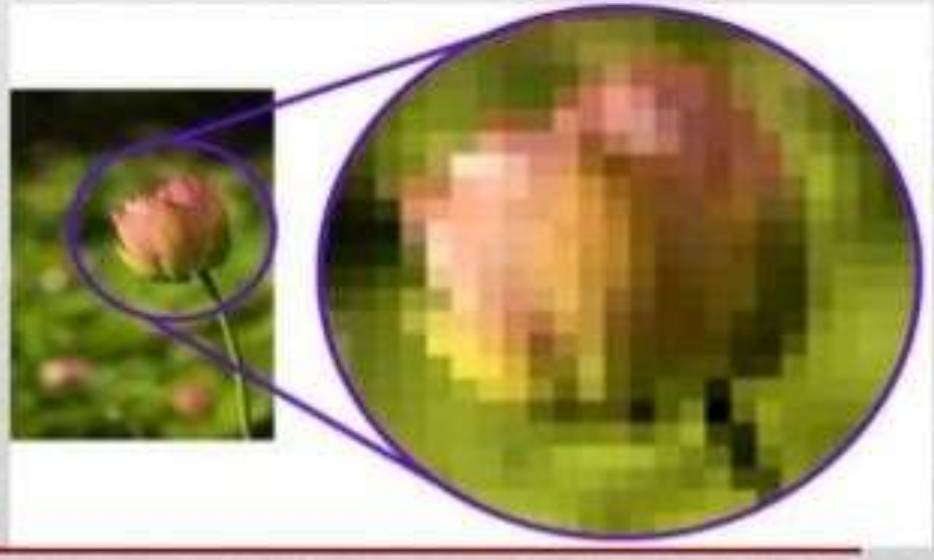
ADVANTAGES

- Raster scan is a technology which is very much efficient.
- It requires little memory.
- This technology is less costlier.

Disadvantage

- To increase size of a raster image the pixels defining the image are be increased in either number or size spreading the pixels over the large area causes the image to lose detail and clarity.
- Produces jagged line that are plotted as discrete points





Pros and Cons

- Advantages to Raster Displays
  - lower cost
  - filled regions/shaded images
- Disadvantages to Raster Displays
  - a discrete representation, continuous primitives must be **scan-converted** (i.e, fill in the appropriate scan lines)
  - **Aliasing** or "**jaggies**" Arises due to sampling error when converting from a continuous to a discrete representation

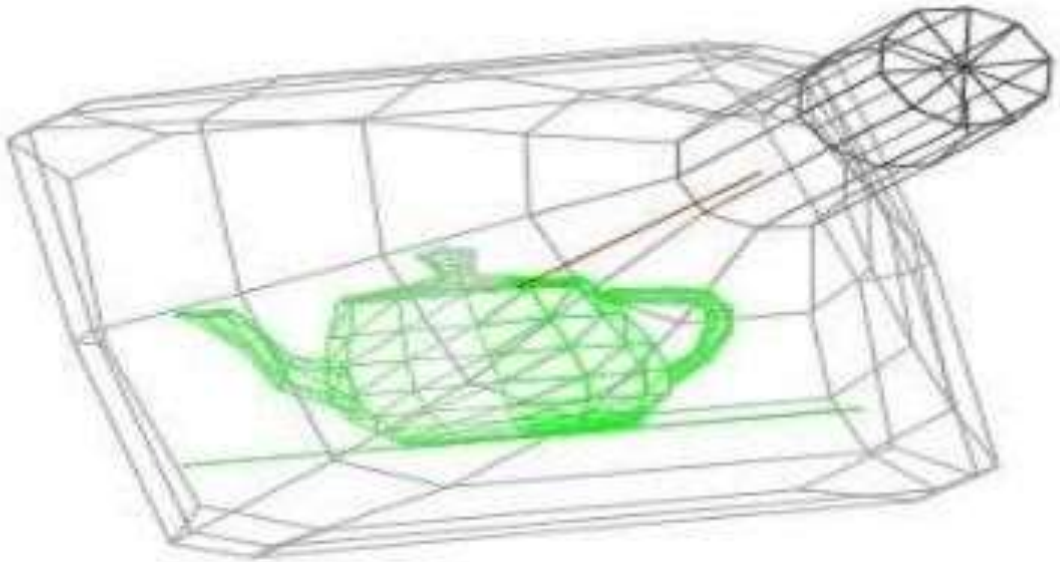
APPLICATIONS

- Suited for realistic display of screens
- Home television computer printers create their images basically by raster scanning. Laser printers use a spinning polygonal mirror (or an optical equivalent) to scan across the photosensitive drum, and paper movement provides the other scan axis

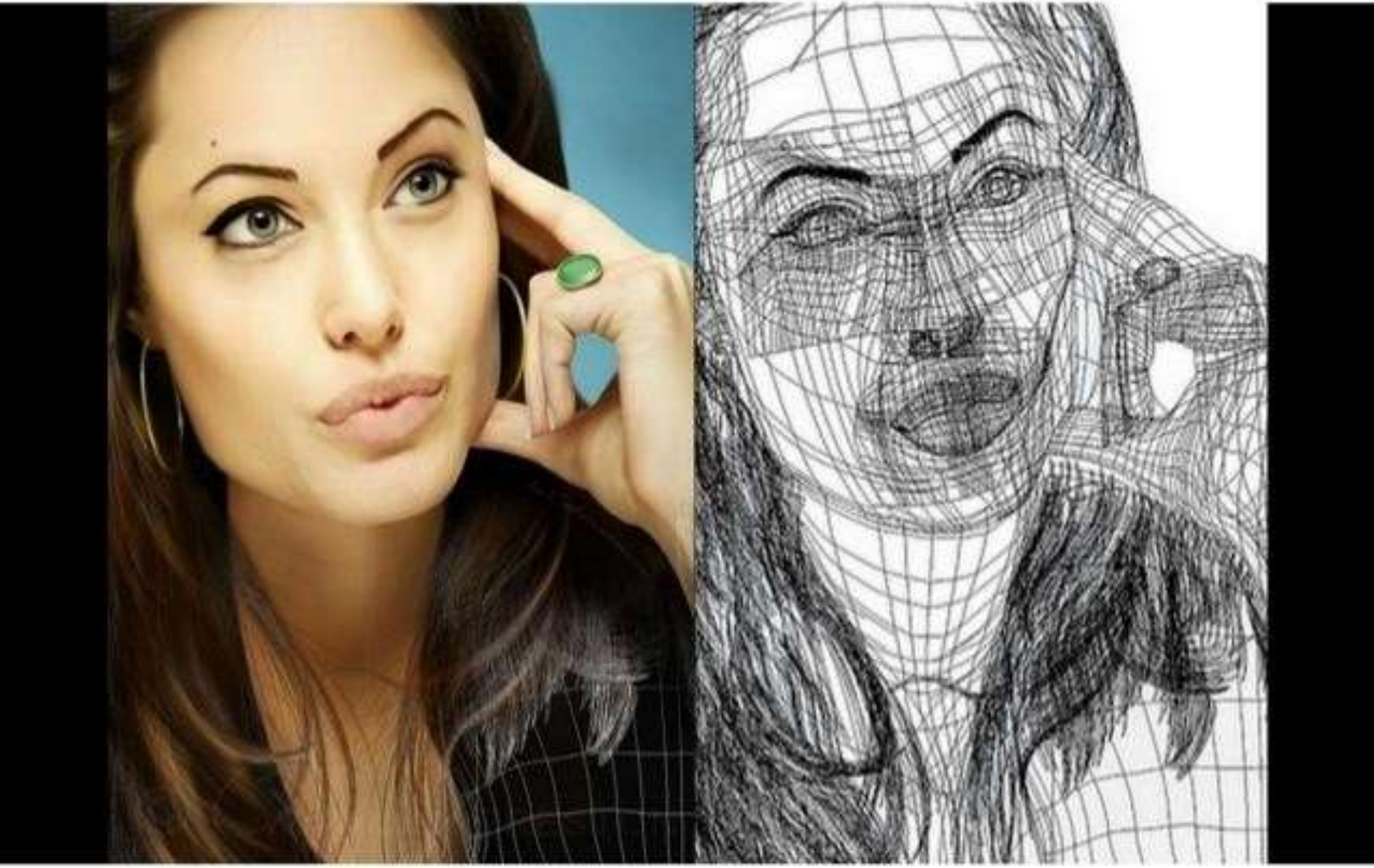
- Common raster image formats include BMP (Windows Bitmap), JPEG (Joint Photographics Expert Group), GIF (Graphics Interchange Format) , PNG (Portable Network Graphic), PSD (Adobe PhotoShop)

RANDOM SCAN DISPLAY

- Random scan display is the use of geometrical primitives such as points, lines, curves, and polygons, which are all based upon **mathematical**

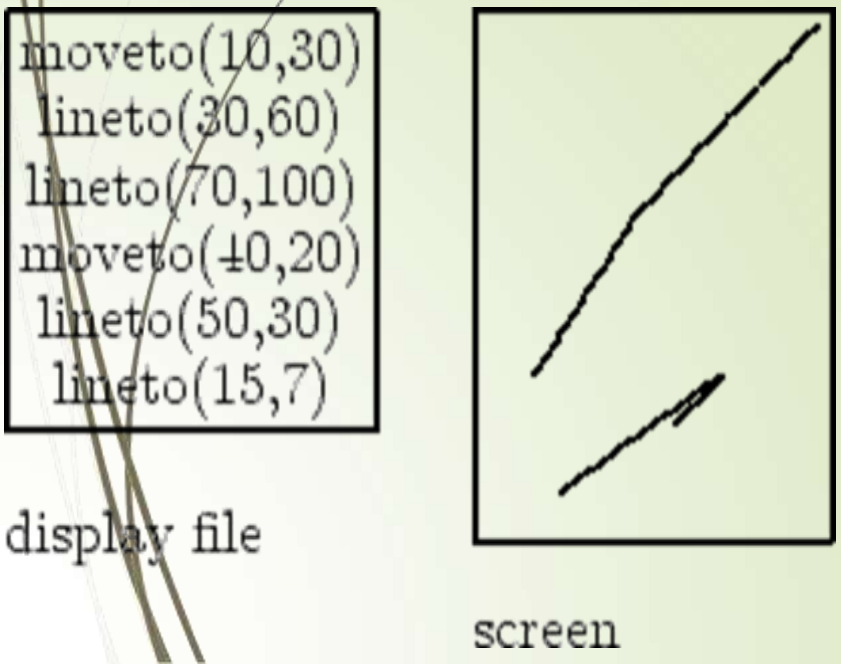


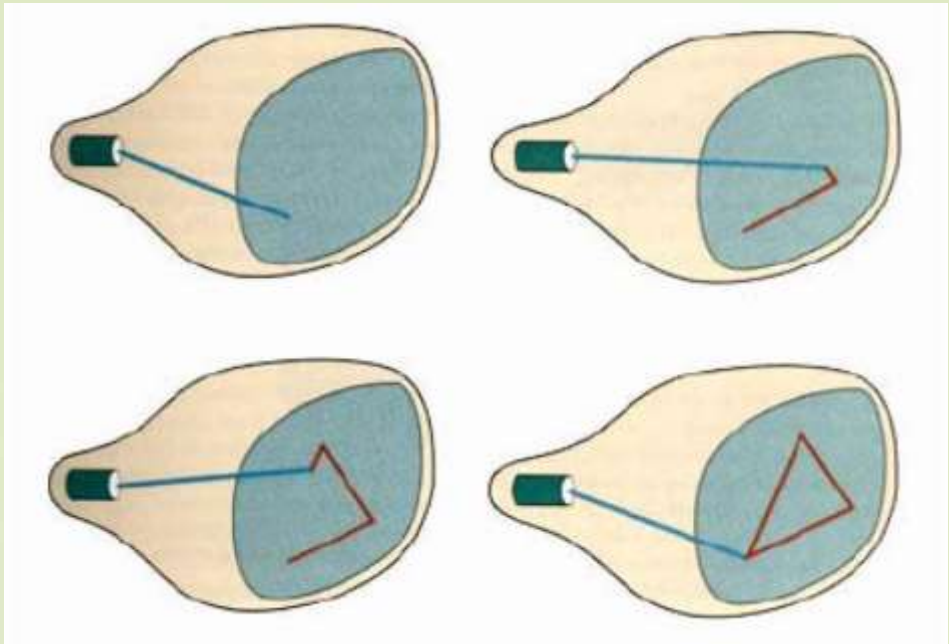
VECTOR IMAGE



Random Scan/ Calligraphic/ Vector CRT

- Beam can be moved from any position to any other position, by turning beam on and off.
- Very expensive but very fast because no scan conversion.





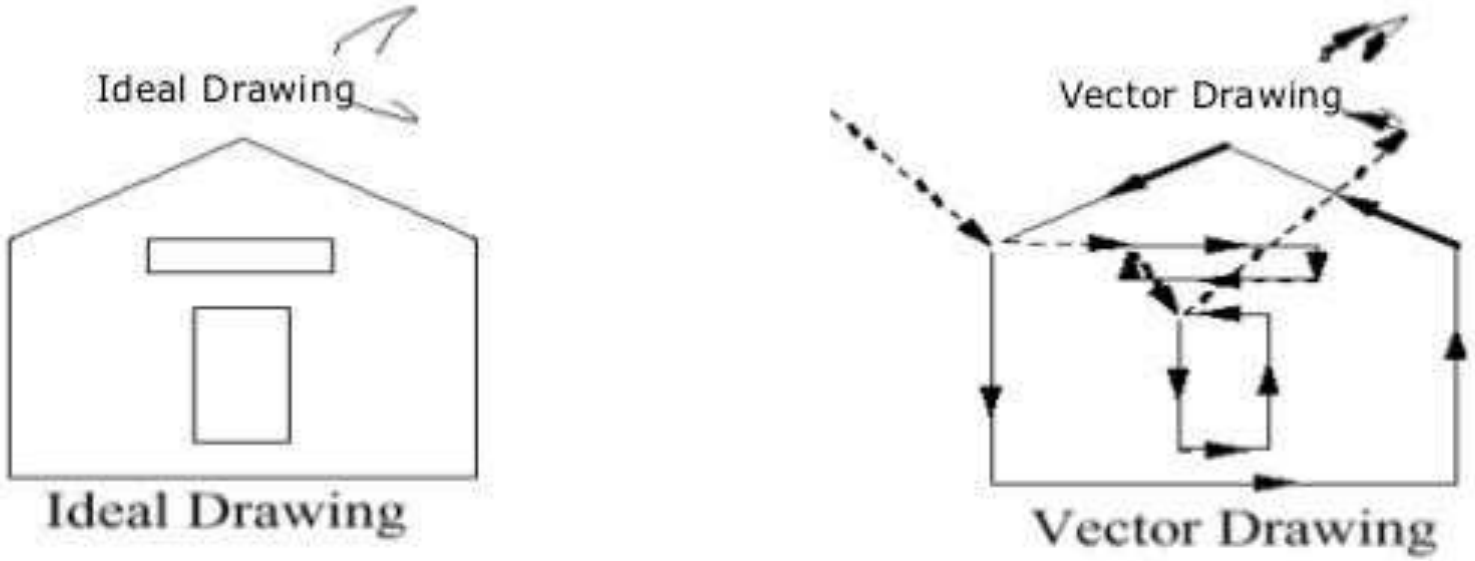
WORKING

- When operated as a random-scan display unit, a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn.
- Random-scan monitors draw a picture one line at a time and for this reason are also referred to as vector displays (or stroke-writing or calligraphic displays).

- **Refresh rate** depends on the number of lines to be displayed.
- Picture definition is now stored as a line-drawing commands an area of memory referred to as **refresh display file** (**display list**).
- To display a picture, the system cycle through the **set of commands** in the display file, drawing each component line in turn.
- Random scan displays are designed to draw all the component lines of a picture 30 to 60 times each second

➢A Raster system produces jagged lines that are plotted as discrete points sets.

➢Vector displays product smooth line drawing



- Random scan displays are designed for **line-drawing applications** and can not display realistic shaded scenes

Advantages

- Random scan displays have higher resolution than raster systems.
- Vector displays product smooth line drawing.
- This minimal amount of information translates to a much smaller file size. (file size compared to large raster images)
- On zooming in, and it remains smooth
- The parameters of obje.cts are stored and can be later modified.

Color CRT Monitors

- Using a combination of phosphors that emit different-colored light
- Beam-penetration
  - Used in random-scan monitors
  - Use red and green phosphors layers
  - Color depends on the penetrated length of electrons
- Shadow mask
  - Used in raster-scan systems
  - Produce wide range of color with RGB color model

. Beam Penetration Method:

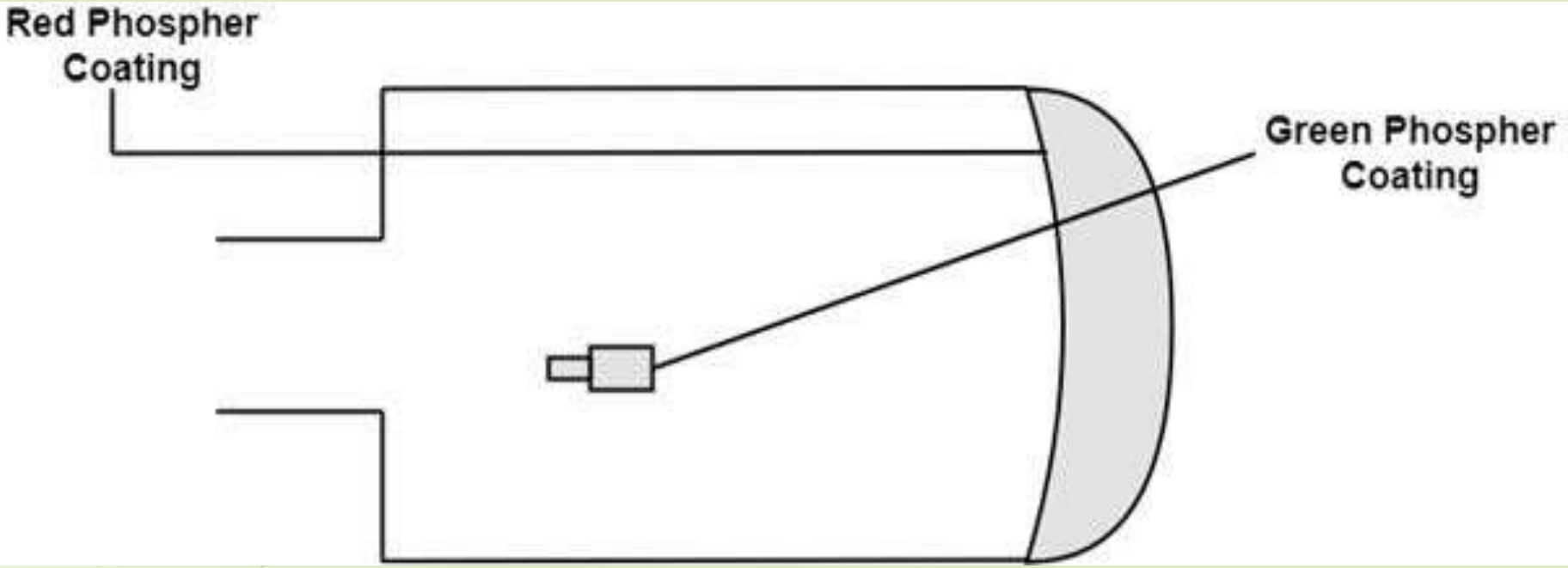
The Beam-Penetration method has been used with random-scan monitors.

In this method, the CRT screen is coated with two layers of phosphor, red and green and the displayed color depends on how far the electron beam penetrates the phosphor layers.

This method produces four colors only, red, green, orange and yellow.

A beam of slow electrons excites the outer red layer only; hence screen shows red color only.

A beam of high-speed electrons excites the inner green layer. Thus screen shows a green color.



Advantages:

1. Inexpensive

Disadvantages:

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

Shadow-Mask Method:

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

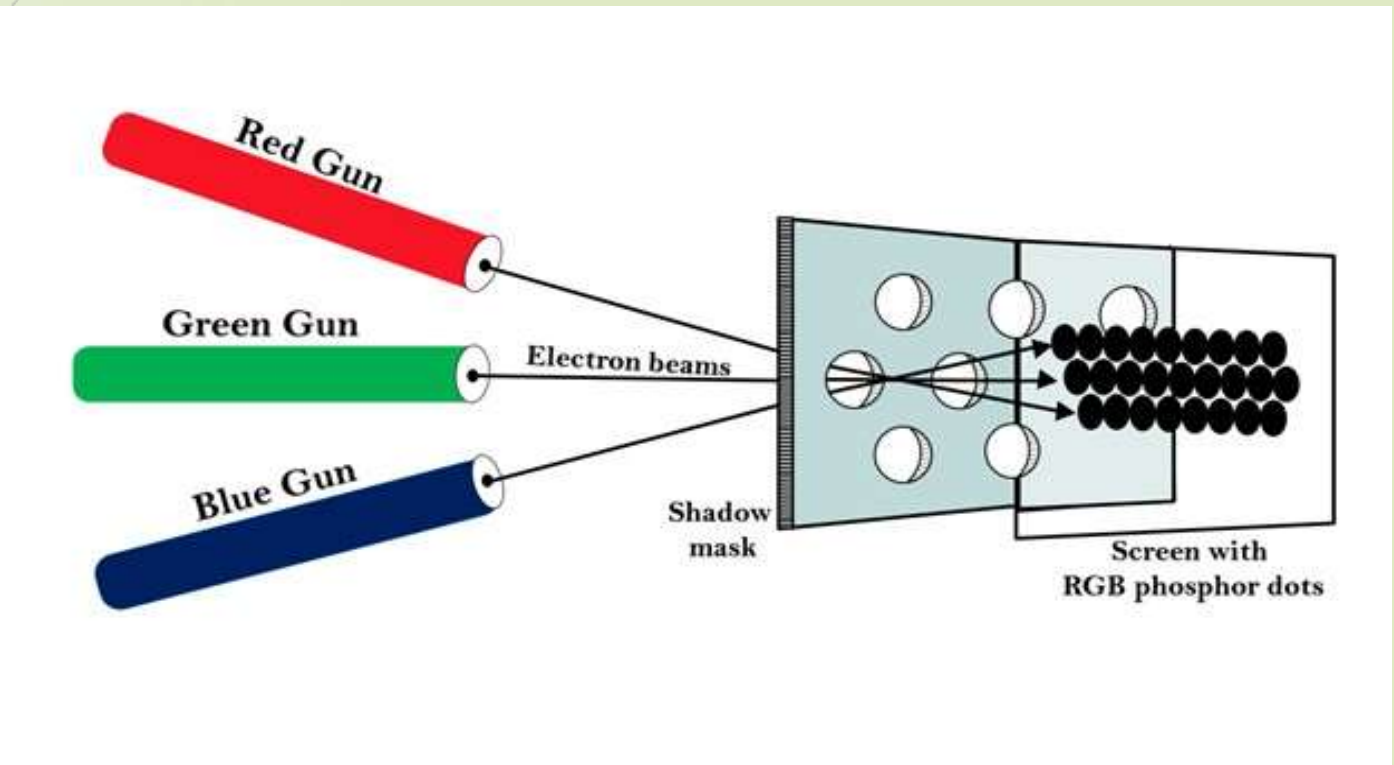
Construction: A shadow mask CRT has 3 phosphor color dots at each pixel position.

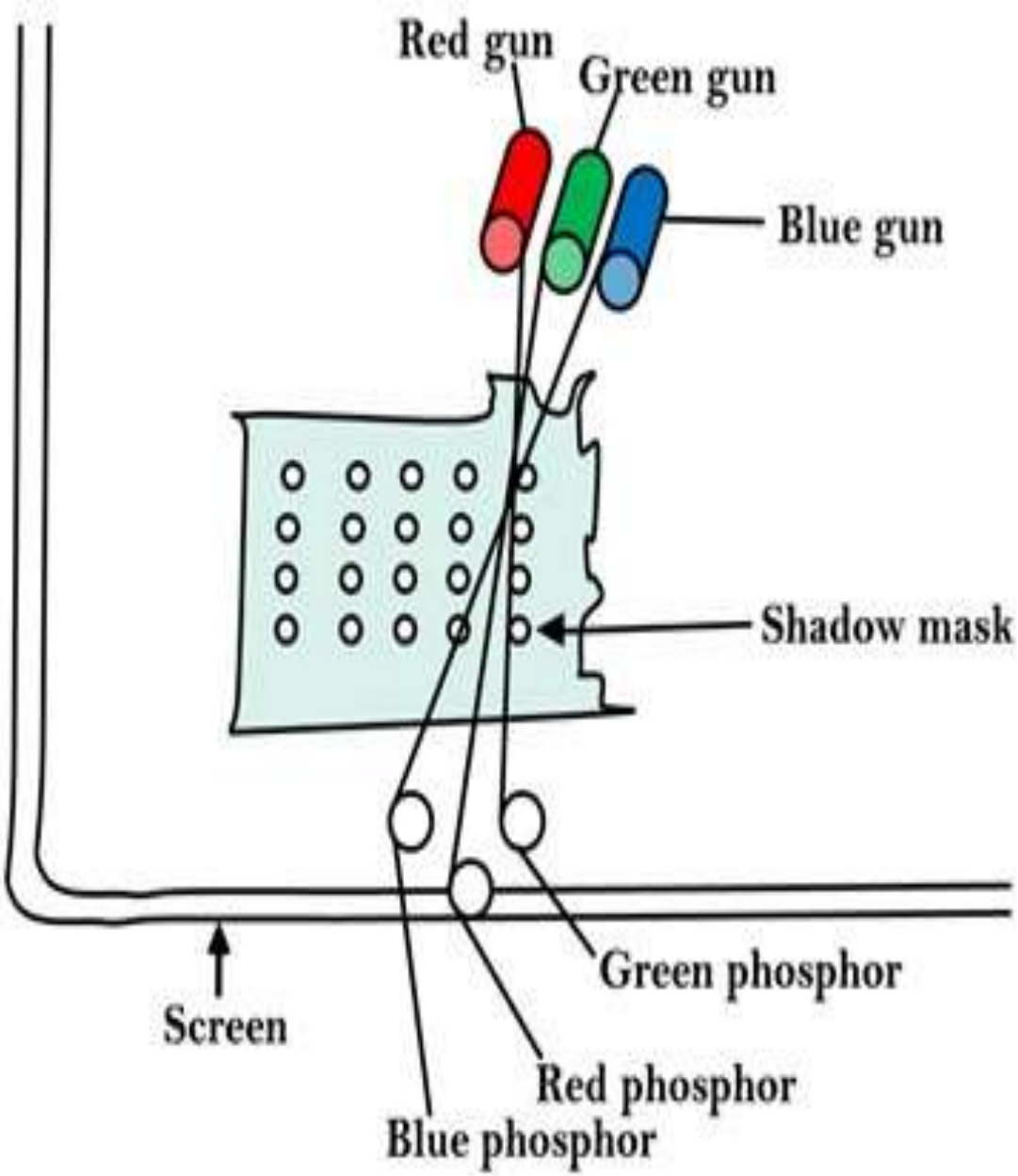
- One phosphor dot emits: red light
- Another emits: green light
- Third emits: blue light

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

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

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





The Shadow mask CRT

Working:

Triad arrangement of red, green, and blue guns.

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

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

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

Inline arrangement:

Another configuration for the 3 electron guns is an Inline arrangement in which the 3 electron guns and the corresponding red-green-blue color dots on the screen, are aligned along one scan line rather of in a triangular pattern.

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

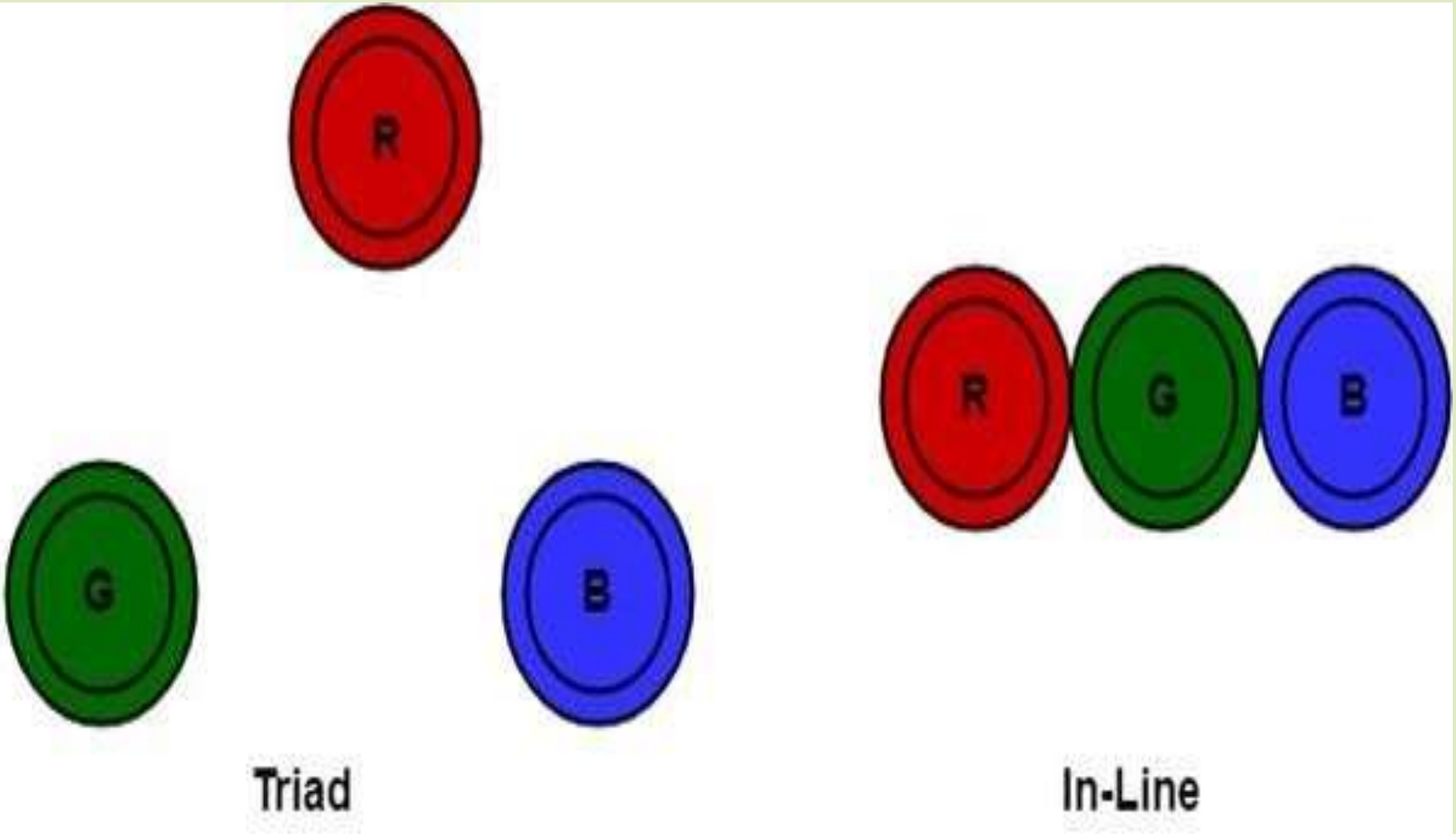


Fig: Triad-and -in-line arrangements of red, green and blue electron guns of CRT for color monitors.

Advantage:

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

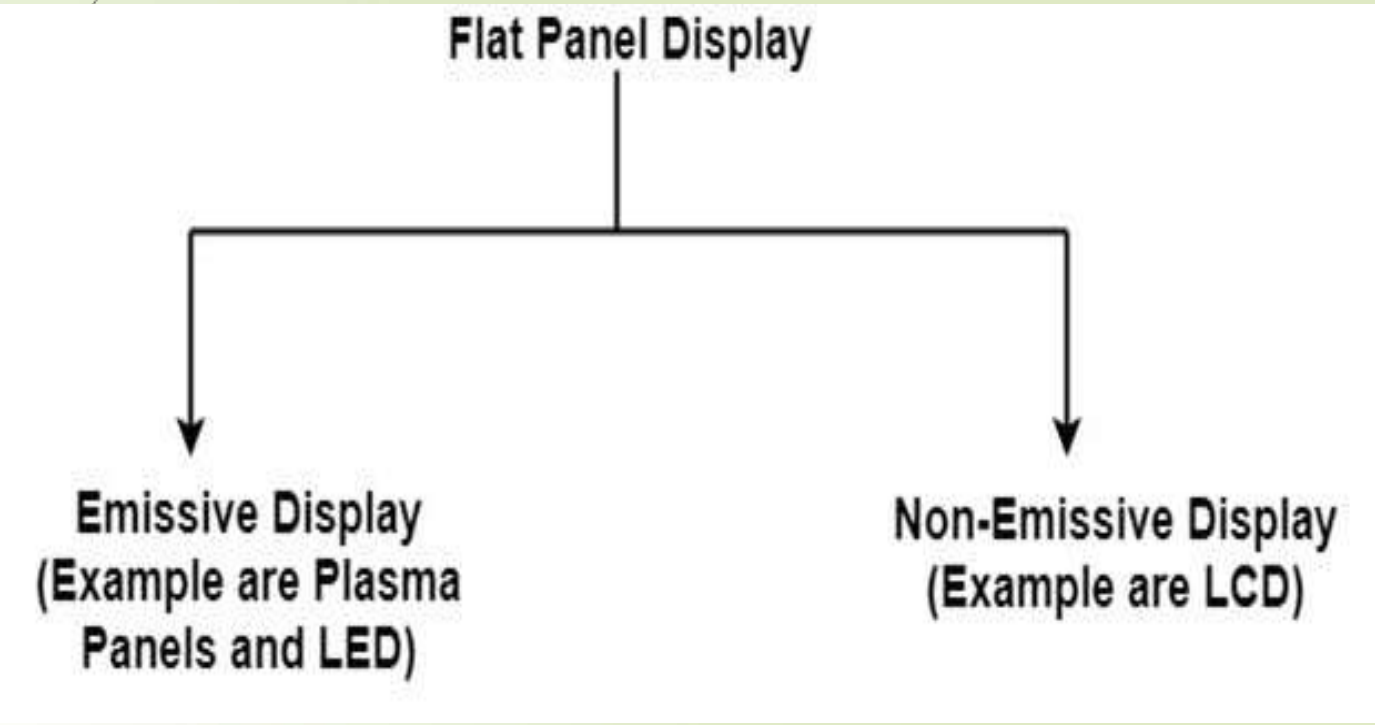
Disadvantage:

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

Flat Panel Display:

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

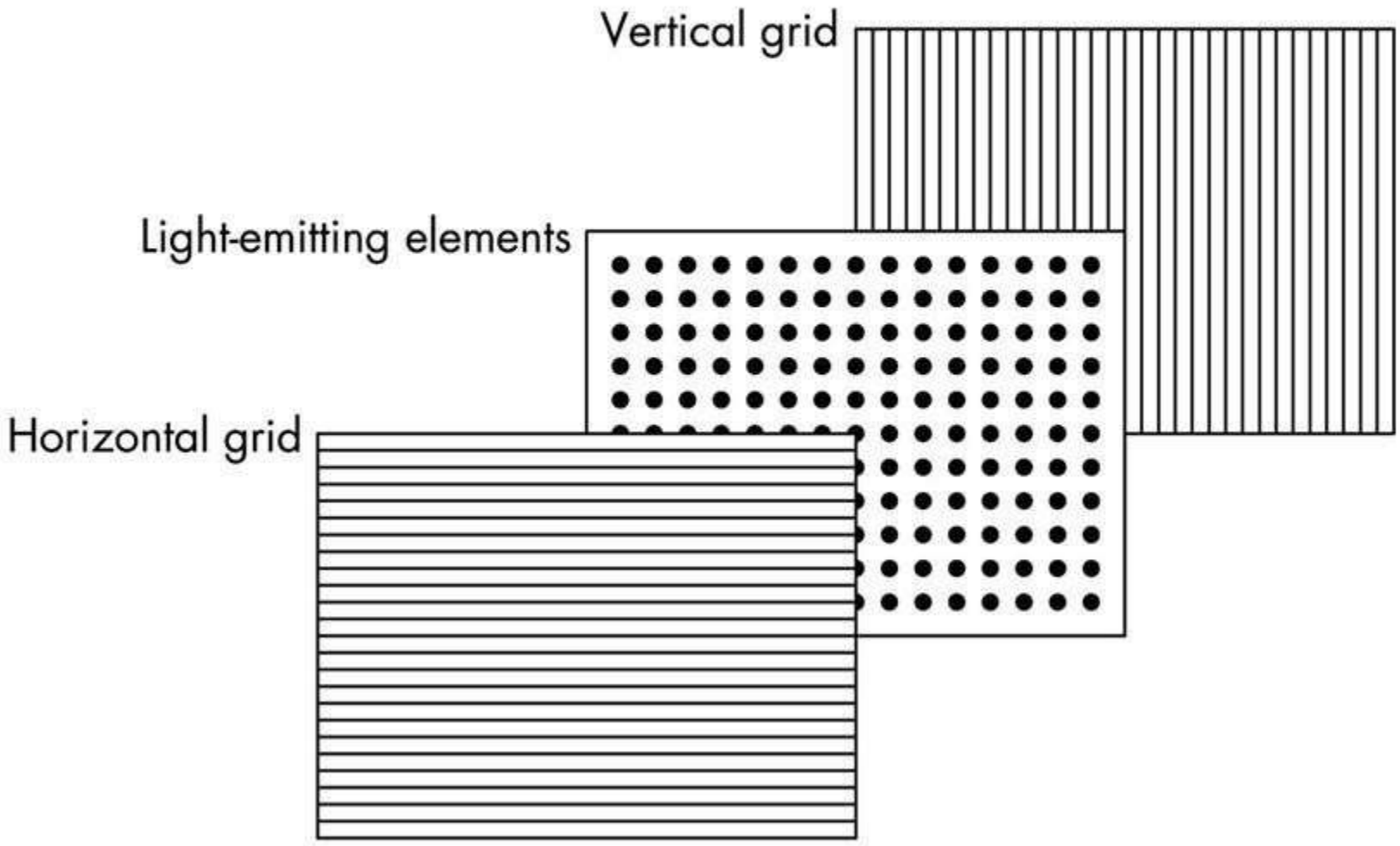
Example: Small T.V. monitor, calculator, pocket video games, laptop computers, an advertisement board in elevator.



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

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

FIGURE 1.5 Generic flat-panel 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 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.

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

LCD uses the liquid-crystal material between two glass plates; each plate is the right angle to each other between plates liquid is filled.

One glass plate consists of rows of conductors arranged in vertical direction.

Another glass plate is consisting of a row of conductors arranged in horizontal direction.

The pixel position is determined by the intersection of the vertical & horizontal conductor. This position is an active part of the screen.

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

Advantage:

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

Disadvantage:

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

Types of Frame buffer

Color buffer

They contain either color-index or RGB color data and may also contain alpha values.

Depth buffer

Stores a depth value for each pixel, mainly used for hidden surface removal, depth is usually measured in terms of distance to the eye, so pixels with larger depth-buffer values are overwritten by pixels with smaller values.

Stencil Buffer

One use for the stencil buffer is to restrict drawing to certain portions of the screen.

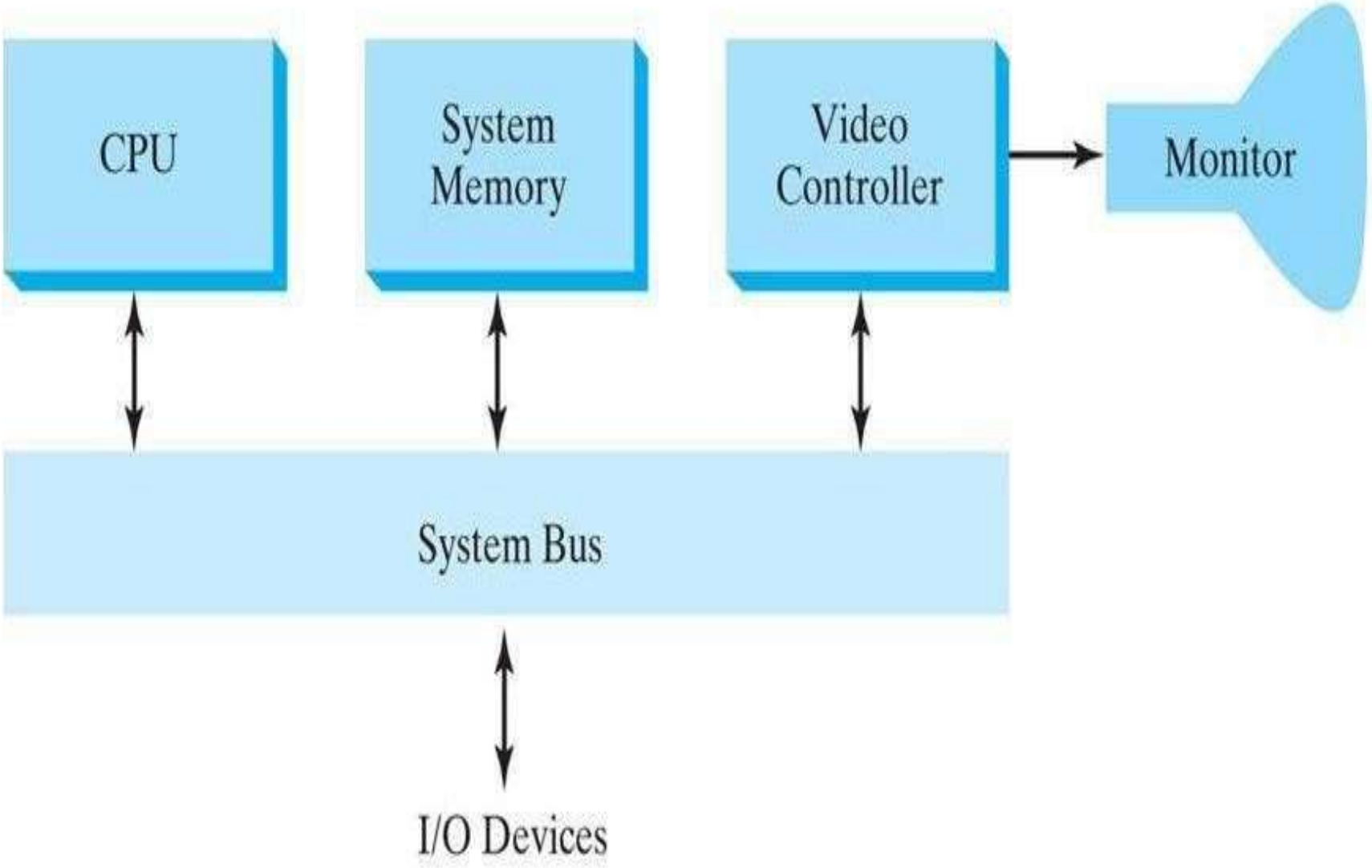
Accumulation Buffer

The accumulation buffer holds RGBA color data just like the color buffers do in RGBA mode It's typically used for accumulating a series of images into a final, composite image.

Raster Scan System

Organization of a simple raster system is shown in Figure.

→



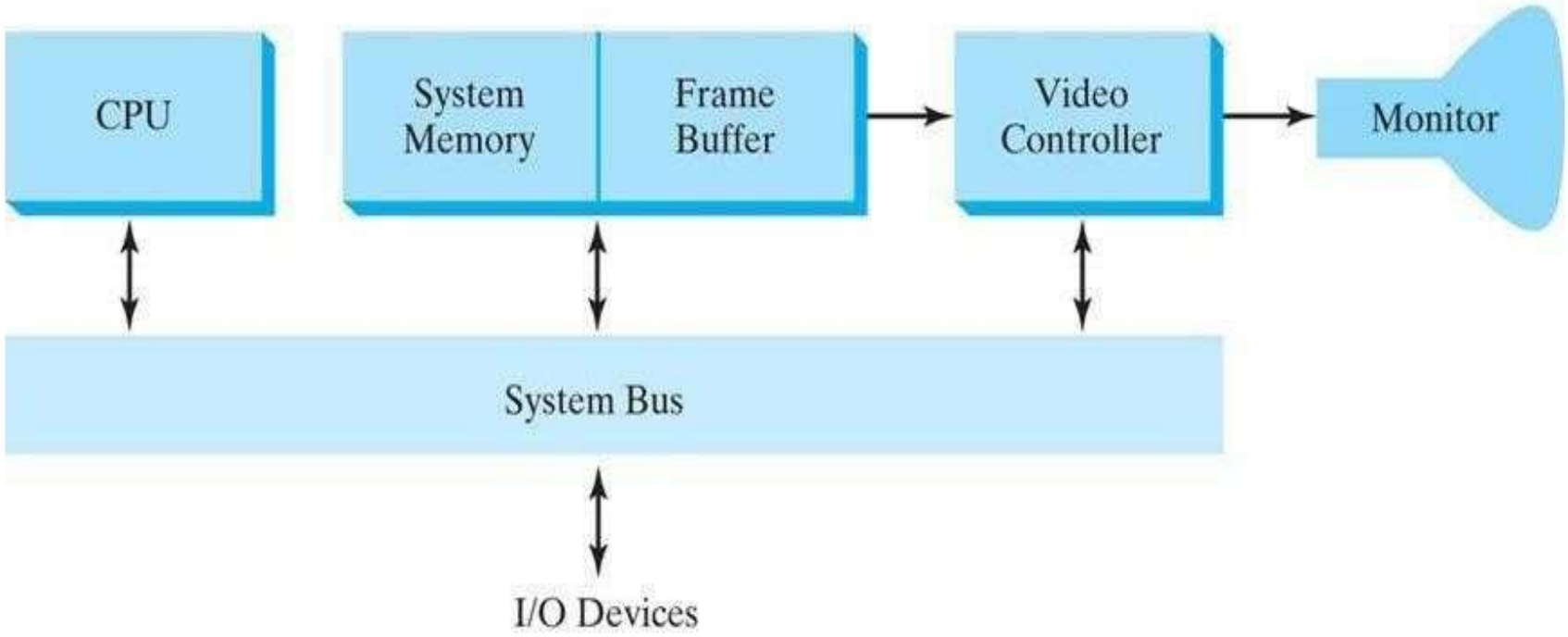
Here, the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen..

In addition to the video controller, raster systems employ other processors as coprocessors and accelerators to implement various graphics operations.

The figure below shows a commonly used organization for raster systems.

✓A fixed area of the system memory is reserved for the frame buffer, and the video controller is given direct access to the frame-buffer memory.

✓Frame-buffer locations, and the corresponding screen positions, are referenced in the Cartesian coordinates.



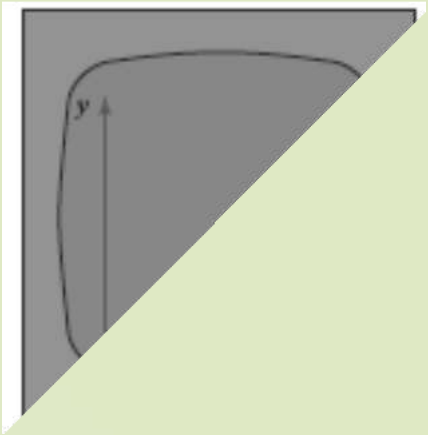
Cartesian reference frame:

✓Frame-buffer locations and the corresponding screen positions, are referenced in Cartesian coordinates.

✓In an application (user) program, we use the commands within a graphics software package to set coordinate positions for displayed objects relative to the origin.

✓The coordinate origin is referenced at the lower-left corner of a screen display area by the software commands, although we can typically set the origin at any convenient location for a particular application.

Working: Figure shows a two-dimensional Cartesian reference frame with the origin at the lowerleft screen corner.

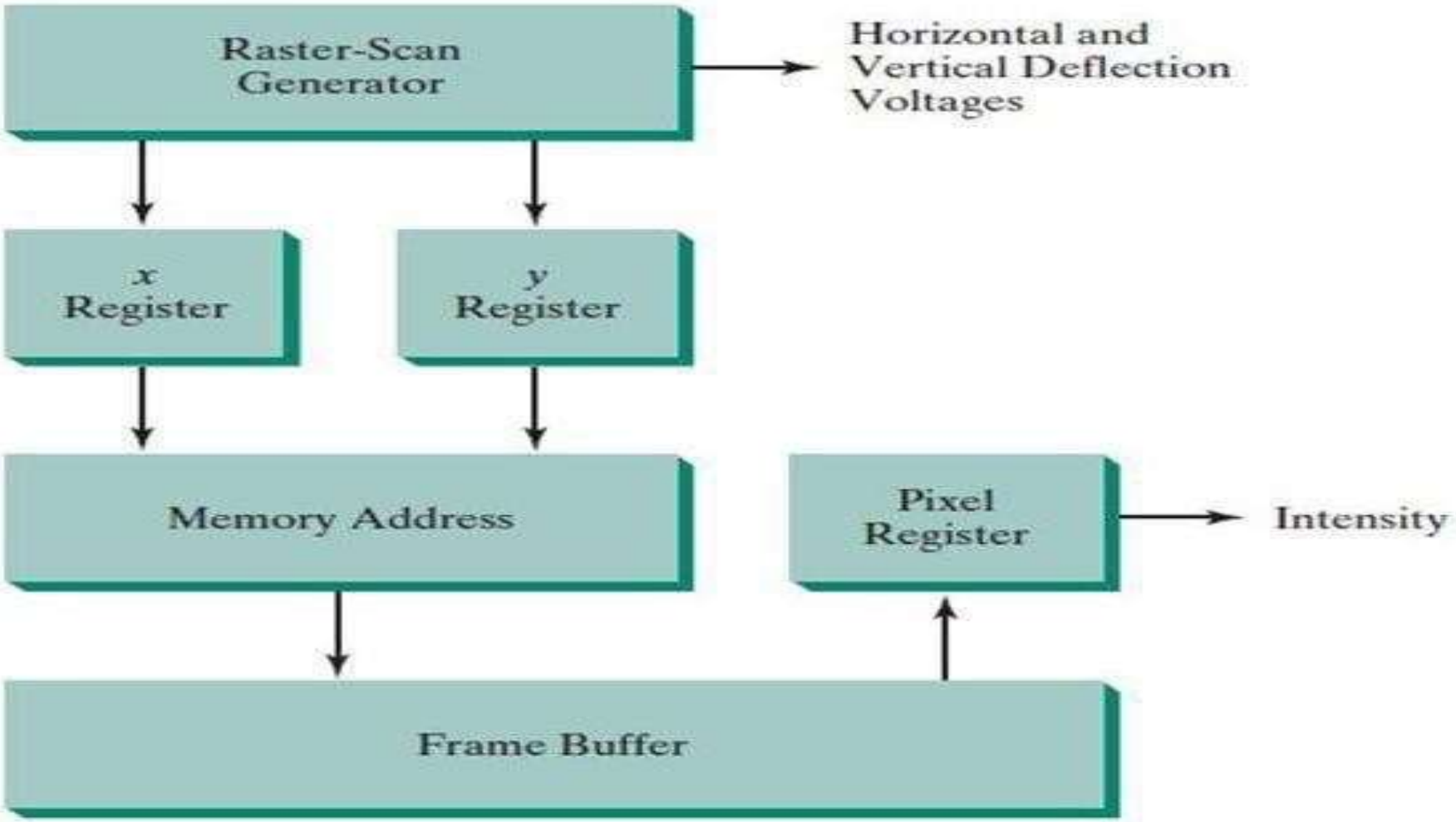


The screen surface is then represented as the first quadrant of a two-dimensional system with positive x and y values increasing from left to right and bottom of the screen to the top respectively.

Pixel positions are then assigned integer x values that range from 0 to xmax across the screen, left to right, and integer y values that vary from 0 to ymax, bottom to top.

Basic Video Controller Refresh Operations

✓ The basic refresh operations of the video controller are diagrammed



Two registers are used to store the coordinate values for the screen pixels.

Initially, the x register is set to 0 and the y register is set to the value for the top scan line.

✓The contents of the frame buffer at this pixel position are then retrieved and used to set the intensity of the CRT beam.

✓Then the x register is incremented by 1, and the process is repeated for the next pixel on the top scan line.

✓This procedure continues for each pixel along the top scan line.

✓After the last pixel on the top scan line has been processed, the x register is reset to 0 and the y register is set to the value for the next scan line down from the top of the screen.

✓ The procedure is repeated for each successive scan line.

✓After cycling through all pixels along the bottom scan line, the video controller resets the registers to the first pixel position on the top scan line and the refresh process starts over .

a. Speed up pixel position processing of video controller:

✓Since the screen must be refreshed at a rate of at least 60 frames per second, the simple procedure illustrated in above figure may not be accommodated by RAM chips if the cycle time is too slow.

✓To speed up pixel processing, video controllers can retrieve multiple pixel values from the refresh buffer on each pass.

✓When group of pixels has been processed, the next block of pixel values is retrieved from the frame buffer.

Advantages of video controller:

✓A video controller can be designed to perform a number of other operations.

✓For various applications, the video controller can retrieve pixel values from different memory areas on different refresh cycles.

✓This provides a fast mechanism for generating real-time animations.

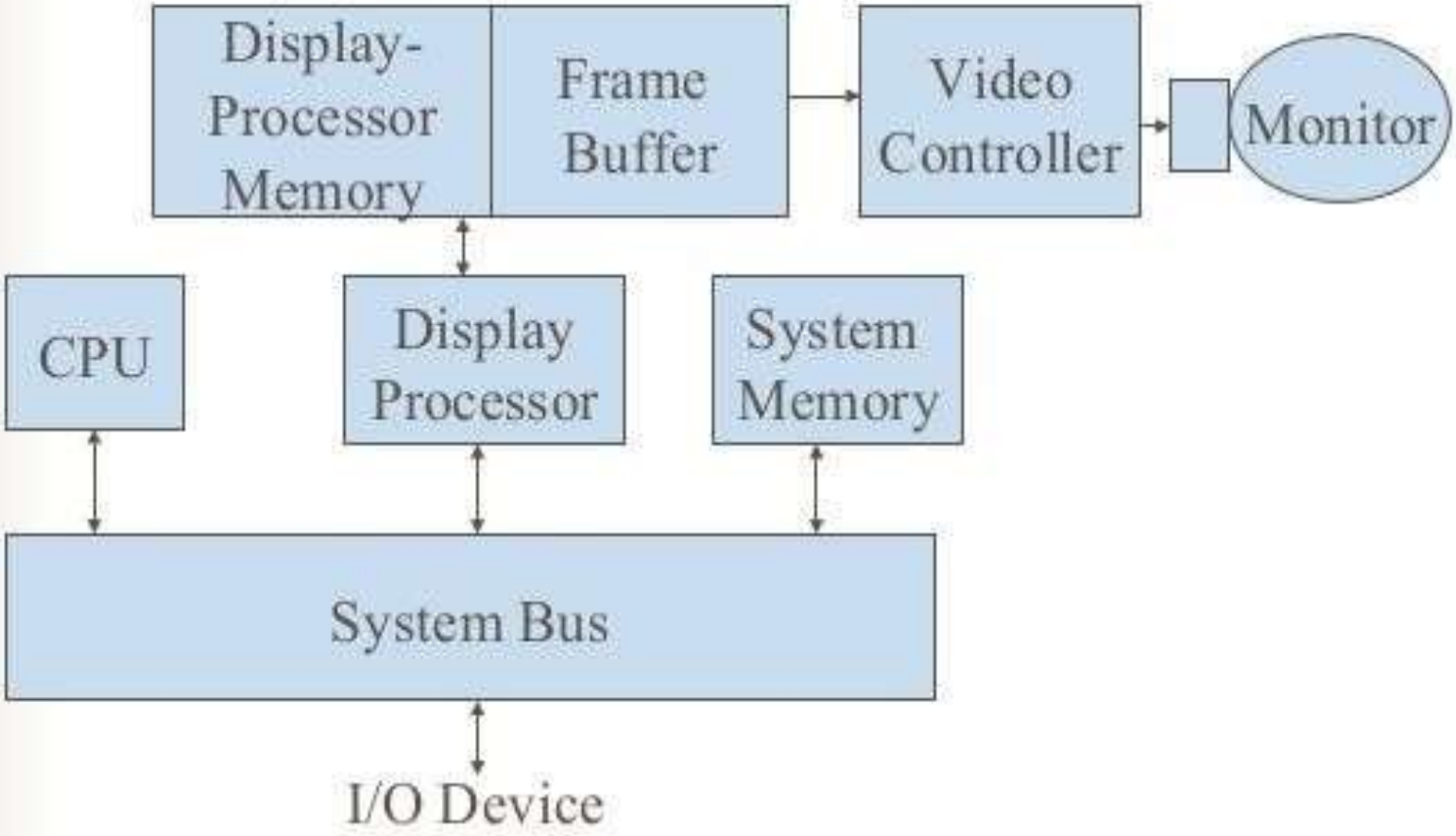
✓Another video-controller task is the transformation of blocks of pixels, so that screen areas can be enlarged, reduced, or moved from one location to another during the refresh cycles.

\

In addition, the video controller often contains a lookup table, so that pixel values in the frame buffer are used to access the lookup table. This provides a fast method for changing screen intensity values.

Finally, some systems are designed to allow the video controller to mix the framebuffer image with an input image from a television camera or other input device .

Raster Graphic system with Display processor



Raster-Scan Display Processor

✓Figure shows one way to organize the components of a raster system that contains a separate display processor, sometimes referred to as a graphics controller or a display coprocessor.

The purpose of the display processor is to free the CPU from the graphics chores.

✓In addition to the system memory, a separate display-processor memory area can be provided.

Scan conversion:

✓A major task of the display processor is digitizing a picture definition given in an application program into a set of pixel values for storage in the frame buffer.

✓ This digitization process is called scan conversion.

Example 1: displaying a line

➔Graphics commands specifying straight lines and other geometric objects are scan converted into a set of discrete points, corresponding to screen pixel positions.

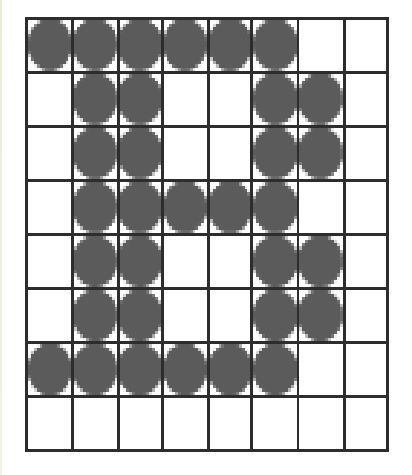
➔ Scan converting a straight-line segment.

Example 2: displaying a character

➔ Characters can be defined with rectangular pixel grids .

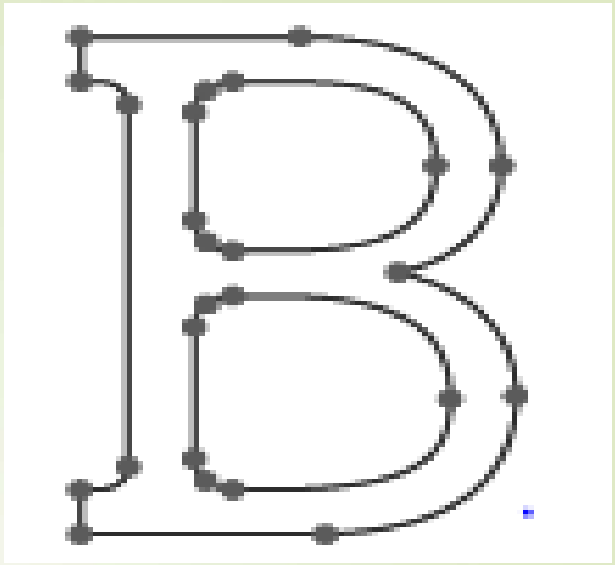
➔The array size for character grids can vary from about 5 by 7 to 9 by 12 or more for higher-quality displays.

➔A character grid is displayed by superimposing the rectangular grid pattern into the frame buffer at a specified coordinate position.



Using outline:

➔For characters that are defined as outlines, the shapes are scan-converted into the frame buffer by locating the pixel positions closest to the outline.



Additional operations of Display processors:

➔Display processors are also designed to perform a number of additional operations.

➔These functions include generating various line styles (dashed, dotted, or solid), displaying color areas, and applying transformations to the objects in a scene.

➔Display processors are typically designed to interface with interactive input devices, such as a mouse.

Graphics workstations and viewing systems

✓Most graphics monitors today operate as raster-scan displays, and both CRT and flat panel systems are in common use.

✓Graphics workstation range from small general-purpose computer systems to multi monitor facilities, often with ultra –large viewing screens.

High-definition graphics systems, with resolutions up to 2560 by 2048, are commonly used in medical imaging, air-traffic control, simulation, and CAD.

Graphics workstations and viewing systems

✓Many high-end graphics workstations also include large viewing screens, often with specialized features.

Multi-panel display screens are used in a variety of applications that require “wall-sized” viewing areas. These systems are designed for presenting graphics displays at meetings, conferences, conventions, trade shows, retail stores etc.

✓A multi-panel display can be used to show a large view of a single scene or several individual images. Each panel in the system displays one section of the overall picture .

Graphics workstations and viewing systems

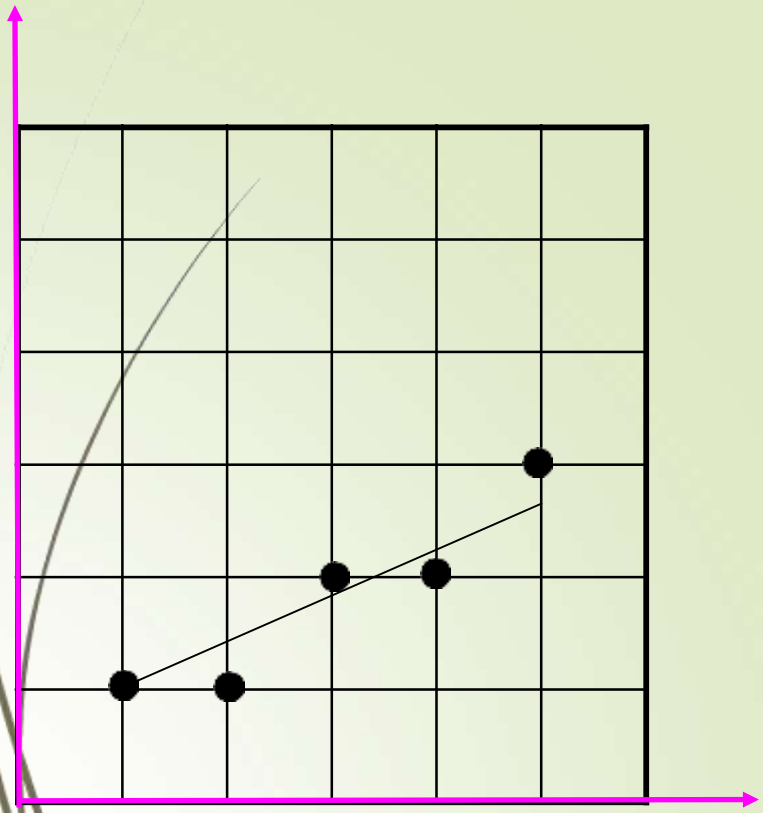
✓A large, curved-screen system can be useful for viewing by a group of people studying a particular graphics application.

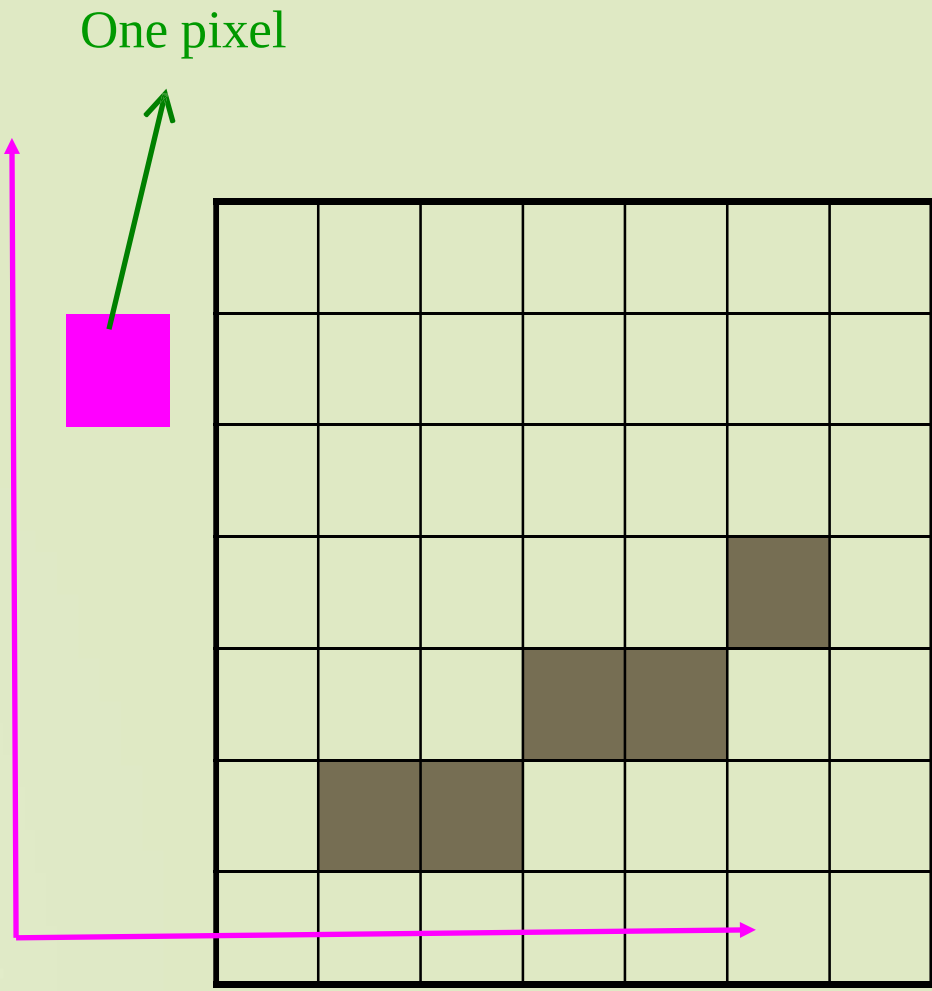
✓A 360 degree paneled viewing system in the NASA control-tower simulator, which is used for training and for testing ways to solve air-traffic and runway problems at airports.

Rasterization

- Rasterization (scan conversion)
  - Scan conversion is combination of rasterization and generating the picture in the scanline order.
  - Rasterization is a process of determining which pixels provide the best approximation to a desired line on the screen. The general requirements of lines are
    - Start and End should be accurate .
    - All pixels should have constant brightness along their length.

Scan Conversion of Line Segments





LINE EQUATION

If the end points of line segments are (x1,y1) and (x2,y2), the line can be represented as

$Y=mX+c$

Where m slope is

$m= \frac{(y2-y1)}{(x2-x1)} = \frac{\Delta y}{\Delta x}$, assume $0 \leq m \leq 1$

For a given x interval Δx along a line, calculate the corresponding Δy

$\Delta y = m * \Delta x$, for slope $|m| < 1$

Similarly for a given y interval Δy along a line , corresponding Δx

$\Delta x = \Delta y / m$ for slope $|m| > 1$

Scan Conversion algorithm for Line segment

- Algorithms are
- 1. DDA (Digital Differential Analyzer)
- 2. Bresenham's Algorithm

- DDA was a electro mechanical device for digital simulation of differential equations.

$$\frac{dy}{dx} = m,$$

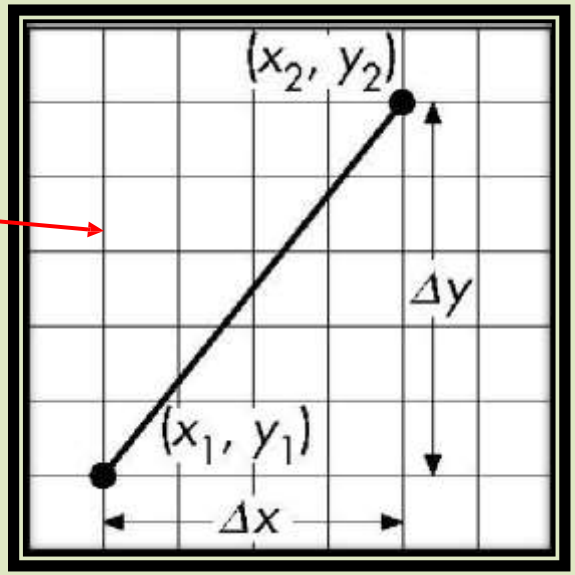
Where ' m ' is the slope.

Scan Conversion of Line Segments

- **Start with line segment in window coordinates with integer values for endpoints**
- **Assume implementation has a `write_pixel` function**

y = mx + c

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



DDA Algorithm

- Slope $|m| < 1$ or $|m| = 0$(horizontal line),

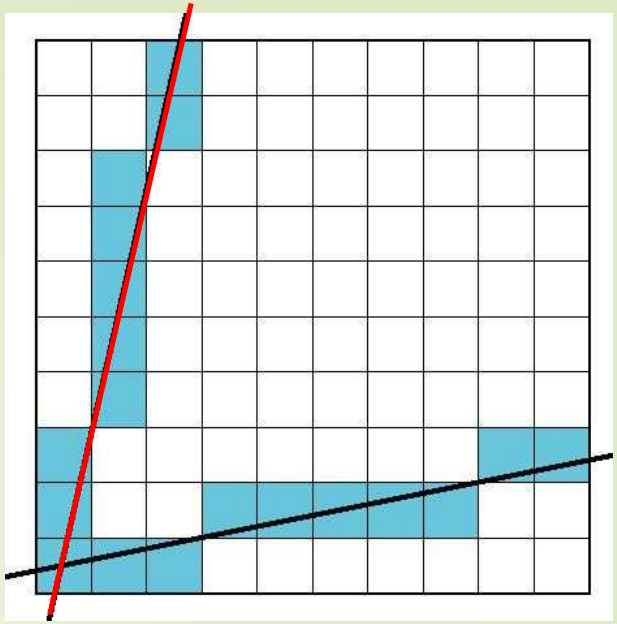
  $\Delta y = m \Delta x$,

  moving from x1 to x2 and x is increase by 1 for each iteration. And y is increase by $\Delta y = m$.

```
For(x=x1; x<=x2, x++)
{
   y += m;
//note:m is float number
 write_pixel(x,round(y),line_color);
}
```

Using Symmetry

- Use for $1 \geq m \geq 0$
- For $m > 1$, swap role of x and y
  - For each y, plot closest x



DDA Algorithm

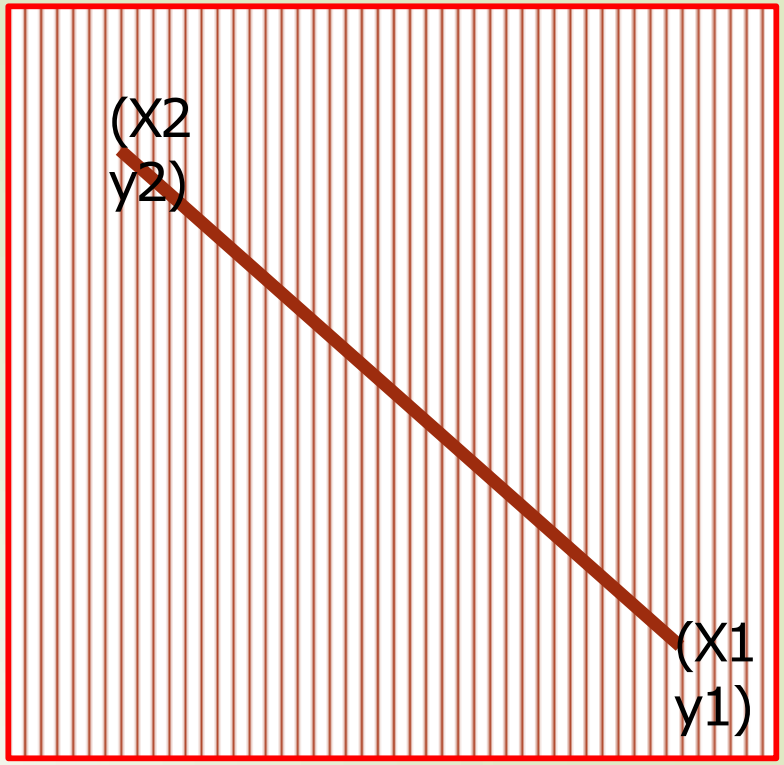
- Slope |m| >1

  Δx= Δy/m ,

moving from y1 to y2 and y is increase by 1 for each iteration. And x is increase by Δx =1/m.

```
For(y=y1; y<=y2, y++)
{
   x += 1/m;
//note:m is float number
 write_pixel(round(x),y,line_color);
}
```

When X1 >X2 or Y1 >Y2



DDA Algorithm

- Slope less than or equal to -1 (negative)

  $\Delta y = m\,\Delta x$ moving from x2 to x1 and x is decrease by 1 for each iteration. And y is decrease by $\Delta y = m$.

```
For(x=x2; x<=x1, x--)
{
   y += m;
//note:m is float number
 write_pixel(x,round(y),line_color);
}
```

DDA Algorithm

- Slope m > -1

  Δx= Δy/m , where y1>y2

moving from y2 to y1 and y is decrease by 1 for each iteration. And x is decrease by Δx =1/m.

```
For(y=y2; y<=y1, y--)
{
   x += 1/m;
//note:m is float number
 write_pixel(round(x),y,line_color);
}
```

Advantages

- **Easy to implement, simple & faster method compared to direct use of Equation.**
- **Does not involve any floating point multiplication and no direct use of intercept.**

Disadvantages

Involves floating point addition.(round off error)

Time consuming technique.

Rasterization

Rasterization

- The raster display is a matrix of picture elements also called ***pixels.*** Each pixel has a color value assigned.

- A frame buffer stores the values for each pixel.

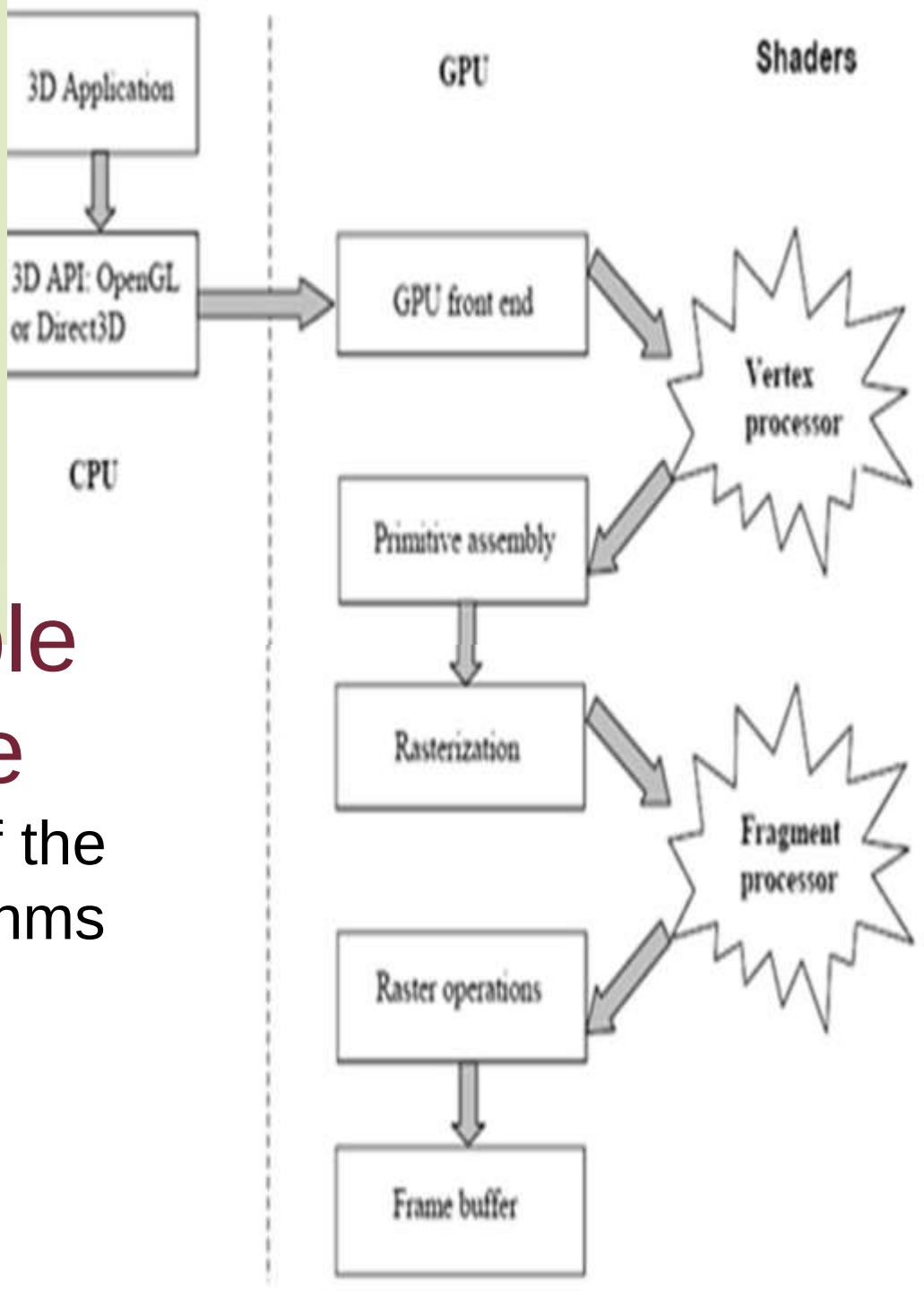
- The task of displaying a world modelled using primitives like lines, polygons, filled/patterned areas, etc. can be carried out in two steps

- ***Rasterization is a process of determining which pixels provide the best approximation to a desired line on the screen.***

- Determine the color value to be assigned to each such pixel.

Programmable GPU Pipeline

GPUs have most of the rasterization algorithms implemented in hardware!

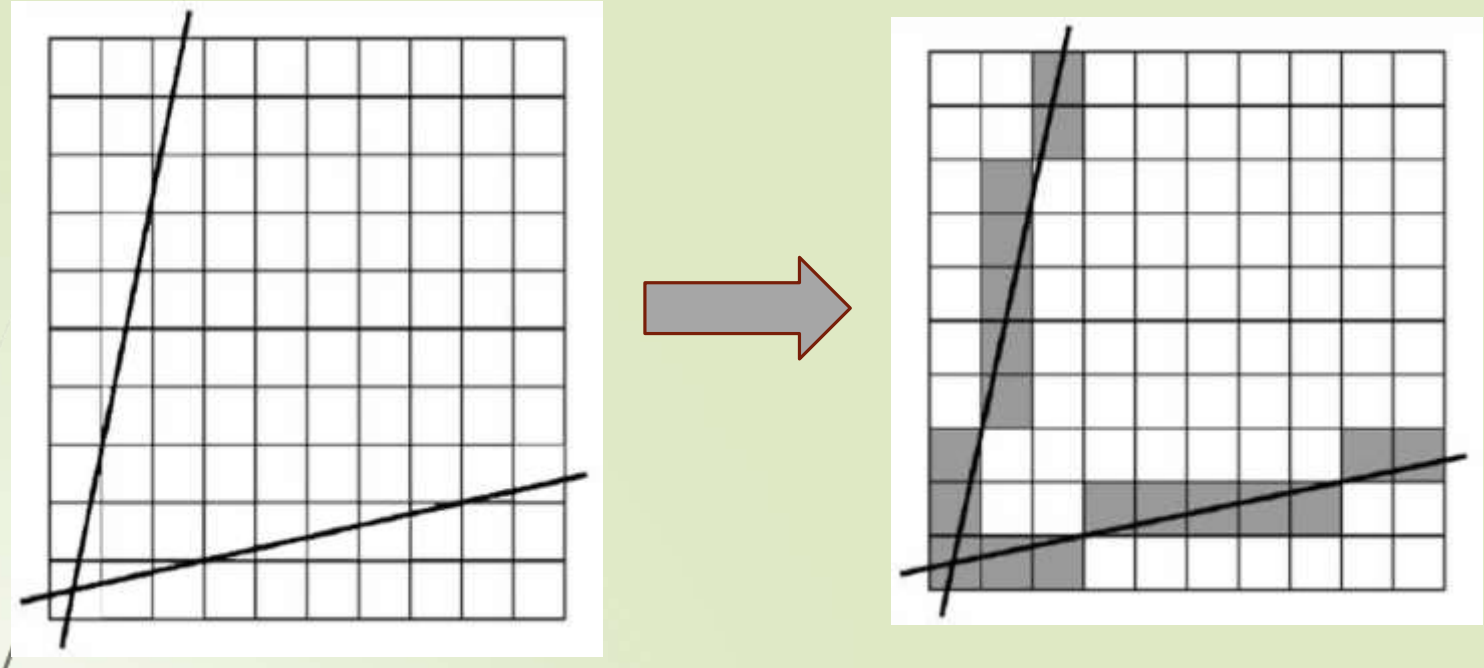


Scan converting lines

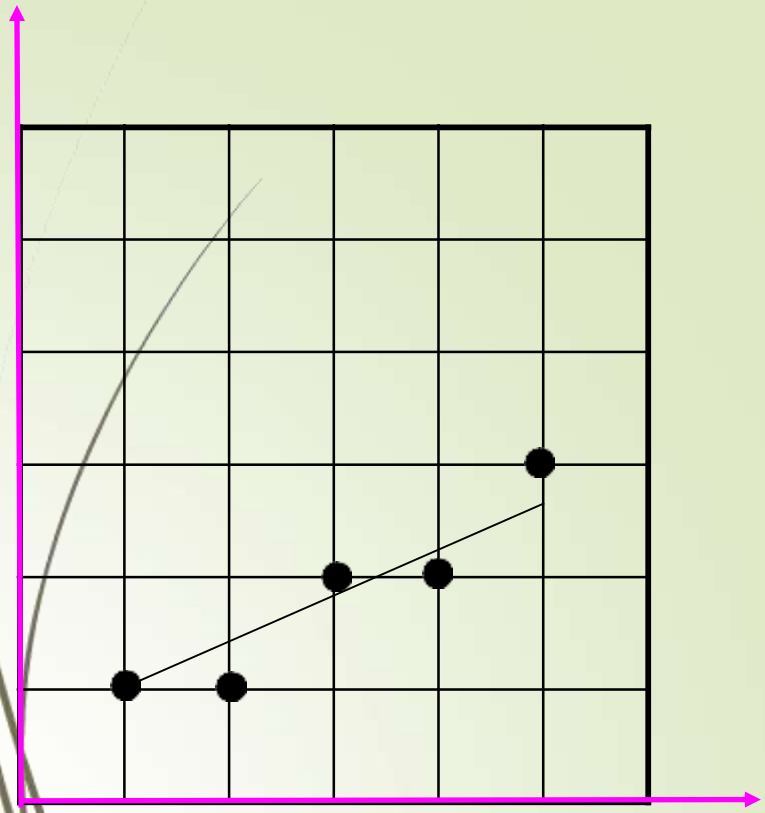
Requirements

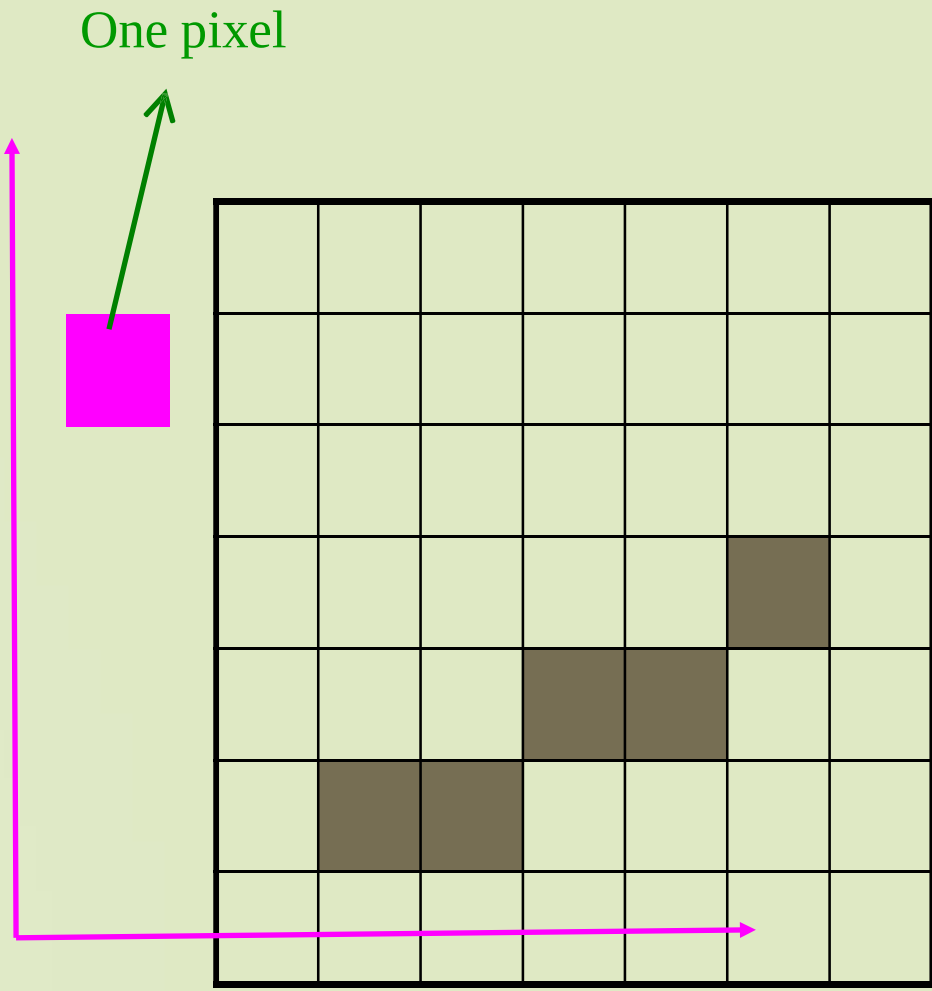
- ▪The chosen pixels should lie as close to the ideal line as possible.
- ▪The sequence of pixels should be as straight as possible.
- All lines should appear to be of constant brightness. Independent of their length and orientation.
- ▪should start and end accurately.
- ▪should be drawn as rapidly as possible.
- ▪should be possible to draw lines with different width and line styles.

Rasterizing a line



Scan Conversion of Line Segments





What is Line?

1. A line in Computer graphics is a portion of straight line that extends indefinitely in opposite direction.

2. It is defined by its two end points.

3. Its density should be independent of line length.

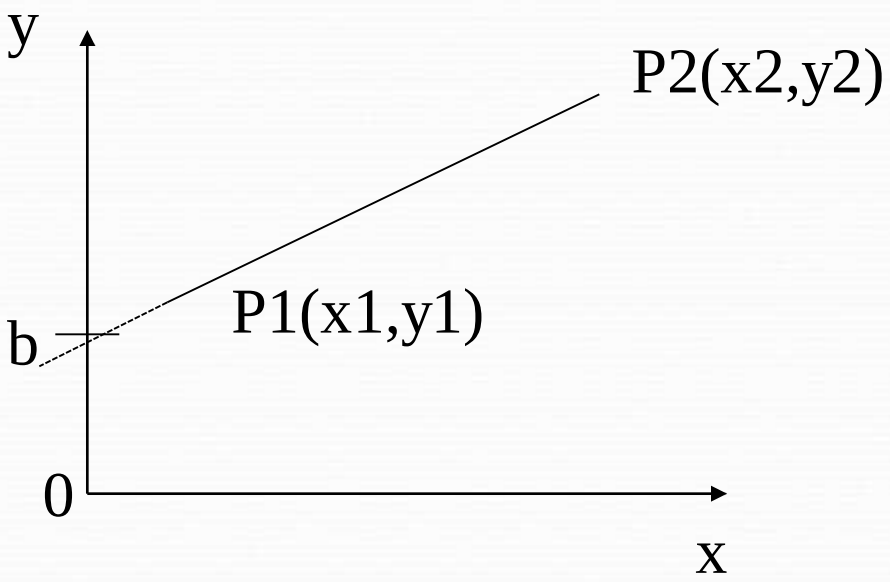
the slope intercept equation for a line:

$$y = mx + b \qquad (1)$$

where, **m** = Slope of the line

b = the y intercept of a line

The two endpoints of a line segment are specified at positions **(x1,y1)** and **(x2,y2).**



We can determine the value for slope m & b intercept as

$$m = y_2 - y_1 / x_2 - x_1$$

$$\text{i.e. } m = \Delta y / \Delta x \qquad (2)$$

LINE EQUATION

If the end points of line segments are (x1,y1) and (x2,y2), the line can be represented as

$Y=mX+b$

Where m slope is

$m= \frac{(y2-y1)}{(x2-x1)} = \frac{\Delta y}{\Delta x}$, assume $0 \leq m \leq 1$

For a given x interval Δx along a line, calculate the corresponding Δy

$\Delta y = m * \Delta x$, for slope $|m| < 1$

Similarly for a given y interval Δy along a line , corresponding Δx

$\Delta x = \Delta y/m$ for slope $|m| > 1$

Example 1 The endpoints of line are(0,0) & (6,18). Compute each value of y as x steps from 0 to 6 and plot the result.

Solution : Equation of line is y= mx +b

m = y2-y1/x2-x1= 18-0/6-0 = 3

Next the y intercept b is found by plugging y1& x1 into the equation y = 3x + b,

0 = 3(0) + b. Therefore, b=0, so the equation for the line is y= 3x.

The challenge is to find a way to calculate the next x,y position by previous one as quickly as possible.

Scan Conversion algorithm for Line segment

- **Algorithms are**

- 1. **DDA (Digital Differential Analyzer)**
- **2. Bresenham's Algorithm**

- DDA was a electro mechanical device for digital simulation of differential equations.

$$\frac{dy}{dx} = m,$$

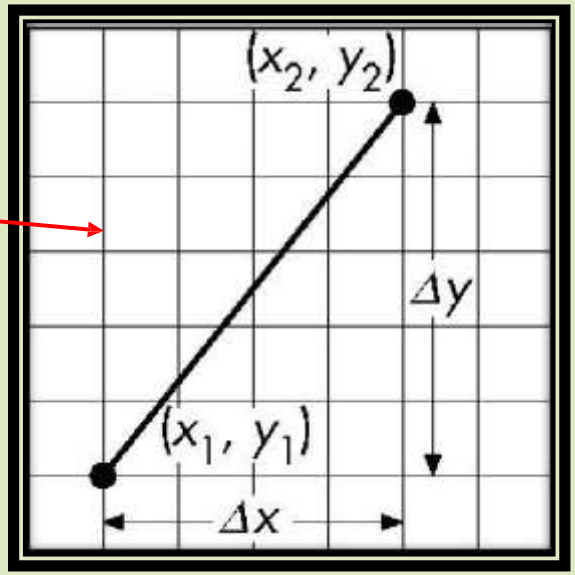
Where ' m ' is the slope.

Scan Conversion of Line Segments

- Start with line segment in window coordinates with integer values for endpoints
- Assume implementation has a **`write_pixel`** function

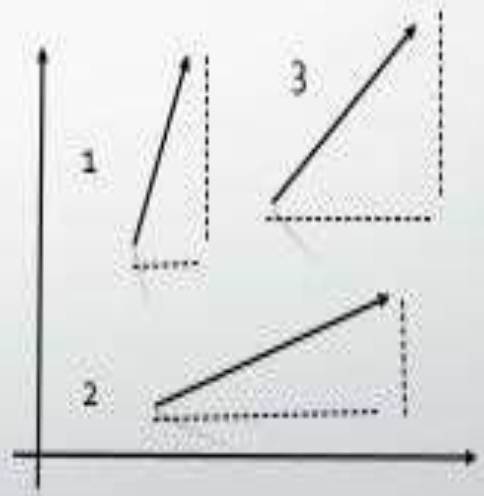
$$y = mx + c$$

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



Slope Conditions for Algorithms

- 1. $\Delta y / \Delta x > 1$ when $0 > 45$
- 2. $\Delta y / \Delta x < 1$ when $0 < 45$
- 3. $\Delta y / \Delta x = 1$ when $0 = 45$



DDA Algorithm

The Digital differential analyzer (DDA) algorithm is an incremental scan-conversion method.

Such an approach is characterized by performing calculations at each step using results from the preceding step.

DDA Algorithm

- Slope $|m| < 1$ or $|m| = 0$(horizontal line),

  $\Delta y = m \Delta x$,

  moving from x1 to x2 and x is increase by 1 for each iteration. And y is increase by $\Delta y = m$.

```
For(x=x1; x<=x2, x++)
{
   y += m;
//note:m is float number
 write_pixel(x,round(y),line_color);
}
```

For Horizontal line

(x1,y1)(x2,y2)

(2,2) (9,2)

Δ x=9-2=7

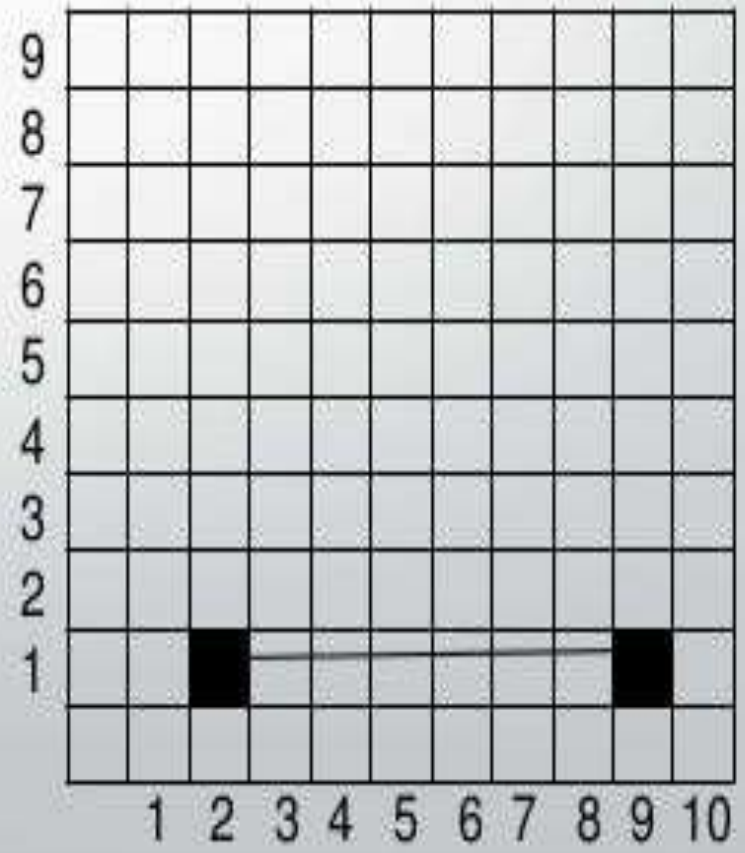
Δ y=2-2=0

m= Δ y/ Δ x=0/7=0

xinc=7/7=1

yinc=0/7=0

X	Y
2	2
3	2
4	2
5	2
6	2
7	2
8	2
9	2



For Horizontal line

(x1,y1)(x2,y2)

(2,2) (9,2)

Δ x=9-2=7

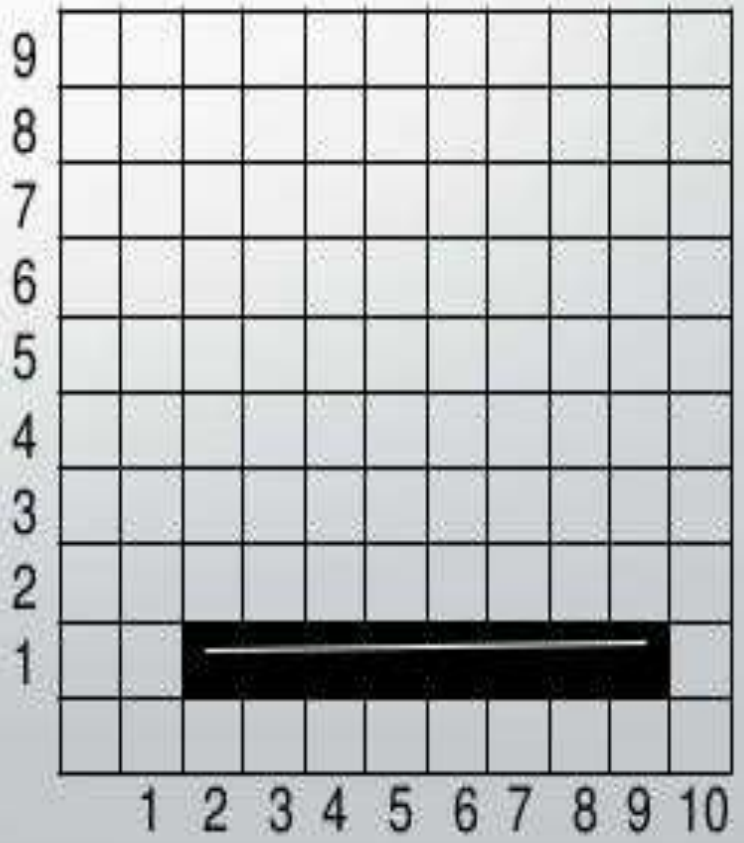
Δ y=2-2=0

m= Δ y/ Δ x=0/7=0

xinc=7/7=1

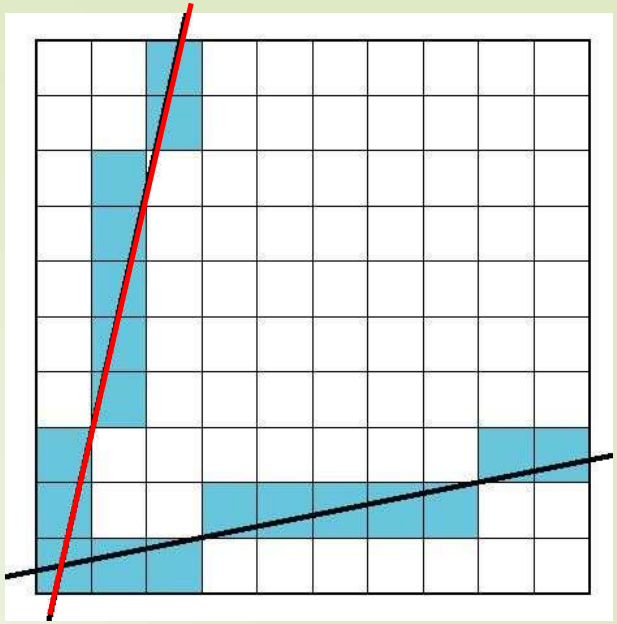
yinc=0/7=0

X	Y
2	2
3	2
4	2
5	2
6	2
7	2
8	2
9	2



Using Symmetry

- Use for $1 \geq m \geq 0$
- For $m > 1$, swap role of x and y
  - For each y, plot closest x



DDA Algorithm

- Slope |m| >1

  Δx= Δy/m ,

moving from y1 to y2 and y is increase by 1 for each iteration. And x is increase by Δx =1/m.

```
For(y=y1; y<=y2, y++)
{
   x += 1/m;
//note:m is float number
 write_pixel(round(x),y,line_color);
}
```

For Vertical line

(x1,y1)(x2,y2)

(2,3) (2,8)

$\Delta x=2-2=0$

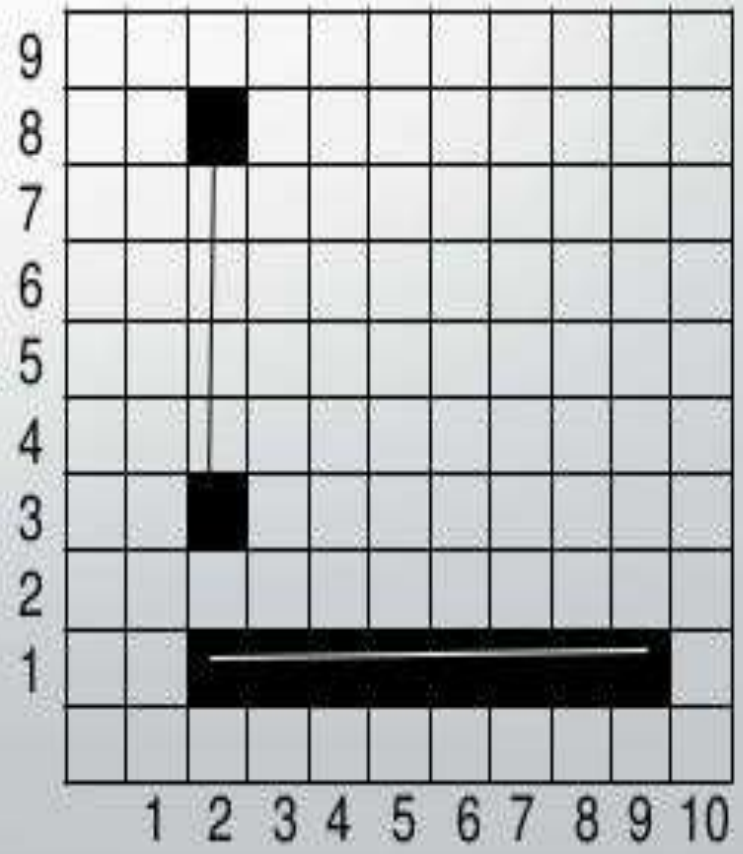
$\Delta y=8-2=6$

$m=\Delta y/\Delta x=6/0=\infty$

xinc=0/6=0

yinc=6/6=1

X	Y
2	3
2	4
2	5
2	6
2	7
2	8



For Vertical line

(x1,y1)(x2,y2)

(2,3) (2,8)

$\Delta x = 2-2 = 0$

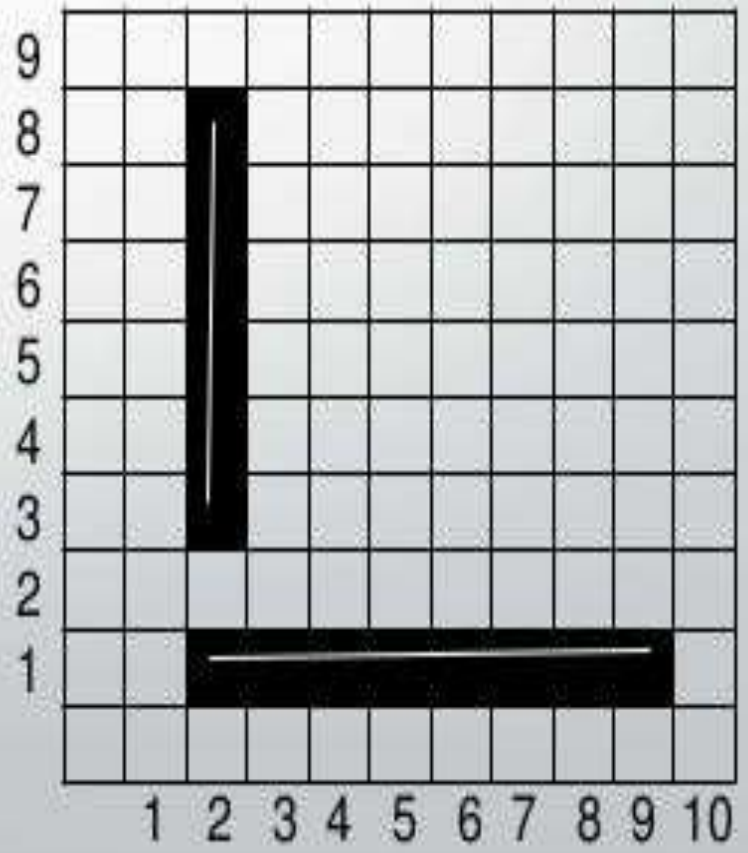
$\Delta y = 8-2 = 6$

$m = \Delta y / \Delta x = 6/0 = \infty$

xinc=0/6=0

yinc=6/6=1

X	Y
2	3
2	4
2	5
2	6
2	7
2	8



For any Diagonal line

(x1,y1)(x2,y2)

(1,2) (9,5)

Δ x=9-1=8

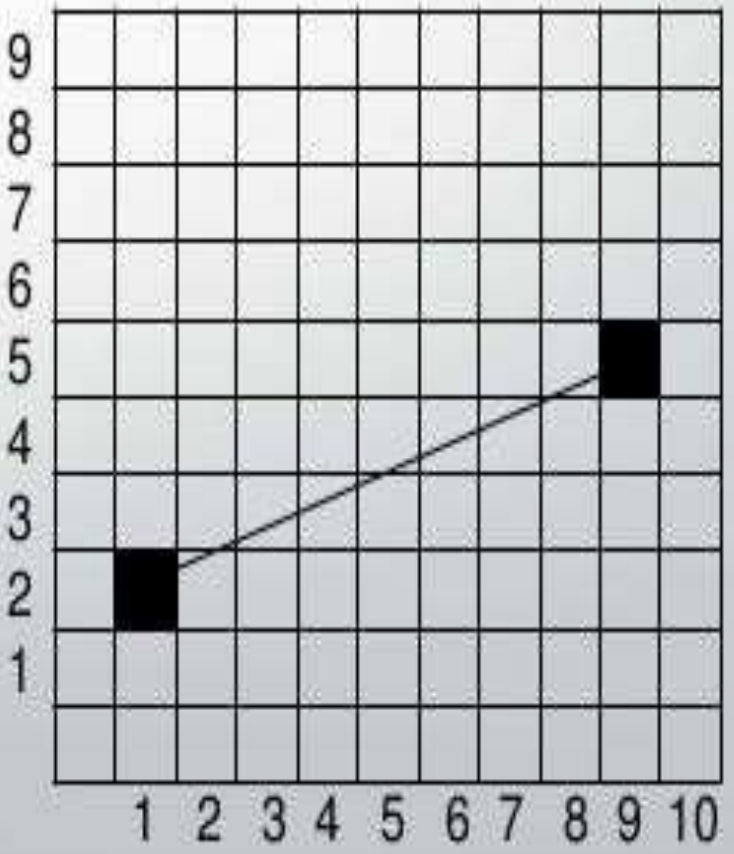
Δ y=5-2=3

m= Δ y/ Δ x=3/8=0.37

xinc=8/8=1

yinc=3/8=0.37

X	Y
1	2
2	2.37=2
3	2.74=3
4	3.11=3
5	3.48=3
6	3.85=4
7	4.22=4
8	4.59=5
9	4.96=5



For any Diagnal line

(x1,y1)(x2,y2)

(1,2) (9,5)

Δ x=9-1=8

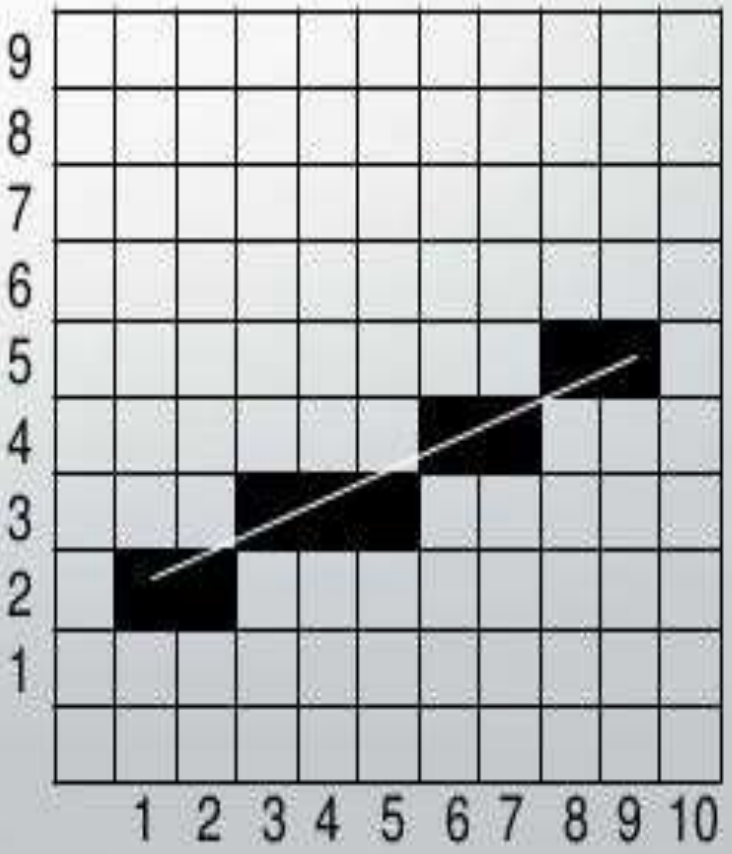
Δ y=5-2=3

m= Δ y/ Δ x=3/8=0.37

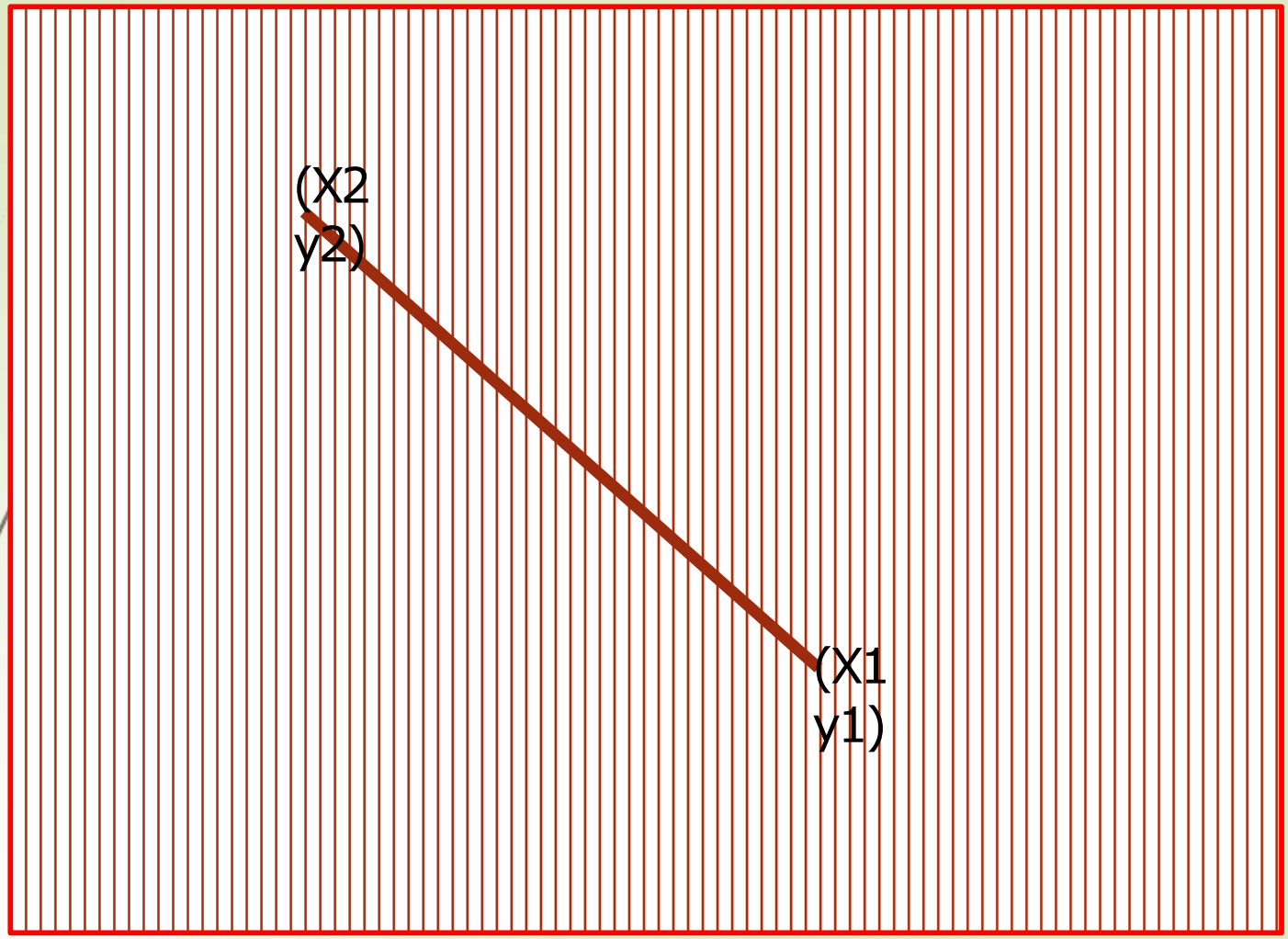
xinc=8/8=1

yinc=3/8=0.37

X	Y
1	2
2	2.37=2
3	2.74=3
4	3.11=3
5	3.48=3
6	3.85=4
7	4.22=4
8	4.59=5
9	4.96=5



When X1 >X2 or Y1 >Y2



DDA Algorithm

- Slope less than or equal to -1 (negative)

  $\Delta y = m\,\Delta x$ moving from x2 to x1 and x is decrease by 1 for each iteration. And y is decrease by $\Delta y = m$.

```
For(x=x2; x<=x1, x--)
{
   y += m;
//note:m is float number
 write_pixel(x,round(y),line_color);
}
```

DDA Algorithm

- Slope m > -1

  Δx= Δy/m , where y1>y2

moving from y2 to y1 and y is decrease by 1 for each iteration. And x is decrease by Δx =1/m.

```
For(y=y2; y<=y1, y--)
{
   x += 1/m;
//note:m is float number
 write_pixel(round(x),y,line_color);
}
```

DDA Algorithm

```
dx-=x2-x1;
dy=y2-y1;
If(abs(dx) > abs(dy))
    Steps=abs(dx);
else
    Steps=abs(dy);
xinc=dx/steps;
yinc=dy/steps;
For(i-1, i<=step, i++)
{
    putpixel(x1,y1);
    x1=x1+xinc;
    y1=y1+yinc;
}
```

Algorithm:

(x1,y1) (x2,y2) are the end points and dx, dy are the float variables.

Where dx= abs(x2-x1) and dy= abs(y2-y1)

(i) If dx >=dy then

length = dx

else

length = dy

endif

(ii) Xinc = (x2-x1)/length

Yinc= (y2-y1)/length

(iii) i = 0

(iv) Plot ((x), (y))

(v) x = x + Xinc

y = y + Yinc

(vi) i = i + 1

(vii) If i < length then go to step (iv)

(vii) Stop

Example 1: Scan convert a line having end points (3,2) & (4,7) using DDA.

Solution: $dx = x2 - x1 = 4-3 = 1$

$dy = y2 - y1 = 7-2 = 5$

As, $dx < dy$ then

$length = y2-y1 = 5$

$Xinc = (x2-x1)/ length = 1/5 = 0.2$

$Yinc = (y2-y1)/ length = 5/5 = 1$

x1	y1	x2	y2	L	Xinc	Yinc	i	x	y	setpixel
3	2	4	7	5	0.2	1	0	3	2	(3,2)
							1	3.2	3	(3,3)
							2	3.4	4	(3,4)
							3	3.6	5	(4,5)
							4	3.8	6	(4,6)
							5	4.0	7	(4,7)

Example 2: Scan convert a line having end points (0,0) & (5,10) using DDA.

$X_1=0, y_1=0$ & $x_2=5, y_2=10$

Solution: $dx = x_2 - x_1 = 5-0 = 5$

$dy = y_2 - y_1 = 10-0 = 10$

As, $dx < dy$ then

$length = y_2-y_1 = 10$

$Xinc = (x_2-x_1)/ length = 5/10 = 0.5$

$Yinc = (y_2-y_1)/ length = 10/10 = 1$

x1	y1	x2	y2	L	Xinc	Yinc	i	x	y	setpixel
0	0	5	10	10	0.5	1	0	0	0	(0,0)
							1	0.5	1	(1,1)
							2	1.0	2	(1,2)
							3	1.5	3	(2,3)
							4	2.0	4	(2,4)
							5	2.5	5	(3,5)
							6	3.0	6	(3,6)
							7	3.5	7	(4,7)
							8	4.0	8	(4,8)
							9	4.5	9	(5,9)
								5.0	10	(5,10)

Example 3: Scan convert a line having end points (0,0) & (5,5) using DDA.

$X_1=0, y_1=0$ & $x_2=5, y_2=5$

Solution: $dx = x_2 - x_1 = 5-0 = 5$

$dy = y_2 - y_1 = 5-0 = 5$

As, $dx < dy$ then

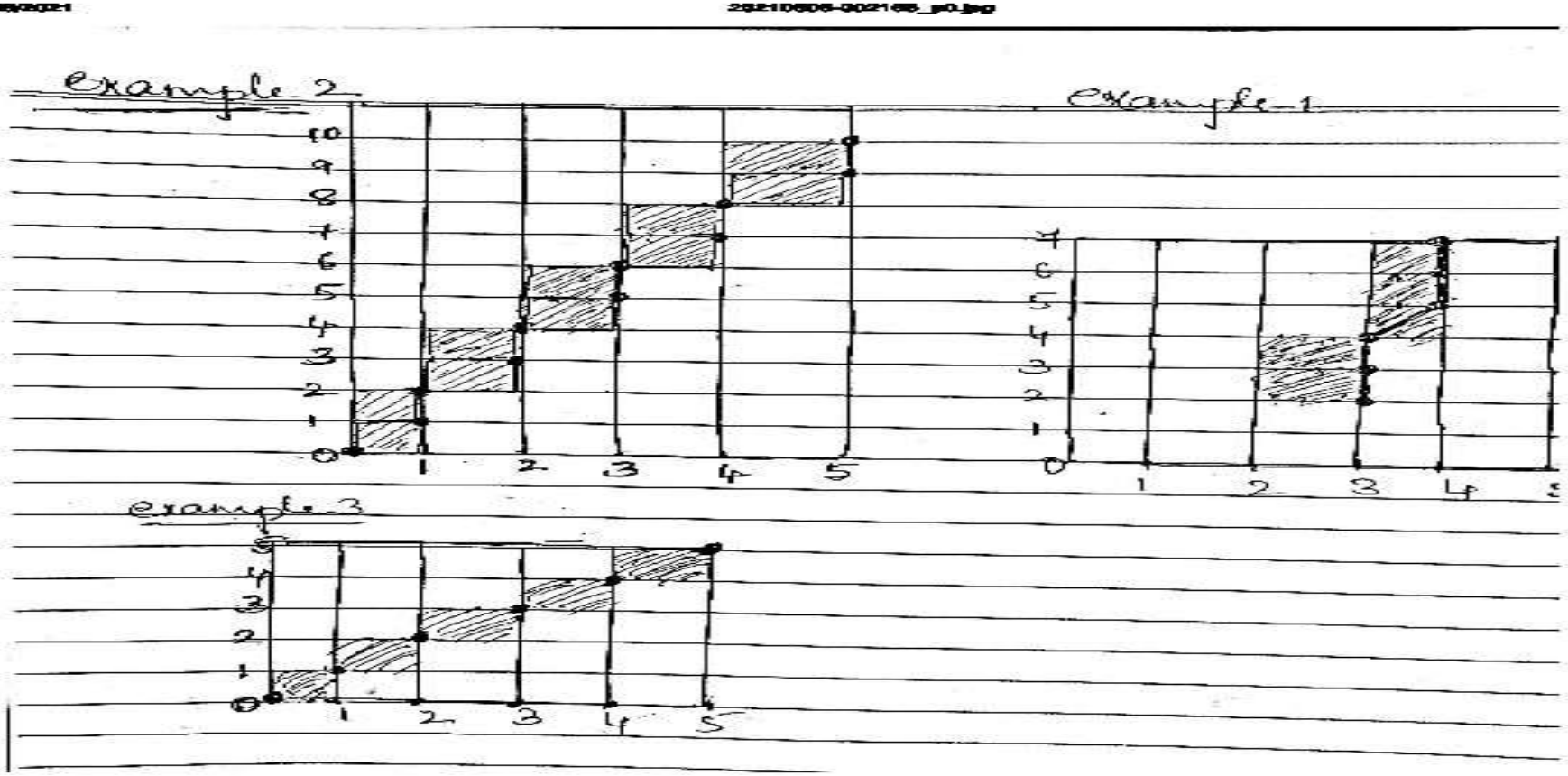
$length = x_2-x_1 = 5$

$Xinc = (x_2-x_1)/ length = 5/5 = 1$

$Yinc = (y_2-y_1)/ length = 5/5 = 1$

x1	y1	x2	y2	L	Xinc	Yinc	i	x	y	setpixel
0	0	5	5	5	1	1	0	0	0	(0,0)
							1	1	1	(1,1)
							2	2	2	(2,2)
							3	3	3	(3,3)
							4	4	4	(4,4)
							5	5	5	(5,5)

Results



Advantages and Disadvantages of DDA

- Advantages :
  1. Simplest line drawing algorithm
  2. No special skills required for its implementation
  3. DDA draws the line faster than drawing the line by directly using the line equation.
- Disadvantages :
  1. It dependents on orientation which makes the end point accuracy poor.
  2. It requires floating point addition to determine each successive point which is time consuming.
  3. Error due to limited precision in floating point representation may cause calculated points to shift away from their actual position when the line is relatively long.

Limitations of DDA:

(1) The rounding operation & floating point arithmetic are time consuming procedures.

(2) Round-off error can cause the calculated pixel position to drift away from the true line path for long line segment.

DDA Example

- Suppose we want to draw a line starting at pixel (2,3) and ending at pixel (12,8).
- What are the values of the variables x and y at each timestep?
- What are the pixels colored, according to the DDA algorithm?

numsteps = 12 – 2 = 10
xinc = 10/10 = 1.0
yinc = 5/10 = 0.5

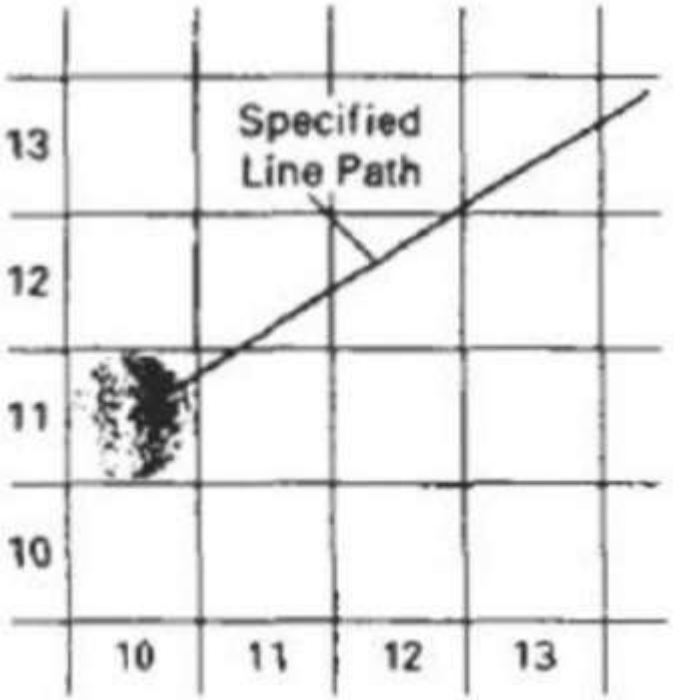
t	x	y	R(x)	R(y)
0	2	3	2	3
1	3	3.5	3	4
2	4	4	4	4
3	5	4.5	5	5
4	6	5	6	5
5	7	5.5	7	6
6	8	6	8	6
7	9	6.5	9	7
8	10	7	10	7
9	11	7.5	11	8
10	12	8	12	8

Bresenham's Line Algorithm

- DDA algorithm has a disadvantage, that is, the round off error. This error happens because the algorithm rounds off the actual floating point line values to integer values (for example, if actual value of a point of line is 12.34, the DDA algorithm rounds it to 12 and if the value is 12.62, it will round it to 13). This causes the calculated pixel positions to move away from the actual line path for long lines.
- For this reason we use another accurate and efficient line drawing algorithm called Bresenham's algorithm because it was developed by Bresenham.
- This algorithm draws line by using only integer calculations.

The Bresenham Line Algorithm

- The Bresenham algorithm is another incremental scan conversion algorithm.
- Developed by Jack Bresenham.
- The big advantage of this algorithm is that it uses only integer calculations.
- Accurate and efficient than DDA.

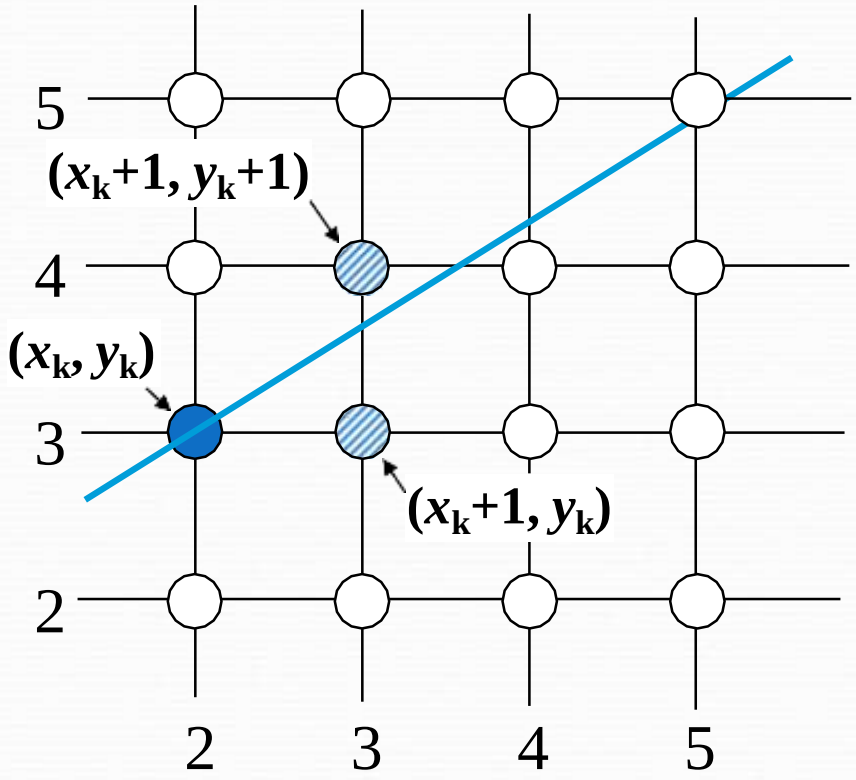


- Horizontal axis shows the pixel columns
- Vertical axis shows the line position.
- If we take unit x-intervals and we plot a point at (x_k,y_k), then in order to calculate the y values we need to check which of the two points is closer to the line path at each sample step. That is, we need to decide at the next sample position x_{k+1} whether to choose the pixel (x_{k+1},y_k) or (x_{k+1},y_{k+1}), that is, whether to choose point (11,11) or (11,12) for the next step.

- These issues are solved by Bresenham's algorithm.
- It decides which value to take at next position whether (x_{k+1}, y_k) or (x_{k+1}, y_{k+1}). It does this by checking the sign of a decision parameter (p_k) which is equal to the difference between the separations of the two pixel positions (y_k and y_{k+1}) from the actual line path.
- If p_k is negative it takes the point as (x_{k+1}, y_k).
- If p_k is positive it takes point as (x_{k+1}, y_{k+1})

The Big Idea

Move across the *x* axis in unit intervals and at each step choose between two different *y* coordinates

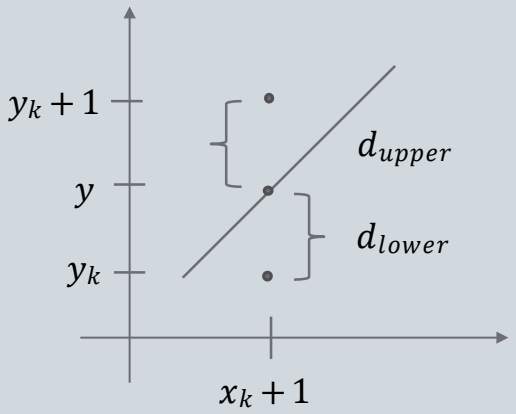


For example, from position (2, 3) we have to choose between (3, 3) and (3, 4)

We would like the point that is closer to the original line

DERIVATION

- Starting from the left endpoint (x0, y0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.



At sample positions Xk+ 1 the vertical separations from the line are labelled d_{upper} and d_{lower} y coordinate on the line at $x_k + 1$ is,

y= m(Xk+1)+b

So

dlower= Y-Yk = m(Xk+1)+b-Yk

dupper= (Yk+1)-Y = Yk+1-m(Xk+1)-b

DERIVATION

- It can be used to make decision about which pixel is closer to the line
- This decision is based on the difference between the two pixel positions,

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

- By substituting $m = \Delta y/\Delta x$ and both are differences of end points,

$$\begin{aligned}\Delta x\left(d_{upper} - d_{lower}\right) &= \Delta x\left(2\left(\frac{\Delta y}{\Delta x}\right)(x_k + 1) - 2y_k + 2b - 1\right)\\ &= 2\Delta y.x_k - 2\Delta x.y_k + 2\Delta y + \Delta x(2b - 1)\\ &= 2\Delta y.x_k - 2\Delta x.y_k + C\end{aligned}$$

DERIVATION

- Now, a decision parameter P_k for the kth step along a line,

$$P_k = \Delta x\left(d_{upper} - d_{lower}\right)$$
$$= 2\Delta y.\, x_k - 2\Delta x.\, y_k + C$$

- The sign of P_k is same as that of $d_{upper} - d_{lower}$
- If P_k is –ve then we choose the lower pixel i.e. y_k only, otherwise we choose the upper pixel i.e. $y_k + 1$
- So, for $P_k + 1$ at step $k + 1$,

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

- Subtracting P_k,

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

DERIVATION

- x_{k+1} is same as $x_k + 1$ so,

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

- Here, $y_{k+1} - y_k$ is either 0 or 1 depending on the sign of P_k
- If $P_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and new value of P is,

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

- If $P_k > 0$, the next point to plot is $(x_k + 1, y_k + 1)$ and new value of P is,

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

- The first decision parameter P_0 is evaluated at (x_0, y_0) is,

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

BRESENHAM'S LINE DRAWING ALGORITHM

1. Input the two line end-points, storing the left end-point in (x_1, y_1)

2. Calculate the constants Δx *i.e.* dx, Δy *i.e.* dy, $2\Delta y$ and $2\,\Delta x$, get the first value for the decision parameter as:

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

3. Initialize starting

4. Initialize i=1 as a counter,

   If Pk<0, next point to plot is (x_k+1, y_k) and

$$p_{k+1} = p_k + 2\Delta y$$

If Pk>0, the next point to plot is (x_k+1, y_k+1) and:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 ($\Delta x - 1$) times

The algorithm and derivation above assumes slopes are less than 1. for other slopes we need to adjust the algorithm slightly.

Adjustment

For m>1, we will find whether we will increment x while incrementing y each time.

After solving, the equation for decision parameter p_k will be very similar, just the x and y in the equation will get interchanged.

Example1:

Using Bresenham's algorithm, generate the coordinates of the pixels that lie on a line segment having the endpoints (2, 3) and (5, 8).

Case: When slope (m) > 1

Now let's solve the same numerical using BLAAlgorithm.

S-1: x1=2; y1=3; x2=5; y2=8.

S-2: dy=y2-y1= 8-3= 5 and dx = x2-x1 = 5-2 = 3

dy-dx = 5-3 = 2; and 2 * dy = 10;

m(slope) = dy/dx => 5/3

Slope is more than 1 so we will follow the following method.

S-3: Calculate d = 2*dx-dy , so d=2*3 – 5 = 1.

S-4: Always remember the rule for any line algorithm, If m is less than 1 then always increment x and calculate y. If m is more than 1 then do opposite , which is, always increment y and calculate x.

In this case, we will increase y by 1 every step as m (Slope) is more than 1 and calculate y as follows.

a) If d >=0 then x1 = x1 + 1
and y1 = y1 +1 with new d = d + 2*(dx-dy)

b) If d<0 then x1 = x1 (remains same) and y1=y1+1 with new d = d + 2*dx

Note: y is always increasing ->Why?, its because for m>1, always increase y.

Sl.no	X1	Y1	d	Pixel Plotted
1	2	3	d = 2*dx – dy = 1	2,3
2	3	4	From step 4 (a) d =1+ 2*(3-5) = -3	3,4
3	3	5	From step 4 (b) d =-3+ 2* 3= 3	3,5
4	4	6	From step 4 (a) d=3+ 2*(3-5)=-1	4,6
5	4	7	From step 4 (b) d = -1 + 2*3 = 5	4,7
6	5	8	From step 4 (a) d = 5+ 2*(3-5) = 1	5,8

Cant increase x as x has reached final Cant increase y as y has reached final So algo will stop here.

Example 2:

Draw a line from (1,1) to (8,7) using Bresenham's Line Algorithm.

Case - When Slope (m) <1
Now let's solve the same numerical using BLAAlgorithm.

S-1: x1=1; y1=1; and x2=8; y2=7.

S-2: dy=7-1 = 6 and dx = 8-1 = 7

dy-dx = 6-7 = -1; and 2 * dy = 12;

S-3: Calculate d = 2*dy-dx , so
d=2*6 – 7 = 5 (Note the change here for m<1)

S-4: We will increase x by 1 every step as m is less than 1 and calculate y as follows

Rule: If slope (m) is less than 1 (m<1) then always increase x and calculate y.

a) If d >=0 then x1 = x1 + 1 and y1 = y1 + 1
with new d = d + 2*(dy-dx)

b) If d<0 then x1 = x1 + 1 and y1 will not change
with new d = d + 2*dy

Sl.no	X1	Y1	d	Pixel Plotted
1	1	1	d = 2*dy – dx = 5	1,1
2	2	2	From step 4 (a) d = 5 + 2*(6-7)= 3	2,2
3	3	3	From step 4 (a) d = 3 +2*(6-7) = 1	3,3
4	4	4	From step 4 (a) d = 1 + 2*(6-7) = -1	4,4
5	5	4	From step 4 (b) d = -1 + 2*6 = 11	5,4
6	6	5	From step 4 (a) d = 11 +2(6-7) = 9	6,5
7	7	6	d=9+2*(6-7)=7	7,6
8	8	7	Algorithm will stop here after plotting final pixel(8,7).	8,7

Bresenham Example

Let's have a go at this

Let's plot the line from (20, 10) to (30, 18)

First off calculate all of the constants:

- Δx: 10
- Δy: 8
- $2\Delta y$: 16
- $2\Delta y - 2\Delta x$: -4

Calculate the initial decision parameter p_0:

- $p0 = 2\Delta y - \Delta x = 6$

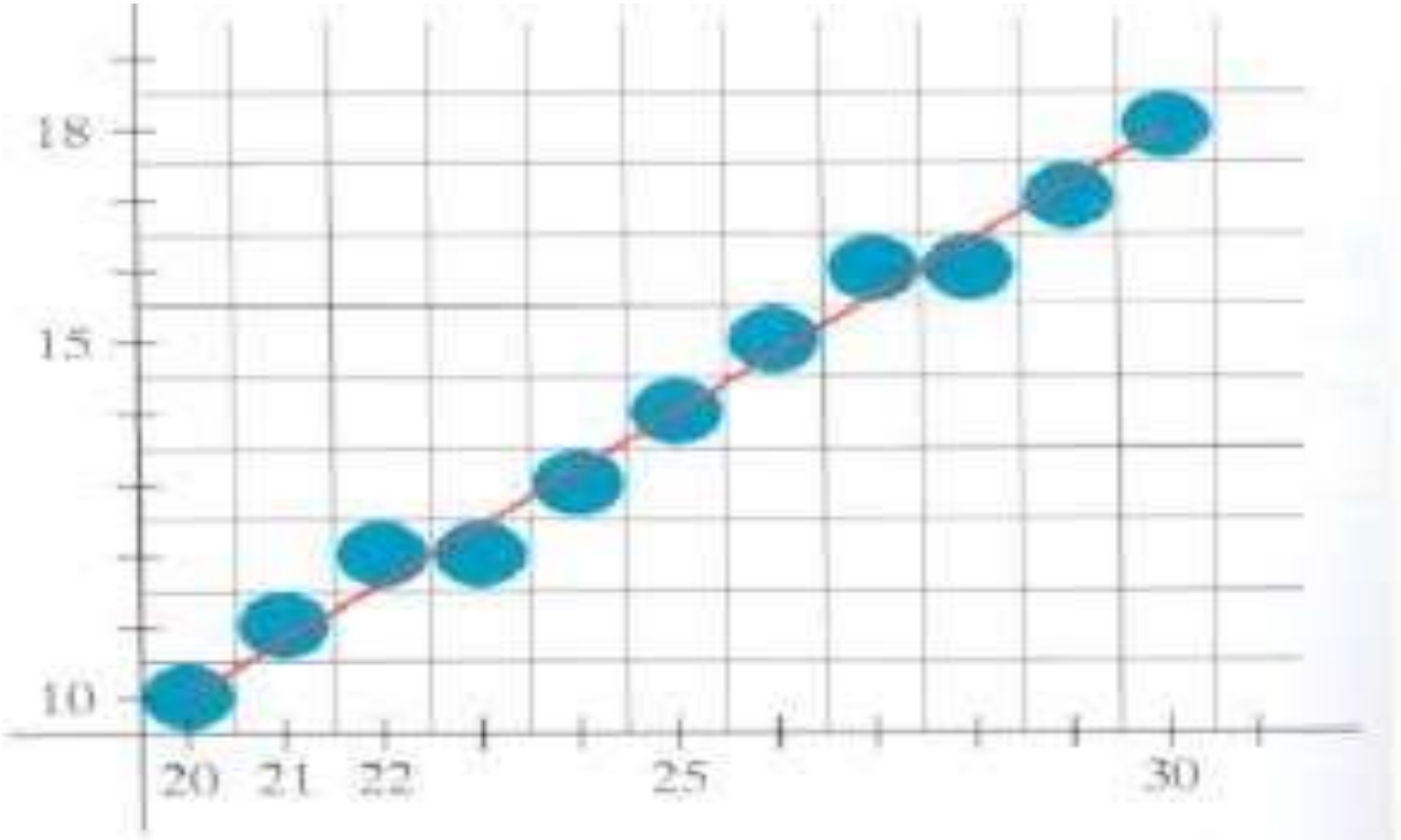
EXAMPLE

- End points (20,10) and (30,18)
  - Δx=x2-x1 =30-20 =10
  - Δy=y2-y1 =18-10 =8
  - m= $\Delta y/\Delta x$=8/10=0.8

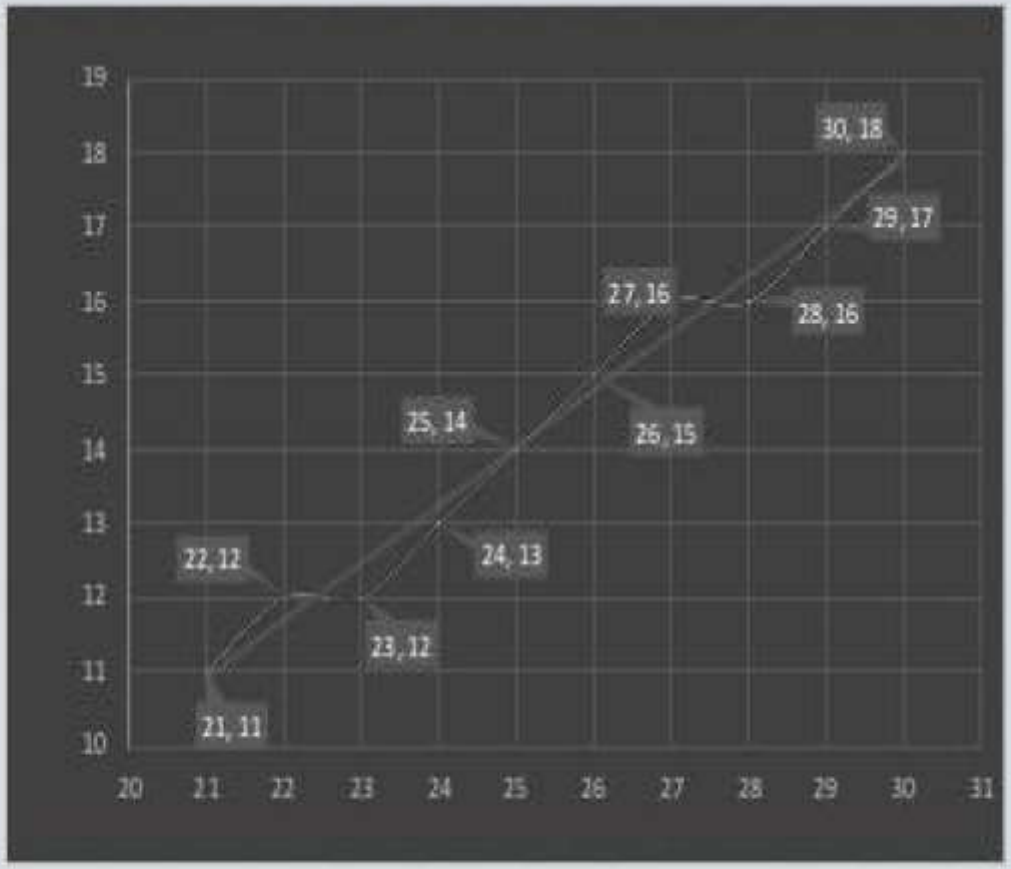
k	P_k	(x_{k+1}, y_{k+1})
0	$6 > 0$	(21,11)
1	$2 > 0$	(22,12)
2	$-2 < 0$	(23,12)
3	$14 > 0$	(24,13)
4	$10 > 0$	(25,14)

k	P_k	(x_{k+1}, y_{k+1})
5	$6 > 0$	(26,15)
6	$2 > 0$	(27,16)
7	$-2 < 0$	(28,16)
8	$14 > 0$	(29,17)
9	$10 > 0$	(30,18)

Example (cont.)



EXAMPLE


19
18
17
16
15
14
13
12
11
10
20
21
22
23
24
25
26
27
28
29
30
31
30, 18
29, 17
27, 16
28, 16
25, 14
26, 15
22, 12
24, 13
23, 12
21, 11

ADVANTAGES

- Uses fixed points
- Easy to calculate (only addition & subtraction)
- Fast execution compare to DDA
- More accurate and efficient

DISADVANTAGES

- Drift away from actual line path
- Causes stair-case pattern

Bresenham Line Algorithm Summary

The Bresenham line algorithm has the following advantages:

- An fast incremental algorithm
- Uses only integer calculations

Comparing this to the DDA algorithm, DDA has the following problems:

- Accumulation of round-off errors can make the pixelated line drift away from what was intended
- The rounding operations and floating point arithmetic involved are time consuming

Differentiate between DDA Algorithm and Bresenham's Line Algorithm:

DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic.	1. Bresenham's Line Algorithm use fixed point, i.e., Integer Arithmetic
2. DDA Algorithms uses multiplication & division its operation	2.Bresenham's Line Algorithm uses only subtraction and addition its operation

DDA Algorithm	Bresenham's Line Algorithm
4. DDA Algorithm is not accurate and efficient as Bresenham's Line Algorithm.	4. Bresenham's Line Algorithm is more accurate and efficient at DDA Algorithm.
5.DDA Algorithm can draw circle and curves but are not accurate as Bresenham's Line Algorithm	5. Bresenham's Line Algorithm can draw circle and curves with more accurate than DDA Algorithm.

Circle Generating Algorithms

- What is a circle?
- It is a set of points that are all at a given distance ***r*** from center position (x_c, y_c).
- The distance relationship equation of a circle is expressed by the Pythagorean theorem in Cartesian coordinates as:

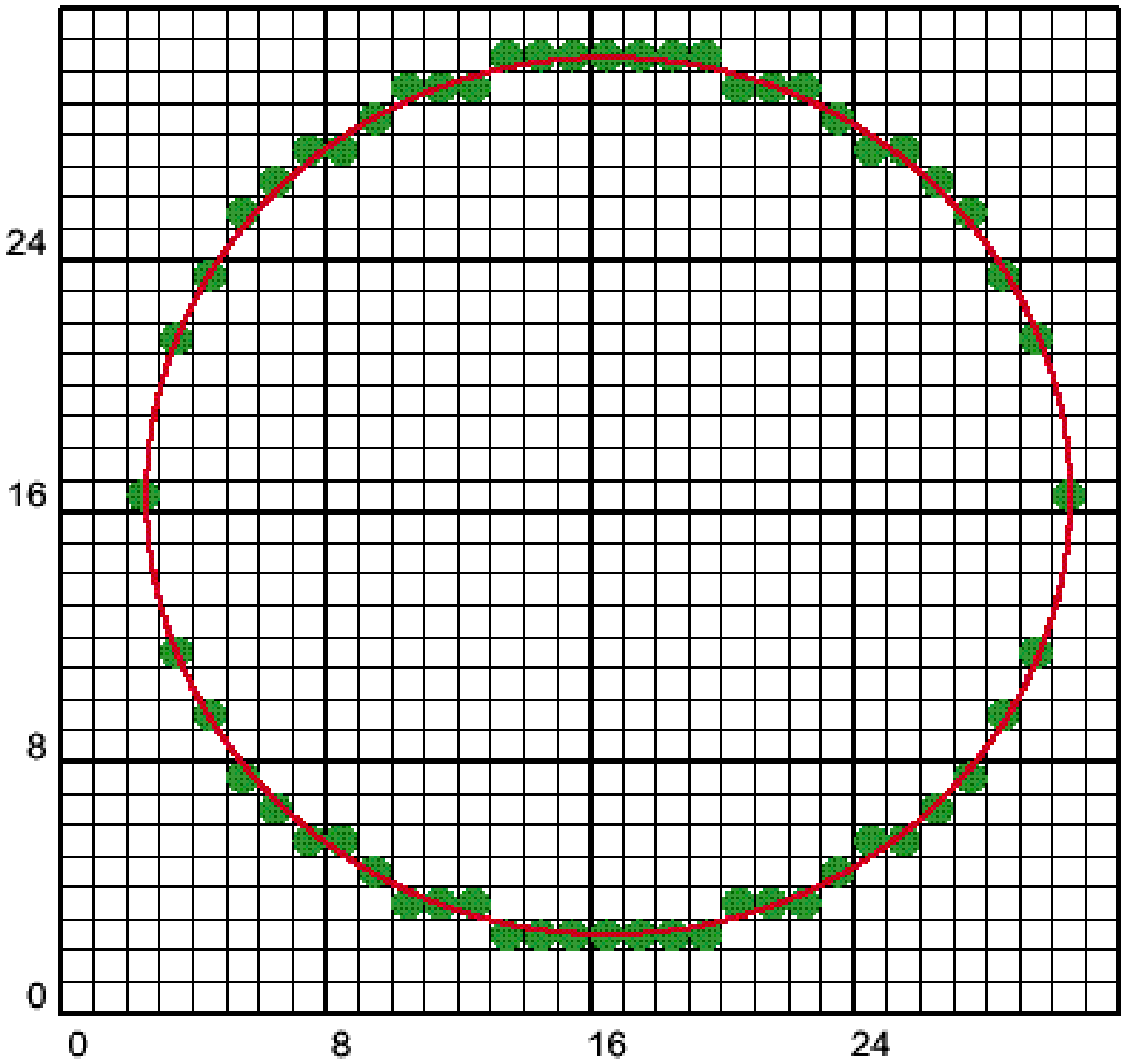
$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- We can re-write the circle equation as:

$$y = y_c \pm (r^2 - (x - x_c)^2)^{0.5}$$

- By substitution with x , x_c and y_c we can get y.

- Two problems with this approach:
  - it involves considerable computation at each step.
  - The spacing between plotted pixel positions is not uniform, as demonstrated below

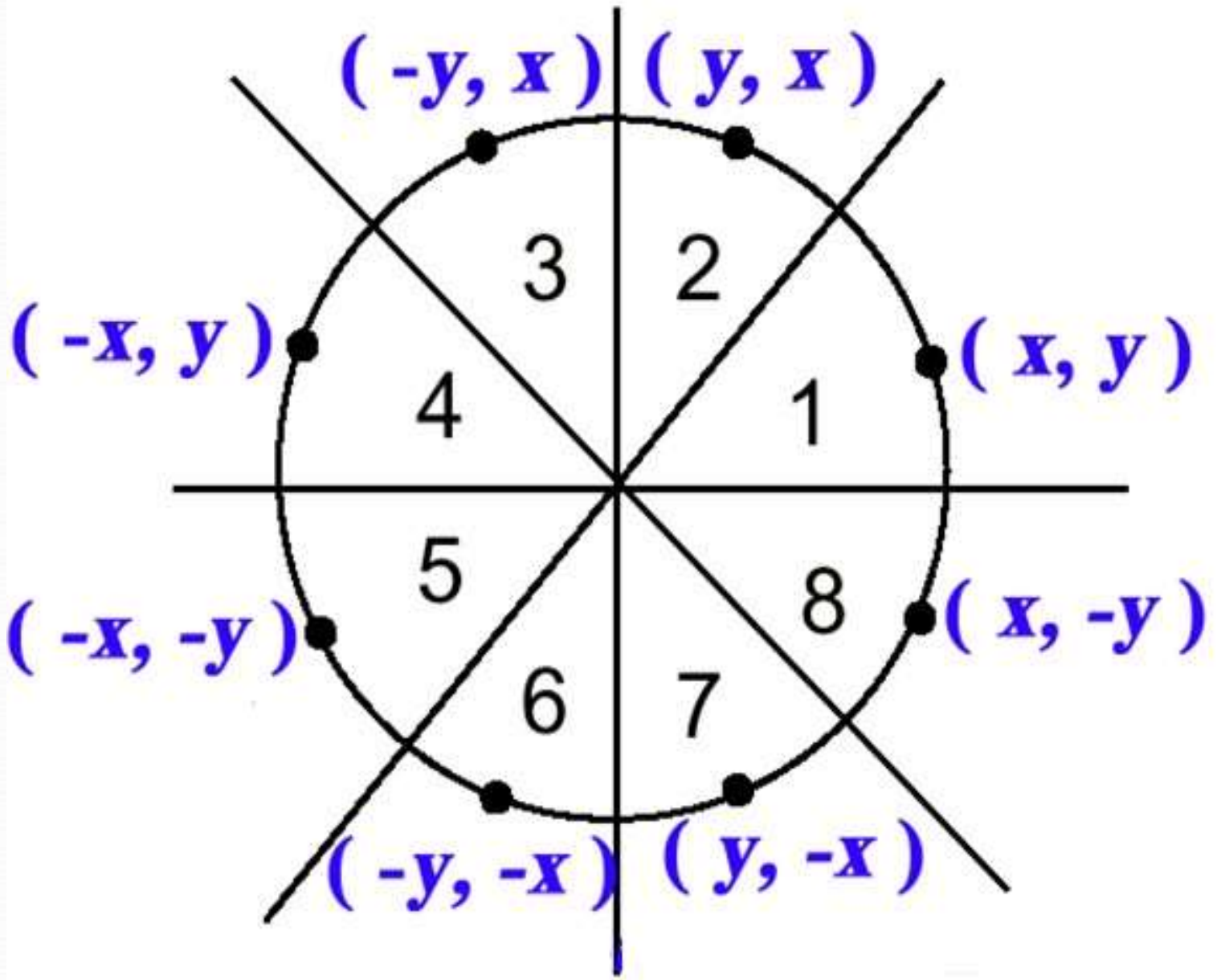

24
16
8
0
0
8
16
24

- Polar coordinates **(r and θ)** are used to eliminate the unequal spacing shown above.

- Expressing the circle equation in parametric polar form yields the pair of equations

  - $\mathbf{x = x_c + r \cos \theta}$
  - $\mathbf{y = y_c + r \sin \theta}$

- When a circle is generated with these equations using a fixed angular step size, a circle is plotted with equally spaced points along the circumference.
- The step size chosen θ depends on the application and the display device.
- Computation can be reduced by considering the symmetry of circles. The shape of the circle is similar in each quadrant.
- We can take this one step further and note that there is also symmetry between octants.


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

- We effectively make use of this 8 fold symmetry to generate a circle.

- We have $(x_c + x , y_c + y)$, the other points are:
  - $(x_c - x , y_c + y)$
  - $(x_c + x , y_c - y)$
  - $(x_c - x , y_c - y)$
  - $(x_c + y , y_c + x)$
  - $(x_c - y , y_c + x)$
  - $(x_c + y , y_c - x)$
  - $(x_c - y , y_c - x)$

Mid-point circle algorithm

- A method for direct distance comparison is to test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary.
- This method is more easily applied to other conics, and for an integer circle radius.
- we sample at unit intervals and determine the closest pixel position to the specified circle path at each step.

Mid-point circle algorithm (cont.)

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

- Then each calculated position (x, y) is moved to its proper screen position by adding x_c to x and y_c to y.

- Along the circle section from x = 0 to x = y in the first quadrant, the slope of the curve varies from 0 to - 1.

Mid-point circle algorithm (cont.)

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

- Positions in the other seven octants are then obtained by symmetry.

Mid-point circle algorithm (cont.)

- To apply the midpoint method. we define a circle function:

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

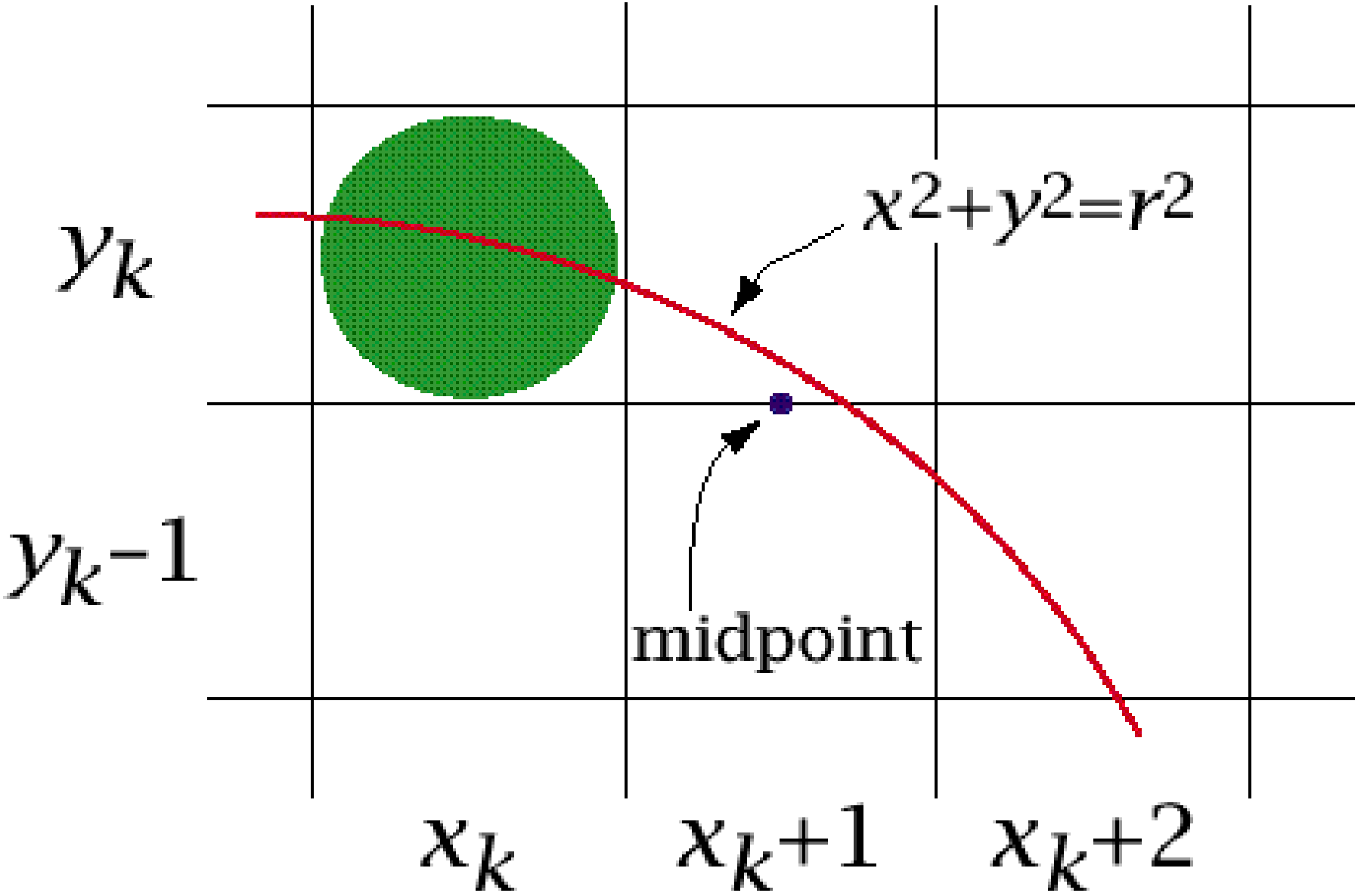
- Any point (x, y) on the boundary of the circle with radius r satisfies the equation $f_{circle}(x, y) = 0$.

- If $f_{circle}(x, y) < 0$, the point is inside the circle boundary ,

  If $f_{circle}(x, y) > 0$, the point is outside the circle boundary,

  If $f_{circle}(x, y) = 0$, the point is on the circle boundary.

Mid-point circle algorithm (cont.)



Calculating p_k

First, set the pixel at (x_k , y_k), next

determine whether the pixel $(x_k + 1, y_k)$ or the pixel $(x_k + 1, y_k – 1)$ is closer to the circle using:

$$p_k = fcircle\ (x_k + 1, y_k – ½)$$
$$= (x_k + 1)^2 + (y_k – ½)^2 – r^2$$

Assuming we have just plotted point at (xk,yk) we determine whether move E or SE by evaluating the circle function at the midpoint between the two candidate pixel positions.

pk is the decision variable

if *pk* <0 the midpoint is inside the circle

Thus the pixel above the midpoint is closer to the ideal circle, and we select pixel on scan line *yk*. i.e. Go E

If *pk* >0 the midpoint is outside the circle.

Thus the pixel below the midpoint is closer to the ideal circle, and we select pixel on scan line *yk*-1. i.e. Go SE

Successive decision parameters are obtained using incremental calculations.

➔We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$:

Calculating P_{k+1}

$$p_{k+1} = F_{circ}(x_{k+1} + 1, y_{k+1} - 1/2)$$

$$= [(x_k + 1) + 1]^2 + (y_{k+1} - 1/2)^2 - r^2$$

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

Where yk+1 = yk if p<0 (move E)

yk+1 = yk-1 if p>0 (move SE)

yk+1 and xk+1 can also be defined recursively

The initial decision parameter is obtained by evaluating the circle function at the start position (x0, y0) = (0, r):

- $p_0 = fcircle(1, r - ½) = 1 + (r - ½)^2 - r^2$

or

- $p_0 = 5/4 - r \cong 1 - r$

For integer radius r p_0 can be rounded to $p_0 = 1-r$ since all increments are integer.

Mid-point Circle Algorithm - Steps

1. Input radius **r** and circle center **(x_c, y_c).** set the first point

   $$(x_0, y_0) = (0, r).$$

2. Calculate the initial value of the decision parameter as $p_0 = 1 - r$.

3. At each x_k position, starting at $k = 0$, perform the following test:

   If $p_k < 0$, plot $(x_k + 1, y_k)$ and $p_{k+1} = p_k + 2x_{k+1} + 1$,

Mid-point Circle Algorithm - Steps

Otherwise,

plot $(x_k + 1, y_k - 1)$ and

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1},$$

where $2x_{k+1} = 2x_k + 2$ and

$2y_{k+1} = 2y_k - 2$.

Mid-point Circle Algorithm - Steps

4. Determine symmetry points on the other seven octants.

5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values: $\boldsymbol{x = x + x_c}$, $\boldsymbol{y = y + y_c}$

6. Repeat steps 3 though 5 until $\boldsymbol{x} \geq \boldsymbol{y}$.

7. For all points, add the center point (x_c, y_c)

Mid-point Circle Algorithm - Steps

- Now we drew a part from circle, to draw a complete circle, we must plot the other points.
- We have $(x_c + x , y_c + y)$, the other points are:
  - $(x_c - x , y_c + y)$
  - $(x_c + x , y_c - y)$
  - $(x_c - x , y_c - y)$
  - $(x_c + y , y_c + x)$
  - $(x_c - y , y_c + x)$
  - $(x_c + y , y_c - x)$
  - $(x_c - y , y_c - x)$

Code:

```
void draw_pixel(GLint cx, GLint cy)
{    glColor3f(0.5,0.5,0.0);
     glBegin(GL_POINTS);
     glVertex2i(cx, cy);
     glEnd();
}
void plotpixels(GLint h, GLint k, GLint x, GLint y)
{
      draw_pixel(x+h, y+k); draw_pixel(-x+h, y+k);
      draw_pixel(x+h, -y+k); draw_pixel(-x+h, -y+k);
      draw_pixel(y+h, x+k); draw_pixel(-y+h, x+k);
      draw_pixel(y+h, -x+k); draw_pixel(-y+h, -x+k);
}
```

```
void circle_draw(GLint xc, GLint yc, GLint r)
{  GLint  d=1-r,    x=0,    y=r;
    while(y>x)
    {
     plotpixels(xc, yc, x, y);
        if(d<0)
        d+=2*x+3;
   else
        {
            d+=2*(x-y)+5;
            --y;
         }
            ++x;
    }
          plotpixels(xc, yc, x, y);
}
```

Mid-point circle algorithm (Example)

- Given a circle radius r = 10, demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from x = 0 to x = y.

Solution:

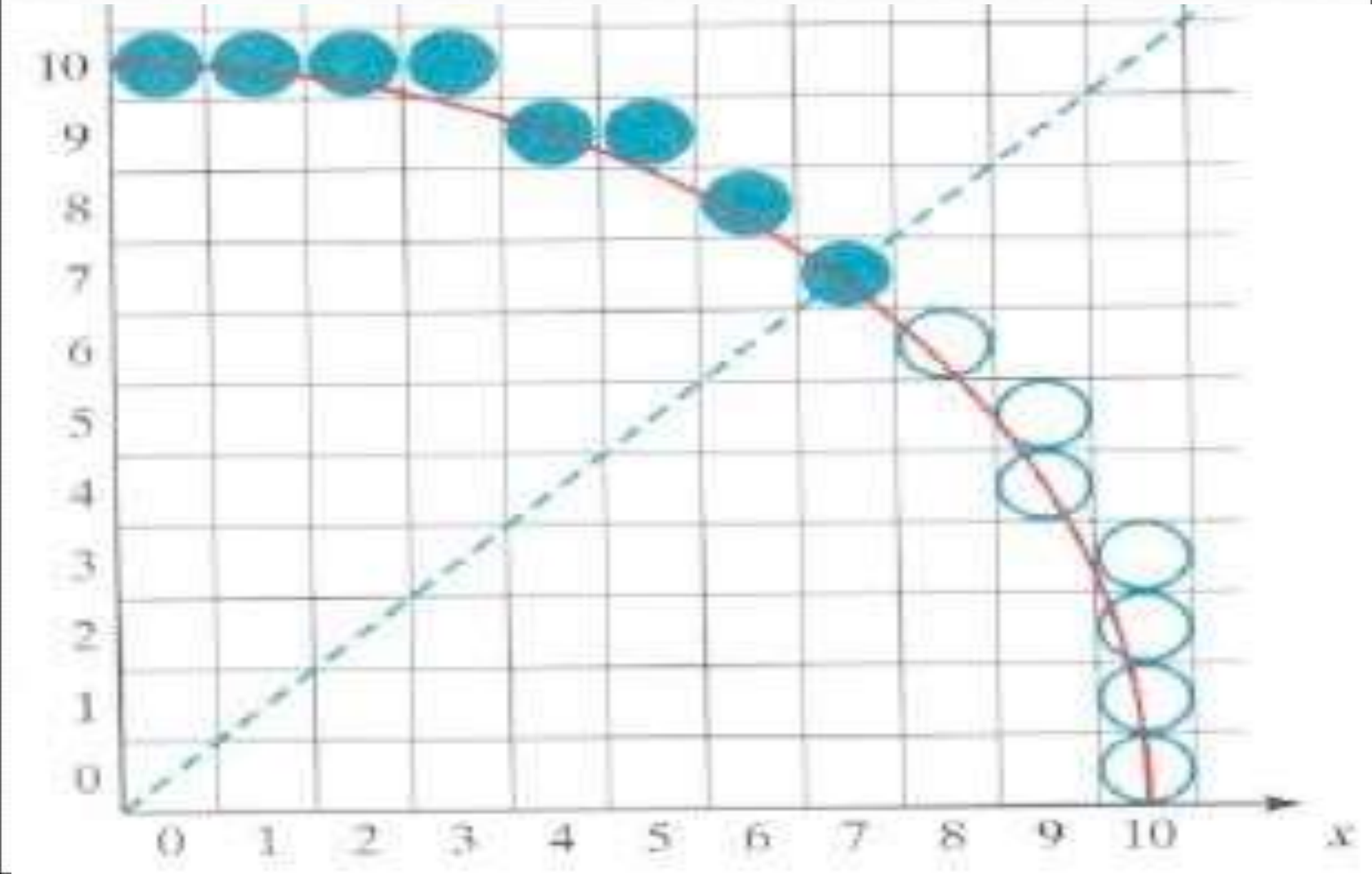
- $p_0 = 1 - r = -9$
- Plot the initial point $(x_0, y_0) = (0, 10)$,

- $2x_0 = 0$ and $2y_0 = 20$.
- Successive decision parameter values and positions along the circle path are calculated using the midpoint method as appear in the next table:

Mid-point circle algorithm (Example)

K	P_k	(x_{k+1}, y_{k+1})	$2\,x_{k+1}$	$2\,y_{k+1}$
0	– 9	(1, 10)	2	20
1	– 6	(2, 10)	4	20
2	– 1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	– 3	(5, 9)	10	18
5	8	(6,8)	12	16
6	5	(7,7)	14	14

Mid-point circle algorithm (Example)



Mid-point circle algorithm (Example2)

- Given a circle radius r = 15, demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from x = 0 to x = y.

Solution:

- $p_0 = 1 - r = -14$
- plot the initial point $(x_0 , y_0) = (0, 15)$,
- $2x_0 = 0$ and $2y_0 = 30$.
- Successive decision parameter values and positions along the circle path are calculated using the midpoint method as:

Mid-point circle algorithm (Example2)

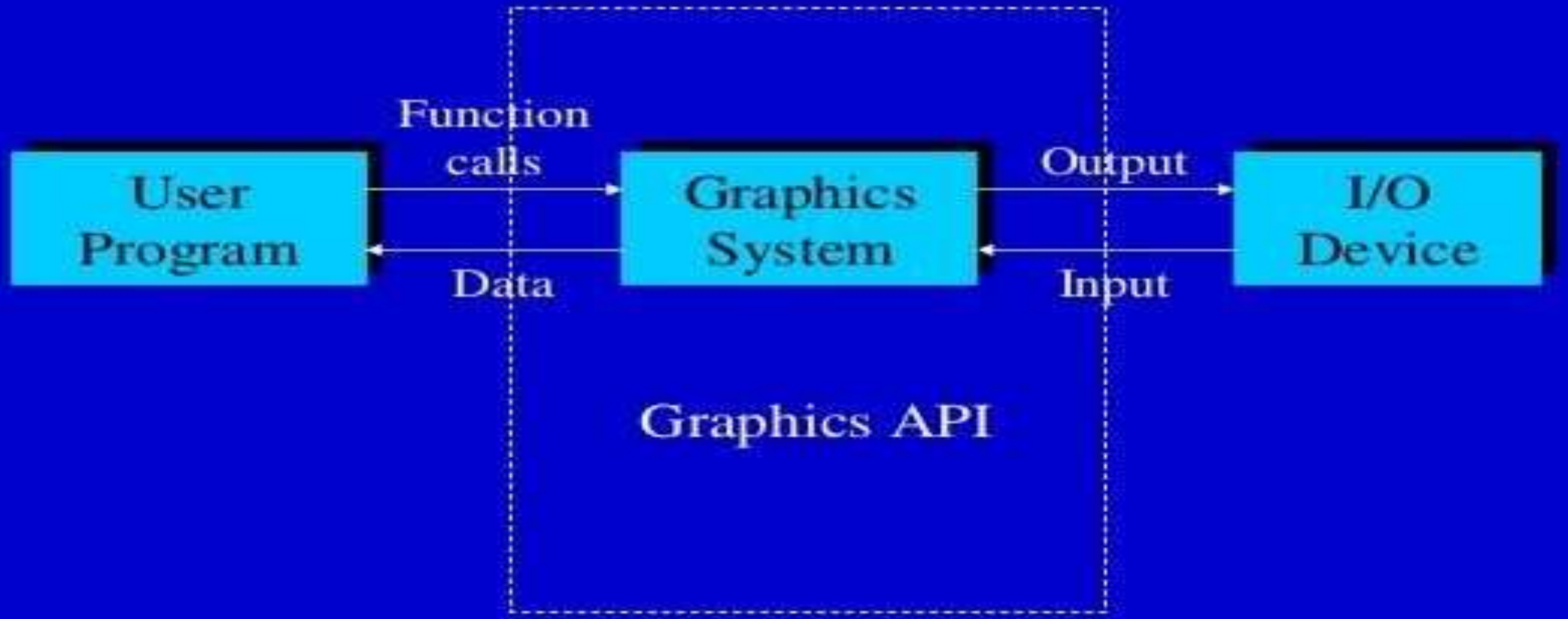
K	P_k	(x_{k+1}, y_{k+1})	$2\,x_{k+1}$	$2\,y_{k+1}$
0	– 14	(1, 15)	2	30
1	– 11	(2, 15)	4	30
2	– 6	(3, 15)	6	30
3	1	(4, 14)	8	28
4	– 18	(5, 14)	10	28

Mid-point circle algorithm (Example2)

K	P_k	(x_{k+1}, y_{k+1})	$2\,x_{k+1}$	$2\,y_{k+1}$
5	– 7	(6,14)	12	28
6	6	(7,13)	14	26
7	– 5	(8,13)	16	26
8	12	(9,12)	18	24
9	7	(10,11)	20	22
10	6	(11,10)	22	20

Introduction to OpenGL

Graphics System



What is OpenGL

- OpenGL is a software interface to graphics hardware.

- Graphics rendering API

- Rendering?-converting geometric or mathematical object descriptions into frame buffer values.
  - high-quality color images composed of geometric and image primitives
  - window system independent
  - operating system independent

- This interface consists of 150 distinct commands that is used to specify the object and operations needed to produce interactive 2D & 3D graphics application.

Graphics Process

Geometric Primitives

Image Primitives

Rendering

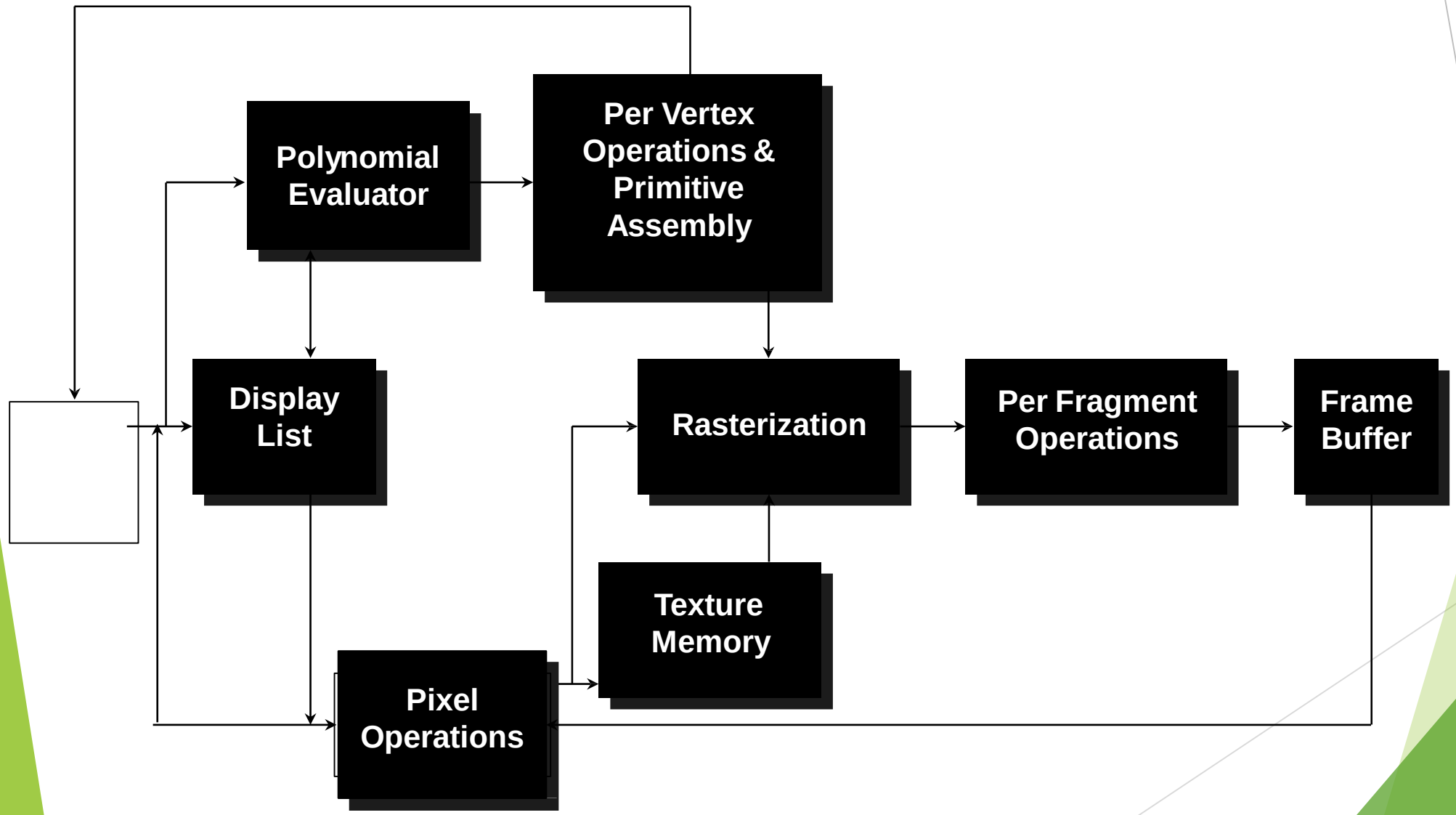
Frame Buffer

A History of OpenGL

- Was SGI's Iris GL – "Open"**GL**
- "Open" standard allowing for wide range hardware platforms
- OpenGL v1.0 (1992)
- OpenGL v1.1 (1995)
- OpenGL v1.4 (latest)
- Governed by OpenGL Architecture Review Board (ARB)

"Mesa" – an Open source (http://www.mesa3d.org)

OpenGL Architecture



Window Management

- OpenGL is window and operating system independent
- OpenGL does *not* include any functions for window management, user interaction, and file I/O
- Host environment is responsible for window management

OpenGL is Not a Language

It is a Graphics Rendering API

Whenever we say that a program is *OpenGL-based* or *OpenGL applications*, we mean that it is written in some programming language (such as C/C++) that makes calls to one or more of OpenGL libraries

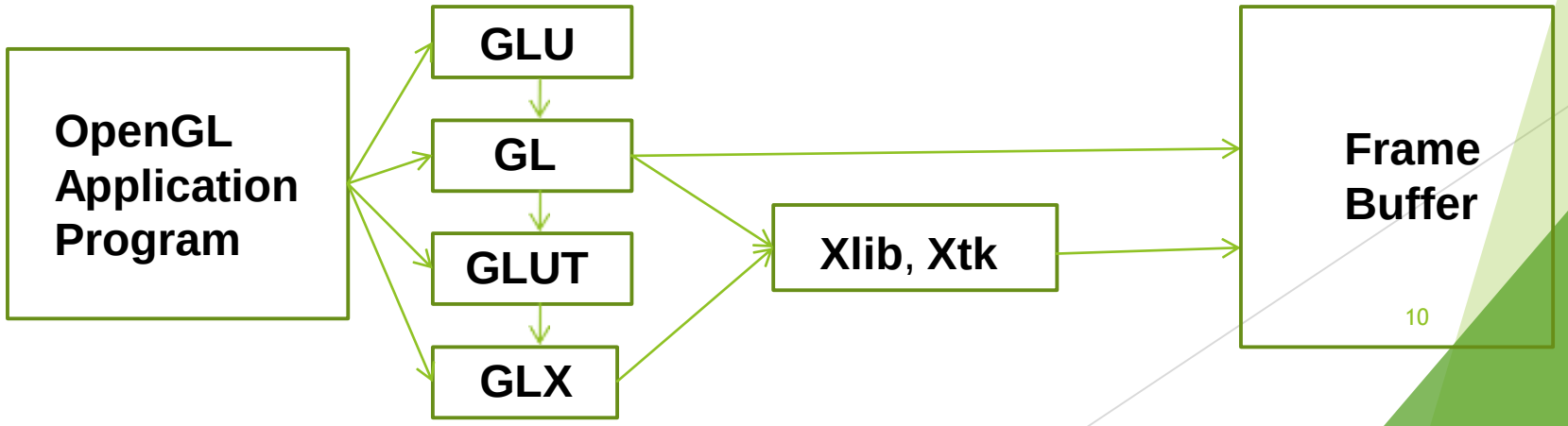
OpenGL Division of Labor

- **GL**
  - "core" library of OpenGL that is platform independent
- **GLU**
  - an auxiliary library that handles a variety of graphics accessory functions
- **GLUT/AUX**
  - utility toolkits that handle window managements

(The OpenGL Interface)

In OpenGL all graphic functions are stored in three Libraries

1. GL (OpenGL in windows)- The functions in this library have names that begin with letters gl and are stored in library GL



(The OpenGL Interface)

In OpenGL all graphic functions are stored in three Libraries

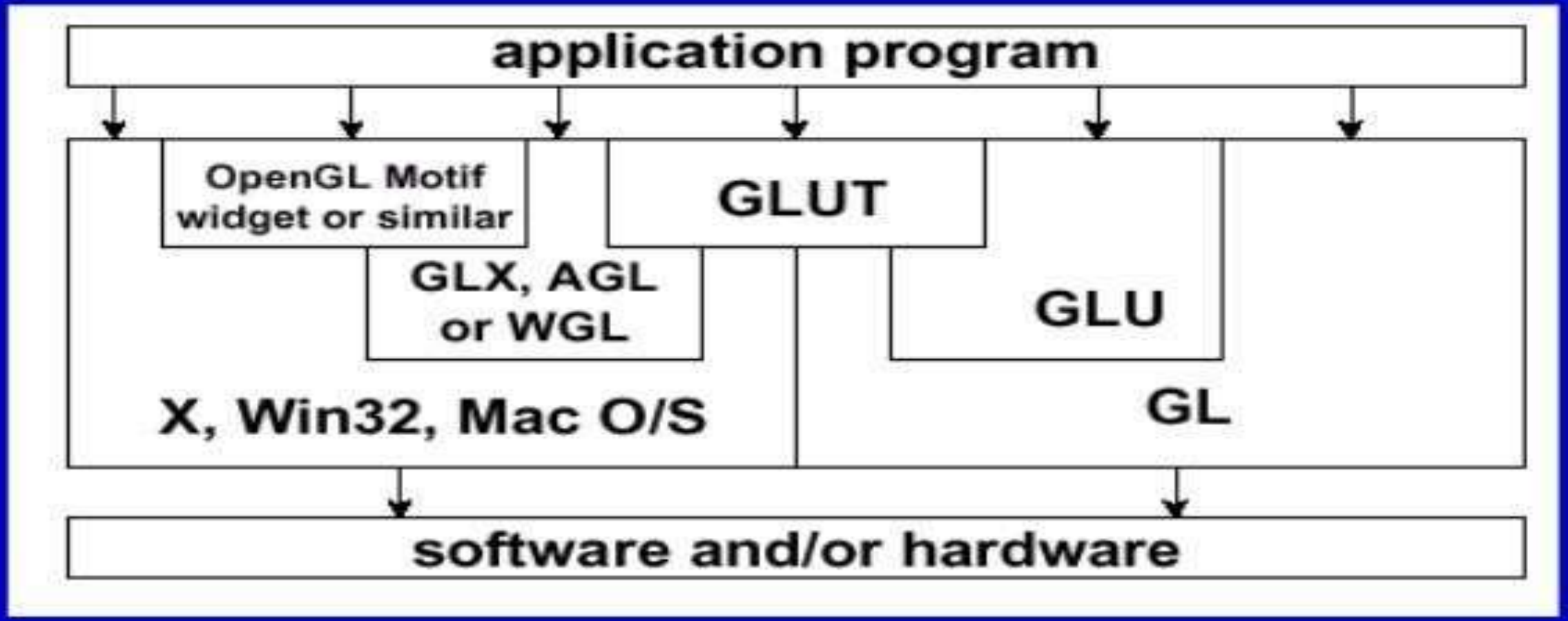
2. **GLU (OpenGL Utility Library)-** This library uses GL functions and contains code for generating objects and simplifying views. Function GLU library begin with “glu". They are used for

    1. Setting up matrices for viewing transformation
    2. Rendering surfaces
    3. performing polygon tessellation.

3. **GLUT(OpenGL Utility Toolkit)-** Used to interface with the window system and to get input from external devices.

- GLX, Xlib and Xtk are used by x-windows

OpenGL API Hierarchy



Libraries and Headers

Library Name	Library File	Header File	Note
OpenGL	opengl32.lib (PC) -lgl (UNIX)	gl.h	"core" library
Auxiliary library	glu32.lib (PC) -lglu	glu.h	handles a variety of accessory functions
Utility toolkits	glut32.lib (PC) -lglut (UNIX) glaux.lib (PC) -lglaux (UNIX)	glut.h glaux.h	window managements

All are presented in the C language

Environment Setup

- All of our discussions will be presented in C/C++ language
- Use GLUT library for window managements
- Files needed

  gl.h, glu.h, glut.h
  opengl32.lib, glu32.lib, glut32.lib
- Go to http://www.opengl.org download files
- Follow the Setup instruction to configure proper path

Usage

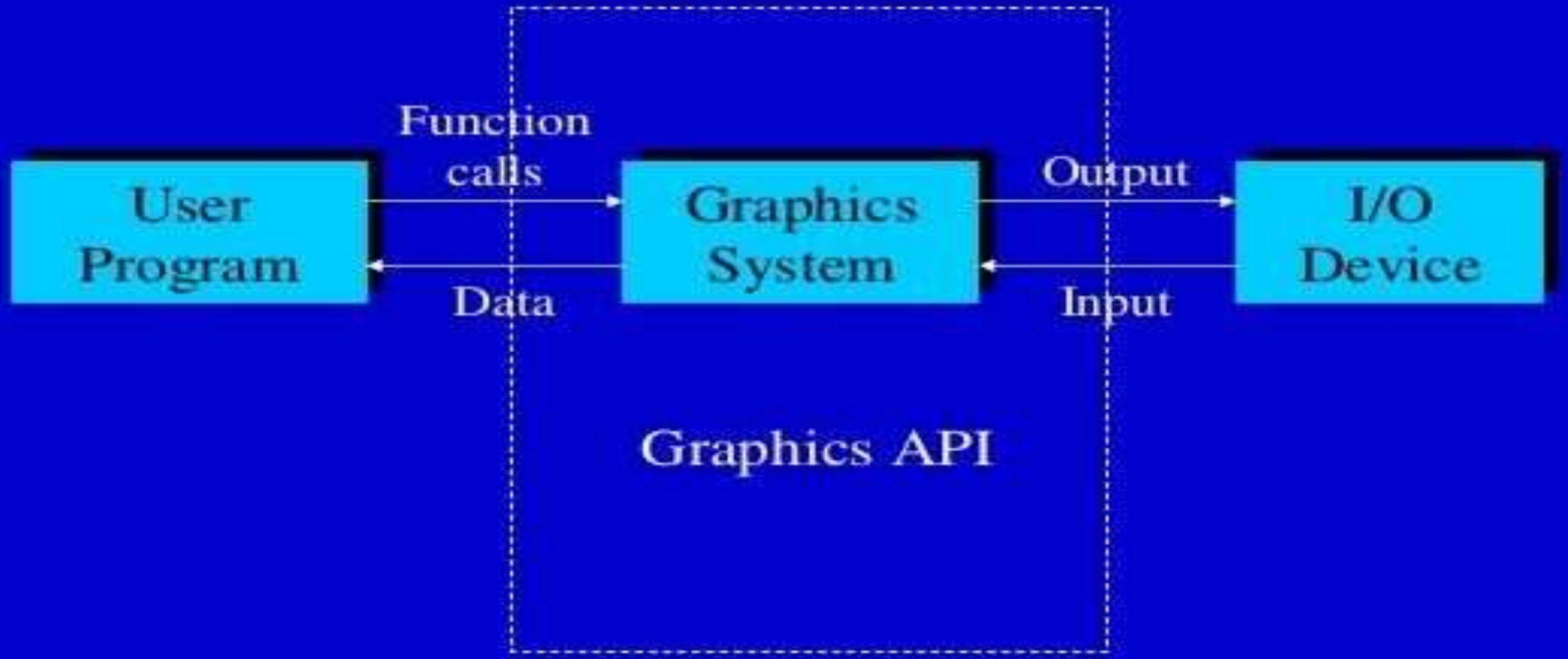
Include the necessary header files in your code

```
#include <GL/gl.h>          // "core", the only thing is required
#include <GL/glu.h>         // handles accessory functions
#include <GL/glut.h>        // handles window managements

void main( int argc, char **argv )
{
      .....
}
```

Only the "core" library (opengl32.lib, gl.h) are required

Graphics System



Graphics Functions

- OpenGL has seven group of functions
  1. Primitive functions
  2. Attribute functions
  3. Viewing functions
  4. Transformation functions
  5. Input functions
  6. Control functions
  7. Query functions

1. **Primitive functions-** Define the low-level object entities such as points, line segments ,polygons, pixels, text and various types of curves and surfaces.

2. **Attribute functions-** Govern the way that a primitive appears on the display (color, pattern, filling, typeface, text styles etc.,).

3. **Viewing functions-** API allows us to clip out objects that are too close or too far away (synthetic camera position and degree of orientation).

4. **Transformation functions-** API should provides the user with a set of transformations functions that allows to carry out transformation of objects such as rotation, scaling and translation.

5. **Input functions-** An API must provide input functions to deal with different input devices (key boards, mouse, light pen, tablets etc.,).

6. **Control functions-** These 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-** A good API must provide information through a set of query functions to write device-independent programs , to use various camera parameters and values in the frame buffer.

Abstractions

GLUT

Windowing toolkit (key, mouse handler, window events)

GLU

- Viewing –perspective/orthographic
- Image scaling, polygon tessellation
- Sphere, cylinders, quadratic surfaces

GL

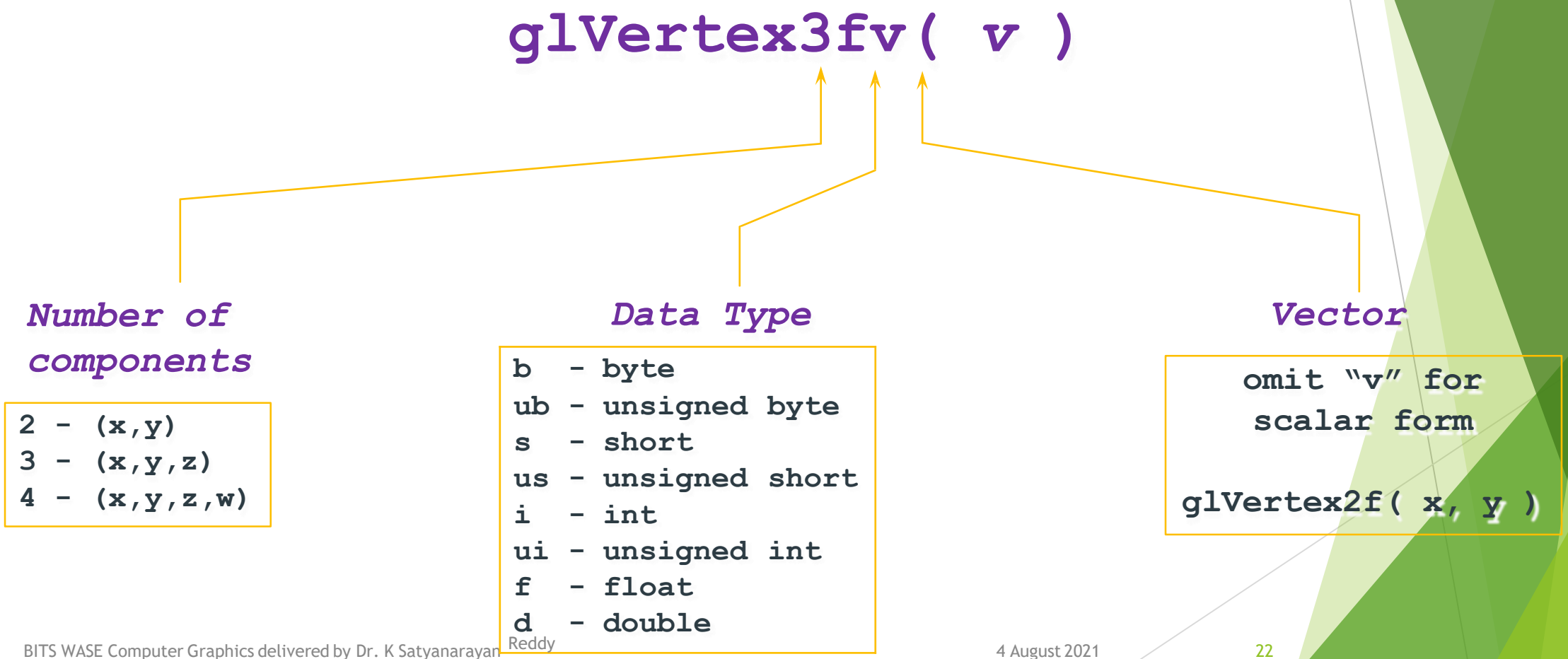
- Primitives - points, line, polygons
- Shading and Colour
- Translation, rotation, scaling
- Viewing, Clipping, Texture
- Hidden surface removal

OpenGL Function Naming

OpenGL functions all follow a naming convention that tells you which library the function is from, and how many and what type of arguments that the function takes

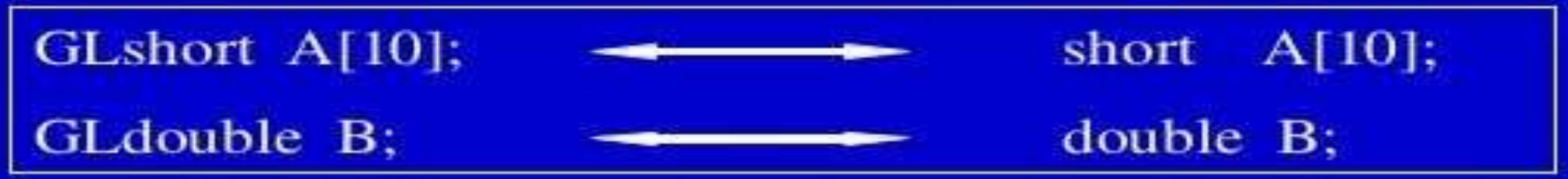
<Library prefix><Root command><Argument count><Argument type>

OpenGL Command Formats



OpenGL Data Types

To make it easier to convert OpenGL code from one platform to another, OpenGL defines its own data types that map to normal C data types



Data types supported in OpenGL

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	Short	GLshort
i	32-bit integer	int or long	GLint
f	32-bit floating-point	Float	GLfloat
d	64-bit floating-point	Double	GLdouble
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, Glbitfield

Control functions

- Interface between the graphics system and operating system (GLUT)
- Interaction with the window system
- Window displays the content of frame buffer
- Position of window are measured in pixels

1. **glutInit**(int **argc*, char ***argv*)
   - initializes GLUT
   - processes any command line arguments.
   - should be called before any other GLUT routine.

   Eg: glutInit(&argc, argv)

Control functions

2. **glutInitDisplayMode**(unsigned int *mode*)
   - specifies whether to use an *RGBA* or color-index color model.
   - specify whether we want a single- or double-buffered window.
   - routine will indicate window is associated with depth, stencil, and/or accumulation buffer.
   - Eg: window with double buffering, the RGBA color model, and a depth buffer, can be

     glutInitDisplayMode(*GLUT_DOUBLE* | *GLUT_RGB* | *GLUT_DEPTH*).
   - Eg: Single buffer with RGB

3. **glutInitDisplayMode** (*GLUT_SINGLE* | *GLUT_RGB*).

- **4.glutInitWindowPosition** (int *x*, int *y*)

- Specifies the screen location for the upper-left corner of window.

Eg: glutInitWindowPosition(0,0);

// Place window top left on display

- **5.glutInitWindowSize** (int *width*, int *size*)

- Specifies the size in pixels of the window.

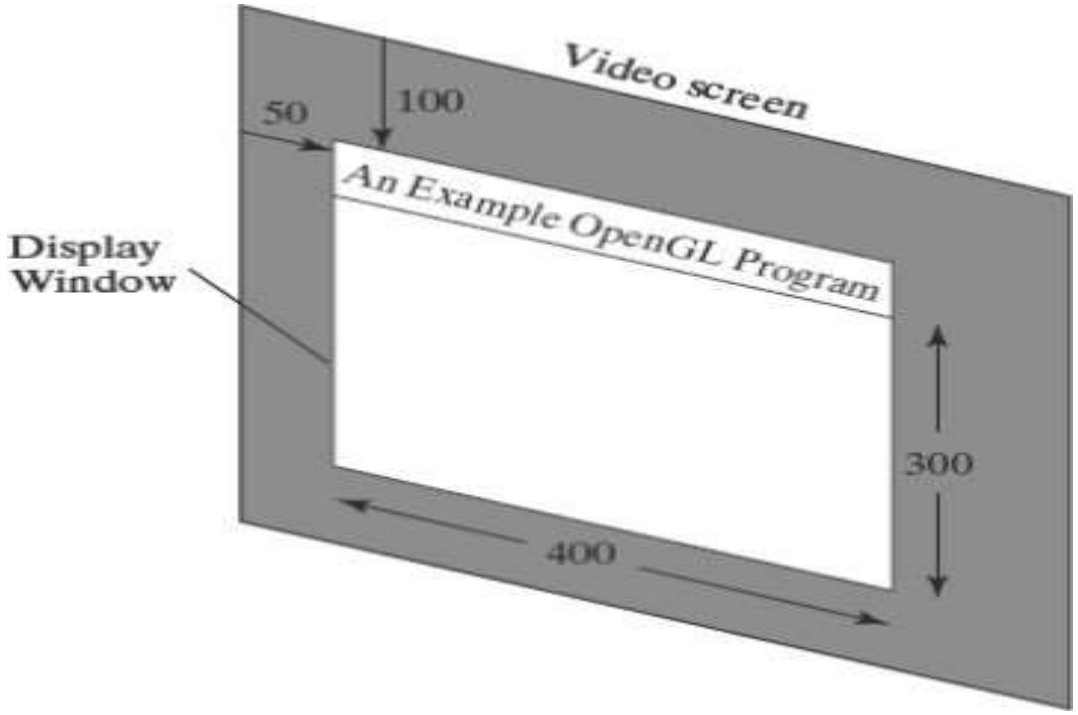
Eg: glutInitWindowSize(500,500); //500x500 window

6. **glutCreateWindow** (char **string*)

- Creates a window with an OpenGL context.
- It returns a unique identifier for the new window.
- Until **glutMainLoop()** is called , the window is not yet displayed.

  Eg: glutCreateWindow("An Example OpenGL Program");

- glutInit (&argv, argc);
- glutInitWindowSize(400,300);
- glutInitWindowPosition(50,100);
- glutCreateWindow(“An Example OpenGL Program”);



7.glutDisplayFunc(void (**func*)(void)) –

It is the first and most important event callback function.

Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc()** is executed. Therefore, we should put all the routines you need to redraw the scene in the display callback function.

8.glutMainLoop(void)- All windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display callback is triggered. Once this loop is entered, it is never exited.

OpenGL API

- Program Structure
  - Step 1: Initialize the interaction between windows and OpenGL.
  - Step 2: Specify the window properties and further create window.
  - Step 3: Set the callback functions
  - Step 4: Initialize the program attributes
  - Step 5: Start to run the program

Program Framework

```
void myDisplay(){
/* clear the display */
glClear(GL_COLOR_BUFFER_BIT);
glFlush();
}
/* End of GasketDisplay */
```

```
void myInit(){
/* set colors */
glClearColor(1.0, 1.0, 1.0, 0.0);
}
/* End of myInit/*
```

Program Framework:

Color Manipulation

- **glClearColor():** establishes what color the window will be cleared to.
- **glClear():** actually clears the window.

- **glColor3f():** establishes what color to use for drawing objects.
- **glFlush():** ensures that the drawing command are actually executed.

Remark: OpenGL is a state machine.
You put it into various states or modes that remain in effect until you change them

OpenGL API

Program Framework

```
#include <GL/glut.h>

int main(int argc, char** argv)
{
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(0,0);
   glutCreateWindow("simple");

   glutDisplayFunc(myDisplay);

   myInit();

   glutMainLoop();
}
```

includes gl.h

interaction initialization

define window properties

display callback

set OpenGL state

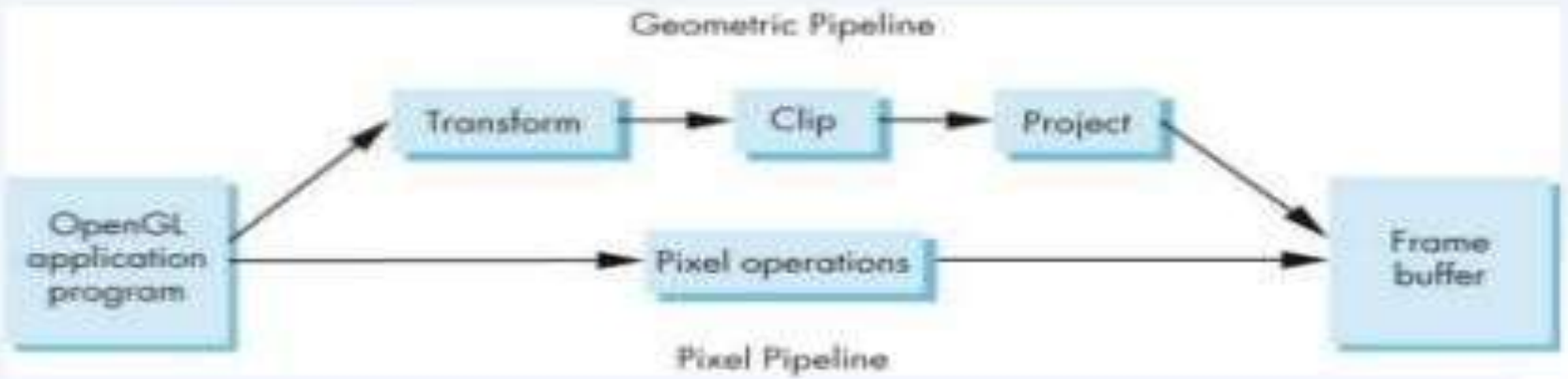
enter event loop

The following is a main program that works for most graphics applications

```
#include <GL/glut.h>
void main(int  *argc, char  **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow(“ Sample  program”);
glutdisplayFunc(display);
myinit();
glutMainLoop();
  }
```

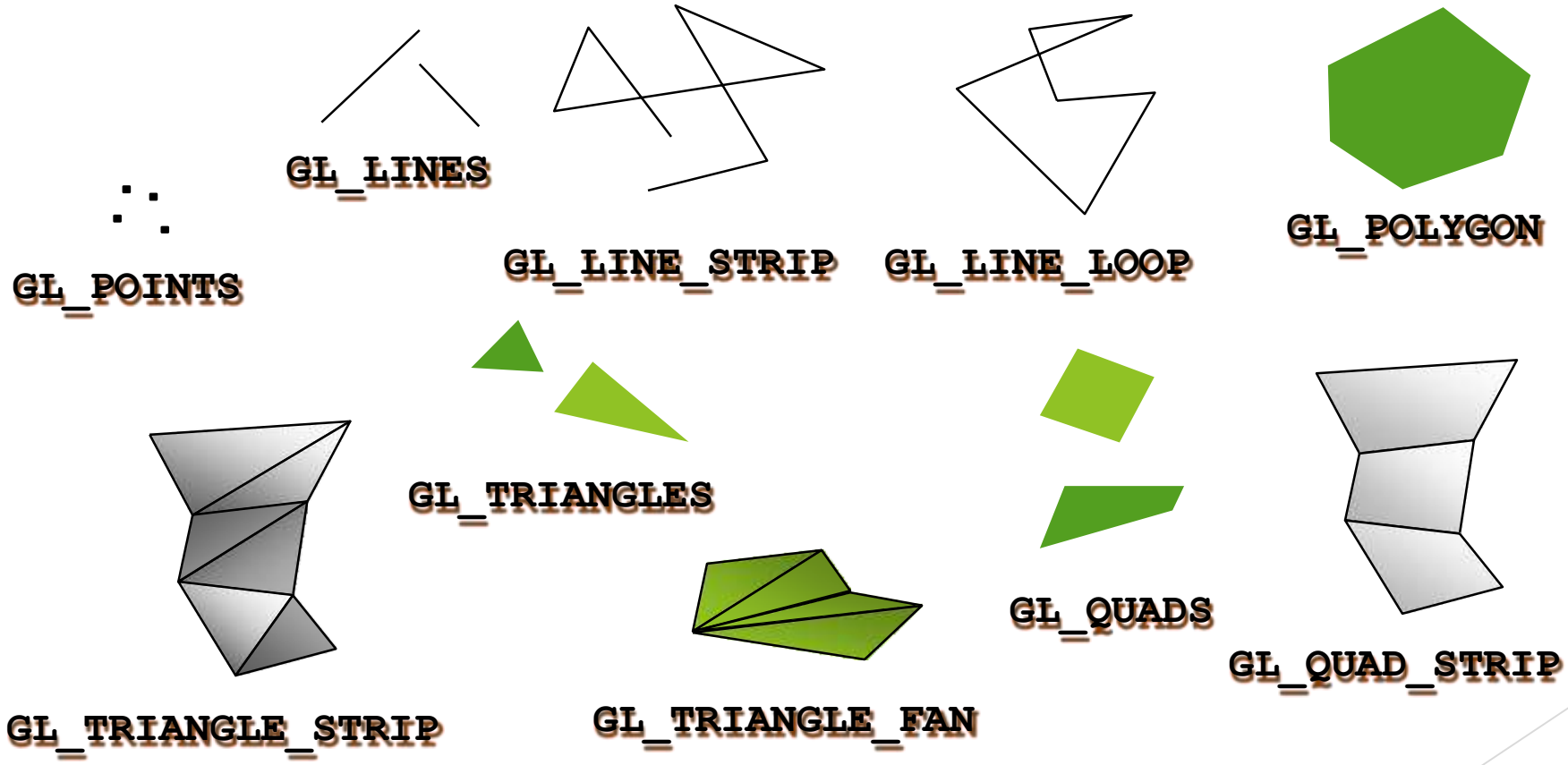
Primitives and Attributes

- Primitive Classes
  - Geometric Primitives
    - They are subject to series of geometric operations.
    - They include points, line segments, curves, etc.
  - Raster Primitives
    - They are lack of geometric properties
    - They may be array of pixels.



OpenGL Geometric Primitives

- All geometric primitives are specified by vertices



GL_POINTS →each vertex is displayed as one pixel

GL_LINES → Takes successive pair of vertices(lines are disconnected)

GL_LINE_STRIP →Successive vertices are connected

GL_LINE_LOOP→ polyline are closed.

POLYGON

Polygon is an object that has

1. Border that can be describe by line loop.

2. It has a well defined interiors.

3 properties of a polygon

Simple: If no two edges of a polygon cross each other it's a simple polygon.

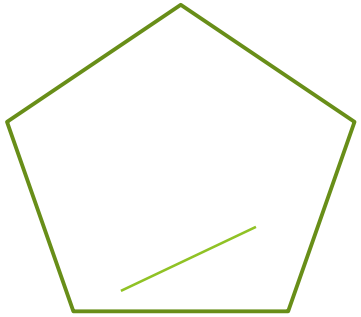
- simple

- not simple

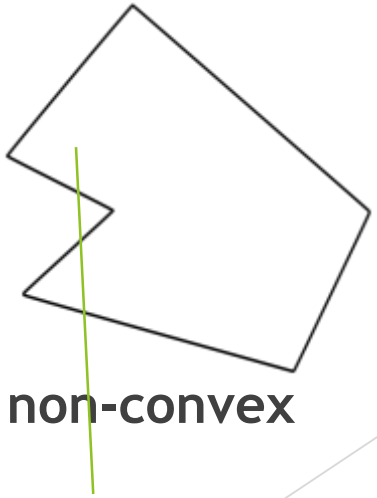
Flat : Any 3 non-colinear determines a plane where that triangle lies.

Convex:

An object is convex, a line segment between two points on the boundary never goes outside the polygon.



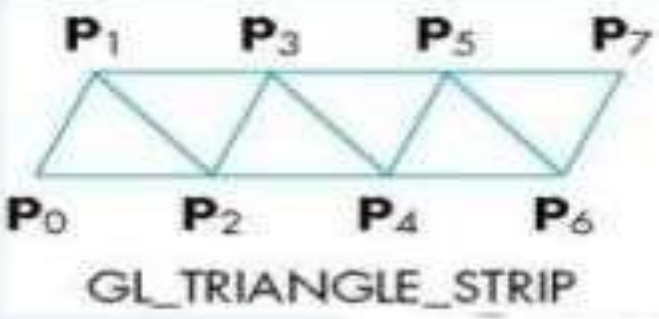
convex



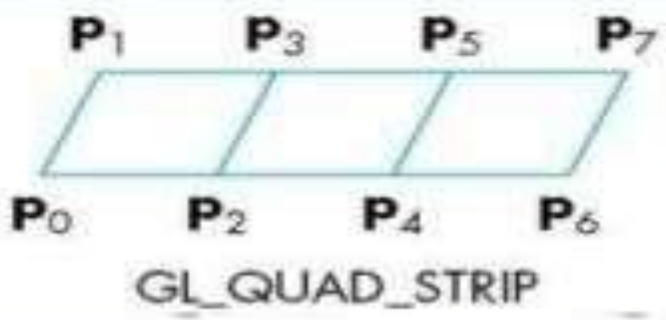
non-convex

Primitives and Attributes

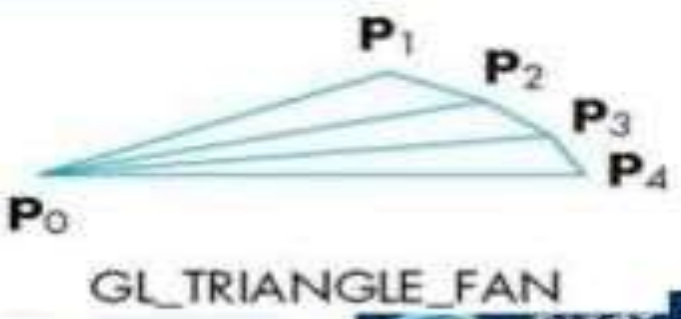
- Polygon Primitives
  - Polygons: GL_POLYGON
  - Triangles: GL_TRIANGLES
  - Quadrilaterals: GL_QUADS
  - Stripes: GL_TRIANGLE_STRIP
  - Fans: GL_TRIANGLE_FAN



GL_TRIANGLE_STRIP



GL_QUAD_STRIP



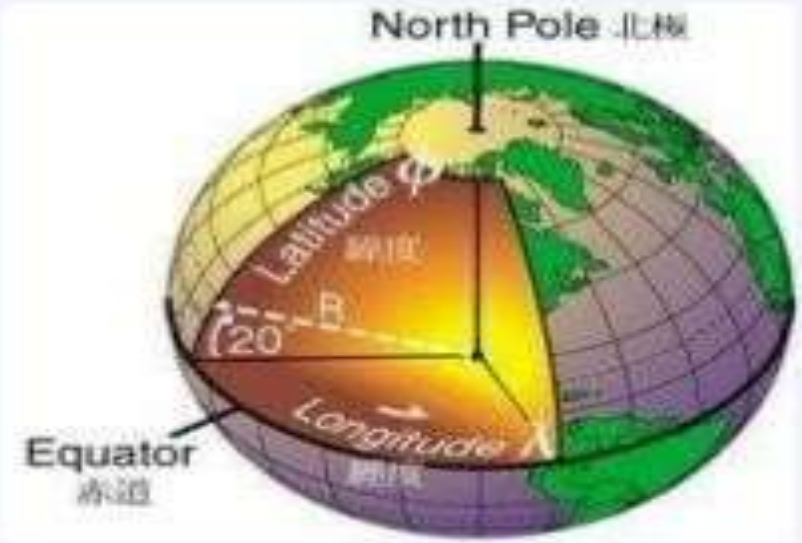
GL_TRIANGLE_FAN

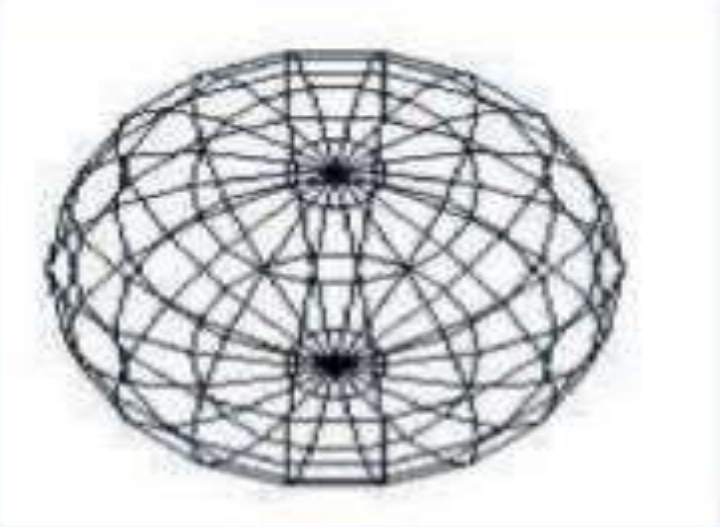
Primitives and Attributes

- Attributes
  - An attribute is any property that determines how a geometric primitive is to be rendered.
  - Each geometric primitive has a set of attributes.
    - Point: Color
    - Line Segments: Color, Thickness, and Pattern
    - Polygon: Pattern

Primitives and Attributes

- Example: Sphere Approximation
  - A set of polygons are used to construct an approximation to a sphere.
    - Longitude
    - Latitude





Specifying Geometric Primitives

- Primitives are specified using

  glBegin(*primType*);

  glEnd();

  - *primType* determines how vertices are combined

```
GLfloat red, green, blue;
Glfloat coords[3];
glBegin( primType );
for ( i = 0; i < nVerts; ++i ) {
  glColor3f( red, green, blue );
  glVertex3fv( coords );
}
glEnd();
```

```
glBegin(GL_LINES);    // Specify what to draw,
                      // here lines

// Geometric info via vertices:
   glVertex*(); // 1
   glVertex*(); // 2
   ...          // ...
glEnd();

glVertex[234][isfd]

[234]: 2D, 3D, 4D
[isfd]: integer, short, float, double
```

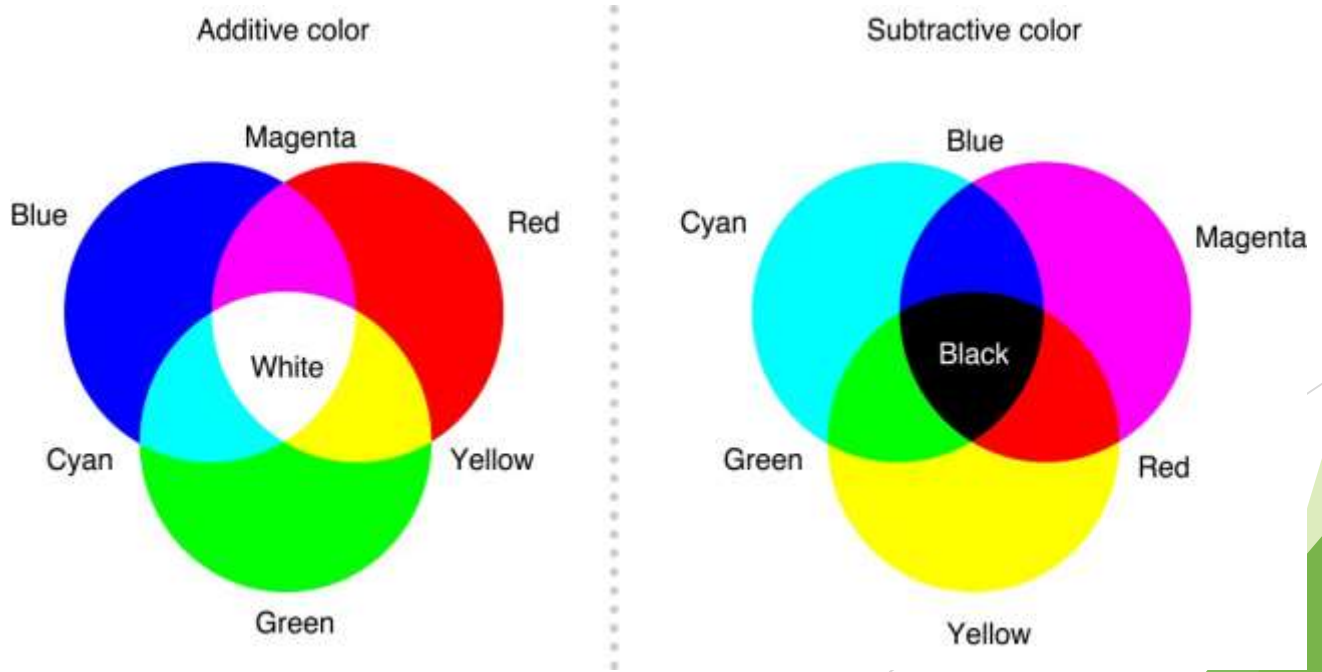
For instance: glVertex2i(100, 25);

Simple Example

```
void drawRhombus( GLfloat color[] )
{
 glBegin( GL_QUADS );
 glColor3f(1.0,0.0,0.0);
 glVertex2f( 0.0, 0.0 );
glVertex2f( 1.0, 0.0 );
 glVertex2f( 1.5, 1.118 );
 glVertex2f( 0.5, 1.118 );
 glEnd();
}
```

COLOR

- **Color attribute:**
  - **Additive color-** Primary colors (RGB)are added to give the perceived color.
  - **Subtractive color (CMY)-** colored pigments remove the color components from light that is striking the white surface (printing and painting).



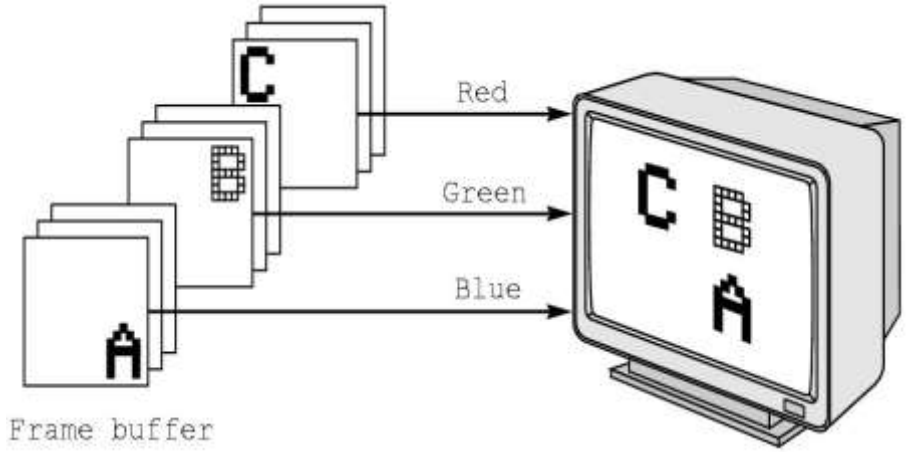
- Color in Graphics system is handled through the API.

- There are 2 approaches
  - 1. RGB Color Model

  - 2.Indexed Color Model → easier to support in hardware because of its lower memory requirement and limited colors available on display.

RGB color (Tristimulus)

- Each color component is stored separately in the frame buffer.
- Usually 8 bits per component in buffer (16million colors).
- In `glColor3f` the color values range from 0.0 (none) to 1.0 (all), whereas in `glColor3ub` the values range from 0 to 255



- Eg: glColor3f(0.0,0.0,0.0)----black
  glColor3f(1.0,1.0,1.0)-----white
  glColor3f(1.0,0.0,0.0)------Red

glClearColor(1.0,1.0,1.0,1.0)- (to clear the window before drawing a new frame)

glClearColor(GL_COLOR_BUFFER_BIT);

GL_COLOR_BUFFER_BIT Indicates the buffers currently enabled for color writing.

Four Color system(RGBA), A-Alpha [Opacity (Opaque-No light passes through) or Transparency value]

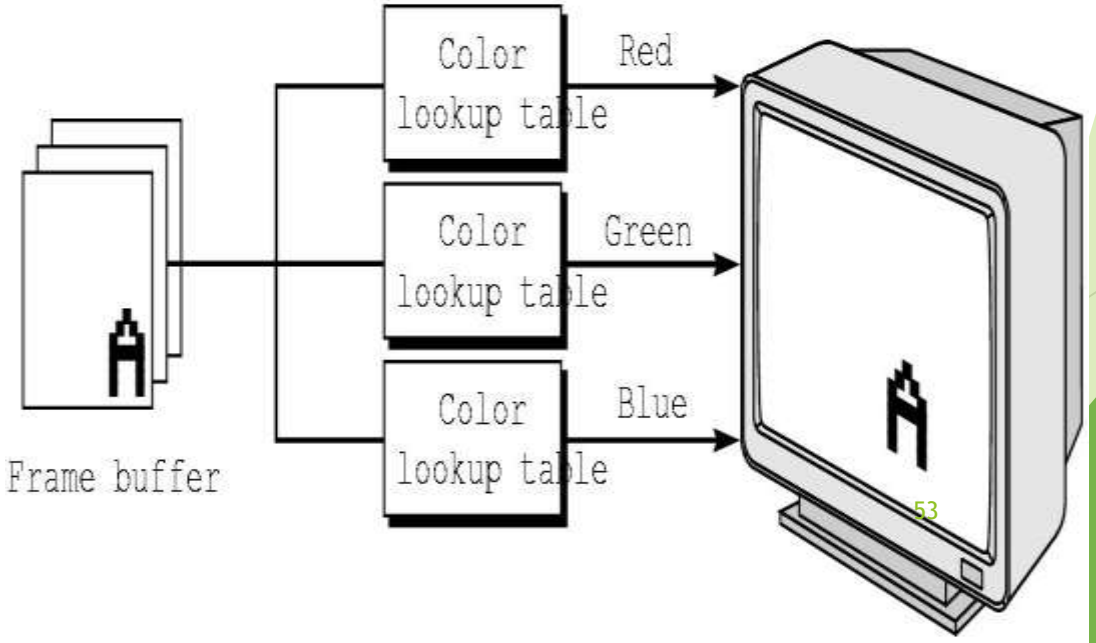
- A=0 .0(Transparent), A=1.0 (Opaque)
- Alpha value will be considered only if Blending is enabled. By default blending is disabled → overwrites any existing color.

Indexed color

- Each pixel has 8-bits.
- Divide each pixel's 8-bit into smaller groups and assign RGB to each.
- Not flexible with color assignment.
- Provides wide range of colors.
- Depth pixel are used as index to color lookup table.

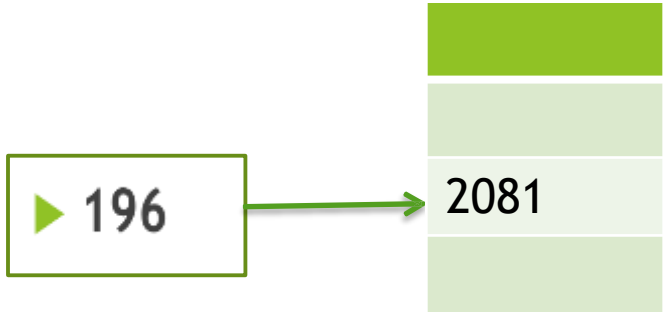
Indexed color

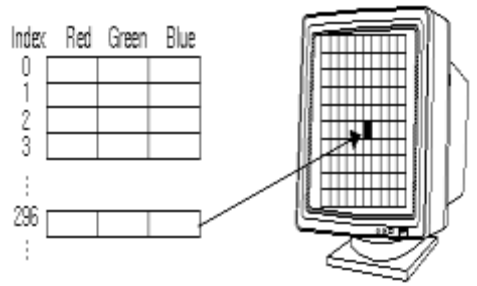
- Color is selected from look-up table
- For example : if Frame buffer has K bits/pixel, each pixel value or index, is an integer between 0—2^k -1
- If precession is m bits, can select 2^m red, 2^m blue, 2^m green, which produces 2^{3m} colors on the display, but frame buffer can specify on 2^K colors.



Eg: glIndexi(element)- assign present color to element, this index value is stored in the frame buffer for subsequent operation.

glIndexi(196);





red gun green gun blue gun

00000000	00001000	00100001

GLUT allows to set the entries in the color table for each window by using

glutSetColor(int color, Glfloat red, Glfloat green, Glfloat blue);

- glClearIndex(index) specified index is cleared.

- Advantages

The Color index mode requires less memory for the frame buffer and hardware components.

- Disadvantages

Interaction with window system is more complex than RGB color.

Colors: RGBA vs. Color-Index

Color mode

RGBA mode

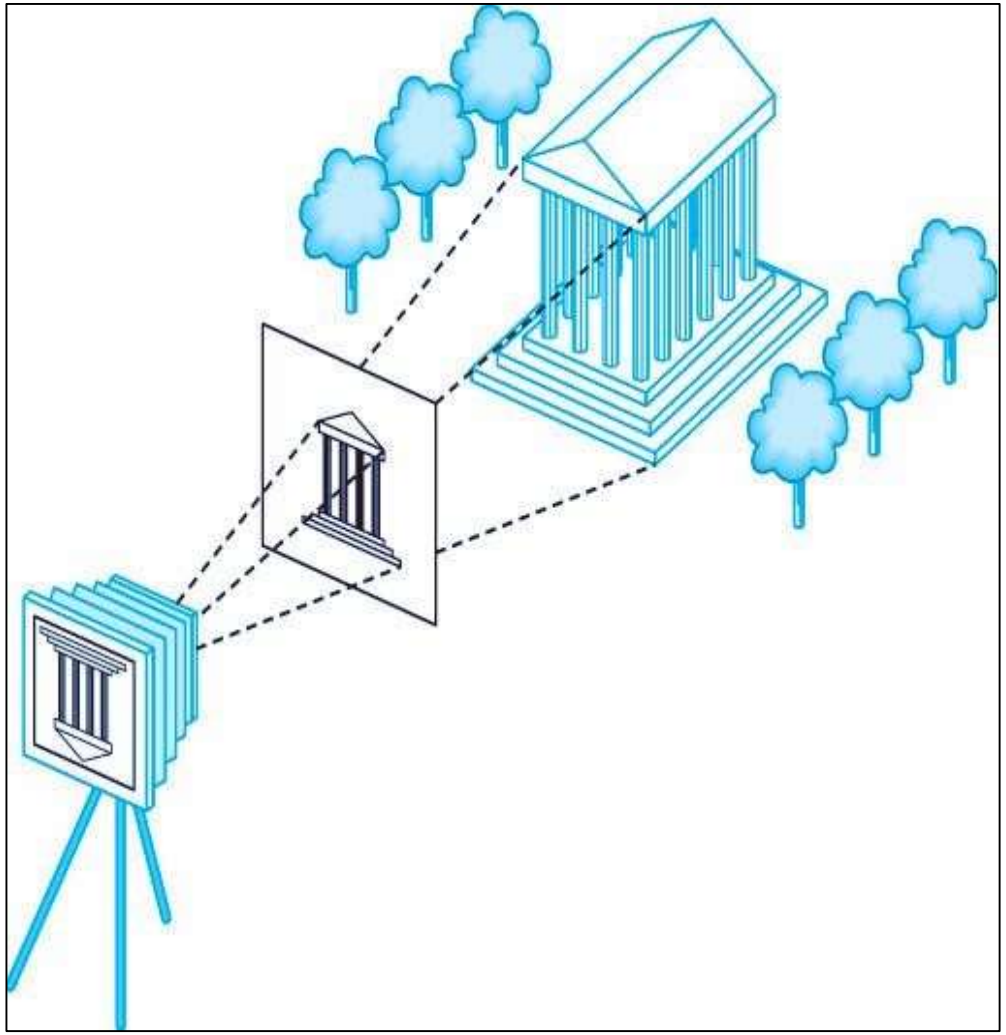
Color-Index Mode

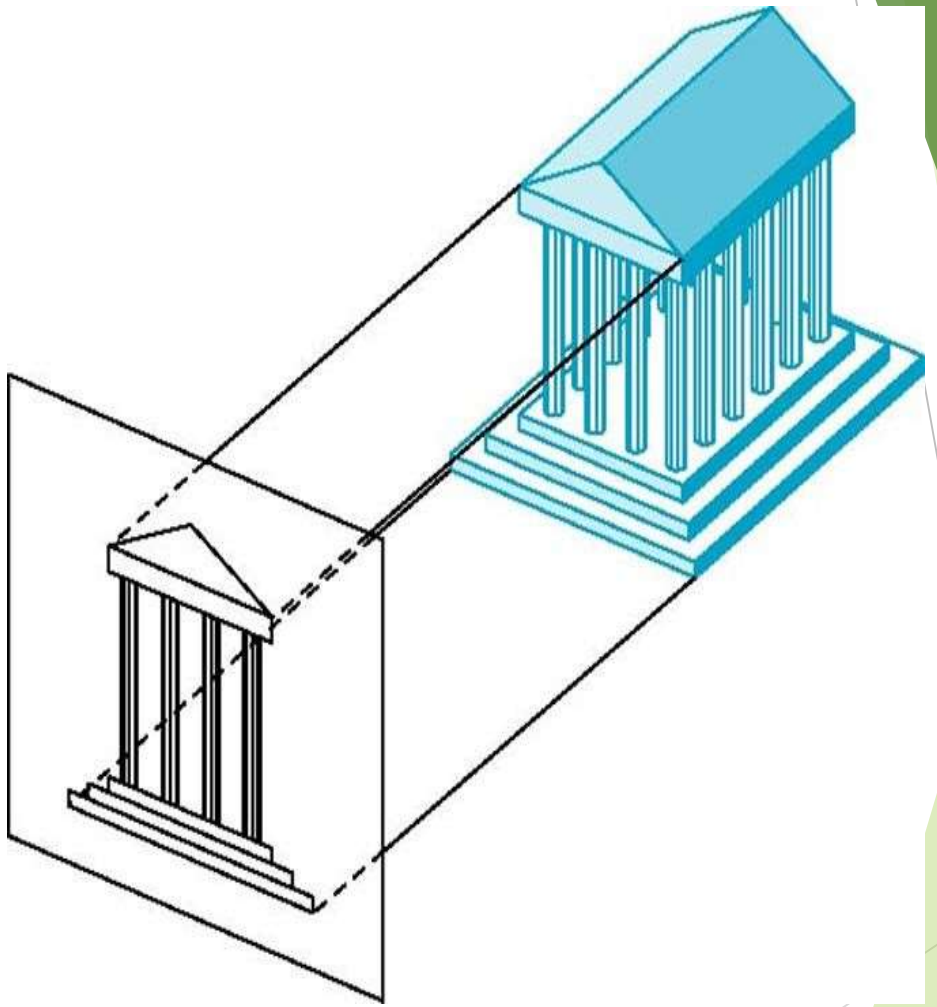
Viewing

Viewing

- Describes how objects appear on the display.
- Camera forms an image by exposing film, but computer forms an image by carrying out a sequence of operations in its pipeline.
- OpenGL default view is the **orthographic projection** (Image plane is fixed and moving camera far from this plane)

ORTHOGRAPHIC Projection

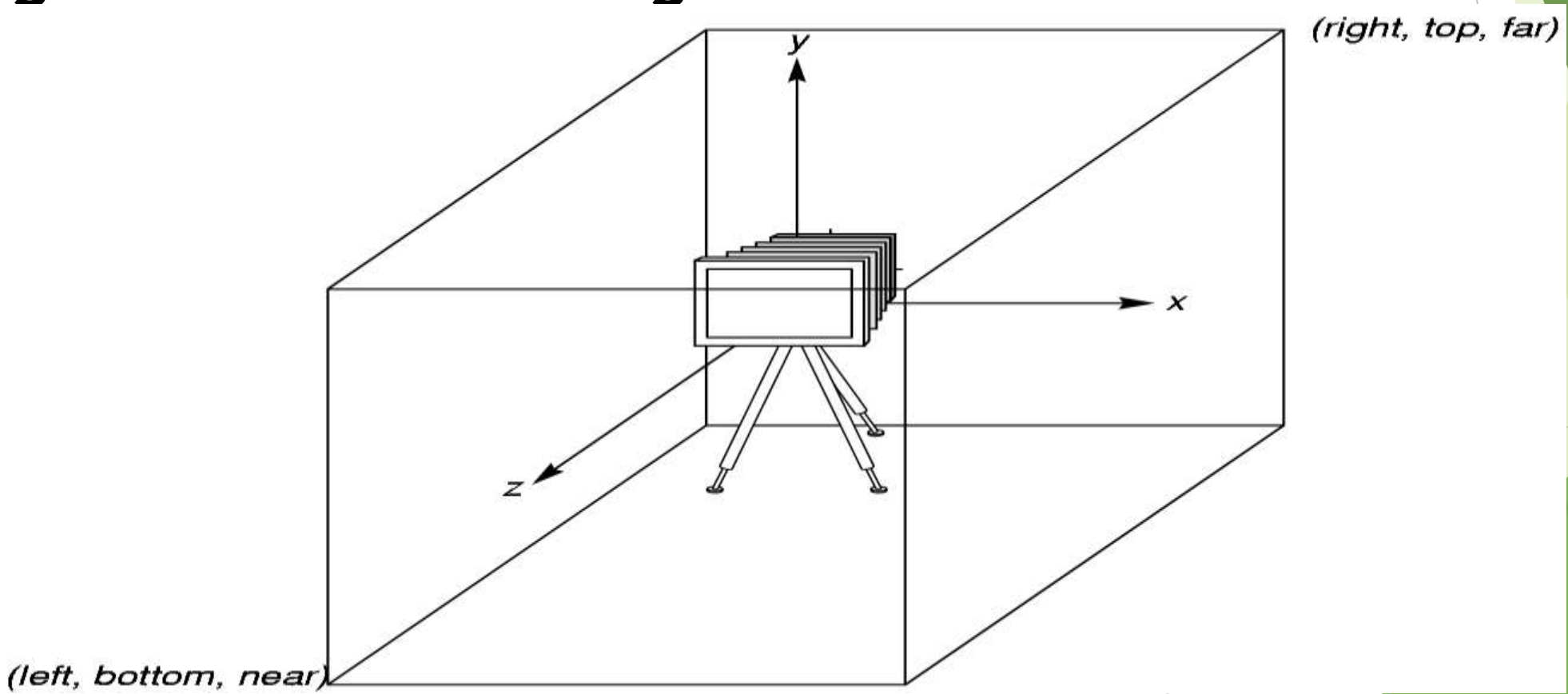


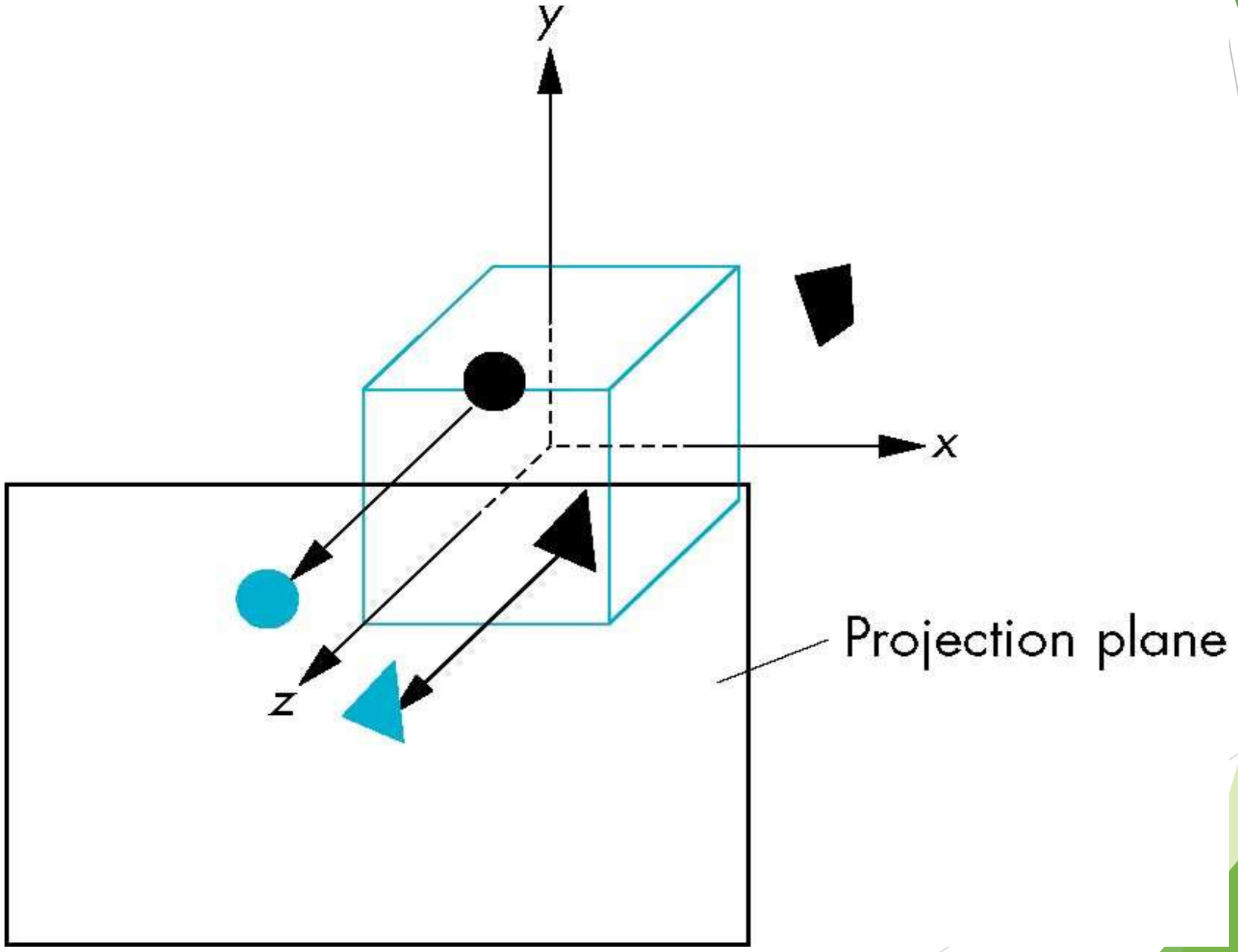


OpenGL Camera

- OpenGL places a camera at the origin in object space pointing in the negative z direction.

The default viewing volume is a box centred at the origin with sides of length 2



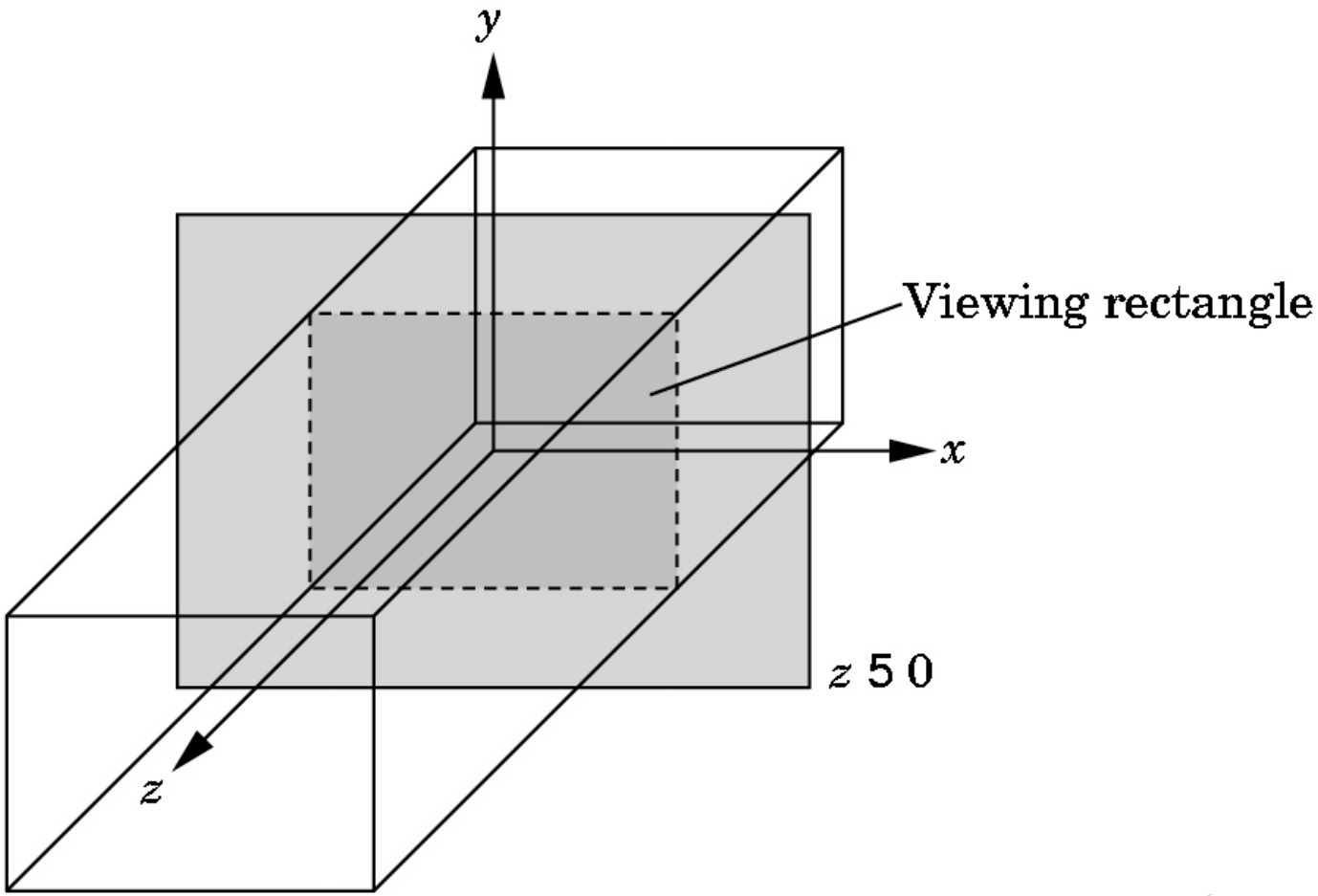

y
x
z
Projection plane

```
Void glOrtho(GLdouble left, Gldouble right,
Gldouble bottom, Gldouble top, Gldouble near,
Gldouble far)
```

- **All the parameters are distance measured from the camera .**
- **Orthographic projection sees only those objects in the volume specified by the view volume.**

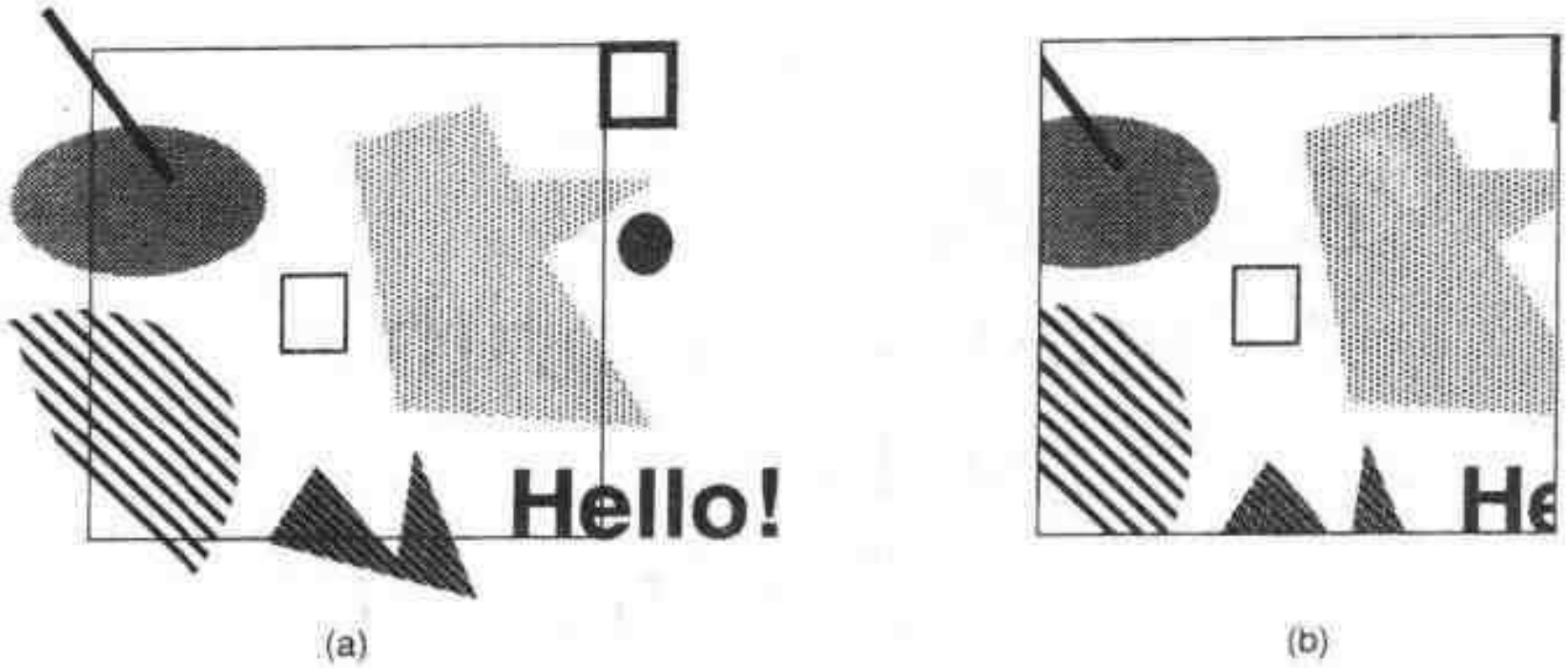
- **OpenGL uses default view volume 2x2x2 cube with the origin in the center. (left, bottom, near)=(-1,-1,-1), (right, top, far)=(+1,+1,+1).**

gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)



Orthographic Viewing

- In the default orthographic view, points are projected forward along the z axis onto the plane z=0



Matrix modes

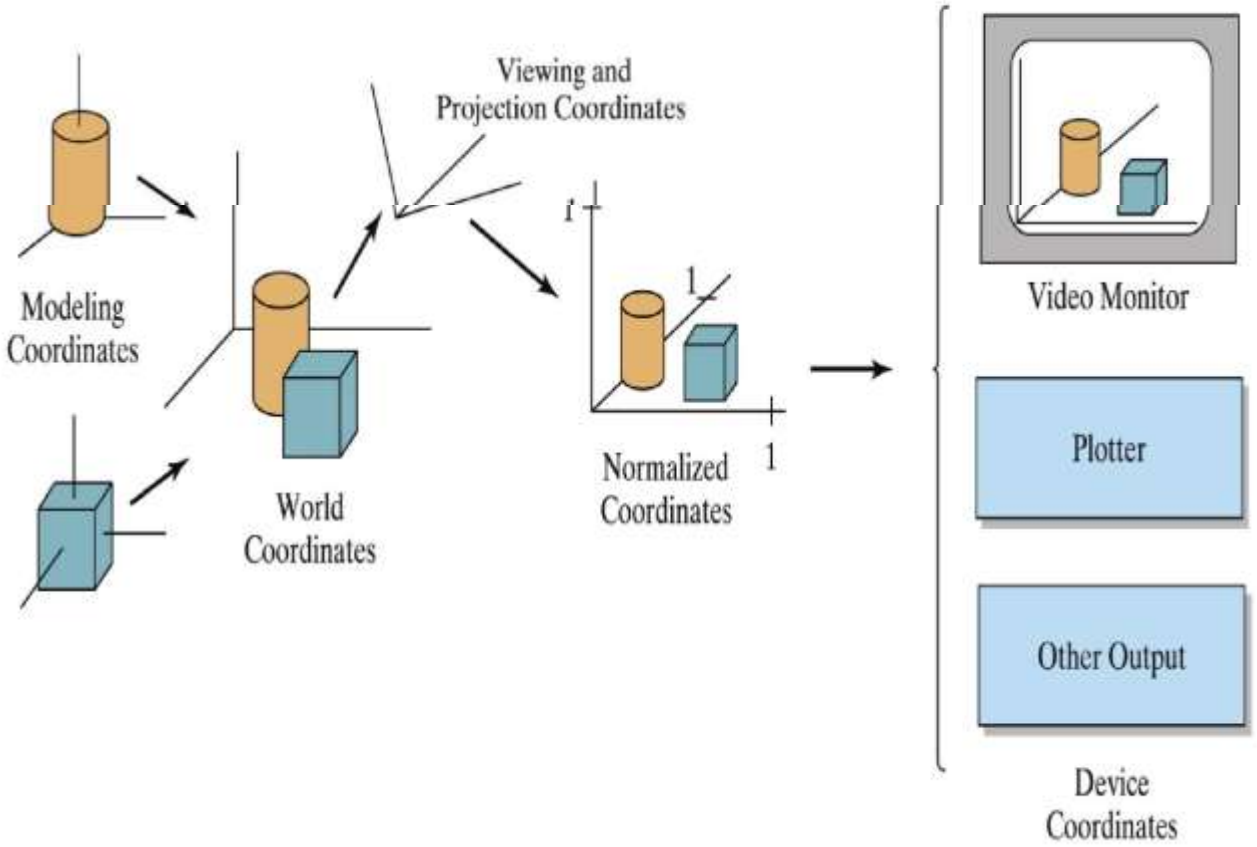
- OpenGL pipeline architecture depend on multiplying or concatenating a number of transformation matrices to achieve the desired image or primitive.
- The values of these matrices are part of the state of the system
- In OpenGL **Model-view** and **Projection** are two important matrices used.
- Initially these matrices are identity matrices.
- The set commands used for two-dimensional viewing are

```
glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
  gluOrtho2D(0.0,50.0,0.0,50.0);
    glMatrixMode(GL_MODELVIEW);
```

- This sequence defines 50.0x50.0 viewing rectangle with the lower left corner of the rectangle at the origin of the 2-D system

- It then switches the matrix mode back to model-view mode

OpenGL Geometric Transformations
glMatrixMode(GL_MODELVIEW);



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 known as coordinates.

✓If coordinate values for a picture are given in some other reference frame (spherical, hyperbolic, etc.), they must be converted to Cartesian coordinates.

✓Several different Cartesian reference frames are used in the process of constructing and displaying.

✓First we define the shapes of individual objects, such as trees or furniture, These reference frames are called modeling coordinates or local coordinates.

✓

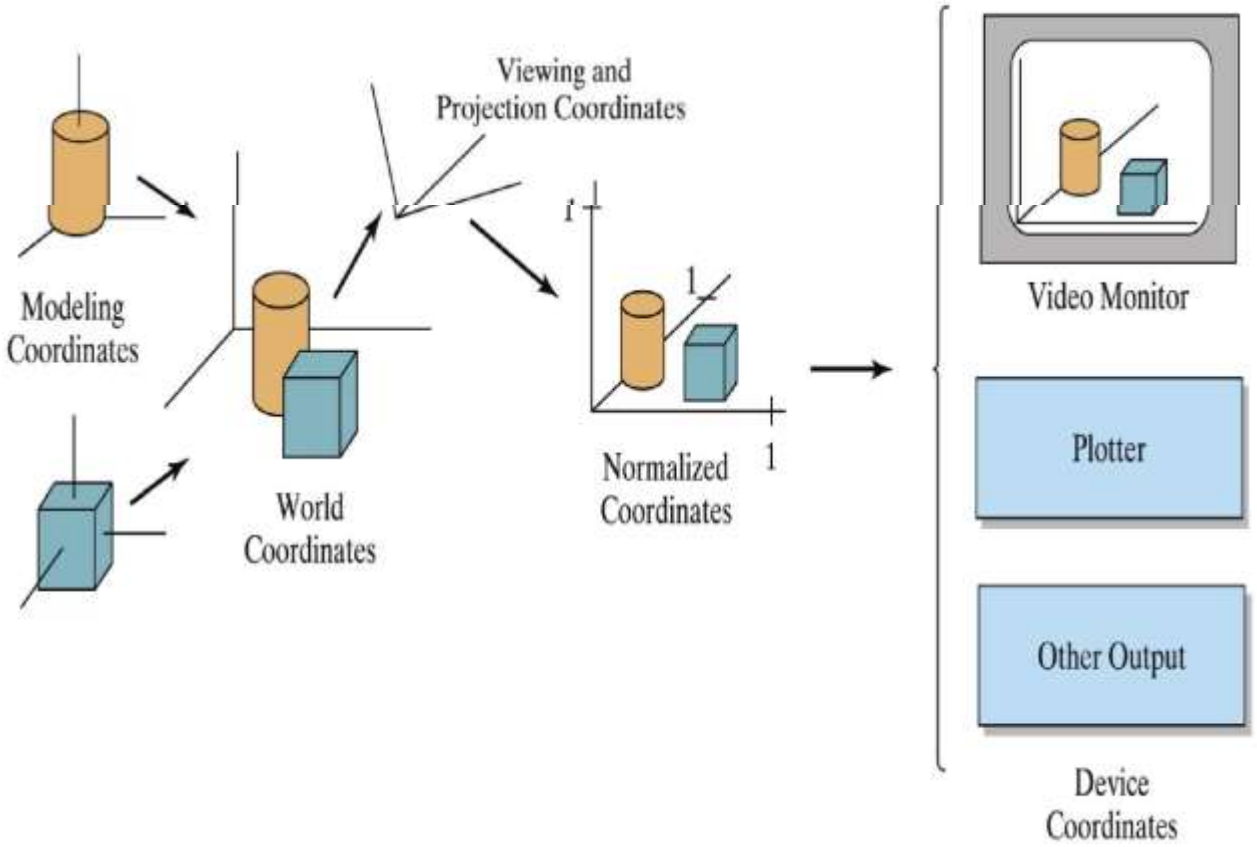
✓Then we place the objects into appropriate locations within a scene reference frame called world coordinates.

✓After all parts of a scene have been specified, it is processed through various output- device reference frames for display. This process is called the viewing pipeline.

✓ The scene is then stored in normalized coordinates. Which range from −1 to 1 or from 0 to 1 Normalized coordinates are also referred to as normalized device coordinates.

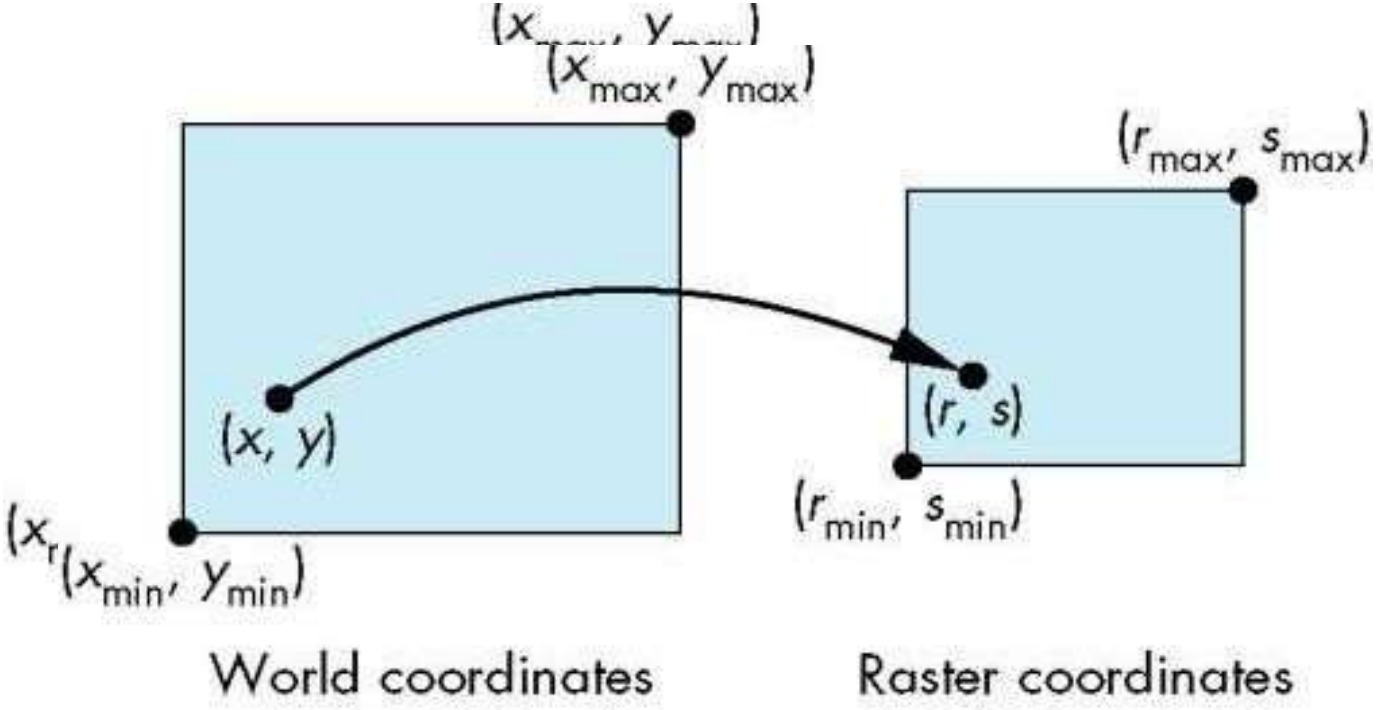
✓The coordinate systems for display devices are generally called device coordinates, or screen coordinates.

Figure briefly illustrates the sequence of coordinate transformations from modeling coordinates to device coordinates for a display



Coordinate System

- It is difficult to specify the vertices in units of the physical device.
- Device-independent graphics makes users easy to define their own coordinate system.



Rendering Process

Coordinate Systems

The units in `points` are determined by the application and are called

– ***object*** **(or** ***model*****) coordinates** **model view transform**

– ***world*** **coordinates**

Viewing specifications usually are also in object coordinates transformed through

-- *eye* (or *camera*) coordinates

--*clip* coordinates **projection transform**

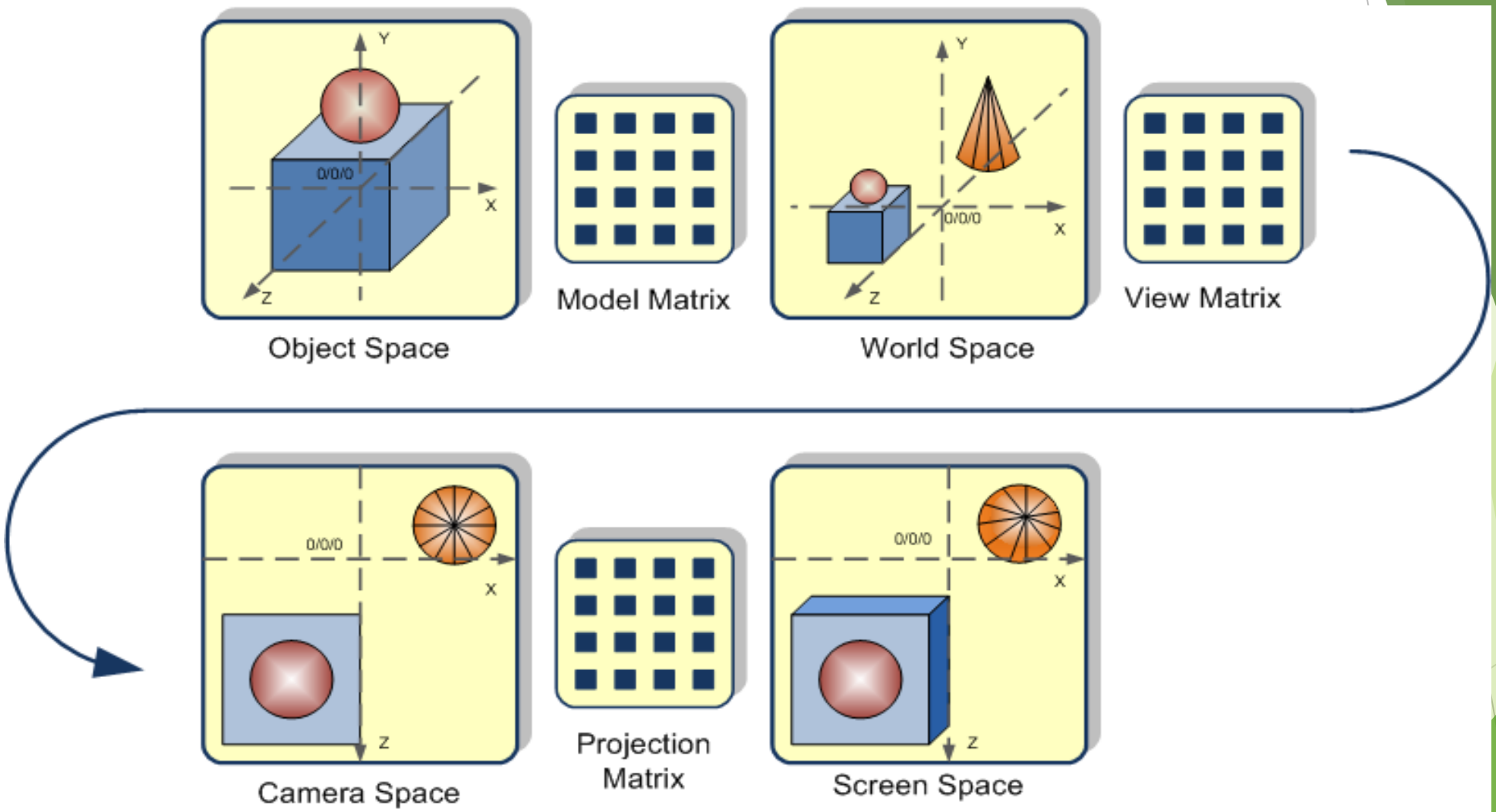
--normalized *device* coordinates

--*window* (or *screen*) coordinates

•OpenGL also uses some internal representations that usually are not visible to the application but are important in the shades

Coordinate Systems and Transformations

- Steps in Forming an Image
  - specify geometry (world coordinates)
  - specify camera (camera coordinates)
  - project (window coordinates)
  - map to viewport (screen coordinates)
- Each step uses transformations
- Every transformation is equivalent to a change in coordinate systems (frames)


Y
0/0/0
X
Z
Object Space
Model Matrix
Y
0/0/0
X
Z
World Space
View Matrix
0/0/0
X
Z
Camera Space
Projection
Matrix
0/0/0
X
Z
Screen Space

Camera Analogy and Transformations

- **Viewing transformations**
  - tripod–define position and orientation of the viewing volume in the world.
- **Modeling transformations**
  - moving the model.
- **Projection transformations**
  - adjust the lens of the camera.
- **Viewport transformations**
  - enlarge or reduce the physical photograph.

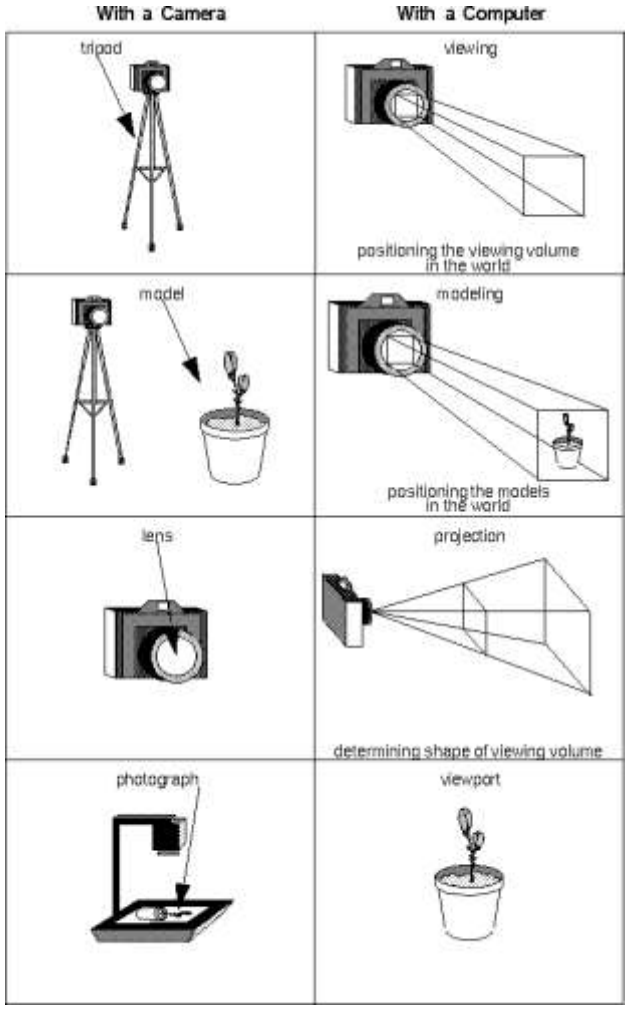
Camera Analogy and Transformations

Positioning the Camera

Positioning the Model

Choose a camera lens and adjust zoom

Mapping to screen



Viewing Transformation

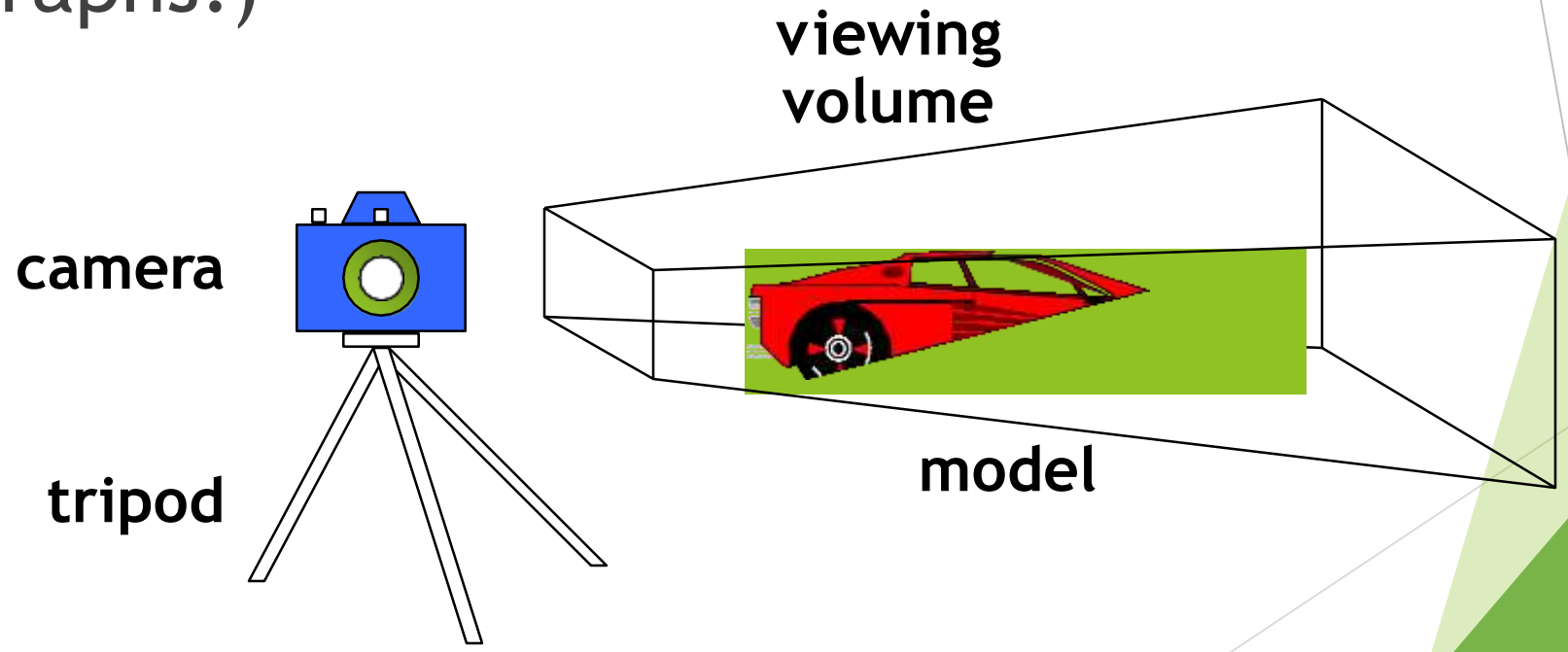
Modeling Transformation

Projection Transformation

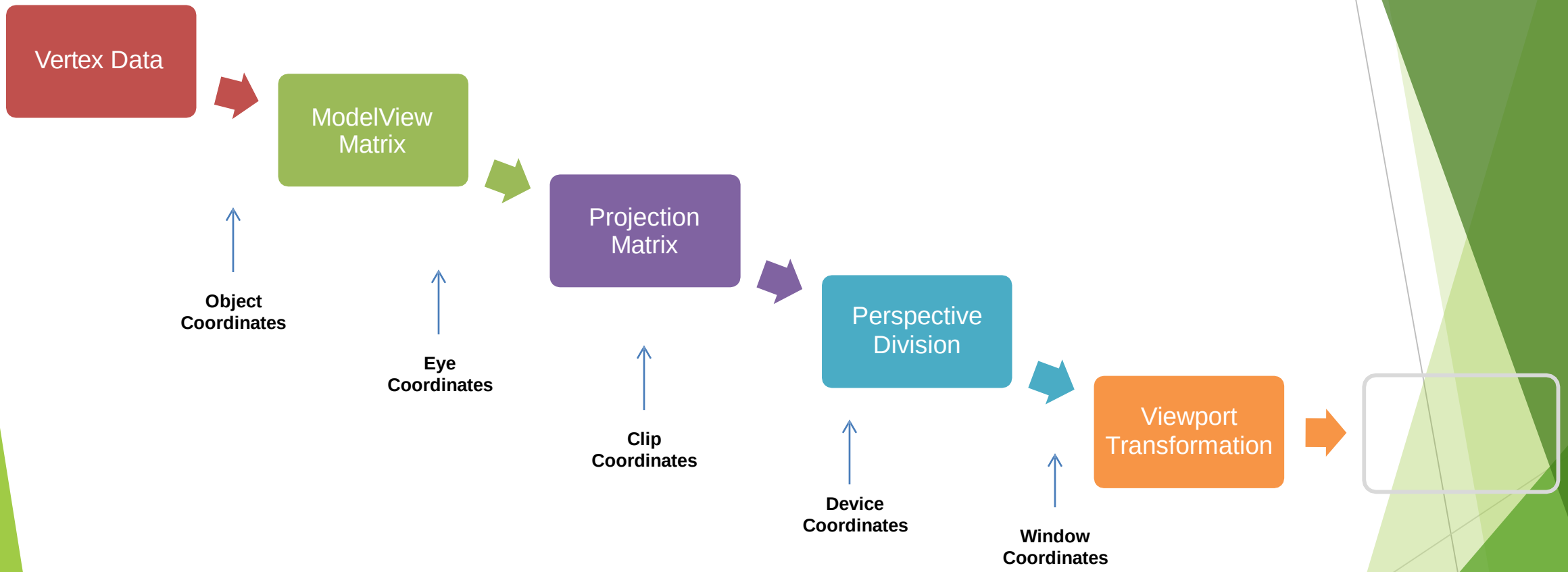
Viewport Transformation

Camera Analogy

- 3D is just like taking a photograph (lots of photographs!)



OpenGL Transformations



View Ports

- The window manager, not OpenGL, is responsible for opening window on the screen.

- By default the viewport is set to the entire pixel rectangle of the window that is opened.

- Do not have use the entire window for the image:

- We use the **glViewport()** command to choose a smaller drawing region.

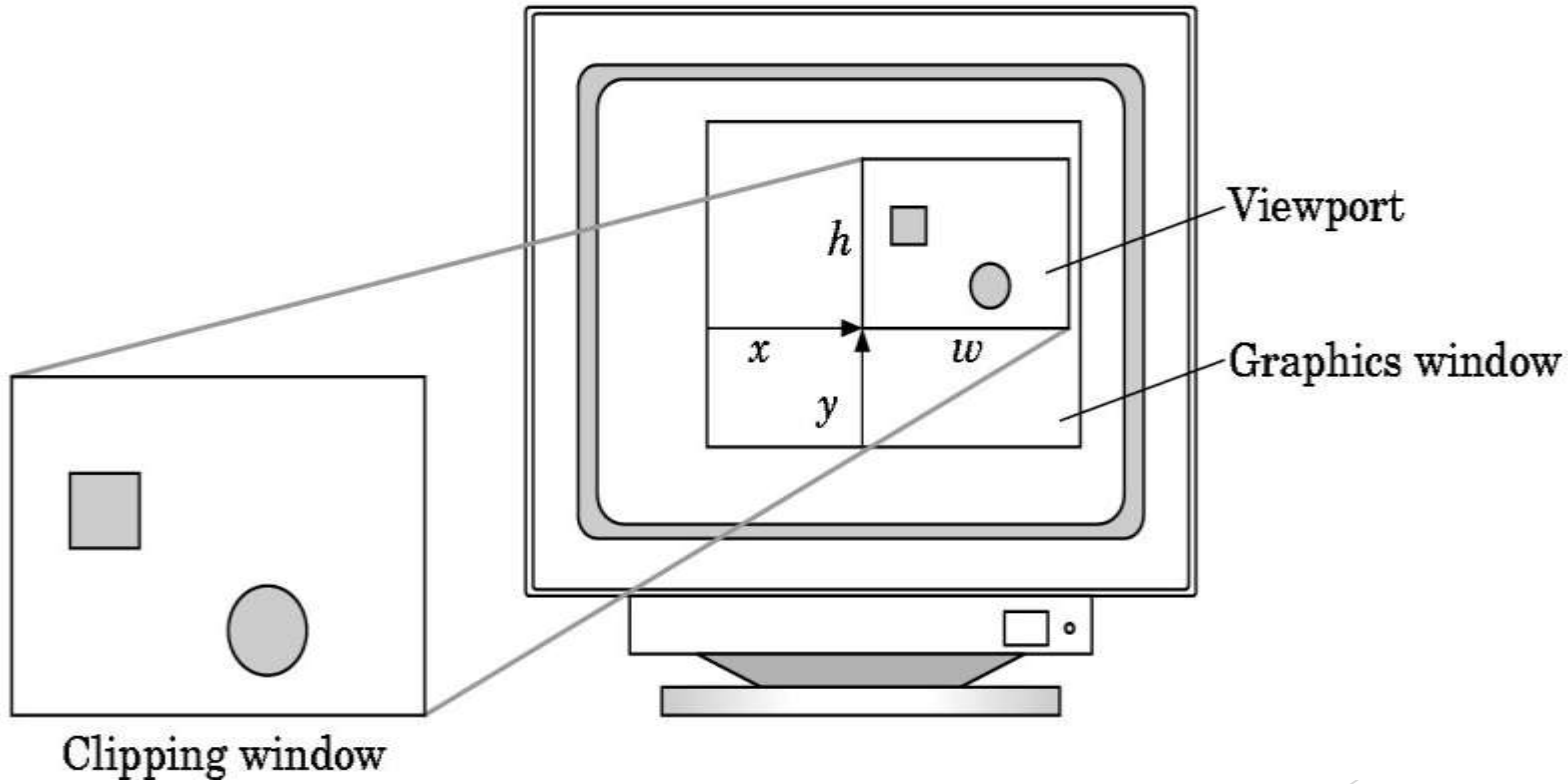
```
glViewport(x,y,w,h)
```

- It defines a pixel rectangle in the window into which the final image is mapped.

View Ports

- **void glViewport(Glint x, Glint y, Glsizei w, Glsizei h)**
  (x,y)-Lower left corner of view port, w:h is the aspect ratio

- By default, the initial viewport values are(0,0,Window W , Window H)



- The aspect ratio of a viewport should generally equal the aspect ratio of the view volume.

- **If the aspect ratio of the viewing rectangle, specified by glOrtho, is not the same as the aspect ratio of the window specified by glutInitWindowSize.**

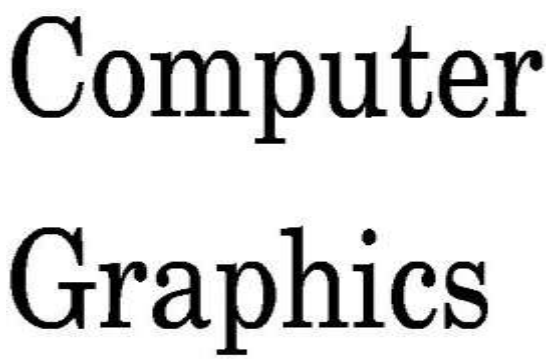
- The independence of the object, viewing and workstation specifications can cause undesirable side effects.

- If they differ, the projected image will be distorted as it is mapped to the viewport.

Text

There are two forms of text:
stroke and **raster.**

The stroke text is constructed the same way as other graphics primitives.

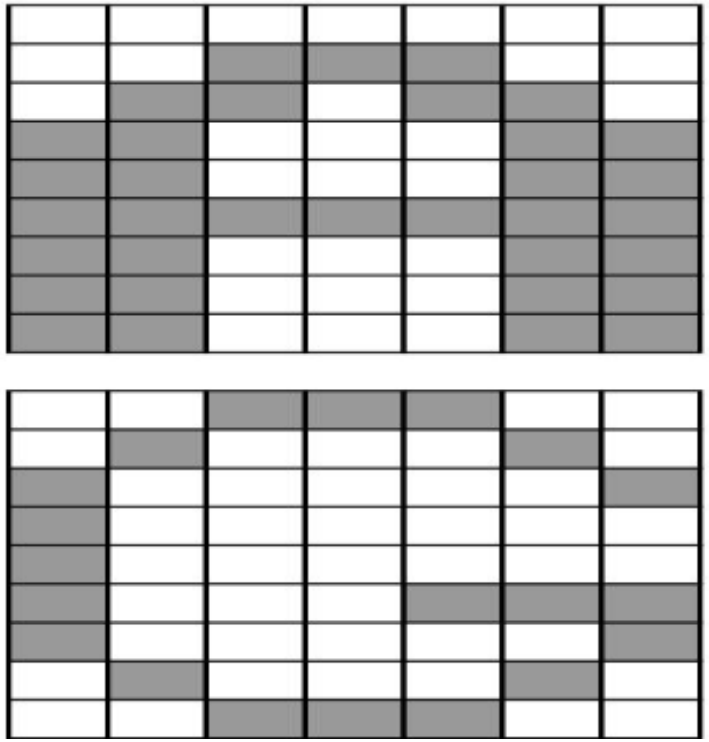


We use vertices to define line segments or curves that outline each character.

- Advantages of having **stroke text** are:
  - It can be defined to have all the details of any other object,
  - It can be manipulated by our standard transformations, and viewed like any other graphical primitive.
  - A character can be defined once and can be transformed to obtain a desire size or orientation.
  - Postscript fonts are good examples for this type of text.

Text – cont.

Raster text is simple and fast. Characters are defined as rectangles of bits called **bit blocks**. A raster character can be placed in the frame buffer rapidly by a **bit-block-transfer (bitblt)**.



If we use bitblt, then we can modify the contents of the frame buffer directly.

The size of characters can be increased by replicating or duplicating pixels.

Due to the fact that raster characters are often stored in the ROM, this may create a problem with the portability of some of the characters.

glutBitmapCharacter(GLUT_BITMAP_8_BY_13,c)

OpenGL does not have a text primitive, however, GLUT provides a few bitmap and stroke character sets that are defined in software and are portable. In the above example, c is the number corresponding to the ASCII characters that we want to display.

Curved Objects

The primitives in our basic set have all been defined through vertices. With the exception of the point type. They all either consist of line segments or use line segments to define the boundary of a region that can be filled with a solid color or a pattern.

We can take two approaches to create a richer set of objects:

1) Use the primitives that we have to approximate curves and surfaces.

Example: to create a circle simply use a polygon of n sides.

More generally, a curved surface can be approximated by a mesh of convex polygons – a **tessellation**.

2) use the mathematical definitions of curved subjects.

Most graphics systems will provide both options. In OpenGL, we can use the utility library GLU for a collection of approximations to common curved surfaces. We also can write our own functions to define new ones.